Résumé

On propose tout d'abord un module de traitement du langage, qui s'appuie sur des méthodes de comparaison sémantique pour interpréter les requêtes en langage naturel de l'utilisateur. La variabilité avec laquelle un locuteur s'exprime constitue en effet l'une des difficultés centrales de ces assistants apprenants. Notre module leur permet de disposer d'un outil de généralisation sur le langage qui s'adapte en continu à l'utilisateur.

Un autre enjeu pour ces agents apprenants est leur capacité à transférer leur connaissance à de nouveaux objets et contextes. On propose alors une série d'architectures d'agents de Deep Reinforcement Learning apprenant à exécuter des tâches exprimées en langage naturel dans des environnements variés. En exploitant le langage comme outil d'abstraction pour représenter les tâches, on montre que lorsque l'environnement est suffisamment structuré, ces agents sont capables de transférer les compétences acquises sur certains objets à de nouveaux.

On développe enfin un cas d'usage dans un environnement domotique. On propose un assistant apprenant qui intègre les systèmes mentionnés précédemment.

Abstract

The rapid development of digital assistants (DA) opens the way to new modes of interaction. Some DA allows users to personalise the way they respond to queries, in particular by teaching them new procedures. This work proposes to use machine learning methods to enrich the linguistic and procedural generalisation capabilities of these systems. The challenge is to reconcile rapid learning skills, necessary for a smooth user experience, with a sufficiently large generalisation capacity. Though this is a natural human ability, it remains out-of-reach for artificial systems and this leads us to approach these issues from the perspective of developmental Artificial Intelligence. This work is thus inspired by the cognitive processes at work in children during language learning.

First, we propose a language processing module, which relies on semantic comparison methods to interpret the user's natural language requests. The variability of a user speech is indeed one of the main difficulties of these learning assistants. We provide them with a generalisation tool to continuously adapt to the user language.

Another challenge for these learning agents is their ability to transfer their knowledge to new objects and contexts. We propose a series of architectures for Deep Reinforcement Learning agents that learn to perform tasks expressed in natural language in various environments. By exploiting language as an abstraction tool to represent tasks, we show that in structured environment, these agents are able to transfer their skills to new objects.

Finally, we develop a use case in a home automation environment. We propose a learning assistant that integrates the systems mentioned above.

Remerciements

Bien qu'une dissertation de thèse soit considérée comme le produit d'un individu, c'est en réalité le fruit de collaborations, d'échanges et d'opportunités. Je souhaite ainsi remercier en premier lieu Peter Ford Dominey, mon directeur de thèse, dont la confiance et l'enthousiasme m'auront été précieux. J'ai eu la chance d'intégrer une équipe pluridisciplinaire aux frontières entre Intelligence Artificielle, Robotique et Sciences Cognitives, et d'apprendre énormément de nos échanges. Merci aussi pour ton soutien et ton intelligence des situations.

Je remercie aussi Christophe Nicolle, co-directeur de cette thèse, pour son accueil lors de notre arrivée à Dijon et sa disponibilité sans faille.

Je remercie bien sûr Jean-Michel Dussoux qui porte ce projet au sein de Cloud Temple, et qui, en réunissant une équipe de recherche d'horizons différents, m'aura permis d'évoluer dans un environnement très riche. La profusion d'idées qui le caractérise ont mené à des discussions que j'ai appréciées et furent une source d'inspiration.

Je remercie évidemment Pierre-Yves Oudeyer qui a joué un rôle des plus actifs dans ce projet. Il est à l'origine d'une collaboration fructueuse et je le remercie pour sa confiance, sa rigueur et son regard critique. Son expérience et son expertise, complémentaires de celles de Peter, furent d'une grande aide.

Merci à Clément Delgrange, mon compagnon de route, qui a ouvert la voie à ces travaux. Son flegme, sa patience et son soutien auront été tout aussi précieux que son amitié et la tasse de café qu'il m'a « laissée ».

Je souhaite aussi remercier Pr. Olivier Sigaud et Dr. Rémi Van Trijp qui ont accepté d'évaluer ces travaux et avec qui je me réjouis de pouvoir échanger.

Enfin, je souhaite remercier tous ceux avec qui j'ai pu échanger et collaborer ces dernières années, et m'ont permis de nourrir ma réflexion, notamment Cédric, Tristan, Takahisa, Jocelyne, Carol, David & David...

Merci à l'équipe du RCL et pour l'atmosphère « familiale » que Peter et Jocelyne ont su y créer. Merci aussi à toute l'équipe du CAPS pour leur accueil malgré l'éloignement de nos thématiques de recherche, avec une mention toute particulière pour les doctorants et l'ambiance joyeuse du labo.

Je termine en remerciant à ma famille et notamment mes parents qui ont su me transmettre leur goût pour la connaissance et la science et dont l'écoute attentive fut d'un réel réconfort.

Mes derniers mots seront pour Mathilde qui aura cru en moi en toutes circonstances. Patiemment, elle m'aura aidé à traverser les périodes de doutes qui jalonnent une thèse. Son soutien est pour moi le plus précieux.

Table des matières

Table des figures

Liste des tableaux

Introduction

Ok Google ! Peux-tu rédiger ma thèse ?

— Anonyme, 2020

Les assistants numériques

Dans un monde toujours plus connecté, même les objets les plus banals sont maintenant capables de communiquer entre eux. Allumer une ampoule, contrôler la température de sa maison, ou vérifier que la porte d'entrée est bien fermée, tout cela devient possible à partir de son smartphone, même depuis l'autre bout du monde. Si ces technologies sont encore en voie de démocratisation, elles ouvrent un champ de tous nouveaux usages. Les enceintes connectées d'Amazon et Google fleurissent dans les salons pour permettre à leurs propriétaires de commander par une simple demande vocale les différents objets connectés de la maison. Plus généralement, ce sont nos modes d'interaction avec les objets qui évoluent. Alors que nos interactions ont longtemps été contraintes par la machine (et les boutons de la télécommande), c'est dorénavant la machine qui s'adapte à notre langage. On a ainsi observé le développement rapide ces dernières années des chatbots qui proposent d'assister un utilisateur (ou un consommateur) dans l'utilisation d'un service numérique. Ces transformations touchent tous les domaines : de la médecine à l'éducation (Underwood, 2017) en passant par les plateformes d'e-commerce (Goksel-Canbek et Mutlu, 2016). Ces assistants numériques prennent des formes très diverses et l'on retiendra pour la suite la définition qu'en donne Hauswald et al. (2015) : « une application qui fournit une assistance en répondant à des questions en langage naturel, en proposant des recommandations ou en effectuant des actions à partir de données comme la voix de l'utilisateur, des images ou des informations contextuelles ».

Personnaliser son assistant numérique

Si la qualité d'interaction avec ces assistants progresse rapidement, elle reste insatisfaisante à plusieurs égards (Claessen et al., 2017). Il n'est pas rare, en effet, que la

réponse obtenue soit sans rapport avec la question posée par l'utilisateur créant un certain sentiment de frustration. De manière générale, ces assistants échouent à remplir une mission dès qu'elle s'écarte des cas d'usages prévus par le fabricant. Il est ainsi très difficile de personnaliser son assistant personnel (une enceinte connectée par exemple) au-delà de la prise en compte de certaines préférences des utilisateurs. Hormis pour des utilisateurs à l'aise avec les outils de programmation, il est notamment impossible de leur enseigner de nouvelles capacités. Cette rigidité dans l'usage nuit à l'adoption de ces systèmes et constitue un domaine d'évolution encore peu exploré. C'est dans ce contexte que l'entreprise Cloud Temple entame en 2015 une collaboration avec l'équipe du Robot Cognition Laboratory (RCL) de l'unité Inserm U1208. L'objectif est de développer un assistant intelligent personnalisable qui puisse apprendre en fonction des besoins de l'utilisateur. Cette collaboration conduit à une première thèse CIFRE : Apprentissage basé sur l'usage en interaction humaine avec un assistant adaptatif (Delgrange, 2018). Dans l'objectif de poursuivre ces travaux, Cloud Temple propose alors une seconde thèse CIFRE, toujours avec l'équipe du RCL (maintenant intégrée à l'unité Inserm U1093) et en partenariat avec l'équipe Flowers de l'INRIA.

Delgrange et al. (2018 ; 2020) propose Jarvis, un assistant numérique pouvant apprendre à effectuer des actions selon les préférences de l'utilisateur. Concrètement, cet assistant peut apprendre la correspondance entre une requête formulée en langage naturel et la séquence d'actions permettant de réaliser la tâche. Cet assistant est disponible via une interface chatbot dans un navigateur web. Il s'appuie sur les démonstrations, que l'utilisateur effectue directement sur l'interface graphique, pour apprendre un ensemble de tâches élémentaires. Il offre ensuite la possibilité à l'utilisateur de réaliser des tâches de plus en plus complexes en composant simplement ces tâches élémentaires. Son fonctionnement est illustré en Figure 0.1 : à partir de la requête de l'utilisateur, l'assistant a appris à rechercher l'information demandée sur Wikipédia et à envoyer un mail avec son contenu au destinataire souhaité. Cet agent est très efficace dans son apprentissage puisqu'une seule démonstration lui permet d'apprendre une nouvelle requête. Les travaux de Delgrange s'inscrivent dans le champ des assistants apprenants par instructions et démonstrations avec l'utilisateur : PLOW (Allen et al., 2001), HILC (Intharah et al., 2017), LIA (Azaria et al., 2016 ; Labutov et al., 2018) ou Sugilite (T. J.-J. Li et al., 2017 ; 2020). Tout comme l'agent de Delgrange, ces différents systèmes apprennent à exécuter des procédures à partir de démonstrations de l'utilisateur sur une interface graphique, des instructions détaillées, et peuvent pour certains demander à l'utilisateur de clarifier ses intentions (Intharah et al., 2017 ; T. J.-J. Li et al., 2020).

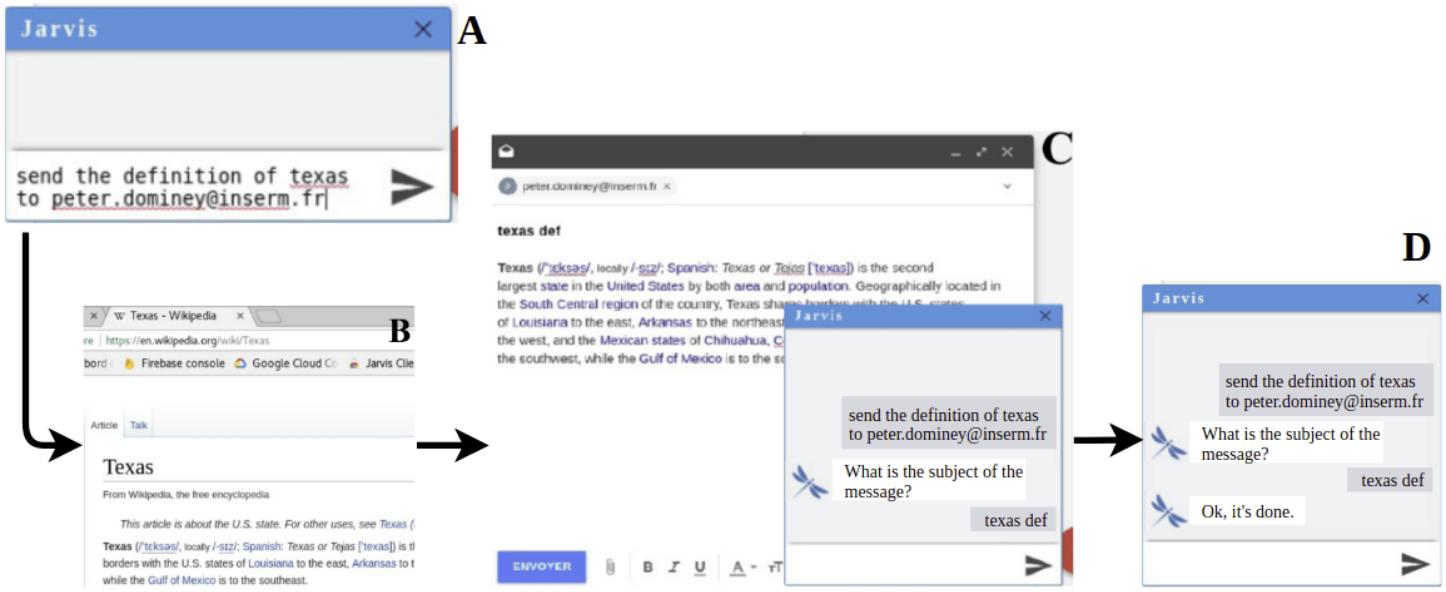


Figure – Exemple d'utilisation de l'assistant JARVIS proposé par Delgrange (2018)

A : Requête à l'agent via une interface chatbot
B : L'agent navigue sur wikipedia pour trouver l'information
C : L'agent rédige l'e-mail et demande à l'utilisateur l'objet à indiquer
D : L'agent indique qu'il a terminé

Les limites de ces assistants personnalisables

Les possibilités de personnalisation offertes par cette famille d'agents se paient cependant par certaines lourdeurs dans l'interaction due à un défaut de généralisation à la fois sur le langage et l'apprentissage des procédures.

Tout d'abord, en ce qui concerne l'interprétation des requêtes de l'utilisateur, la majorité de ces systèmes s'appuient sur des parseurs sémantiques, c'est-à-dire sur une fonction qui associe une commande en langage naturel à une forme logique exécutable par l'assistant. Aussi expressifs soient-ils, ces parseurs limitent, de l'aveu même des auteurs (Azaria et al., 2016 ; Delgrange et al., 2020 ; T. J.-J. Li et al., 2020), la capacité de ces systèmes à comprendre le langage. Ainsi, l'agent de Delgrange et al. (2020) parvient interpréter les requêtes uniquement si elles sont formulées exactement de la même manière que durant l'apprentissage. Cela impose à l'utilisateur de se souvenir en permanence de la forme exacte des requêtes apprises. Cette rigidité dans la compréhension du langage est certainement le frein majeur à l'utilisabilité de ces systèmes. Elle empêche un utilisateur de s'exprimer naturellement et contraint fortement l'interaction.

Ensuite, ces systèmes éprouvent des difficultés pour généraliser l'exécution de procédures apprises dans des contextes légèrement différents. Ainsi, l'agent de Delgrange et al. (2020) n'est pas en mesure, après avoir appris à envoyer un mail depuis un certain service de messagerie, d'inférer la procédure équivalente pour un service de messagerie différent.

A notre connaissance, aucun assistant numérique apprenant par interaction au contact de l'utilisateur n'en est aujourd'hui capable. Pour un humain, au contraire, le transfert de procédures entre différents services est quasiment immédiat, il aura identifié les concepts qui sous-tendent chacune des actions : création d'un mail, choix du destinataire, rédaction du mail et envoi. Avec un nouveau service de messagerie (ou simplement une mise à jour de l'interface), un humain est capable de retrouver ses repères en identifiant rapidement le rôle des différents champs et boutons. Pour un agent apprenant au contraire, il est difficile de généraliser sa perception à différents contextes uniquement sur la base de quelques démonstrations. De manière générale, la perception d'un assistant numérique est un enjeu difficile. Tout récemment, T. J.-J. Li et al. (2020) a proposé une forme de conceptualisation où l'assistant peut apprendre à moduler l'exécution d'une procédure en fonction du résultat d'une autre comme commander une boisson chaude ou froide suivant la température extérieure. Cela ne résout cependant pas la problématique de généralisation de l'exécution de procédures qui constitue un autre handicap quant à l'utilisabilité de ces assistants. Cela force en effet l'utilisateur à répéter des démonstrations très similaires les unes avec les autres.

Enfin, l'apprentissage par démonstration peut se révéler relativement laborieux à l'usage. En effet, l'utilisateur doit éviter toute action parasite ou erreur au risque que l'agent ne les reproduise à chaque fois. Même dans le cas de l'assistant de Delgrange et al. (2020), qui propose d'exprimer une nouvelle procédure comme la composition de procédures déjà apprises plutôt que de la démontrer entièrement, l'exercice requiert de l'utilisateur un certain niveau d'attention et de préparation. Pour bénéficier au mieux de cette fonctionnalité, l'utilisateur doit découper au niveau le plus fin l'apprentissage des procédures élémentaires qu'il démontre et planifier avec soin l'ordre dans lequel il les apprend à l'agent. Cet exercice se rapproche finalement d'un exercice de programmation en langage naturel.

De par leur conception, ces agents rencontrent donc des difficultés à généraliser les comportements appris au contact de l'utilisateur à des situations nouvelles. On identifie deux limites de généralisation :

— *Généralisation sur le langage* : Ces agents ont des difficultés pour faire le lien entre différentes formes verbales qui signifient la même chose ;

— *Généralisation sur les procédures* : Ces agents ont des difficultés pour exécuter une procédure dans une situation qui ne correspond pas exactement à la situation vue durant l'apprentissage.

Ces problématiques de généralisation sont classiques en ML (*Machine Learning*), ou apprentissage automatique, dont l'enjeu est justement de développer des systèmes capables d'apprendre un comportement à partir d'exemples très précis et de généraliser aux exemples

plus ou moins proches. Cette seconde thèse CIFRE est l'occasion pour Cloud Temple d'essayer de répondre à ces limitations en s'appuyant sur des techniques d'apprentissage automatique. Dans cette thèse, on s'attaque donc successivement à ces deux problématiques de généralisation sur le langage (Chapitre 2) et sur les procédures (Chapitres 3 et 4). On souhaite toujours donner la possibilité à des utilisateurs, potentiellement non spécialistes, de personnaliser un assistant numérique pour qu'il puisse effectuer des tâches dans un environnement numérique. Dans ce contexte, on s'intéresse essentiellement aux interactions en langage naturel, l'utilisateur jouant le rôle de professeur pour l'assistant. Ces travaux s'inscrivent à l'intersection de deux domaines : l'Intelligence Artificielle développementale et l'iML (*Interactive Machine Learning*). Dans les deux parties suivantes, on décrit ce que sont ces deux domaines et pourquoi nos travaux s'y inscrivent naturellement.

L'approche développementale en intelligence artificielle

Aujourd'hui, une part importante de la recherche en Intelligence Artificielle consiste à développer des systèmes qui répondent aussi précisément que possible à une question donnée en fonction d'un contexte. Les techniques de ML, et notamment de Deep Learning, ont démontré leur efficacité dans un grand nombre de ces tâches. Aussi appelé apprentissage automatique ou statistique, elles permettent de détecter des régularités statistiques dans de larges ensembles de données et d'inférer les règles qui permettent de répondre à une question. Plutôt que de programmer directement les règles qui permettrait de répondre à la question posée (comme pour les systèmes experts), le système est doté d'outils statistiques (un modèle) pour les déterminer automatiquement. Le paradigme de ML le plus courant est l'apprentissage supervisé, il consiste à apprendre à un modèle à répondre à une question en associant une donnée (un exemple) à un label (une réponse). Pendant une phase d'apprentissage, ou d'entraînement, il est exposé à de nombreuses paires exemples-réponses afin d'identifier les clés de décision. En reconnaissance d'images, par exemple, on peut développer des systèmes qui, après avoir vu un grand nombre d'images et le nom de ce qu'elles représentent, savent répondre à la question *Que représente cette image ?*.

Le fait de ne pas encoder explicitement les règles nécessite du concepteur moins de connaissances spécifiques du domaine et laisse le modèle s'adapter en fonction des données vues pendant l'apprentissage. Dans de bonnes conditions, c'est-à-dire notamment lorsque l'ensemble des exemples d'entraînement est suffisamment vaste et cohérent, les systèmes d'apprentissage automatique démontrent de très bonnes capacités de généralisation. La généralisation est la capacité à répondre à la question dans un contexte qui n'a pas été vu durant l'entraînement. On mesure d'ailleurs la performance d'un système de ML par sa capacité de généralisation. C'est une manière de s'assurer que le système a inféré les

bonnes règles de décision. Sur certaines tâches, ces techniques permettent d'atteindre un niveau de performances exceptionnel, supérieur à celui des humains. L'université de Stanford a ainsi développé un logiciel capable de détecter des mélanomes cancéreux avec plus de précision que des dermatologues (Esteva et al., 2017).

Ces systèmes sont cependant très dépendants de la question, du type de contexte pour lesquels ils ont été conçus ainsi que de la disponibilité et de la qualité des données. Alors que les systèmes experts étaient dépendants des règles écrites par le concepteur, les modèles de ML sont dépendants des données utilisées pendant l'entraînement. Si le contexte change, c'est-à-dire si la distribution des données évoluent par rapport à celles utilisées pour l'entraînement, les performances de ces systèmes ne sont pas garanties, il faut alors ré-entraîner le modèle sur de nouveaux exemples. De plus, il est rare que les modèles puissent répondre à plus d'une question à la fois et un nouveau modèle est nécessaire dès que l'on modifie la question posée. Ces systèmes manquent donc d'adaptabilité à l'évolution du contexte.

Dans le cadre d'assistant apprenant par interaction avec des utilisateurs, on souhaite au contraire disposer de systèmes non spécialisés, multi-tâches et robustes aux évolutions du contexte (Laird et al., 2017). Une approche consiste notamment à concevoir des systèmes, qui une fois placés dans un environnement, se développent de manière autonome au fur et à mesure de leurs interactions avec l'environnement. Ils sont en quelque sorte façonnés directement par leur expérience. On qualifie cette approche de *développementale* (Asada et al., 2009) car elle est inspirée par, et présente certaines similarités avec le développement biologique : à sa naissance, un enfant ne dispose pas de connaissance ou de savoir-faire spécifique, mais il est doté des outils nécessaires pour les acquérir (Weng et al., 2001). En fonction de son contexte personnel et de ses interactions, il développe tout au long de sa vie un ensemble de compétences adaptées aux problématiques qu'il rencontre. Turing (1950) insistait déjà sur l'intérêt d'imiter le cerveau de l'enfant :

> « Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's ? If this were then subjected to an appropriate course of education one would obtain the adult brain. Our hope is that there is so little mechanism in the child brain that something like it can be easily programmed. (Turing, 1950) »

Sans pour autant prétendre reproduire le fonctionnement cognitif d'un jeune enfant, l'objectif de cette approche est donc de concevoir des systèmes qui, à la manière d'un enfant, démarrent non pas avec les solutions aux problèmes qu'ils vont rencontrer, mais avec la capacité d'apprendre à les résoudre. L'autonomie, la capacité d'apprendre tout au long de leur existence (*life-long learning*) (Weng et al., 2001 ; Silver et al., 2013) et celle de s'adapter rapidement sont des caractéristiques importantes de ces systèmes. La philosophie

derrière un assistant numérique apprenant est parfaitement compatible avec cette approche. En effet, on souhaite disposer d'un système capable de s'adapter à l'environnement dans lequel il est employé (environnement de services numériques, domotique, éventuellement une combinaison des deux, etc.), aux utilisateurs avec lesquels il interagit et à la manière dont ces derniers évoluent.

C'est en ce sens que l'Intelligence Artificielle développementale constitue un bon cadre d'étude pour notre problématique. Ce champ de recherche multidisciplinaire, qui mêlait initialement robotique et psychologie développementale, explore le développement de systèmes intelligents autonomes. Elle s'est construite depuis la fin du XX^{e} siècle autour de deux motivations : développer des systèmes intelligents plus autonomes et fournir des outils pour modéliser les processus cognitifs des agents incarnés (*embodied agents*) (Lungarella et al., 2003). La robotique développementale s'inscrit largement dans une démarche biomimétique où les allers-retours entre les systèmes biologiques et artificiels permettent à la robotique comme à la psychologie développementale de faire des avancées (Pfeifer et al., 2007 ; Asada et al., 2009). La notion d'*embodiment* joue un rôle central dans cette discipline : les processus cognitifs sont envisagés en interaction avec les systèmes de perception, qui fournissent les représentations nécessaires (Pfeifer et al., 2007). À la différence des approches *analytiques* qui étudient les phénomènes en isolation les uns des autres, l'approche *synthétique*, qui consiste à modéliser explicitement l'objet d'étude, est privilégiée en robotique développementale (Asada et al., 2009). La robotique développementale conçoit l'étude des processus cognitifs au sein d'un système intégré, comprenant le corps physique du système, les modes de représentations des perceptions et les interactions (éventuellement sociale) dans l'environnement du robot. Aujourd'hui, la démarche développementale dépasse le cadre de la robotique pour s'étendre plus largement au ML et à l'Intelligence Artificielle (Smith et Slone, 2017 ; Oudeyer, 2019 ; Kiani et al., 2020)

Interactive Machine learning

Le terme iML (*Interactive Machine Learning*) a lui aussi fait son apparition il y a une vingtaine d'années. Partant du constat que les utilisateurs des modèles de ML étaient plus souvent experts du domaine d'application que des algorithmes d'apprentissage (Amershi et al., 2014), la motivation initiale était de replacer l'utilisateur-expert au centre du processus d'apprentissage (Fails et Olsen, 2004). L'objectif était de donner la possibilité à des non-spécialistes de concevoir leur propre modèle d'inférence. Les premières approches proposent à l'utilisateur de construire son propre classifieur en visualisant les données et en traçant les frontières séparant les différentes classes, puis le système reconstruit l'arbre de décision

correspondant (Ankerst et al., 1999 ; Ware et al., 2001). L'idée d'exploiter la connaissance spécifique d'un humain pour améliorer l'apprentissage d'un modèle statistique est assez largement applicable. L'iML est ainsi un domaine transverse et plusieurs définitions ont été proposées. Pour certains, la présence de l'humain dans le processus d'apprentissage est essentiel (Amershi et al., 2014). À l'inverse, on préfère la définition moins contrainte de Holzinger (2016, p. 1) qui ne suppose pas qu'un humain soit impliqué dans le processus d'apprentissage :

> Des algorithmes qui peuvent interagir avec d'autres agents et peuvent optimiser leur apprentissage grâce à ces interactions et où les agents peuvent éventuellement être humains.

En effet, on peut envisager notamment deux cas dans lesquels le paradigme d'iML est pertinent sans qu'il n'implique un humain. Tout d'abord, certains systèmes multi agents apprennent à collaborer via les interactions qu'ils ont entre eux (F. Li et Bowling, 2019 ; Lowe et al., 2020). Ensuite, la collecte de données auprès d'un véritable humain est parfois trop coûteuse et il est préférable de s'appuyer sur un agent qui simule le retour d'un humain. Certains algorithmes actuels sont en effet trop gourmands en données pour être appris dans des conditions d'interactions réelles (Chevalier-Boisvert et al., 2019 ; Lowe et al., 2020). Enfin, dans l'optique d'étudier un phénomène d'apprentissage par interaction et de pouvoir itérer rapidement, il est souvent plus facile de s'appuyer sur un oracle simulé plutôt que sur un humain (R. Kaplan et al., 2017 ; Misra et al., 2018 ; H. Yu et al., 2018). Réintégrer l'humain dans l'apprentissage fait partie des étapes suivantes (Chevalier-Boisvert et al., 2019).

Plusieurs revues ont été réalisées sur l'iML sous des angles différents, en fonction de la manière dont le système est envisagé : un modèle à entraîner ponctuellement ou un agent avec lequel interagir dans la durée. Dans la première catégorie, la majorité de la recherche se concentre sur la visualisation interactive de données et la calibration des modèles (Sacha et al., 2017 ; L. Jiang et al., 2019). L'objectif, fidèle à la motivation originale du domaine, permet à l'utilisateur de personnaliser un modèle et l'apprentissage en interaction est donc concentré pendant la phase de personnalisation. D'autres travaux exploitent les techniques d'Active Learning qui permettent à un modèle de solliciter l'utilisateur pour labelliser certaines données ou obtenir une démonstration (Settles, 2009). Misra et al. (2018) propose par exemple un modèle apprenant à répondre à des questions sur une image en interrogeant un oracle. L'agent est doté d'un « budget » de questions qui limite la quantité de retours qu'il peut recevoir.

Dans la seconde catégorie, sur laquelle on se concentre par la suite, le système est envisagé comme un agent avec lequel l'apprentissage et les interactions se font dans la durée. Les différents assistants apprenants évoqués jusqu'ici s'intègrent naturellement

à cette catégorie (Azaria et al., 2016 ; Delgrange et al., 2020 ; T. J.-J. Li et al., 2020). Laird et al. (2017) en proposent une revue des agents apprenant en interaction, restreinte aux systèmes qui sont capables d'apprendre une nouvelle tâche à partir d'une unique interaction (*one-shot learning*), excluant de facto la majorité des systèmes s'appuyant sur du Deep Learning. Pourtant, de plus en plus de travaux, notamment en Deep RL (*Deep Reinforcement Learning*), s'intéressent à un paradigme dans lequel un agent s'adapte à un utilisateur (Shah et al., 2016), ou est guidé dans son apprentissage par un oracle (Cruz et al., 2014 ; Cruz et al., 2015 ; R. Kaplan et al., 2017 ; Misra et al., 2018 ; H. Yu et al., 2018). Dobrovsky et al. (2017) propose aussi un cadre d'apprentissage par renforcement interactif pour les personnages simulés dans des jeux sérieux (*serious game*).

On se place donc à l'intersection entre le *Machine Learning* développemental et *Interactive Machine Learning* On décrit maintenant l'organisation des travaux de la thèse.

Organisation des travaux

Chapitre 1 - Perspectives croisées IA et sciences cognitives sur le langage naturel

On commence par passer en revue différentes approches de traitement du langage naturel. On montre les limites d'une approche qui ne s'appuie que sur des sources textuelles pour interpréter et acquérir la sémantique de la langue. On s'intéresse aussi aux processus cognitifs et sociaux qui conduisent le jeune enfant à apprendre le langage. Il existe en effet de nombreux parallèles entre l'acquisition du langage par un enfant et les problématiques que l'on rencontre en Intelligence Artificielle. On détaille une série de travaux, notamment ceux de Vygotsky, Bruner, Nelson et Tomasello qui ont influencé la suite de nos travaux.

Chapitre 2 - Apprentissage en interaction et détection de l'intention de l'utilisateur

Dans ce chapitre, on cherche à proposer une solution à l'une des limites des assistants apprenants que nous avons identifiée précédemment : la difficulté de généraliser sur le langage et à gérer la variabilité des formes verbales employées par un utilisateur. Le langage possède une richesse sémantique et syntaxique qui permet à un locuteur d'exprimer de multiples manières une même requête et l'on souhaite doter des agents apprenants de la capacité à les comprendre. On propose un système de détection interactif de l'intention de l'utilisateur, AidMe, qui repose sur une comparaison de la similarité sémantique entre deux phrases pour interpréter une nouvelle requête de l'utilisateur en fonction de

celles déjà connues. AIDME peut être intégré à un agent apprenant par interaction pour remplacer une fonction de traitement du langage naturel. Ces travaux ont donné lieu à la publication d'un article lors de la conférence Intelligent User Interface en 2020 (Lair et al., 2020).

Chapitre 3 - Le langage, outil d'apprentissage pour un agent autonome

On s'attaque ensuite au second défaut de généralisation des assistants apprenants : la généralisation sur les procédures. Le RL (*Reinforcement Learning*) est un paradigme d'apprentissage adapté à cette problématique. Il consiste à permettre à un modèle d'interagir dans un environnement et d'apprendre par essai-erreur. Les environnements dans lesquels évoluent les assistants numériques posent cependant des difficultés non triviales, notamment pour modéliser les perceptions et les actions. On se tourne vers des environnements classiques utilisés en RL pour développer un agent apprenant à effectuer des tâches à partir d'interactions langagières simples avec un oracle.

Ces travaux ont été menés en collaboration avec Cédric Colas et Tristan Karch, de l'équipe INRIA Flowers de Pierre-Yves Oudeyer. Nous avons proposé LE2 et IMAGINE, qui, au-delà d'apprendre à suivre une instruction exploite le langage comme un outil d'apprentissage et de généralisation. Les idées exploitées dans ces travaux découlent directement des théories développées en Sciences Cognitives notamment par Vygotsky (voir Chapitre 1). Ces travaux ont conduit à la présentation de résultats préliminaires lors du workshop ViGIL (Visuallly Grounded Interaction and Language) à NeurIPS en 2019 (Lair, Colas et al., 2019) et la publication d'un article à NeurIPS 2020 (Colas, Karch, Lair et al., 2020).

Chapitre 4 - Application dans un environnement domotique réaliste

On exploite les résultats obtenus dans le développement de LE2 et IMAGINE pour concevoir un agent adapté à un environnement domotique et tenter de résoudre aux problématiques qui y sont spécifiques. Il est notamment nécessaire de modéliser des espaces d'observations et d'actions qui soient exploitables par un agent de RL. Contrairement à la robotique simulée, dans un environnement numérique, les perceptions de l'agent et d'un utilisateur sont différentes : l'agent perçoit les fonctions d'une API (*Application Programming Interface*) qui lui permettent d'agir et l'utilisateur perçoit leurs effets dans le monde physique. Cette asymétrie dans la perception constitue une difficulté surtout dans le cadre d'un apprentissage en interaction où l'utilisateur doit guider l'agent.

On développe un environnement de domotique réaliste inspiré d'OpenHAB, un standard de la gestion d'appareils connectés employé par Cloud Temple, permettant de simuler l'interaction d'un agent avec une variété d'objets connectés et en présence d'un oracle. On propose aussi ILEAD, un agent adapté à ces espaces d'observation et d'action atypiques.

On propose enfin un protocole pour intégrer AIDME, ILEAD et JARVIS, l'assistant développé par Delgrange et al. (2020), permettant de tirer le meilleur parti des différentes approches.

Chapitre 1

Perspectives croisées IA et sciences cognitives sur le langage naturel

Qui dit homme dit langage, et qui dit langage dit société.

— Claude Lévi-Strauss
Tristes Tropiques, 1955

Sommaire

1.1 Introduction

Le traitement du langage naturel fait l'objet d'une recherche active en Intelligence Artificielle depuis de très nombreuses années. Cambria et White (2014) propose une revue de différentes approches qui sont investiguées par la communauté, et notamment l'utilisation d'ontologie, de pattern matching, etc. Dans ce chapitre, nous nous concentrons sur les approches qui sont plus directement en lien avec la problématique des assistants numériques dialogiques, apprenants ou non, ainsi que le développement récent des modèles de langage neuronaux (Mikolov et al., 2013).

Nous débutons ainsi en exposant le formalisme employé pour traiter l'interprétation des requêtes de l'utilisateur d'un système de dialogue. Cette tâche est modélisée sous la forme d'une double tâche de détection de l'intention de l'utilisateur et d'identification des paramètres de la requête, aussi appelée *slot filling* ou *slot tagging* (Partie 1.2).

Les développements récents des modèles de langage (Mikolov et al., 2013), qui fournissent une représentation de la sémantique des mots d'une langue à partir d'un large corpus de texte, ont permis d'importants progrès dans de nombreuses tâches classiques en NLP (*Natural Language Processing*) (Partie 1.3). Ces modèles, entraînés uniquement sur des données textuelles, manquent cependant d'un certain sens commun (Bisk et al., 2020) qui peut s'expliquer par l'absence d'expérience sensorielle du langage.

En robotique, au contraire, la problématique du traitement du langage naturel s'est naturellement concentrée sur la capacité des robots à relier la sémantique de la langue avec leurs perceptions sensorimotrices de l'environnement (Partie 1.4). L'ancrage du langage naturel dans une expérience sensorimotrice (et éventuellement sociale) permet de rapprocher un système artificiel de la manière dont les enfants apprennent le langage et le domaine a bénéficié de l'influence des théories en Sciences Cognitives (Cangelosi et Schlesinger, 2018).

Nous concluons enfin ce chapitre en se tournant vers la psycholinguistique développementale, une discipline de la psychologie qui étudie l'acquisition du langage par les enfants (Partie 1.5). Nous montrons que les problématiques de cette discipline trouvent naturellement leur écho en Intelligence Artificielle où elles y sont déjà étudiées, parfois en ignorant les travaux en Sciences Cognitives. Nous évoquons notamment pourquoi l'environnement social et l'apparition d'une attention partagée sont primordiales pour l'acquisition du langage par l'enfant (Bruner, 1983 ; Tomasello, 1999). Nous nous attardons aussi sur la formation des concepts et l'usage du langage comme outil cognitif par l'enfant (Vygotsky, 1978 ; Nelson, 1996). La suite de nos travaux sera ainsi en partie inspirée par les théories de Vygotsky.

1.2 Traitement du langage naturel par les assistants numériques dialogiques

Les assistants numériques dialogiques, ou assistants conversationnels, sont des systèmes avec lesquels un utilisateur peut échanger en langage naturel. Ils peuvent prendre des formes diverses comme des assistants vocaux ou des chatbots.

1.2.1 Module de NLU : détection de l'intention et des paramètres

Phrase	Trouve	-	moi	les	vols	de	demain	entre	Rome	et	Paris
Label	O		O	O	O	O	B.date	O	B.dép.	O	B.arr
Pattern	Trouve	-	moi	les	vols	de	DATE	entre	DEP	et	ARR

Table 1.1 – Exemple d'analyse d'une requête utilisateur
Trouve-moi les vols de Rome à Paris demain
Intention : recherche de vol

Le traitement du langage naturel (NLU (*Natural Language Understanding*)) est évidemment au cœur des problématiques de ces assistants et constitue l'étape incontournable en vue de répondre intelligemment à l'utilisateur. Il s'agit pour un assistant conversationnel d'interpréter ce que dit l'utilisateur en transformant une phrase en langage naturel dans une représentation utilisable par l'assistant. Cette phase s'effectue en deux étapes : tout d'abord détecter l'intention de l'utilisateur puis identifier les différents mots de la phrase. Cette étape est d'autant plus importante que c'est à partir de l'interprétation faite par le module de NLU que l'assistant décidera d'une éventuelle action à exécuter et de sa réponse.

La détection de l'intention est traditionnellement formulée comme une tâche de classification dont l'objectif est de déterminer la classe d'intention parmi un ensemble d'intentions prédéterminées (Schapire et Singer, 2000). Le *slot tagging* consiste à attribuer un tag (un label) à chacun des mots de la phrase, il s'agit d'une tâche plus difficile résolue généralement par l'utilisation de réseaux récurrents (Yao et al., 2013). Lorsque l'agent doit exécuter une action, l'étape de *slot tagging* permet notamment d'identifier les paramètres de l'action. Nous exploiterons ce formalisme par la suite et notamment dans le Chapitre 2.

Le Tableau 1.1 illustre, par un exemple inspiré du Airline Travel Information System, le résultat que produit un système de NLU classique dans le format Inside-Outside-Beginning (Ramshaw et M. P. Marcus, 1995). L'utilisateur souhaite connaître les horaires pour des vols et le système identifie l'origine, la destination et le jour afin de répondre correctement à

la requête. Le système identifie ainsi l'intention (*recherche de vol*) et extrait les arguments liés à cette intention.

1.2.2 Différents types d'agents conversationnels

Les agents conversationnels neuronaux

Aujourd'hui, la plupart des agents conversationnels s'appuient sur des techniques de Deep Learning et constituent ce qu'on appelle les agents conversationnels neuronaux (Gao et al., 2019). Nous distinguons deux formes d'assistants dialogiques.

Tout d'abord, les agents de type chatbots qui sont conçus pour entretenir une conversation avec un utilisateur sur un ensemble varié de sujets (Vinyals et Le, 2015 ; Ram et al., 2017). Ils sont évalués sur leur capacité à maintenir l'engagement de l'utilisateur dans la discussion. Ils s'appuient pour l'essentiel sur les approches dites *sequence to sequence (seq2seq)* : un unique modèle traite la phrase de l'utilisateur comme une séquence de mots et produit une réponse sous la forme d'une séquence de mots (Sutskever et al., 2014). Ces systèmes sont entraînés sur d'importants corpora et doivent permettre à un modèle de dialoguer de manière naturelle avec un utilisateur sur une large variété de sujets (Ram et al., 2017). Certaines approches récentes suggèrent de conditionner la génération de la réponse par le graphe de connaissance pour que l'agent soit capable de répondre intelligemment dans de nombreux domaines (Moon et al., 2018).

D'autres agents conversationnels ont pour objectif d'aider l'utilisateur à réaliser des tâches particulières (*task-oriented dialog system*) comme réserver un vol, un restaurant, etc. et répondre à ses questions (Peng et al., 2020 ; Y. Zhang et al., 2020 ; Z. Zhang et al., 2020). Ils sont évalués sur leur capacité à remplir la tâche assignée ou à apporter une réponse correcte. C'est sur ce type d'assistants, plus en accord avec notre problématique, que l'on se concentre ensuite. Ces agents ont besoin de faire appel à une base de connaissance, une base de données externe par exemple, pour répondre aux requêtes de l'utilisateur. Ils adoptent une architecture généralement plus modulaire constituée d'un modèle de NLU pour identifier l'intention de l'utilisateur, d'un manager de dialogue qui maintient le contexte courant et exécute une requête sur la base de connaissance et d'un modèle de génération de langage qui répond à l'utilisateur (Z. Zhang et al., 2020). De récents travaux ont montré qu'une architecture non modulaire *seq2seq* pouvait aussi produire d'excellents résultats (Hosseini-Asl et al., 2020) mais ces modèles semblent souffrir d'un manque de stabilité (Z. Zhang et al., 2020).

Les agents conversationnels grand public

Les agents conversationnels que l'on retrouve dans le commerce s'appuient sur une combinaison entre modèles statistiques, notamment pour détecter l'intention de l'utilisateur, et règles de dialogue, pour gérer la discussion courante (Gao et al., 2019). En effet, si les approches purement neuronales présentées précédemment permettent de disposer d'un agent couvrant plusieurs domaines à la fois, ils sont en pratique difficiles à utiliser dans un produit grand public. Ils ne sont pas facilement configurables ni leurs réponses interprétables (Gao et al., 2019).

Il existe aussi de nombreuses plateformes qui permettent de prototyper très rapidement un chatbot en exploitant les modèles de Google, Amazon, Microsoft, etc. en définissant un ensemble d'intentions et en fournissant quelques exemples de phrases pour chacune des intentions. Ces plateformes proposent ainsi un module de NLU quasiment clef en main. Ces assistants sont conçus pour fonctionner dans un domaine particulier et pour réaliser des tâches prédéfinies. Cela permet à leurs concepteurs de contrôler les contextes dans lesquels ont lieu les interactions et donc de maintenir une certaine qualité d'interaction. Allen et al. (2001) définit cinq niveaux de gestion de dialogues adaptés à la complexité des tâches à accomplir. Les agents grand public peuvent être considérés au niveau 3 (« *sets of contexts* ») : ils sont capables de gérer l'interaction dans un ensemble de contextes prédéfinis. Ces outils permettent de définir l'ensemble des intentions qui seront reconnues par l'agent. De par leur conception, ces assistants sont donc limités aux seules intentions prédéfinies et ne peuvent s'adapter ni apprendre des interactions avec l'utilisateur.

Les assistants apprenants

Dans le cadre des assistants personnels et notamment ceux apprenants en interaction auprès de l'utilisateur, le module de NLU n'exploite pas d'apprentissage automatique mais repose sur des ensembles de règles écrites pour le système ou sur un parseur standard. Elles permettent ainsi de disposer d'un module de NLU opérationnel dès le début de l'interaction avec l'utilisateur.

PLOW, développé par Allen et al. (2007), propose ainsi de convertir les phrases en langage naturel en des formes logiques générés à partir d'un ensemble de règles et d'un filtrage lié au contexte (Allen et al., 2001). Cette méthode requiert un important effort de la part de développeur notamment pour concevoir l'ensemble des règles et n'est pas adapté à un langage non contraint comme celui d'un utilisateur. Pour éviter un système basé sur l'implémentation de règles grammaticales au sein du système, des systèmes comme LIA (Azaria et al., 2016) propose de s'appuyer sur un parseur (*Combinatory Categorical Grammar*, CCG (Steedman, 1996)) permettant d'associer à une phrase en langage naturel à une fonction primitive. Delgrange et al. (2020) s'appuie sur un formalisme proche des

grammaires de construction (Goldberg, 2003). Les constructions sont inférées en fonction de la démonstration effectuée par l'utilisateur. À chaque nouvelle requête de l'utilisateur, l'agent tente de faire coïncider la requête sur l'une des constructions connues pour en inférer automatiquement l'intention et les paramètres.

On peut considérer que ces agents se situent au niveau 2 (« *frame based* ») sur l'échelle proposée par Allen et al. (2001). Il est ainsi difficile, voire impossible, pour les assistants apprenants d'interpréter des requêtes en langage naturel lorsque l'utilisateur n'utilise pas une requête dans une forme déjà apprise.

Ces agents sont en difficulté pour s'adapter à la variabilité des formes verbales. Le langage possède pourtant une richesse sémantique et syntaxique qui permet à un locuteur d'exprimer de multiples manières une même requête. De ce point de vue l'adaptabilité d'un système semble antagoniste à la fluidité de l'interaction. Dans le Chapitre 2, nous proposons AIDME, un module de NLU adapté aux agents apprenants, qui permet d'apprendre dynamiquement à interpréter les formes verbales employées par l'utilisateur en fonction des intentions déjà apprises. AIDME s'appuie sur des techniques de ML et notamment sur les modèles de langage neuronaux dont nous détaillons le principe dans la partie suivante.

1.3 Représentation langage naturel en Deep Learning

1.3.1 Les modèles de langage

Les progrès récents en NLP sont notamment dues à l'utilisation des techniques de Deep Learning pour représenter et raisonner sur le langage. Les modèles de langage comme word2vec (Mikolov et al., 2013), Glove (Pennington et al., 2014), BERT (Devlin et al., 2019), GPT 1 à 3 (Radford et al., 2018 ; Radford et al., 2019 ; Brown et al., 2020), etc. y ont largement contribué. Ces modèles de langage, permettent de projeter la sémantique d'une langue dans des espaces de dimension finie. Ces modèles encodent chaque mot par un vecteur, manipulable par des fonctions numériques classiques, et notamment les modèles d'apprentissage automatique. L'utilisation de ces modèles donne des résultats impressionnants sur de nombreuses tâches de NLP (He et al., 2021) et applications nécessitant un traitement du langage.

Une sémantique basée sur la co-occurence

Étant donné un corpus de textes, un modèle de langage est entraîné à calculer la distribution de probabilité d'un mot conditionnellement à un contexte (un ensemble de mots). La version la plus simple de cette tâche consiste, à partir d'un début de phrase, à prédire le mot suivant en fonction des mots qui précèdent. Par exemple, la phrase « Bob

mange une glace » sera présentée au modèle sous la forme « Bob X » et « Bob mange X », il doit deviner que X est successivement « mange » et « glace ». Plus précisément, il cherche la distribution de probabilité du mot caché par X. Cette tâche de prédiction conduit le modèle à estimer la probabilité de présence d'un mot en fonction des autres mots présents dans les contextes similaires. Cette méthode s'appuie sur l'hypothèse que la représentation sémantique des mots découlent naturellement de la distribution des mots co-occurrents, c'est-à-dire qui apparaissent dans le même contexte (Landauer et Dumais, 1997). Évidemment cette hypothèse ne tient que sur de grands corpus de texte et pas sur un exemple unique comme celui présenté.

Du point de vue technique, un modèle neuronal est entraîné de manière auto-supervisée (auto-régressive) sur un large corpus de textes. Les représentations apprises par le modèle sont ensuite exploitées pour associer à chaque mot un vecteur qui encode sa sémantique. Plus précisément, le modèle commence par associer au vocabulaire un espace vectoriel de dimension égale à la taille du vocabulaire et dans lequel chaque mot est associé à un vecteur de la base canonique. On dénomme cette représentation *one-hot* et le premier mot du vocabulaire est par exemple associé au vecteur $(1, 0, ...0)$. Étant donné le i-ème mot w_i du vocabulaire, le modèle applique une transformation matricielle *apprise* A au vecteur one-hot X_i correspondant à w_i. Le vecteur AX_i obtenu correspond ainsi à la représentation apprise du mot w_i. X_i étant le i-ème vecteur de base, AX_i s'obtient naturellement comme la i-ème colonne de A. Cette transformation matricielle A est souvent nommée *embedding layer*. Cette méthode revient donc à compresser la représentation sparse one-hot des mots du vocabulaire dans une représentation dense, appelée *embedding*, selon une transformation linéaire bien choisie (i.e. apprise). Les progrès effectués par les modèles successifs s'expliquent essentiellement par le raffinement obtenu en complexifiant les architectures des modèles, la tâche de prédiction et surtout en intégrant des modules attentionnels capable de filtrer l'information pertinente (Graves et al., 2014 ; Bahdanau et al., 2015 ; Luong et al., 2015 ; Vaswani et al., 2017).

Etude de ces modèles sous l'angle Sciences Cognitives

La possibilité de compresser la sémantique sous une forme dense a rapidement interrogé sur les liens éventuels entre les représentations apprises par ces modèles et les représentations utilisées par le cerveau. Plusieurs séries de travaux ont ainsi essayé d'évaluer les capacités linguistiques de modèles comme les LTSM (Linzen et al., 2016) ou BERT (Ettinger, 2020) selon des standards de linguistiques. À l'inverse, les modèles de langage ont aussi été utilisés pour interpréter une activité cérébrale relative au traitement du langage naturelle (Dehghani et al., 2017 ; Schwartz et al., 2019 ; Goldstein et al., 2020). Enfin, ces modèles ont aussi été employés pour modéliser la dynamique neuronale de

compréhension d'un discours chez l'homme (Ettinger et al., 2016 ; Uchida et al., 2021). Nous avons ainsi montré que l'utilisation d'un *embedding* simple simple comme *word2vec* permettait de modéliser l'intégration de l'information au fur et à mesure de la lecture d'un texte (Uchida et al., 2021).

Les modèles de langage héritent des biais des corpus d'entraînement

L'une des limites rapidement atteintes par ces modèles, et notamment pour les assistants digitaux, est leur niveau de sens commun. On désigne par sens commun un ensemble de connaissances implicites que possèdent les locuteurs d'une langue et qui leur permettent de se comprendre sans avoir besoin d'expliquer certaines prémisses. Il s'agit d'un contexte commun qui n'apparaît quasiment jamais explicitement dans les textes auxquels sont exposés les modèles de langage. Le « problème du mouton noir » (*Black sheep problem* illustre cette problématique, Daumé III, 2016) : pour un modèle de langage entraîné uniquement à partir de données textuelles, il est pratiquement impossible de comprendre qu'un mouton est généralement blanc. En effet, dans la littérature courante, et notamment anglaise, l'expression *mouton noir* est bien plus courant que *mouton blanc*. Il est très difficile pour un modèle de langage appris selon le principe décrit ci-dessus d'acquérir cette forme de connaissance implicite. La représentation vectorielle d'un mot w est en effet uniquement calculée à partir des mots qui apparaissent dans le contexte de w. La méthodologie de calcul des modèles de langage et l'utilisation de sources textuelles ne permettent pas, à elles seules, d'intégrer ces connaissances de sens commun. Fonder la sémantique de la langue uniquement sur la co-occurence des mots conduit aussi à encoder des biais importants et notamment des biais sexistes ou racistes hérités des données utilisées (Garg et al., 2018). La capacité à raisonner à partir de sens commun (*Commonsense Reasoning*) fait partie des tâches qui restent difficiles pour les modèles de langage (Zellers et al., 2018 ; Bisk et al., 2020) et l'idée que l'on ne peut pas se reposer uniquement sur des corpus de données toujours plus importants gagne de l'intérêt (Klein et Nabi, 2018).

1.3.2 Modèles de langage multimodaux

Pour tenter de résoudre cette difficulté, certains ont proposé d'apprendre un modèle de langage à partir de sources multimodales comme des images ou des sons préalablement labellisés sémantiquement par des humains... (Hill et Korhonen, 2014 ; Lazaridou et al., 2015 ; Wang et al., 2018a ; Wang et al., 2018b ; Tan et Bansal, 2020). Ces modèles peuvent ainsi corréler la sémantique de la langue à une source visuelle ou sonore. L'une des difficultés majeures derrière cette approche est l'évidente difficulté de disposer de corpus

de données multimodaux aussi larges que pour du seul texte. Pour les tâches de génération de langage, certaines approches récentes combinent les modèles de langage avec des graphes de connaissances pour développer des architectures neuronales capable de raisonner sur les deux (Ji et al., 2020).

D'autres travaux s'intéressent à l'apprentissage d'un modèle de langage dans le cadre d'une autre tâche d'apprentissage. Celui-ci n'a alors pas vocation à être ré-employé dans un contexte différent. Les tâches de question-réponse sur des scènes visuelles (VQA (*Visual Question Answering*))(Misra et al., 2018 ; Johnson et al., 2019 ; Z. Yu et al., 2019) ou de navigation guidée (*Vision and Language Navigation*) (Anderson et al., 2018 ; H. Chen et al., 2019 ; Huang et al., 2019 ; Hong et al., 2020) permettent ainsi d'apprendre un modèle de langage spécifique de la tâche cible. L'apprentissage se fait non plus par une tâche de prédiction du mot suivant mais directement par rétropropagation de l'erreur de la tâche principale.

Une dernière approche consiste à permettre à des agents apprenants (souvent des agents apprenants par renforcement) d'apprendre leur propre modèle de langage contextuellement aux expériences vécues dans leur environnement. Il s'agit essentiellement d'*instruction-following agents*, c'est-à-dire des agents qui sont guidés par des instructions fournies en langage naturel avant chaque épisode. Luketina et al. (2019) propose une revue des différentes approches dans ce domaine. Cette approche en phase avec une démarche développementale est celle que l'on adoptera notamment dans les Chapitres 3 et 4.

1.4 Traitement du langage en robotique

1.4.1 *Symbol Grounding Problem*

En robotique, l'idée d'associer le langage avec d'autres perceptions n'est pas nouvelle. En effet, permettre aux robots de nommer et reconnaître les objets de leur environnement a rapidement constitué un objectif assez naturel. Ce problème fait partie d'un problème plus large : le problème d'ancrage des symboles (*Symbol Grounding Problem*, Harnad, 1999). Coradeschi et al. (2013) a proposé une revue des différentes approches et différencie l'ancrage *physique* et l'ancrage *social* du langage. Le premier, celui qui nous intéresse plus spécifiquement ici, désigne l'association des symboles à des objets de l'environnement physique dans lequel un robot évolue (Vogt, 2002). Le second désigne la négociation au sein d'un groupe de robots de la signification des symboles (Steels et Vogt, 1997 ; Cangelosi, 2006) et recoupe les questions d'émergence du langage dans une population d'agents artificiels.

Avant de constituer à proprement parler le langage, la problématique était donc d'abord d'apprendre à associer des symboles (mots) à des objets en fonction des données

perceptuelles du robot, souvent visuelles. Les approches sont très variées et l'on peut donner les quelques exemples suivants. Siskind (1996) propose un système de règles d'apprentissage à partir des « situations croisées » (*cross situational model*) : en reconnaissant le sens d'un mot par recoupement des différentes situations dans lesquelles ce mot est rencontré. Roy (2005) développe une théorie des schémas sémiotiques : une médiation entre le langage, comme acte de parole, la pensée et les perceptions dans le monde physique. Il définit l'*ancrage* comme une tâche de prédiction de l'adéquation entre les perceptions et le sens attribué au langage.

Comme pour l'apprentissage des modèles de langage, l'apprentissage du langage peut être traité non pas comme une fin en soi mais comme une tâche annexe de la tâche d'intérêt, par exemple lorsque le langage est un moyen de donner des instructions à un robot. L'enjeu est alors que le robot exécute ces instructions en fonction de ses perceptions. Tellex et al. (2011) propose ainsi un modèle graphique permettant d'interpréter les instructions en langage naturel et d'identifier les aspects de l'environnement auxquels les mots font référence.

1.4.2 Robotique développementale

Comme évoqué en Introduction, la notion d'*embodiment* joue un rôle central en robotique développementale. Le langage est donc naturellement envisagé non pas comme une capacité extrinsèque mais directement liée aux perceptions et à l'environnement social du robot. À nouveau, il existe une diversité importante de travaux et on en mentionne une sélection. Roy et Pentland (2002) propose un modèle d'apprentissage en ligne CELL basé sur un réseau récurrent qui cherche à corréler la présence de formes dans le champ visuel avec les mots d'une phrase. C. Yu et Ballard (2004) couple l'apprentissage de la reconnaissance visuelle des objets avec l'apprentissage des symboles qui les désignent et montre que l'intégration de ces données multimodales permet de résoudre ces deux tâches simultanément. Enfin, Dominey et Boucher (2005) met à profit des grammaires de construction pour apprendre le nom des objets et leurs relations spatiales à partir de vidéos racontées par un narrateur.

La très grande majorité de ces travaux en robotique revendique l'influence de l'apprentissage du langage chez l'enfant. En sciences cognitives, on retrouve en effet assez naturellement l'idée que le langage ne s'apprend pas uniquement à travers les mots mais aussi et surtout par les expériences que l'on en a. L'apprentissage du langage par l'enfant s'effectue en parallèle d'une multitude d'autres apprentissages qui ne sont pas indépendants les uns des autres. Dans la suite de ce chapitre, nous évoquons ainsi des théories en psychologie développementale qui traite de l'acquisition du langage chez l'enfant. Sans

chercher à dresser un tableau exhaustif de cet immense domaine, nous nous concentrons sur les différents points qui nous paraissent pertinents au regard de nos problématiques. Nous nous intéressons ainsi plus particulièrement aux mécanismes qui font de l'acquisition du langage un processus éminemment social et à ses interactions avec d'autres processus développementaux de l'enfant. Nous discutons notamment des travaux de Vygotsky, Bruner, Nelson et Tomasello. L'inspiration que l'on peut tirer de ces travaux ne consiste pas à tenter de reproduire tel quel les mécanismes décrits par ces travaux mais à les adapter au contexte qui nous intéresse : les agents autonomes apprenant en interaction.

1.5 L'apprentissage du langage en sciences cognitives

Dans cette partie, nous évoquons cinq grandes questions issues des théories du développement : le débat entre inné et acquis à travers la controverse béhavioriste/nativiste, le rôle de l'attention partagée, la formation des concepts, la vision vygotskienne du langage comme outil cognitif et l'importance de l'entourage social de l'enfant. Ces différents sujets pourraient chacun faire l'objet d'une thèse, notre objectif est ici de présenter une sélection d'idées qui ont retenu notre attention et d'expliciter leurs liens avec des problématiques en Intelligence Artificielle.

1.5.1 Acquisition du langage : entre inné et acquis

Le débat entre béhavioriste et nativiste – L'acquisition du langage chez l'enfant a donné lieu à de vifs débats dans la communauté scientifique. Le plus célèbre est peut-être l'opposition entre les béhavioristes (ou comportementalistes) et les nativistes. Remontant jusqu'à Saint Augustin, la théorie béhavioriste, notamment développée par Skinner (1957), considère le langage comme une compétence apprise uniquement en imitant ses parents et guidée par leur retour. Au fur et à mesure que l'enfant entend ses parents parler, il apprendrait la signification des mots en corrélant leur apparition avec les éléments du contexte qu'il observe. L'apprentissage du langage serait donc simplement appris par renforcement, c'est-à-dire par essai-erreur, comme le sont une part importante des capacités humaines. Du côté nativiste, Noam Chomsky (1965) oppose à cette vision que l'apprentissage du langage est trop rapide pour émerger uniquement de l'environnement social dans lequel l'enfant évolue. Un enfant n'a pas rencontré suffisamment d'exemples, ni n'a été assez souvent corrigé par ses parents, pour être possiblement en mesure de vocaliser les mots ou d'intégrer la grammaire aussi vite (*pauvreté du stimulus*). L'apprentissage du langage est envisagé comme une tâche très complexe qui nécessite de segmenter les phrases entendues pour en reconnaître les mots, d'apprendre à vocaliser ces derniers, de les

associer à leur signification, etc. La rapidité avec laquelle l'enfant prononce ses premiers mots suggère donc qu'il possède des prédispositions très particulières.

La théorie de la Grammaire Universelle – On attribue ainsi à Chomsky la théorie de la Grammaire Universelle qui stipule l'existence d'un ensemble de règles (une grammaire) qui régit le fonctionnement de toutes les langues. Innée pour l'enfant, cette Grammaire Universelle lui permet de comprendre beaucoup plus vite la structure des phrases qu'il entend et facilite la tâche d'apprentissage du langage. Envisagée comme une faculté génétique, cette dernière consiste essentiellement à apprendre les mots du vocabulaire, leur sens et un ensemble de règles venant parfaire la grammaire spécifique de la langue maternelle. Les contraintes imposées par la Grammaire Universelle sur le langage permettent ainsi d'expliquer son apprentissage aussi rapide par les enfants. Cette théorie a révolutionné le domaine de la linguistique (Pater, 2019) sans faire pour autant l'unanimité. En effet, la Grammaire Universelle n'a jamais pu être explicitement définie et de nombreuses études en linguistique sont allées à l'encontre de cette idée, elle semble notamment incompatible avec la variété des héritages sociaux-culturels observés dans les différentes langues (Evans et Levinson, 2009).

Entre ces deux visions radicalement opposées – l'une prônant l'apprentissage exclusif par l'expérience et l'autre un important conditionnement génétique – il en existe tout un continuum. S'il existe un consensus pour réfuter la théorie béhavioriste, et qu'il est admis que l'existence de contraintes est nécessaire pour permettre à l'enfant d'apprendre le langage, ces contraintes ne sont pas nécessairement génétiques, de l'ordre d'une Grammaire Universelle (Tomasello, 2009). Elles peuvent être liées à des attributs inhérents du langage tels que l'intentionnalité de la communication et à des processus d'interactions sociales qui se mettent en place entre un adulte et un nourrisson comme l'attention partagée. Dans ses travaux, Bruner (1983) donne ce qu'il considère être quatre capacités cognitives innées des nourrissons à la base de l'apprentissage du langage : l'action du nourrisson orientée par les buts, la transactionnalité, la systématisation et l'abstraction. Enfin, les preuves de l'existence d'une forme d'apprentissage statistique chez l'enfant participent à une vision hybride entre apprentissage par l'expérience et conditionnement biologique (Yang, 2004).

Théorie constructionniste – Parmi les théories qui remettent en cause les origines génétiques de la faculté du langage, les théories dites *usage-based* considèrent le langage comme un processus empirique conditionnée par l'usage du langage. Ils partent du constat que le langage de l'enfant et de l'adulte diffèrent de manière assez fondamentale (Tomasello, 2000a). Le langage de l'enfant s'articule d'abord autour d'expressions courtes, simples et centrées autour d'objets (Tomasello, 2000b) comme « Chien parti ». L'enfant fait preuve d'une créativité en combinant ces premières expressions, il prend conscience d'une forme

de catégorie grammaticale et peut ainsi substituer dans ces premières expressions les mots qu'il considère équivalents (Tomasello et Olguin, 1993). En grandissant, sa parole devient plus abstraite et se rapproche du langage adulte (Tomasello, 2000b ; Goldberg, 2003). Ces théories suggèrent que l'enfant acquiert le langage à travers des unités élémentaires qu'il compose pour complexifier sa parole (Tomasello, 2003).

Les théories des grammaires de construction (CxG) en linguistique sont particulièrement adaptées à ces idées. Les théories constructionnistes se sont développées à partir des années 80 pour défendre une interprétation du langage sur la base d'unités élémentaires, appelées des constructions (Fillmore, 1988). Goldberg (2003) en donne la définition suivante :

> Any linguistic pattern is recognized as a construction as long as some aspect of its form or function is not strictly predictable from its component parts or from other constructions recognized to exist.

Le terme construction rassemble ainsi différents niveaux d'abstraction, des phonèmes jusqu'à la structure complexe d'une phrase (ex : verbe-sujet-complément) en passant par des unités plus petites comme les préfixes (ex : anti-X). Une construction est simplement une fonction du langage, l'association d'une forme avec un sens (*form-meaning mapping*). Certaines de ces fonctions acceptent des arguments et, une fois instanciée, prennent tout leur sens. Par exemple, *anti-X* est une construction dont l'instanciation avec un nom exprime une opposition aux effets décrits : anti-inflammatoire, antivol, anticapitaliste... Nous exploitons dans les Chapitres 2 et 3 un formalisme simple inspiré des grammaires de constructions.

Parallèle en Intelligence Artificielle – Ces débats entre les différentes conceptualisations du langage humain cachent d'une part la question de l'inné et de l'acquis. On la retrouve en intelligence artificielle sous une autre forme : quelle part est-il envisageable d'apprendre et quelle part doit être *hardcodée* dans un système intelligent ? Le débat entre IA symbolique, représentée par les systèmes experts et gouvernée par des systèmes de règles, et IA connexionniste, représentée par les modèles de Deep Learning et les modèles statistiques, en est une émanation (Smolensky, 1987). Les premiers sont dotés à leur conception d'un ensemble de règles leur permettant de raisonner et d'expliciter leur décision, les seconds apprennent un équivalent statistique et distribué de ces règles à partir de large corpus d'exemples et excellent sur les tâches de prédiction. Le langage constitue depuis longtemps un point d'achoppement important dans ce débat (Rumelhart et James L McClelland, 1986 ; Fodor et Pylyshyn, 1988). Récemment, Santoro et al. (2021) propose de redéfinir la manière dont on définit la pensée symbolique et tente de s'affranchir de cette opposition.

D'autre part, ce débat pose la question des ressources nécessaires à l'enfant pour ne pas se perdre dans l'immensité des possibilités du monde qui l'entoure. Ces questions sont aussi importantes en Intelligence Artificielle et traitées par les mécanismes d'attention ou des stratégies d'exploration, notamment basé sur la motivation intrinsèque (Oudeyer et al., 2007) pour des systèmes d'apprentissage par renforcement.

1.5.2 Développement d'une attention partagée

Décrire l'apprentissage du langage comme un acte social peut sembler relever de la banalité. Pourtant, un ensemble de processus non triviaux permet à l'enfant de prendre conscience de la présence de ses parents, de leur intentionnalité, de développer une attention partagée avec eux et d'établir une forme de proto-communication. Ce n'est qu'alors que les processus visibles d'acquisition du langage peuvent débuter.

Les interactions sociales précoces du nourrisson – Dès sa naissance le nourrisson a un comportement « ultra-social » (Tomasello, 1999, Chapitre 3) qui le différencie des autres espèces. Le nourrisson s'engage naturellement avec les adultes qui l'entourent dans une forme de communication non verbale à travers des échanges de regards, de sourires, le partage d'émotions, etc. Mises en évidence par Bateson (1975) et largement étudiées par Trevarthen (1993), ces *proto-conversations* indiquent que le nourrisson possède une motivation intrinsèque pour chercher à se lier socialement avec ses parents. Ces observations ont donné naissance à la théorie de l'*intersubjectivité innée* (Trevarthen et Aitken, 2003), l'idée que les nourrissons perçoivent, de manière innée, la subjectivité des autres personnes.

« The Nine-Month Revolution » Tomasello (1999) – Pour Tomasello (1999, Chapitre 3), cette prise de conscience est cependant plus tardive. Entre neuf et douze mois, l'enfant commence à s'engager dans des activités d'attention conjointe avec ses parents. Pendant les premiers mois, en effet, l'enfant ne partage pas son attention : ces interactions avec un objet ou un parent sont mutuellement exclusives (interaction dyadique). Le développement de l'attention partagée est cruciale pour l'apprentissage du langage par l'enfant et son défaut est considéré comme un signe clinique de possibles troubles autistiques (Bruinsma et al., 2004). L'apparition de l'attention partagée prouve que le nourrisson a compris que les autres personnes sont elles aussi animées par leurs intentions, reconnaissant ainsi leur subjectivité propre (Tomasello, 1999). Prendre conscience de l'intentionnalité des autres permet au nourrisson d'interpréter leurs actions sous l'angle de la causalité. En effet, l'enfant sait alors qu'il existe un lien causal entre les actes et paroles observés et l'intention poursuivie. Carpenter et al. (1998) décrit trois types d'attention partagée qui apparaissent au fur et à mesure : la première consiste pour le nourrisson à

simplement prendre conscience que le parent porte son attention sur la même chose que lui, la seconde consiste à suivre l'attention du parent et la troisième consiste à diriger l'attention du parent.

L'évolution de plus en plus active de cette attention rejoint les observations effectuées par Bruner (1983, Chapitre 3) lors du suivi régulier de jeunes enfants de moins de 24 mois pendant des séances de jeu avec leur mère. Si les enfants semblent conserver une position d'observateur passif au début, ils prennent une part de plus en plus active au fur et à mesure que les scénarios se répètent et qu'ils semblent en comprendre les règles, jusqu'à vouloir eux-mêmes mener ces scénarios et les faire varier. Bruner (1983, Chapitre 3) décrit l'évolution de l'attitude de l'enfant comme un moyen de négocier une forme de communication. Pour Bruner, le développement de l'attention partagée et la négociation des règles du jeu est un moyen de créer un cadre contrôlé dans lequel il peut apprendre. Bruner insiste ainsi sur le rôle joué par les scénarios qui permettent un ancrage social du langage, s'ajoutant à la notion d'ancrage physique déjà évoquée. On retrouve ici l'idée, détaillée plus haut, que l'enfant a besoin de repère pour apprendre le langage, et le jeu est un cadre privilégié dans lequel le nourrisson peut construire ses repères avec l'aide du parent.

Parallèle en Intelligence Artificielle – À nouveau, le parallèle avec les problématiques en Intelligence Artificielle est naturel puisque la notion d'attention a joué un rôle important dans les progrès récents faits dans le domaine. Les mécanismes attentionnels en deep learning ont d'abord été développés dans le contexte de modèle de traduction (Bahdanau et al., 2015) et se sont ensuite diversifiés et étendus à tous les domaines (Chaudhari et al., 2020). Dans ces travaux, l'attention s'y entend comme la capacité à filtrer automatiquement l'information pertinente dans des données. La notion d'attention partagée est légèrement différente parce qu'elle suppose une interaction : elle consiste notamment à se concentrer sur une part de l'environnement en coordination avec un agent extérieur, mais aussi à pouvoir interpréter les actions de ce dernier en lui prêtant une intention. En robotique développementale, Nagai et al. (2003) a proposé un robot apprenant à regarder la même chose qu'un humain. Il s'agit cependant d'une forme relativement simple d'attention partagée (F. Kaplan et Hafner, 2004) et cette notion reste relativement peu exploitée en apprentissage interactif.

1.5.3 La formation des concepts

La formation des concepts est une question centrale en psychologie du développement. Sans chercher à dresser un tableau exhaustif du domaine, nous nous sommes intéressés

aux théories portées par Vygotsky et Nelson, en raison des idées qu'elles permettent de soulever en Intelligence Artificielle.

Au premier abord, on est tenté de considérer le langage comme l'outil idéal à la formation des concepts : les mots constituant des symboles permettant d'évoquer les concepts, de les manipuler et donc de penser. Il semble cependant que l'enfant soit capable de former des concepts pré-linguistiques et que la formation de concepts soit un processus dynamique et non une acquisition ponctuelle.

Concepts spontanés et scientifiques – Vygotsky (1986, Chapitre 5) considère en effet que l'émergence des concepts et du langage ont des origines différentes. Les premiers concepts qui émergent chez l'enfant sont les concepts dits *spontanés*. Ils ont pour origine l'activité quotidienne de l'enfant et ne sont pas issus d'un enseignement explicite. Le langage, quant à lui, émerge des interactions sociales de l'enfant avec son entourage, il a avant tout une fonction communicative. Autrement dit, l'origine du langage est sociale, celui des concepts est individuel. Il existe ainsi une pensée non verbale et une communication pré-intellectuelle chez l'enfant.

En grandissant, et notamment grâce aux interactions sociales permises par le langage, il acquiert une seconde forme de concepts dits *scientifiques*. Ces derniers sont enseignés et transmis par le langage, ils s'organisent au sein d'un système conceptuel global construit au fur et à mesure par l'éducation. Vygotsky n'envisage pas ces deux formes de concepts de manière concurrente mais comme des processus parallèles qui se rejoignent : l'enfant fait coïncider les concepts spontanés et scientifiques en alignant ses expériences vécues et leurs explications culturelles. À l'âge adulte, ce sont les concepts scientifiques qui dominent et qui permettent à l'individu de raisonner. À ce stade, la pensée est alors nécessairement verbale.

Pour mieux comprendre ce qui sépare la pensée de l'enfant de celle de l'adulte, Vygotsky (1978, Chapitre 3) donne l'explication suivante : un concept d'enfant est une collection de souvenirs tandis que pour l'adulte, il est abstrait. Il résume sa vision ainsi : *For the young child, to think means to recall ; but for the adolescent, to recall means to think* (Vygotsky, 1978, p. 51).

Conceptualisation fonctionnelle – Nelson (1996) partage l'idée que la formation des concepts est un processus dynamique qui n'est véritablement réalisée qu'à partir de l'adolescence. Elle reconnaît aussi que l'environnement socio-culturel influence leurs représentations chez l'enfant. Là où Vygotsky donne peu de précision sur l'origine des concepts spontanés, elle propose une vision intéressante, parfaitement résumée par le titre de l'article que Tomasello (2002) lui consacre : « Les choses sont ce qu'elles font ». Elle considère en effet que l'émergence des concepts chez l'enfant se fait sur la base

des fonctions associées aux objets plutôt que sur leurs caractéristiques perceptuelles. Le concept de balle est ainsi associé à ce qui peut rouler et rebondir plutôt qu'à ce qui est de forme ronde. Cette vision fonctionnelle de la formation des concepts et des premiers mots qui leur sont associées peut paraître contre-intuitive pour un adulte. Cette théorie a en effet été éprouvée lors d'expérimentations chez de jeunes enfants (Kemler Nelson et al., 2000) et rejoint la différence que fait Mandler (2000) entre catégorisation perceptuelle et catégorisation conceptuelle. Les caractéristiques perceptuelles des objets permettent de les identifier mais pas de faire émerger des concepts. L'enfant conceptualise à partir de la fonction d'un objet et catégorise à partir de ses caractéristiques perceptuelles.

Parallèle en Intelligence Artificielle – La notion de concept trouve naturellement sa place parmi les préoccupations majeures en Intelligence Artificielle. En effet, l'enjeu pour bon nombre de systèmes d'Intelligence Artificielle est de généraliser un comportement à des situations qu'il n'a jamais rencontrées. Pour y parvenir, il doit extrapoler ce qui a été appris dans les contextes connus à une nouvelle situation. Cela suppose une forme de conceptualisation, qu'elle soit explicite ou non. La notion de conceptualisation rejoint assez naturellement celle de généralisation. La vision de Vygotsky nous invite (1) à envisager le concept comme une notion évolutive et culturelle et (2) à investiguer les interactions entre le langage et les concepts spontanés qui émergent de l'expérience. La communauté est très inventive pour évaluer le niveau de conceptualisation d'un système artificiel, cependant, à notre connaissance, il n'existe pas de travaux qui se soit intéressé à la dynamique de formation des concepts chez une IA. La vision de Nelson, quant à elle, a été assez peu explorée en Intelligence Artificielle : l'apprentissage du langage par des systèmes artificiels est en effet généralement réalisé à partir de régularités détectées dans des observations statiques. Pour parvenir à une conceptualisation fonctionnelle à la Nelson, ces systèmes auraient besoin de lier le langage à des transformations visibles de leur environnement.

1.5.4 Le langage : moteur du développement cognitif

Dans cette partie, nous évoquons enfin le rôle que joue le langage chez l'enfant. Le langage est en effet avant tout envisagé comme un moyen de communication. S'il y trouve son origine, son rôle ne saurait s'y réduire. Chez l'adulte, on observe l'influence de la langue sur ses représentations spatiales et numériques (Gentner, 2016).

Vygotsky (1978 ; 1986) soutient ainsi que le langage est le responsable principal du développement cognitif de l'enfant. Dans le chapitre *Tool and Symbol in Child Development*, il s'est notamment intéressé à la manière dont le langage est employé par de jeunes enfants pour résoudre une tâche. Il montre que plus une tâche est complexe, plus l'enfant aura recours au langage. L'enfant formule ainsi, à voix haute, des phrases qui lui permettent de

décrire la situation, d'évoquer des solutions potentielles et de décrire ses résultats. Cette parole, extériorisée, qui s'écarte de sa traditionnelle fonction communicative correspond à la parole *égocentrique* introduite par Piaget (2002). Ce dernier avait noté que les enfants développaient une forme de parole pour eux-mêmes, peu compréhensible par l'entourage et qui s'internalisait avec le temps. Vygotsky donne ainsi un attribut fonctionnel à la parole égocentrique. Dans ses expériences, il observe des enfants auxquels il donne une tige et la tâche d'attraper un objet en hauteur. Il observe que l'enfant se sert alternativement de la tige et du langage dans ses essais pour y parvenir. Il observe aussi qu'en privant l'enfant de la parole, le taux de succès diminue. Pour Vygotsky, cette parole est un outil à part entière au même titre que les outils physiques (la tige). Il définit en effet un outil comme un « médiateur » entre une situation et un individu. Les outils physiques permettent d'agir sur l'environnement, le langage est un outil psychologique qui permet d'organiser sa pensée, de planifier son action mais aussi d'agir indirectement sur son environnement :

— *Organisation* : Le langage permet à l'enfant de manipuler une situation intellectuellement. Alors que les animaux, et notamment les primates, abordent une situation uniquement à partir des stimuli perceptuels, l'enfant use du langage pour évoquer des souvenirs, se rappeler ce qu'il aurait entendu dans une situation similaire et ainsi aborder la situation sous des angles plus variés. Son approche est significativement plus créative dès le moment où il se met à employer le langage.

— *Planification* : Le langage permet à l'enfant de décomposer, ordonner et planifier. Il peut ainsi simplifier un problème complexe en sous-problèmes plus simples et élaborer un plan pour le résoudre. Le langage permet aussi de simuler le résultat des actions potentielles et d'anticiper la réussite ou l'échec.

— *Communication* : Si l'enfant est laissé seul face à une tâche et qu'il s'aperçoit qu'il n'est pas en mesure de la réaliser seul, il cherchera l'aide d'un adulte en lui décrivant le plan auquel il a réfléchi. La sollicitation sociale est pour l'enfant un moyen d'agir indirectement sur son environnement en tentant d'influencer son interlocuteur pour atteindre son objectif.

L'enfant a besoin d'exprimer à haute voix cette parole égocentrique, comme si le fait de la vocaliser était nécessaire à son usage. À l'âge adulte, au contraire, la réflexion est internalisée. Pour Vygotsky, c'est une preuve supplémentaire de l'origine distincte de la pensée et du langage : l'origine du langage étant sociale, ses premiers usages sont nécessairement extériorisés. L'une des étapes de la fusion entre les deux passe ainsi par l'internalisation de la parole égocentrique et sa transformation en parole dite *intérieure*. Cette évolution est visible dans la manière dont change la synchronisation de l'action et du langage. Au début, la parole suit, ou accompagne, l'action : l'enfant décrit ce qu'il est

en train, ou vient, d'accomplir. En grandissant, c'est la parole qui précède l'action. On retrouve l'idée déjà évoquée plus haut de l'internalisation des processus chez Vygotsky.

Pour Vygotsky, c'est la manière dont le langage est employé qui le rend si important. Il voit ainsi dans la capacité de l'homme à mêler, au sein d'une même activité, intelligence pratique (qu'il définit essentiellement comme l'utilisation d'outils) et la manipulation de symboles (notamment le langage), l'origine de ses capacités cognitives uniques.

Parallèle en Intelligence Artificielle – La vision originale de Vygotsky sur le langage comme un outil cognitif a attiré l'attention de la communauté en robotique développementale. Dautenhahn et Billard (1999) opposent les visions de Vygotsky et de Piaget et défendent que s'inspirer du premier doit permettre le développement de robot socialement intelligent. Pour Lindblom et Ziemke (2003) c'est même la seule façon de créer des robots capables de cognition sociale. Mirolli et Parisi (2011) enfin proposent le développement d'une robotique cognitive et l'utilisation de robots comme des outils de recherche pour investiguer le rôle du langage dans le développement de la cognition.

1.5.5 Le rôle des interactions dans le développement de l'enfant

Dans cette dernière partie, nous quittons à proprement parler le champ de l'apprentissage du langage pour évoquer le développement plus global de l'enfant et le rôle qu'y joue l'entourage. L'enfant est très dépendant de ses parents pendant les premières années de sa vie. Il a aussi besoin d'eux pour structurer son développement. Et le langage, comme moyen de transmission, joue un rôle primordial dans le lien entre l'enfant et ses enseignants. Nous évoquons dans cette partie la notion de *zone de développement proximal* de Vygotsky ainsi que la *théorie de l'échafaudage* de Bruner.

Zone de développement proximal – Pour Vygotsky, le développement de l'enfant consiste à internaliser les compétences. L'acquisition d'une nouvelle compétence est toujours d'origine sociale : découverte au contact des adultes par un enseignement ou une activité collaborative. Au début, l'enfant ne peut pas utiliser seul la compétence. Le rôle de l'enseignant est alors de guider et d'accompagner l'autonomie de l'enfant. Vygotsky (1978, Chapitre 6) s'est cependant aperçu que le potentiel d'apprentissage des enfants était très différent selon chacun. Il a cherché à mesurer cette différence de potentiel. Il a ainsi développé la notion de *Zone de développement proximal* (ZPD (*Zone de développement proximal*)). Cette zone correspond aux tâches qu'un enfant parvient à réaliser avec l'aide d'un enseignant. Il constate en effet que deux enfants qui auraient le même âge mental, ce qu'il définit comme la même capacité à résoudre seul des problèmes, peuvent avoir des ZPD très différentes. Vygotsky prévoit que l'enfant avec la plus grande

ZPD progressera le plus vite, car il a déjà découvert un plus grand nombre de compétences et peut démarrer leur internalisation. Ce processus d'internalisation et d'intégration culturelle, que Vygotsky dénomme *interpersonnel* et *intrapersonnel*, est la pierre angulaire des théories du développement chez Vygostky :

> *An interpersonal process is transformed into an intrapersonal one.* Every function in the child's cultural development appears twice : first, on the social level, and later, on the individual level ; first, between people (interpsychological), and then inside the child (intrapsychological). (Vygotsky, 1978, Chapitre 4, p. 57)

La notion de ZPD est devenue très populaire dans les théories de l'éducation (Bodrova et Leong, 2001) et a été ensuite réinterprété comme étant la zone d'apprentissage optimale pour l'enfant : la frontière entre ce qu'il sait déjà et ce qu'il est en mesure d'apprendre.

La théorie de l'échafaudage – La théorie de l'échafaudage (*scaffolding*) n'a pas été développée ni explicitement mentionnée par Vygotsky, mais lui est souvent associée, car elle complète les idées introduites par la ZPD. Elle est attribuée à Bruner (Wood et al., 1976) et permet de préciser par quels processus l'enfant internalise les compétences. Nous avons vu que le rôle de l'enseignant est d'accompagner l'autonomie de l'enfant. Cet accompagnement s'articule de plusieurs façons, il peut s'agir de simplifier la tâche, d'exécuter des démonstrations que l'enfant peut imiter ou de lui faire remarquer ses succès et erreurs. Comme lorsque Bruner évoquait le rôle des parents dans la structuration des scénarios de jeux avec le nourrisson, il insiste sur le rôle actif que joue l'enseignant. Il ne s'agit pas d'assister passivement l'enfant dans la réalisation, mais d'évaluer et de doser l'aide nécessaire et la réduire progressivement en fonction des progrès de l'enfant. Le rôle de l'enseignant consiste finalement à maintenir l'enfant dans sa zone de développement proximal en lui transférant au fur et à mesure la responsabilité de la réussite de la tâche. Nous retrouvons l'idée que pour Bruner, le potentiel d'apprentissage de l'enfant se trouve dans les contextes familiers.

Parallèle en Intelligence Artificielle – Les idées développées ici sont proches de ce que l'on nomme curriculum learning (Bengio et al., 2009) en Intelligence Artificielle. Cette expression désigne un ensemble de techniques consistant à structurer l'apprentissage d'un système. Nous présentons d'abord au système des tâches simples que l'on complexifie au fur et à mesure. Cette technique permet à des systèmes de résoudre des tâches plus complexes que si les mêmes tâches étaient présentées de manière aléatoire. La difficulté pour mettre en place ce genre de technique est justement d'être en mesure d'établir ce curriculum. Il existe ainsi des techniques de génération automatique de curriculum (Portelas et al., 2020). La

famille de modèles IMGEP (Intrinsically Motivated Goal Exploration Processes) (Forestier et al., 2017 ; Colas et al., 2019a) permet, par exemple, à un agent d'explorer de manière autonome son environnement en choisissant ses propres objectifs basés sur une récompense interne. Basée sur la notion de motivation intrinsèque (Oudeyer et F. Kaplan, 2009), cette récompense interne est calculée à partir d'une heuristique maximisant la surprise, le progrès en apprentissage, ou minimisant l'erreur en prédiction. Assez naturellement, l'agent va ainsi se fixer des tâches qu'il peut réussir et complexifier petit à petit ses objectifs.

Conclusion

A travers ce chapitre, nous avons voulu mettre en évidence certains des processus qui permettent à l'enfant d'acquérir et maîtriser le langage. Les théories évoquées ne sont pas uniquement des descriptions de processus psychologiques. Elles sont « actionnables » et sont appliquées pour développer des outils spécifiques à l'éducation des enfants comme pour améliorer l'apprentissage de la lecture (Bodrova et Leong, 2001). Ces théories ont aussi retenu l'intérêt de la communauté en robotique développementale (Dautenhahn et Billard, 1999 ; Lindblom et Ziemke, 2003 ; Nagai et al., 2003 ; Mirolli et Parisi, 2011), et notamment les théories de Vygotsky. Plus généralement, les liens entre Sciences Cognitives et Intelligence Artificielle peuvent être très féconds et fournir l'un à l'autre des terrains de réflexion et d'expérimentations (Dupoux, 2018).

1.6 Conclusion

Dans ce chapitre, nous avons couvert différents aspects du traitement du langage naturel que nous réemploierons dans la suite de ces travaux. Après avoir posé ce contexte, les chapitres suivants seront consacrés à proposer des solutions aux difficultés de généralisation des assistants autonomes.

Chapitre 2

Apprentissage en interaction et détection de l'intention de l'utilisateur

Les limites de ma langue sont les limites de mon monde.

— Ludwig Wittgenstein
Tractatus Logico-Philosophicus, 1921

Sommaire

2.1 Introduction et contexte

Nous avons évoqué les difficultés des assistants apprenants pour généraliser leur comportement en ce qui concerne (1) l'interprétation du langage naturel et (2) la réalisation d'actions. Dans cette partie, nous nous attelons à proposer une solution à (1) pour permettre à ces agents de disposer d'une interprétation plus souple du langage. Les assistants apprenants s'appuient pour la plupart sur un parseur sémantique qui nécessite que le concepteur écrive à l'avance l'association entre les formes connues et leurs interprétations exécutables par l'assistant. Nous souhaitons employer des techniques de ML pour doter l'agent d'une notion de distance sémantique qui lui permette de comparer des requêtes entre elles. Nous proposons ainsi un système de détection de l'intention de l'utilisateur, AIDME pour *Adaptive Intent Detection in Multi-Domain Environment*. Ce système est conçu comme un module de NLU qui peut s'adapter à un agent apprenant par interaction pour remplacer une fonction de traitement du langage naturel. AIDME repose sur une comparaison de la similarité sémantique entre la requête de l'utilisateur et les requêtes connues pour déterminer l'intention de l'utilisateur.

Lorsque l'agent rencontre une nouvelle requête, AIDME cherche à identifier, si elle existe, la requête connue la plus proche sémantiquement afin d'inférer automatiquement son interprétation. Si aucune requête ne semble suffisamment proche, cela peut signifier que la requête de l'utilisateur est inconnue et l'agent peut ainsi exprimer qu'il ne comprend pas ce que l'on attend de lui et retomber dans mode d'apprentissage de l'agent. Si au contraire AIDME réussit à interpréter correctement la nouvelle requête, cette dernière vient alimenter l'ensemble de requêtes connues et servira comme base de comparaison pour interpréter les requêtes futures. Ces travaux ont donné lieu à la publication d'un article dans les *proceedings* de la conférence Intelligent User Interface en 2020 (Lair et al., 2020).

2.1.1 Modèle d'assistant apprenant par interaction

On commence par proposer un modèle standard d'architecture d'un assistant apprenant par interaction. Cela permettra notamment de bien définir dans quelles conditions AIDME peut s'intégrer à un assistant apprenant existant.

Pour interagir avec le système, l'utilisateur exprime son intention par une phrase. On définit :

— $\mathcal{L}$, l'ensemble des requêtes de l'utilisateur, ouvert et a priori (quasi-)infini ;

— $\mathcal{S} \subset \mathcal{L}$, l'ensemble des requêtes connues par le système ;

— $\mathcal{I}$ l'ensemble des intentions connues ;

— $\mathcal{P}$ l'ensemble des formes verbales connues ;

— $\mathcal{A}$ l'ensemble des actions que l'agent peut effectuer sur un environnement.

On différencie ainsi la requête, une phrase en langage naturel, de l'intention et du pattern, dont des exemples sont donnés dans le Tableau 1.1. Plusieurs requêtes en langage naturel peuvent en effet exprimer la même intention, en revanche chaque requête est associée à un unique pattern. On peut ainsi définir une *fonction d'intention* $\mu : \mathcal{S} \to \mathcal{I}$ qui associe à chaque requête connue son intention et une *fonction d'instanciation* λ qui étant donné un pattern et ses arguments fournit la requête correspondante : $\lambda : \mathcal{P} \times Args \to \mathcal{L}$, $Args$ étant un dictionnaire d'arguments adapté au pattern. Un module de NLU consiste en quelque sorte à disposer de la fonction λ^{-1} qui étant donnée une requête fournit le pattern (et donc l'intention).

On définit deux fonctions : $\Phi : \mathcal{L} \to \mathcal{I} \times Args$, l'interpréteur du langage naturel dont le rôle est d'estimer λ^{-1}, et $\Psi : \mathcal{I} \times Args \to \mathcal{A}$, la politique d'actions, qui exécute les actions étant donnée une requête interprétée. Un assistant se définit donc simplement par la fonction $\Psi \circ \Phi$. La particularité des agents apprenants est que les fonctions Φ et Ψ sont apprises par interactions avec l'utilisateur. Un tel agent est représenté en Figure 2.1.

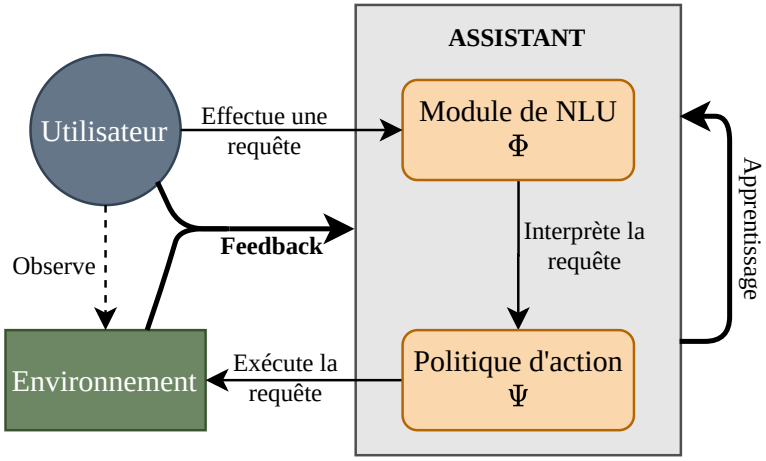


Figure 2.1 – Architecture générale d'un assistant apprenant par interaction

Remarque : Dans la formalisation proposée, on distingue l'interpréteur du langage naturel et la politique d'action comme étant deux processus différents, le premier venant nourrir le second. On verra par la suite que l'on peut proposer des assistants qui ne font pas aussi clairement la distinction mais conçoivent la réponse à l'utilisateur comme un processus unifié.

2.1.2 Retour de l'utilisateur

Les retours de l'utilisateur qui permettent l'apprentissage peuvent être très divers (démonstration, indication langagière...). Pour être compatible avec AidMe, on suppose

la propriété suivante quant au retour de l'utilisateur :

> *Lorsque l'assistant échoue à effectuer la requête demandée par l'utilisateur en raison d'une erreur d'interprétation de* Φ*, il est demandé à l'utilisateur de fournir un retour qui permette à l'assistant d'identifier quels étaient la bonne intention et le bon pattern.*

Hormis cette hypothèse sur la nature du retour fait par l'utilisateur à l'assistant, la formalisation de cet assistant reste volontairement très générale. L'objectif est de fournir un module d'interprétation du langage naturel (Φ) qui s'intègre facilement chez un assistant apprenant par interaction et soit capable d'apprendre en parallèle de la politique d'actions (Ψ).

2.2 *AidMe* : les principes de fonctionnement

Dans cette partie, nous proposons AidMe, un algorithme d'interprétation du langage naturel qui s'inscrit dans la modélisation de Partie 2.1.1 et l'hypothèse énoncée Partie 2.1.2. À partir d'une requête en langage naturel de l'utilisateur, AidMe tente d'inférer l'intention de l'utilisateur et le pattern associé à la requête par rapport aux intentions déjà connues par l'assistant. AidMe fonctionne en deux temps : il tente d'abord d'identifier l'intention de l'utilisateur parmi les intentions connues et, s'il trouve une intention candidate avec seuil de confiance suffisamment élevé, identifie le pattern associé à la requête.

2.2.1 Identification d'une nouvelle intention

Définition du problème

Le problème de détection de l'intention consiste à évaluer la fonction d'intention μ définie en Partie 2.1.1. Il est souvent formulé comme un problème de classification où chaque intention est représentée par une classe. Dans le contexte des assistants apprenants, on ne connaît cependant pas à l'avance l'ensemble des intentions (et donc des classes). De plus, les problèmes de classification avec un grand nombre de classes sont des problèmes difficiles, dont les solutions existantes sont souvent coûteuses du point de vue computationnel. On reformule donc le problème : lorsqu'un assistant apprenant doit interpréter une requête de l'utilisateur, il ne sait pas à l'avance s'il connaît l'intention de l'utilisateur ou si l'utilisateur exprime une intention nouvelle. Une première étape consiste donc à répondre à la question « Suis-je censé comprendre cette requête ? ».

Une manière naturelle de le faire est de comparer cette nouvelle requête aux requêtes précédemment rencontrées et de trouver celle qui s'en rapproche le plus. On se tourne alors

vers le domaine de la similarité sémantique qui consiste justement à évaluer la distance sémantique entre deux mots, deux phrases ou textes.

Détection d'une intention par comparaison sémantique

L'Algorithme 2.1 présente une méthode de détection d'une intention par comparaison sémantique. On se dote d'un modèle $Sim : \mathcal{L} \times \mathcal{L} \to [0, 1]$ qui mesure la similarité sémantique entre deux phrases en langage naturel (0 les phrases sont dissimilaires et 1 les phrases sont exactement similaires). Étant donné une requête de l'utilisateur, $s \in \mathcal{L}$, on compare s à toutes les phrases déjà rencontrées et pour lesquelles on connaît l'intention sous-jacente. On fait l'hypothèse que deux phrases suffisamment similaires expriment la même intention. On peut ainsi facilement inférer l'intention d'une nouvelle requête comme étant l'intention de la requête connue la plus proche. On cherche ainsi l'éventuelle solution du problème suivant :

$$s_0 = \arg\max_{s' \in S} Sim(s, s') \text{ s.c. } Sim(s, s') \geq \epsilon \tag{2.1}$$

où $\epsilon \in [0, 1]$ est le seuil de confiance qui détermine la similarité nécessaire pour considérer que deux phrases expriment potentiellement la même intention. ϵ est à choisir en fonction du comportement souhaité, plus ou moins prudent. Lorsqu'ϵ est proche de 0, il est très probable que le problème ait une solution, et plus ϵ est proche de 1, plus le problème 2.1 risque de se trouver sans solution.

— **Si s_0 existe :** s_0 est la phrase connue la plus proche de s avec un seuil de confiance suffisamment élevé pour considérer que $\mu(s_0)$ est une estimation fiable de $\mu(s)$

— **Si s_0 n'existe pas :** Aucune phrase connue ne semble suffisamment proche de s et il est donc probable que l'utilisateur exprime une nouvelle intention. L'assistant peut alors exprimer qu'il pense ne pas être en mesure de réaliser la requête de l'utilisateur et demander son aide.

Algorithme 2.1 : Détection d'intention par comparaison sémantique

Input : Requête de l'utilisateur s
Output : Intention identifiée $i \in I \cup \{\emptyset\}$
Phrase la plus proche sémantiquement $s_0 \in \mathcal{S} \cup \{\emptyset\}$
$s_0 = \arg\max_{s' \in \mathcal{S}} Sim(s, s')$;
if $d(s, s_0) \geq \epsilon$ **then**
  $i = \mu(s_0)$;
  return i, s_0
else
  return $\emptyset, \emptyset$ (Nouvelle intention détectée)

Remarque : La fonction Sim peut être interprété comme l'inverse d'une fonction de distance sémantique dans l'espace des phrases $\mathcal{L}$.

2.2.2 Identification d'un nouveau pattern

Similarité contextuelle

Une fois l'intention identifiée, l'assistant doit encore déterminer les arguments nécessaires à la réalisation de la requête. On cherche donc à construire le pattern p associé à s. À nouveau, on s'appuie sur la notion de similarité sémantique. En effet, si s et s_0 sont proches et décrivent la même intention, il est raisonnable de penser que p et p_0 sont proches eux aussi. Si l'on dispose d'une métrique de similarité pour les patterns, on peut alors chercher à construire p_0 comme le pattern le plus proche de p. On cherche à définir une notion de similarité sémantique propre aux patterns. On définit ainsi la similarité contextuelle relativement à un ensemble d'arguments $args$ (le contexte) :

$$\begin{aligned} Sim_{|args} : \quad \mathcal{P}^2 \quad &\to [0,1] \\ (p_1, p_2) &\mapsto Sim(\lambda(p_1, args), \lambda(p_2, args)) \end{aligned}$$

où λ est la fonction d'instanciation définie dans la Partie 2.1.1. En première approximation, on peut mesurer la similarité sémantique entre deux patterns en mesurant la similarité contextuelle relative à un jeu d'arguments bien choisi. Cette définition est relativement naturelle puisque cela revient à considérer que deux patterns sont proches si deux phrases instanciées à partir de ces patterns et des mêmes arguments sont proches.

Algorithme d'identification d'un nouveau pattern

En supposant que p et p_0 ont le même nombre d'arguments, on peut construire à partir de s l'ensemble des paires (pattern, arguments) possibles :

$$P_c = \{(p_c, args_c) \mid \lambda(p_c, args_c) = s\}$$

Deux méthodes s'offrent ensuite à nous pour identifier le pattern p et les bons arguments $args$ parmi l'ensemble des patterns et jeux d'arguments candidats (P_c) :

— **Méthode 1 :** On peut tester l'ensemble des arguments candidats $args_c$ en calculant $Sim_{|args_c}(p_c, p_0) = Sim(s, \lambda(p_0, args_c)$. Cela revient à tenter d'aligner p_0 sur p.

— **Méthode 2 :** On peut aussi faire l'inverse et tester l'ensemble des patterns candidats p_c en calculant $Sim_{|args_0}(p_c, p_0) = Sim(\lambda(p_c, args_0), s_0)$. Cette fois-ci, cela revient à tenter d'aligner p sur p_0.

On peut alors choisir la paire (pattern, arguments) qui donne la meilleure similarité contextuelle. L'idée derrière cette heuristique est que la majorité des instanciations $\lambda(p_0, args_c)$ et $\lambda(p_0, args_c)$ ne seront même pas des phrases correctes et mécaniquement la similarité de telles phrases avec s ou s_0 devraient être plus faibles que pour des phrases sémantiquement correctes. Cette méthode d'identification de pattern est décrite dans l'Algorithme 2.2.

Algorithme 2.2 : Identification d'un nouveau pattern (Méthode 1)

Input : Requête s, Plus proche requête connue s_0
Output : Pattern et arguments identifés $p, args$
$p_0, args_0 = \Phi(s_0)$;
$P_c = \{(p_c, args_c) \mid \lambda(p_c, args_c) = s\}$;
$p, args = \arg\max_{(p_c, args_c) \in P_c} Sim_{|args_c}(p_0, p_c)$;
return $p, args$

Remarques : Les deux méthodes ont la même complexité computationnelle et a priori aucun argument théorique ne semble donner l'avantage à l'une ou l'autre des méthodes. Afin de disposer d'un critère de décision symétrique on peut *moyenner* le critère des deux méthodes et proposer un nouveau mode de calcul de la similarité entre deux patterns :

$$1/2 * (Sim_{|args_c}(p_c, p_0) + Sim_{|args_0}(p_c, p_0))$$

2.2.3 Interprétation complète d'une requête

On peut maintenant combiner les Algorithmes 2.1 et 2.2 pour proposer AIDME : un algorithme qui peut interpréter une requête de l'utilisateur, même lorsque celle-ci est exprimée dans une forme nouvelle. Son fonctionnement est décrit dans l'Algorithme 2.3. AIDME tente d'abord de calculer $\Phi(s)$, c'est-à-dire d'interpréter la requête à l'aide de patterns déjà connus ($\mathcal{P}$). Si le calcul de $\Phi(s)$ échoue, l'Algorithme 2.1 d'identification de l'intention par comparaison sémantique prend le relais pour déterminer l'intention connue la plus probable :

— Si ce calcul échoue aussi, l'assistant peut indiquer qu'il est disponible pour apprendre à effectuer cette nouvelle requête.

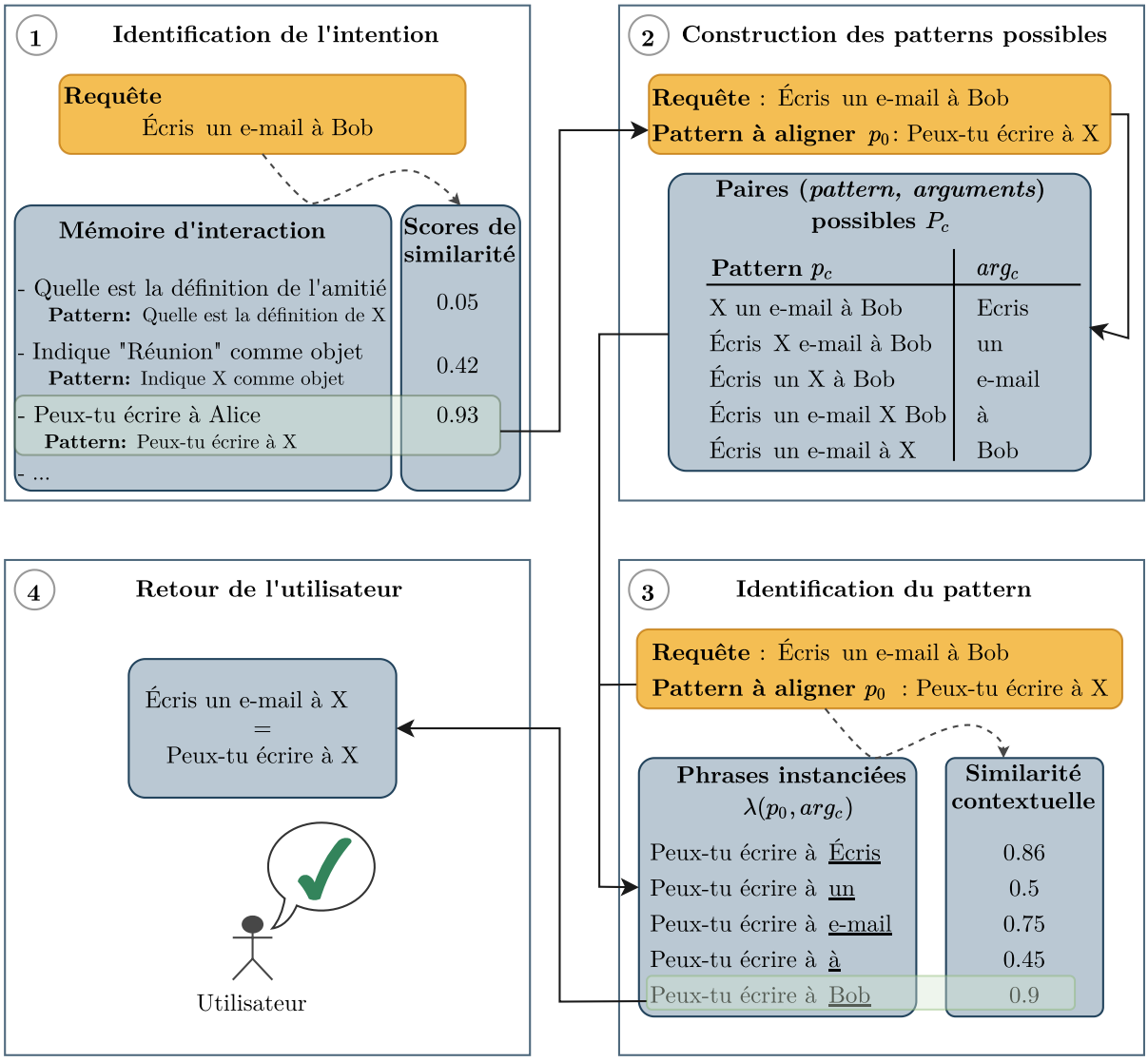


Figure 2.2 – Description du fonctionnement d'AidMe. Interprétation d'une requête de l'utilisateur en 4 étapes : (1) identification de l'intention, (2) construction des patterns candidats possibles, (3) identification du pattern le plus probable, (4) retour de l'utilisateur.

— Si une intention compatible est identifiée, l'Algorithme 2.2 propose une interprétation complète de la requête. La politique d'action Ψ peut alors effectuer l'action correspondante.

L'interaction avec un utilisateur est représentée en Figure 2.2 en illustrant quatre étapes : la détection de l'intention par comparaison avec les requêtes déjà rencontrées, la construction des paires patterns-arguments possibles P_c, l'identification du pattern le plus probable par comparaison de la similarité sémantique contextuelle et le retour de l'utilisateur qui permet de valider l'interprétation d'AidMe.

Algorithme 2.3 : AidMe

Input : Requête de l'utilisateur s
Output : Intention identifiée i, pattern p et les arguments $args$
if $\Phi(s)$ *est défini* **then**
  return $\Phi(s)$
else
  $i, s_0 =$ Algorithm 1(s);
  if s_0 *exists* **then**
    $p, args =$ Algorithm 2(s, s_0);
    return $i, p, args$
  else
    return $\emptyset, \emptyset$ (Nouvelle intention identifiée !)

2.2.4 Apprentissage de Φ

AidMe apprend par interaction avec l'utilisateur et notamment à partir des succès et erreurs dans l'interprétation des requêtes de l'utilisateur. AidMe bénéficie en effet du retour de l'utilisateur pour mettre à jour les ensembles de phrases $\mathcal{S}$, intentions $\mathcal{I}$ et patterns connus $\mathcal{P}$ ainsi que les fonctions d'intention μ et d'interprétation Φ. Si l'utilisateur ne fournit pas de retour, on suppose que l'utilisateur a été satisfait de la manière dont l'assistant a traité la requête et donc que l'interprétation de sa requête était correcte. Si l'utilisateur fournit un retour, conformément à l'hypothèse énoncée en Partie 2.1.2, on suppose que celui-ci permet à l'assistant d'identifier la bonne intention et le bon pattern.

2.3 Modèle de similarité sémantique

Jusqu'à présent, on a supposé que la fonction d'évaluation de la similarité sémantique Sim était donnée. On peut a priori utiliser n'importe quel modèle de similarité sémantique tant qu'il est possible de l'écrire sous la forme : $Sim : \mathcal{L} \times \mathcal{L} \to [0, 1]$ tel que $Sim(s_1, s_2) = 0$ si deux phrases sont totalement dissimilaires et $Sim(s_1, s_2) = 1$ si elles

sont parfaitement similaires sémantiquement. Dans cette section, on détaille le modèle de similarité linguistique qui a été développé pour AIDME. On propose un modèle qui s'appuie sur des techniques d'apprentissage automatique et capable d'être appris en ligne grâce aux données collectées lors des interactions avec l'utilisateur. Ce modèle peut donc s'adapter au mieux au langage de l'utilisateur.

2.3.1 SemEval

En traitement du langage, le domaine de l'évaluation sémantique est actif notamment au travers de différentes compétitions. *SemEval*, organisée par l'*International Workshop on Semantic Evaluation*, est l'une de ces compétitions de référence. Entre 2012 et 2017, le challenge STS (*Semantic Evaluation Similarity*) proposaient des tâches consistant justement à produire des modèles capables d'estimer la similarité sémantique de paires de phrases.

Cette compétition a rassemblé de nombreuses équipes, proposant des méthodes avec des techniques très variées et les résultats produits lors de ces compétitions sont très encourageants. Certains participants s'appuient sur des techniques d'apprentissage automatique : réseaux de neurones avec des mécanismes attentionnels (J. C. Henderson et al., 2017), des réseaux convolutionnels (Shao, 2018) ou apprentissage supervisé après extraction de features sémantiques et syntaxiques (Maharjan et al., 2017). D'autres équipes ont aussi choisi de s'appuyer sur des analyses de tags sémantiques de WordNet (Miller et al., 1990) et BabelNet (Navigli et Ponzetto, 2012). La majorité des équipes, et notamment celles ayant obtenu les meilleurs résultats ont combiné ces différentes techniques dans des modèles ensemblistes (Tian et al., 2017).

Le modèle développé est inspiré de ceux proposés par les meilleures équipes de la Tâche n°1 de SemEval 2017 (Cer et al., 2018) et notamment du modèle ensembliste de l'équipe ECNU (Tian et al., 2017). Il s'agit d'un modèle qui combine :

— des réseaux neurones s'appuyant sur des modèles de langage,

— des modèles d'arbres de décision s'appuyant sur des features d'analyse de la paire de phrases.

Dans la suite, on décrit brièvement chacun des modèles et on évalue la performance de l'ensemble sur les données de la compétition.

2.3.2 Modèles neuronaux

Description des modèles neuronaux

Les modèles neuronaux sont des perceptrons multi-couches (MLP) qui prennent en entrée des paires de phrases. Chaque paire est convertie en un vecteur de la manière suivante :

1. *Embedding d'un mot :* On utilise un modèle de langage pré-appris qui permet d'obtenir une représentation vectorielle (*embedding*) de chaque mot. En pratique, on utilise Paragram (Wieting et al., 2015 ; Wieting et al., 2018), spécialement conçu pour traiter les paraphrases et encoder la similarité entre les mots.
2. *Embedding d'une phrase :* La représentation de chaque phrase est obtenue en moyennant les représentations de chaque mot de ladite phrase.
3. *Embedding d'une paire :* La représentation de chaque paire est obtenue en concaténant le produit de Hadamard (terme à terme) des *embeddings* des phrases et la distance L_1 entre ces phrases.

La représentation d'une paire de phrase passe ensuite à travers 2 ou 3 couches complètement connectées, séparées par des activations ReLu pour les couches cachées et un SOFTMAX pour la couche de sortie. On récupère ainsi en sortie du MLP un vecteur qui peut être vu comme une distribution de probabilité et qui représente le score de similarité. On détaille ensuite la manière dont la valeur de similarité est encodée.

Représentation du score de similarité

Dans les données de STS utilisée pour la compétition SemEval, la similarité sémantique est évaluée entre 0 et 5. Lorsque l'on a travaillé sur ces modèles, cela nous a conduit à représenter la similarité de deux manières différentes :

— *Représentation continue* : dans ce cas, le réseau fournit en sortie un score entre 0 et 1 que l'on peut interpréter comme la probabilité que les deux phrases soit parfaitement similaires. Le score de similarité est alors simplement extrapolé pour s'adapter à l'échelle : une probabilité de 0.8 donne ainsi un score de 4 sur l'échelle de STS.

— *Représentation multi-classe* : Cette représentation est inspirée des travaux de Kiros et al. (2015) et Tai et al. (2015). On distingue alors des classes de similarité qui encodent un degré de similarité, le réseau fournit en sortie un vecteur de taille n qui encode la probabilité de chacune des classes de similarité de la paire de phrases. Dans ce cas le score est simplement l'espérance de la distribution de sortie (facilement calculable comme le produit scalaire entre ce vecteur de probabilité et le vecteur des

classes : $[0, 1, ..., n]$). Ainsi un score de 2.7 est encodé par $[0, 0, 0.3, 0.7, 0, 0]$, ce qui peut être interprété comme $\mathbb{P}(similarity = 2) = 0.3$ and $\mathbb{P}(similarity = 3) = 0.7$.

Fondamentalement, ces deux représentations sont très différentes. La première est certainement la plus naturelle puisqu'elle conserve la notion d'ordre. La seconde, si elle perd cet ordre naturel, permet de discrétiser la notion de similarité et de donner une plus grande finesse au modèle. En pratique, on implémente deux modèles neuronaux pour chacun des modes de représentation de la similarité.

Remarque : Il peut paraître étonnant de s'être tourné vers des perceptrons alors que des réseaux récurrents comme les *Long Short Term Memory* (LSTM) sont largement employés en traitement du langage. L'avantage de ces réseaux simples est qu'ils ont besoin de moins de données pour apprendre et cela est particulièrement important dans le contexte d'assistant apprenant. Par ailleurs ceux-ci peuvent souffrir de performances dégradées lorsque les données d'entraînement sont trop différentes des données d'usage (Tai et al., 2015 ; Wieting et al., 2015). Tian et al. (2017) avait d'ailleurs mentionné les surprenantes mauvaises performance des LSTM.

L'une des faiblesses de ces modèles est qu'ils ne s'appuient que sur des données sémantiques, méconnaissent la syntaxe et la grammaire, et ne prennent pas en compte l'ordre des mots pour représenter les phrases. Les modèles d'arbres de décision permettent de compenser ces manques.

2.3.3 Modèles d'arbres de décision

A l'inverse, les modèles d'arbre prennent en compte la syntaxe et la grammaire à travers différentes features. On utilise un modèle xgboost (T. Chen et Guestrin, 2016) et un modèle de forêt aléatoire (*Random Forest*) et deux types de features différentes calculées à partir des paires de phrases :

Features de distance

On mesure directement la distance ou la similarité entre les deux phrases d'une paire en utilisant plusieurs fonctions simples :

— *Mesures de la qualité de la traduction.* En supposant qu'une phrase est la traduction de l'autre, on mesure la qualité d'une telle traduction avec des outils standards comme le score BLEU (Papineni et al., 2001) ;

— *Superposition des n-grams.* On calcule une série de n-grams sur les caractères et les mots ;

— *Distance de Levenshtein.* Il s'agit du nombre minimum d'insertions, de suppressions ou de remplacements nécessaires pour transformer une chaîne de caractères en une autre. C'est une mesure standard de la distance entre deux chaînes de caractère en traitement du langage.

Features à noyaux

Le second type de features s'appuie sur la transformation avec une fonction noyau de la paire de vecteurs représentant chacune des phrases. On utilise une représentation BoW (*Bag of Words*) du vocabulaire pour représenter chaque mot. La représentation d'une phrase est calculée comme la moyenne des représentations de chaque mot pondérées par l'IDF (*Inverse Document Frequency*). Les phrases sont donc représentées dans des espaces relativement grands (de la taille du vocabulaire). Pour éviter la malédiction de la dimensionnalité (*curse of dimensionality*) et maintenir un équilibre dans les features utilisés par les modèles, on calcule une transformation de chaque paire en utilisant les fonctions noyaux suivantes :

— noyaux linéaires : similarité cosinus (*cosine similarity*), distance de Manhattan, distance euclidienne ;

— mesures statistiques : corrélation de Pearson, corrélation de Spearman ; tau de Kendall ;

— noyaux non linéaires : sigmoïde, fonction à base radiale (RBF : *Radial Basis Function*), noyau de Laplace.

Remarque : On peut s'interroger sur le choix d'un *embedding* BoW plutôt que l'*embedding* d'un modèle de langage pré-appris. Cela permettrait en effet de limiter automatiquement la taille de la représentation des phrases et d'incorporer une sémantique pré-apprise dans le modèle de langage. Cependant, d'un point de vue computationnel, BoW présente l'avantage de produire des *embeddings* sparses et les calculs des transformations à noyaux sont donc très rapides. Dans le cas d'un *embedding* dense, les calculs seraient beaucoup plus longs. C'est un point important à prendre en compte car ces modèles doivent pouvoir tourner en temps réel pour répondre à un utilisateur. Les tests que nous avons conduits, notamment avec Glove (Pennington et al., 2014) ont bien conduit à des gains en performance mais qui ne compensaient pas le temps de calcul supplémentaire. Enfin, la sémantique est déjà prise en compte par les modèles neuronaux, l'objectif est que les modèles d'arbres soient complémentaires.

2.3.4 Le modèle complet

On utilise quatre modèles : un MLP à représentation continue, un MLP à représentation multi-classe, un modèle xgboost et un modèle de forêt aléatoire. Les scores des différents modèles sont moyennés pour obtenir le score final de similarité sémantique. L'architecture du modèle complet est illustrée par la Figure 2.3.

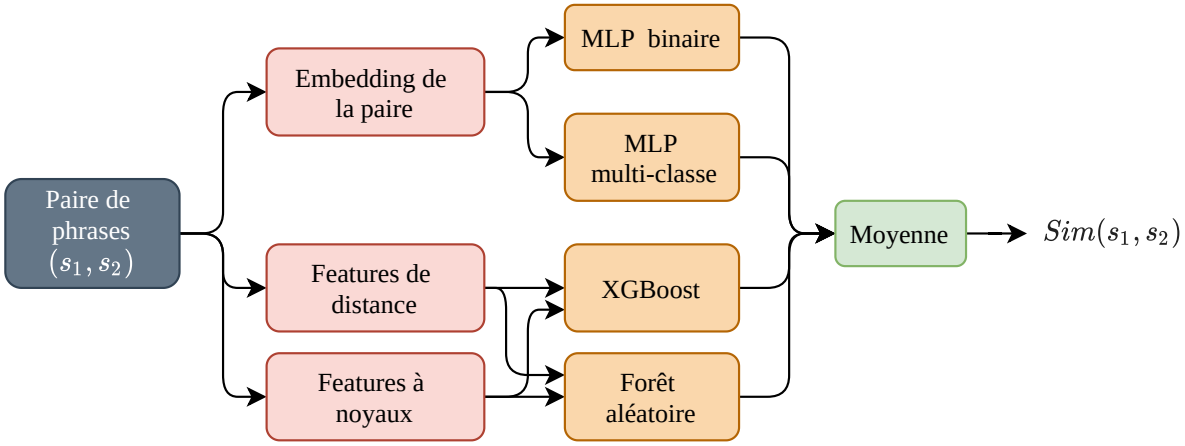


Figure 2.3 – Architecture du modèle de similarité sémantique
Deux réseaux de neurones s'appuyant sur des embedddings de phrases et deux modèles d'arbres s'appuyant sur des features.

2.3.5 Comparaison avec les modèles proposés dans SemEval

Nous avons évalué le modèle proposé sur les données de la compétition SemEval (Benchmak STS [1]). Nous l'avons comparé avec les baselines de la compétition et les scores des meilleures équipes. Nous avons entraîné notre modèle sur le jeu de données d'entraînement et évalué sur le jeu de test. La mesure utilisée par SemEval est la corrélation de Pearson, qui mesure la corrélation linéaire entre deux jeux de données. Le Tableau 2.1 synthétise les résultats obtenus et reprend certains des résultats du Tableau 14 des résultats de SemEval 2017 Task 1 (Cer et al., 2018), notamment les baselines de l'organisateur et les résultats des trois meilleures équipes. Deux baselines sont proposées par les organisateurs : la similarité cosinus calculée sur les représentations des phrases obtenues avec InferSent (Conneau et al., 2017) ou en moyennant les *embeddings* de mots d'un modèle de langage pré-entraîné (Paragram (Wieting et al., 2018)).

L'objectif est de rivaliser avec les modèles des meilleures équipes mais plutôt de valider les choix faits pour simplifier le modèle : rester proche de l'état de l'art tout en garantissant une certaine efficacité computationnelle. Notre modèle fait bien mieux que la baseline simple et s'approche de l'état de l'art mais fait moins bien qu'InferSent. L'utilisation

[1] http://ixa2.si.ehu.es/stswiki/index.php/STSbenchmark

Table 2.1 – Comparaison de différent modèles de similarité sémantique sur les données de SemEval

Modèle	Description	Corrélation de Pearson
ECNU Tian et al., 2017	participant	0.81
BIT H. Wu et al., 2017	participant	0.81
HCTI Tian et al., 2017	participant	0.78
InferSent	baseline	0.76
Paragram	baseline	0.50
Modèle proposé		**0.75**

d'InferSent fonctionne particulièrement bien sur ce jeu de données mais comme nous le montrerons par la suite, il ne donne pas de bons résultats sur la tâche de détection de l'intention et ne peut donc pas être utilisé avec AidMe. De manière générale, les modèles qu'on ne peut ré-entraîner ou *fine-tuner* sur la tâche donnent des performances décevantes.

2.3.6 Apprentissage du modèle dans le contexte d'AidMe

Sur quelles données peut-on entraîner le modèle proposé ?

Comme pour tout modèle d'apprentissage automatique, se pose la question des données sur lesquelles on peut entraîner le modèle. La première solution consisterait à pré-entrainer le modèle sur un jeu de données bien choisi et ainsi d'utiliser un modèle déjà entraîné dans AidMe. On pourrait utiliser le benchmark STS mais cela conduit à des performances médiocres car ces données ne sont pas adaptées à un contexte d'interaction avec un assistant digital. On pourrait aussi constituer un jeu de données propres à ce cas d'usage. Mais outre la difficulté que constitue la création d'un tel jeu de données, sa pertinence à l'usage n'est pas évidente. En pratique le modèle réalise l'apprentissage à partir des données collectées lors des interactions avec l'utilisateur. On pourrait craindre que les seules données collectées lors des interactions avec l'utilisateur ne soient pas suffisantes pour disposer d'un modèle de similarité sémantique performant mais en pratique on montre qu'AidMe peut fonctionner de la sorte. La possibilité de bénéficier de l'interaction avec l'utilisateur pour apprendre le modèle de similarité sémantique permet aussi de disposer d'un assistant plus adapté aux contextes dans lesquels l'assistant est employé par l'utilisateur.

Collection des données d'interaction

AidMe collecte phrases et intentions pour construire une base d'entraînement pour le modèle de similarité sémantique. La fonction d'intention μ étant à valeur discrète, on

définit une relation d'équivalence naturelle $\sim$ tel que $s_1 \sim s_2 \iff \mu(s1) = \mu(s2)$. Chaque intention est donc vue comme une classe d'équivalence qui possède ses représentants à travers les requêtes de l'utilisateur qui expriment cette intention.

On peut ainsi facilement construire un corpus de paires de phrases à partir des requêtes collectées en définissant la similarité entre deux phrases d'une classe d'équivalence à 1 et 0 si les phrases n'appartiennent pas à la même classe d'équivalence :

$$Sim(s_1, s_2) = \begin{cases} 1, & \text{si } s_1 \sim s_2 \\ 0, & \text{sinon.} \end{cases}$$

Remarque : L'un des avantages d'une approche par comparaison AIDME est que le modèle de similarité sémantique n'a pas besoin d'être ré-entraîné à chaque nouvelle interaction avec l'utilisateur. Seules les mises à jour des fonctions μ et Ψ ainsi que des ensembles $\mathcal{S}$, $\mathcal{I}$ et $\mathcal{P}$ sont nécessaires, et elles se font en temps constant.

2.4 Evaluation du système AIDME

Dans cette section, on présente différentes simulations qui permettent d'évaluer les performances et l'adaptabilité d'AIDME. En plus des simulations déjà présentées dans l'article présenté à Intelligent User Interface (Lair et al., 2020), on présente une série de simulations complémentaires. Le code permettant de faire tourner ces simulations a été mis en accès libre sur GitHub [2].

2.4.1 Grammaire utilisateur : *UserGrammar*

Description d'une grammaire de simulation

Afin d'évaluer AIDME, on propose *UserGrammar*, une grammaire générative contenant 239 patterns et 49 intentions ainsi qu'un dictionnaire d'arguments. Cette grammaire permet de générer près de 20 000 requêtes différentes typiques d'un utilisateur couvrant des domaines variés : envoi d'e-mail, recherche web, réservation de concert, de voyages, gestion de calendrier et interaction avec des objets connectés. La grammaire présentée ici est une grammaire anglaise mais la seule dépendance d'AIDME relative à la langue provient des modèles neuronaux du modèle d'évaluation de la similarité sémantique. Pour adapter AIDME à une nouvelle langue, il suffit de remplacer le modèle de langage pré-appris (ici Paragram (Wieting et al., 2018)) par un modèle de langage dans la langue souhaitée.

[2] https://github.com/nicolas-lair/AidMe

Le Tableau 2.2 présente une liste d'exemples d'intentions, de patterns et de requêtes générées par *UserGrammar*. Les deux premiers exemples illustrent une même intention énoncée à l'aide de deux formes verbales différentes et instanciées différemment. Afin de s'assurer que la grammaire génère des phrases qui font sens, les arguments dans les patterns sont typés : device, lieu, date, personne... Cependant, AIDME ne connaît pas le typage des arguments et n'a même aucune idée de l'existence de tel ou tel type. Une description complète de la grammaire est présentée en Annexe A.

TABLE 2.2 – Exemples d'intentions, patterns et requêtes issus de *UserGrammar*.

Intention ***Domaine***	**Pattern** ***Exemple de requête***
Allumer un appareil quelque part *domaine :* IoT	Can you turn the `device` on in `loc` *Can you turn the alarm on in Paris*
Allumer un appareil quelque part *domaine :* IoT	Switch on the `device` in `loc` *Switch on the television in Paris*
Prochaine réunion avec quelqu'un *domaine :* Calendrier	When do I have a meeting with `pers` *When do I have a meeting with Bob*
Prix d'un vol pour une date *domaine :* Voyage	Get me the cost for a ticket to `loc` on `date` *Get me the cost for a ticket to Berlin on Monday*
Obtenir de l'information *domaine :* Recherche web	Do you know what is a `var` *Do you know what is a spider*

Evaluation du modèle de similarité sémantique sur *UserGrammar*

Avant d'évaluer concrètement le fonctionnement d'AIDME, on évalue la performance du seul modèle de similarité sémantique sur un corpus généré à partir de *UsrGrammar*. Le corpus est généré de la manière suivante :

— *Données d'entraînement* : On génère 150 phrases à partir de 100 des 150 patterns disponibles puis on constitue 5000 paires à partir de ces phrases ;

— *Données de test* : On génère 100 phrases à partir de 40 patterns qui n'ont **pas** été utilisés pour générer le train set. On constitue ensuite un ensemble de 1000 paires de test.

Pour chaque paire de phrases, le label de similarité sémantique est calculée comme expliqué en Partie 2.3.6 : une paire de phrases dont les intentions sont similaires est affectée d'un 1, 0 sinon.

Formulé de cette manière, le problème est légèrement différent de celui de SemEval et s'apparente plutôt à une classification binaire : le modèle répond à la question « Ces deux phrases expriment-elles la même intention ? ». Plutôt que la corrélation de Pearson

TABLE 2.3 – Evaluation du modèle de similarité sémantique sur *UserGrammar*

Modèle	F_1 score		Précision	Recall
	moyenne	écart-type	moyenne	moyenne
MLP Binaire	0.77	0.09	0.75	**0.84**
MLP multi-classe	0.78	0.08	0.76	**0.84**
Forêt aléatoire	0.82	0.08	**0.88**	0.77
XGBoost	0.80	0.09	0.85	0.77
Modèle complet	**0.85**	**0.05**	0.87	0.82

utilisée pour SemEval, on mesure le F_1 score, la précision et le rappel (*recall*) des modèles entraînés sur les données d'entraînement et évalués sur les données de test. La précision mesure la proportion de vrais positifs parmi les paires classées positivement et le rappel mesure la proportion de paires positives trouvées. Le F_1 score est une moyenne harmonique de ces deux métriques permettant d'avoir une idée globale du comportement du modèle.

Ces métriques qui peuvent sembler plus difficiles à interpréter sont cependant plus pertinentes qu'une mesure classique d'*accuracy* (proportion de paires correctement classifiées). En effet, les jeux de données sont fortement déséquilibrés avec une proportion de paires similaires oscillant entre 2 et 4%. Un modèle qui prédirait constamment 0 possède ainsi une accuracy d'au moins 96% mais n'est d'aucune utilité. Il possède en revanche une précision de 0 et un recall de 0. On reprend en Figure 2.4, une figure reprise de Wikimedia Commons [3] qui explicite visuellement le sens de ces métriques.

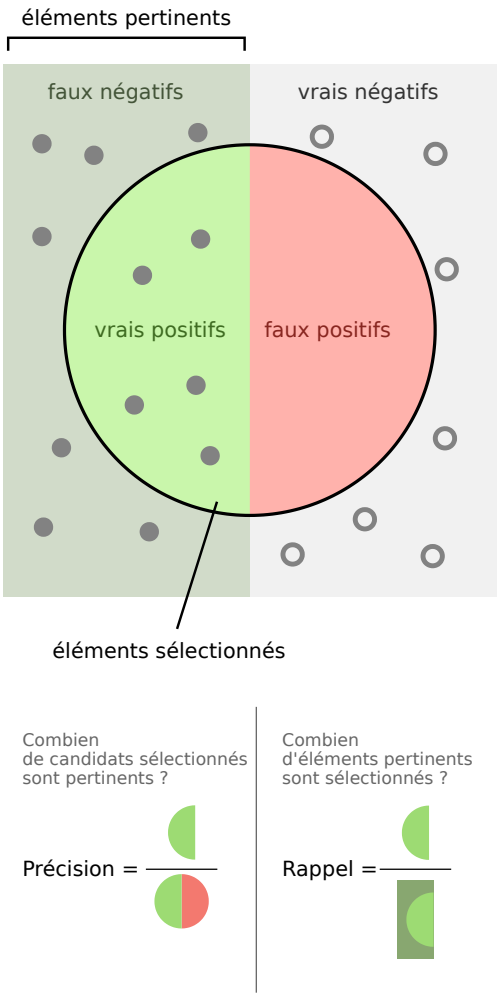


FIGURE 2.4 – Précision et Rappel[3]

Pour obtenir une mesure fiable, on répète cette expérience à partir de 30 corpus d'entraînement et de tests différents, la moyenne et l'écart-type sont indiqués dans le Tableau 2.3. On rappelle qu'aucune des phrases ni constructions de l'ensemble de test n'a été vu pendant l'entraînement.

[3]Source : Par Datamok / User : Walber — File :Precision_recall.svg, CC BY-SA 4.0, Lien

Pour montrer l'intérêt d'un modèle ensembliste, on mesure la performance de chacun des sous-modèles. On observe ainsi que les modèles d'arbres de décision sont plus précis que les modèles neuronaux mais présentent un rappel plus faible. Le modèle global présente le meilleur F_1 score, un équilibre entre précision et rappel, c'est aussi le modèle le plus stable (écart-type le plus faible). L'équilibre entre précision et rappel est important pour disposer d'un modèle ni trop conservateur (haute précision, faible rappel) ni trop optimiste (faible précision, haut rappel).

2.4.2 Simulation des interactions avec un utilisateur

Conditions de la simulation

Dans cette partie on simule des interactions entre AidMe et un utilisateur en s'appuyant sur la grammaire décrite en Partie 2.4.1. On peut ainsi estimer l'apport d'AidMe et le comparer avec des systèmes existants. On génère un ensemble de 700 requêtes-utilisateur qui représentent donc 700 interactions entre l'utilisateur et un assistant. Au début de la simulation, AidMe ne connaît aucune intention ni patterns ($\mathcal{I} = \emptyset$ et $\mathcal{P} = \emptyset$). Les requêtes sont présentées les unes après les autres dans un ordre aléatoire. À chaque fois, AidMe doit déterminer la bonne intention et le pattern associé et lorsque qu'il échoue, on fournit à AidMe la bonne intention et le bon pattern (voir Partie 2.1.2). AidMe peut ainsi se mettre à jour. Ce fonctionnement est illustré en Figure 2.2.

Dans les simulations, on utilise $\epsilon = 0.3$ pour l'Algorithme 2.1. Cet hyperparamètre peut être choisi selon si l'on souhaite disposer d'un système plus ou moins prudent : plus ϵ est faible, moins on demande au système d'être confiant dans ses prédictions et donc moins il est prudent. Le modèle de similarité sémantique est entraîné toutes les 5 interactions.

Comparaison avec des systèmes concurrents

Pour mesurer la performance d'AidMe, on le compare avec différents systèmes :

— OneShotNLU : il s'agit d'un système de pattern matching. Pour interpréter une requête, ce système tente de faire coïncider l'un des patterns connus sur la requête. Ce système a ainsi besoin de voir un pattern exactement une fois avant de pouvoir l'interpréter (*one-shot*). Un tel système ne fait pas la différence entre une intention et un pattern et ne connaît pas non plus la notion de pattern similaires. C'est ce type de système qui est par exemple employé par Delgrange et al. (2020).

— AidMeStat est une variante de AidMe dans laquelle le système utilise toujours les algorithmes de détections de l'intention et du pattern (Algorithmes 2.1 et 2.2)

sans tenter de faire du pattern matching à partir des patterns déjà connus. AidMe combine, en quelque sorte, les avantages de OneShotNLU et AidMeStat

— DFLearner : est l'adaptation d'un système de NLU classique au cas des assistants apprenants. On remplace les algorithmes de détection de l'intention et du pattern (Algorithmes 2.1 et 2.2) par l'API de DialogFlow [4] de Google. Au début de la simulation, l'instance de Dialogflow démarre avec un ensemble vide d'intentions et d'exemples de phrases. Au fur et à mesure que se déroule la simulation, on ajoute intentions et exemples à l'instance pour que DialogFlow puisse s'entraîner. Ce système ne peut découvrir seul de nouvelles intentions car il propose systématiquement une intention connue pour interpréter la requête courante. Il peut en revanche interpréter correctement une requête dont l'intention est connue mais pas le pattern.

— TrainedAidMe est une variante d'AidMe où le modèle de similarité sémantique ne part pas de zéro mais est pré-appris sur un corpus de 2000 paires de phrases. Il est mis à jour au cours de la simulation avec les données collectées. À la différence d'AidMe, le modèle est ré-entraîné toutes les 50 interactions. On précise que dans cette variante, TrainedAidMe n'a pas non plus connaissance des intentions et patterns ne fait pas usage de pattern matching (comme AidMeStat).

2.4.3 Résultats des simulations

On évalue ces différents systèmes et AidMe en réalisant 10 simulations à partir de différents ensembles de requêtes générés avec *UserGrammar*. On analyse les performances en termes d'interprétation complète des requêtes, mais aussi sur les tâches intermédiaires que représente la détection de l'intention et des nouveaux patterns.

Qualité d'interprétation des requêtes

On s'intéresse d'abord à la capacité de ces systèmes à interpréter correctement les requêtes des utilisateurs. On mesure le taux de succès global d'interprétation des requêtes au cours de la simulation. On calcule plus spécifiquement ce taux dans le cas où l'intention était connue mais pas le pattern, et dans le cas où l'intention et le pattern étaient connus. (Dans le cas où l'intention n'est pas connue, aucun système ne peut interpréter la requête.) On reporte les résultats dans le Tableau 2.4. La fréquence indique la proportion des requêtes qui sont concernées.

OneShotNLU étant un système de pattern matching, il interprète correctement une requête si et seulement si le pattern est connu. Les performances de OneShotNLU indiquent ainsi la distribution des nouveaux patterns par rapport aux patterns déjà connus.

[4] https://dialogflow.com/

Taux de succès d'interprétation des requêtes(%)				
	Intention	connue		**Global**
	Pattern	inconnu	connu	
	Fréquence	24%	69%	
Modèle	OneShotNLU	0	100	69
	DFLearner	12	33	26
	AidMeStat	23	31	29
	TrainedAidMe	26	28	28
	AidMe	**23**	**100**	**75**

Table 2.4 – Taux de succès d'interprétation des requêtes

Taux de succès de détection des intentions connues (%)				
	Pattern	inconnu	connu	**Global**
	Fréquence	25%	75%	
Modèle	OneShotNLU	0	100	75
	DFLearner	52	65	63
	AidMeStat	51	92	84
	TrainedAidMe	55	92	82
	AidMe	**51**	**100**	**89**

Table 2.5 – Taux de succès de détection des intentions connues en fonction de la connaissance du pattern

Cela signifie que dans 31% des cas, la requête présentée lors de la simulation était issue d'un nouveau pattern. Il faut noter que 7% des requêtes introduisent une nouvelle intention et qu'aucun des systèmes n'est en mesure de l'interpréter correctement. Le score global maximal atteignable est donc de 93% et non de 100%. AidMe parvient à interpréter correctement 75% des requêtes.

On observe que l'utilisation des seules modèles de similarité sémantique ne permet une interprétation correcte que pour environ 28% des requêtes, et cela indépendamment du pré-entraînement du modèle de similarité sémantique avant la simulation. Cependant on note que la performance globale d'AidMe est 6 points au-dessus de celle de OneShotNLU. Cela indique qu'AidMe a été en mesure d'identifier automatiquement certains nouveaux patterns. Dans la suite, pour mieux comprendre le fonctionnement d'AidMe, on décompose l'analyse des résultats pour chacune des phases : (1) la détection de l'intention (Algorithme 2.1) et (2) l'identification de nouveaux patterns (Algorithme 2.2).

Qualité de la détection de l'intention

On reporte dans le Tableau 2.5 les performances des différents modèles en termes de détection de l'intention de l'utilisateur suivant si le pattern est connu ou non. On reporte aussi la performance globale. Le taux de succès de détection de l'intention est calculé uniquement en prenant en compte les requêtes faisant référence à une intention connue. De même, la fréquence est différente de celle rapportée en Tableau 2.4 car elle est calculée uniquement parmi les intentions connues.

On peut noter que DFLearner et les systèmes basés sur AidMe parviennent à détecter l'intention dans la moitié des cas où ils ne connaissent pas le pattern. On note une véritable progression des performances de DFLearner par rapport aux simulations conduites en 2019 (Lair et al., 2020). En effet, à l'époque DFLearner ne détectait jamais l'intention de l'utilisateur. On peut attribuer cette progression au passage à la v2 de Dialogflow. DFLearner voit ses performances plafonnées à 65% tandis que AidMeStat atteint 90% pour les patterns connus.. On observe un léger avantage pour TrainedAidMe sur AidMeStat pour la détection des intentions pour un pattern inconnu.

Qualité d'identification des patterns

On s'intéresse enfin au taux de détection des patterns de chacun des systèmes : la proportion des patterns correctement identifiés sachant que l'intention avait correctement été identifiée. Il diffère du taux de succès d'interprétation qui mesure simplement la proportion de requêtes correctement interprétées sans tenir compte de si l'erreur est à attribuer à l'étape de détection de l'intention ou d'identification du pattern. On présente les résultats dans le Tableau 2.6.

On observe qu'AidMeStat et TrainedAidMe sont plus performants pour détecter les nouveaux patterns que les patterns connus. Cela paraît étonnant et l'on a pas d'hypothèse réellement convaincante pour expliquer ce phénomène. Il est possible que la dynamique de la simulation et la manière dont les données d'apprentissage sont collectées conduise le modèle de similarité sémantique à une situation de sur-apprentissage sur la tâche de détection de l'intention et que le modèle perde certaines propriétés de généralisation nécessaire pour la tâche de détermination du pattern.

DFLearner ne différencie pas explicitement intention et pattern, la mesure rapportée ici correspond à la proportion des requêtes pour lesquelles les bons arguments ont été identifiés lorsque l'intention a été bien détectée.

Taux de succès de détection des patterns (%)				
	Pattern	inconnu	connu	**Global**
Modèle	DFLearner	26	52	45
	AidMeStat	41	36	37
	TrainedAidMe	45	30	37

Table 2.6 – Taux de succès d'identification des patterns, calculée sur l'ensemble des requêtes dont l'intention a été détectée

Table 2.7 – Taux de succès d'identification des patterns en fonction du nombre d'arguments

Taux de succès de détection des patterns (%)										
	Pattern	inconnu			connu			**Global**		
	N° arguments	1	2	3	1	2	3	1	2	3
	Fréquence	13%	14%	3%	21%	33%	14%	34%	47%	17%
Modèle	AidMeStat	71	28	0	66	29	10	67	28	9
	TrainedAidMe	74	27	18	57	16	5	73	16	4
	AidMe	71	28	0	100	100	100	94	88	92

Qualité d'identification des patterns en fonction de leur complexité

On peut raffiner un peu cette analyse en s'intéressant à la découverte des nouveaux patterns en fonction de leur complexité. L'identification d'un pattern est plus ou moins difficile en fonction, notamment, du nombre d'arguments à identifier dans la requête. Nous avons relevé dans le Tableau 2.7 les performances d'AidMe en différenciant les patterns en fonction du nombre d'arguments et l'on constate qu'il est très difficile de découvrir de nouveaux patterns qui auraient plus de deux arguments. Tout comme précédemment, ce taux est calculé parmi les requêtes pour lesquels l'intention a bien été détectée afin de ne mesurer que les performances de l'Algorithme 2.2. Comme on peut s'y attendre, la capacité de détection des nouveaux patterns décroît avec le nombre d'arguments et il semble qu'au-delà de 2 arguments, AidMe ne soit plus en mesure de découvrir les nouveaux patterns. En effet, avec le nombre d'arguments qui augmente, le nombre de patterns candidats (p_c) à partir d'une requête explose. C'est la plus importante limitation introduite par l'heuristique derrière AidMe.

2.4.4 Notes sur chacun des systèmes concurrents

OneShotNLU – Les performances de OneShotNLU peuvent paraître étonnamment élevées dans ces simulations. Le nombre de patterns pour chaque intention est relativement

faible dans *UserGrammar* et cela permet à OneShotNLU d'être rapidement performant. En effet, la priorité a été donnée de disposer d'une grammaire couvrant plusieurs domaines plutôt que de disposer de nombreux patterns dans une gamme de domaines plus restreinte. C'est en effet précisément dans le cas des assistants multi-domaines qu'une solution d'agent apprenant par interaction, et donc d'AidMe, est le plus pertinent.

Plus le nombre de patterns par intention diminue, et plus les performances de OneShotNLU diminuent aussi. Les performances de OneShotNLU sont donc plutôt à voir comme un indicateur de la distribution des requêtes plutôt que la performance d'un système utilisable en conditions réelles. En effet, on a supposé un retour systématique de l'utilisateur dans ces simulations et OneShotNLU est très dépendant de ce retour. Enfin, en augmentant le nombre de patterns par intention, on fait mécaniquement baisser les performances de OneShotNLU.

AidMeStat – AidMeStat permet d'évaluer la performance et la pertinence du modèle de similarité sémantique pour les tâches de détection de l'intention et des patterns. On observe que seul, AidMeStat ne permet pas de disposer d'un système utilisable. En revanche les performances pour détecter les nouveaux patterns sont prometteuses et 23% des patterns peuvent être découverts automatiquement. La combinaison avec le pattern matching permet à AidMe d'interpréter correctement 75% des requêtes sur les 93% des requêtes interprétables.

DFLearner – Les performances de DFLearner ont véritablement progressé lors ces nouvelles simulations en comparaison avec les simulations de 2019. On observe que DFLearner se rapproche d'AidMe. Pour autant, DFLearner ne constitue pas une solution viable à notre problème et AidMe conserve son avantage en identifiant près de deux fois plus de patterns inconnus. Cela illustre la difficulté de cette tâche d'apprentissage en interaction à partir d'une quantité de données très restreinte. On peut avancer plusieurs raisons pour expliquer les difficultés de DialogFlow sur cette tâche :

— DialogFlow s'appuie sur une combinaison entre apprentissage automatique et système de règles. Lors des simulations, nous avons eu des avertissements de DialogFlow indiquant que le nombre d'exemples fourni par intention était trop faible pour que le modèle soit entraîné correctement. Cela confirme l'avantage d'une méthode d'identification de l'intention indirecte (comme celle d'AidMe par comparaison de la similarité sémantique) par rapport à une méthode directe qui classifie directement l'intention. En effet, avec n phrases, une approche directe peut s'appuyer sur un corpus de n exemples tandis que pour une approche indirecte, on peut construire un corpus de n^2 exemples.

— *UserGrammar* couvre volontairement une série de domaines et les intentions exprimables sont donc très variées, parfois très proches les unes des autres et parfois sans aucun rapport. DialogFlow ne semble pas en mesure gérer cette variabilité, en tout cas avec une réactivité suffisante.

— DialogFlow a besoin de connaître le typage des arguments de chaque intention. Par souci d'égalité avec AidMe qui ne connaît pas le typage des arguments et parce qu'il est difficile d'imaginer apprendre les types DialogFlow des arguments par interaction, on ne donne pas les types d'arguments à DialogFlow (type par défaut `@sys.any`). L'idée de pouvoir apprendre le typage des arguments au cours des interactions est cependant une piste qu'il serait intéressant de poursuivre dans de futurs travaux.

TrainedAidMe – Les performances de TrainedAidMe sont légèrement meilleures que celles d'AidMeStat mais le gain paraît marginal au regard de la quantité de données avec lesquelles TrainedAidMe démarre. Cela conforte l'idée que la capacité d'adaptation du modèle en temps réel est suffisante et préférable à un modèle pré-appris.

2.4.5 Etude de la dynamique des simulations

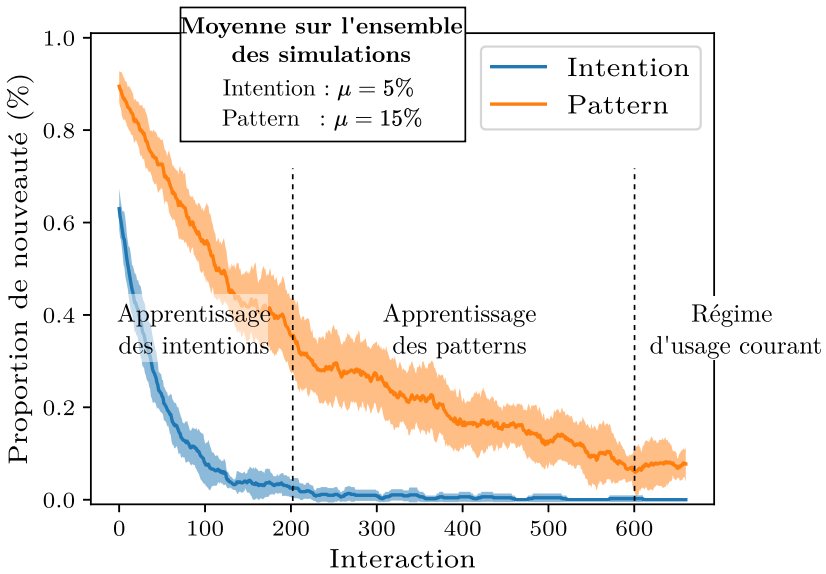


Figure 2.5 – Différents régimes durant la simulation

Dans cette expérience, on simule l'interaction entre un utilisateur qui emploie pour la première fois un assistant digital personnalisable. On peut s'intéresser à la dynamique des interactions et identifier trois phases :

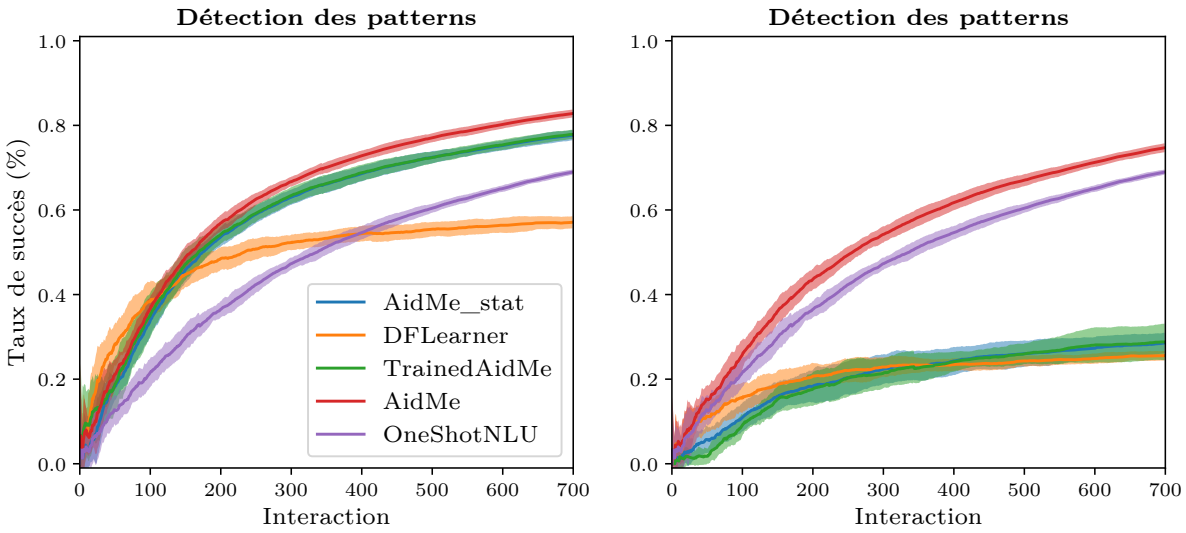


FIGURE 2.6 – Moyenne cumulée du taux de succès au cours des simulations

1. **Découverte des intentions** : Durant cette phase, la plupart des requêtes font référence à des intentions de l'utilisateur qui sont nouvelles pour le système. À ce moment, quel que soit le système employé, l'assistant doit déclencher un mode d'apprentissage ;
2. **Découverte des patterns** : Durant cette phase, les intentions sont essentiellement connues mais les patterns employés par l'utilisateur ne le sont pas. C'est dans cette phase-là que le gain permis par AIDME est le plus important ;
3. **Régime d'usage courant** : Durant cette phase, l'essentiel des intentions et des patterns employés par l'utilisateur sont connus. L'utilisateur a alors appris à son assistant l'essentiel des commandes pour lesquelles il souhaite l'utiliser.

On peut constater ces différentes phases en traçant la proportion des nouvelles intentions et nouveaux patterns au cours de la simulation. La Figure 2.5 présente cette proportion comme la moyenne roulante sur 25 interactions. On peut tenter de visualiser la dynamique d'apprentissage en traçant les taux de succès cumulés ou instantanés au cours de la simulation. Le taux de succès cumulé est simplement le taux moyen de succès depuis le début de la simulation et le taux de succès instantané correspond à la moyenne locale des succès sur une fenêtre bien choisie.

2.4.6 Curriculum Learning : Ordonner les requêtes par difficulté croissante

Dans cette partie, on s'interroge sur la pertinence de mettre en place une forme de curriculum learning pour AIDME. On peut en effet imaginer qu'un assistant bénéficierait

de découvrir d'abord les requêtes considérées comme plus « faciles » avant d'apprendre les requêtes plus « difficiles ».

On a déjà évoqué que la complexité d'identification d'un pattern était notamment liée au nombre d'arguments à détecter. Plus généralement, on peut définir la complexité d'une requête en considérant le nombre de domaines qui sont concernés par la requête ainsi que le nombre d'arguments nécessaires à l'interprétation de la requête. Le nombre de domaines donne une indication sur la richesse sémantique de la requête et le nombre d'arguments sur la complexité computationnelle de la tâche mais aussi sur la finesse de compréhension nécessaire à son interprétation. Par exemple, la requête « *Can you buy* `num1` *tickets for* `pers1` *concert, then inform* `pers2` *about it* » mélange deux domaines (réservations de billets et envoi de mail) et possède trois arguments. Elle est ainsi affectée d'un score de complexité élevée.

On compare ensuite des simulations dans lesquelles les requêtes sont présentées à AidMe dans un ordre aléatoire comme précédemment ou par ordre de difficulté croissante. Les résultats sont présentés en Figure 2.7. Les deux premiers graphiques présentent les taux moyens de détection de l'intention et du pattern (sans pattern matching - correspond à AidMeStat) de la requête. On observe que l'assistant qui a reçu le corpus trié est plus performant au début de la simulation. Cet avantage se maintient légèrement au cours de la simulation uniquement pour la détection d'intention. Le troisième graphique représente le gain en termes de succès d'interprétation par rapport à un système qui ne ferait que du pattern matching. On observe à nouveau que le gain du curriculum ne se fait qu'au début de la simulation

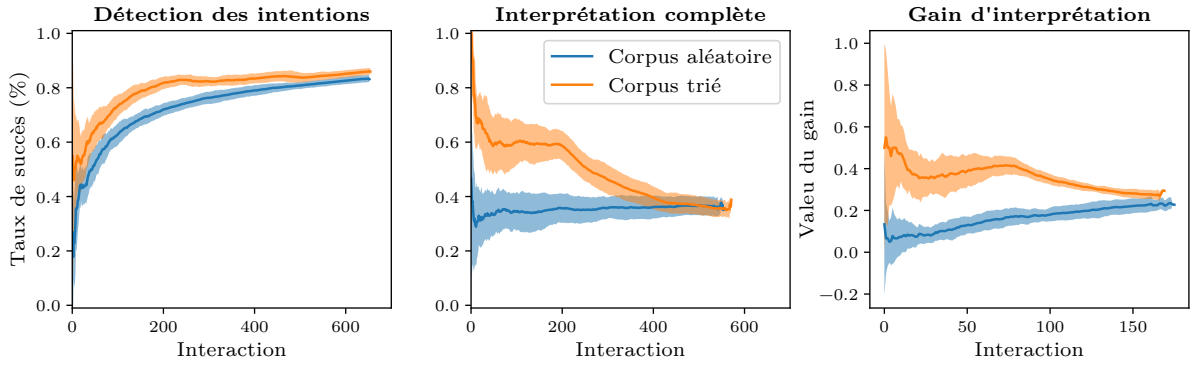


Figure 2.7 – Effet de l'ordre dans laquelle sont présentées les requêtes

2.4.7 Effet de la fréquence d'entraînement

Dans cette partie, on s'intéresse à l'impact de la fréquence d'entraînement du modèle de similarité sémantique sur les performances en termes de détection de l'intention et

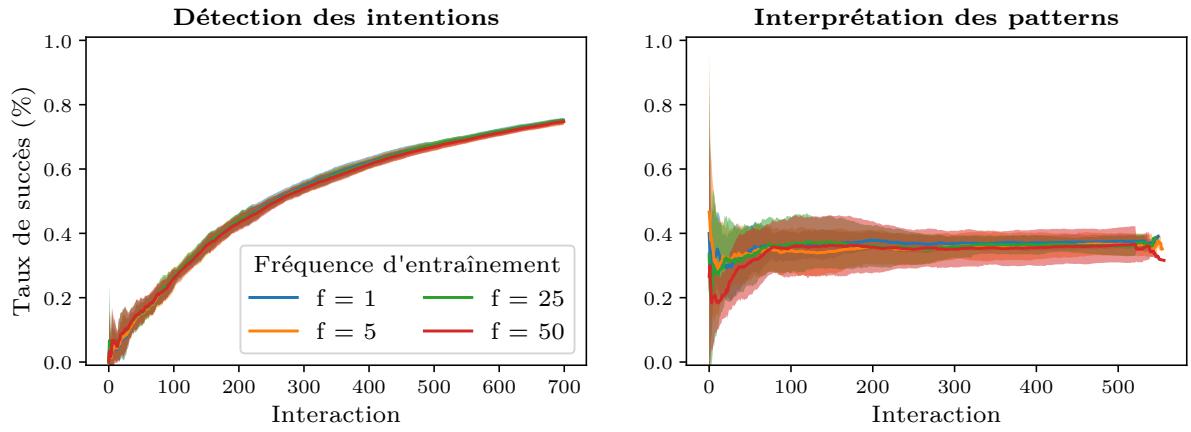


FIGURE 2.8 – Moyenne cumulée sur la simulation

d'interprétation complète de la requête. On étudie 4 fréquences d'entraînement : après chaque interaction, toutes les 1, 5, 25 et 50 interactions. Pour les fréquences d'entraînement 5, 25 et 50, le modèle de similarité sémantique est initialisé au début de la simulation avec un exemple positif et un exemple négatif. Cela évite que ces modèles ne soient inutilisables pendant les premières interactions (avant le premier entraînement effectué à partir des données collectées) et explique pourquoi les modèles ne commencent pas à 0 au début des simulations.

On reporte dans la Figure 2.8 le taux de succès cumulés sur la simulation (comme pour laFigure 2.6). On observe pas de différence significative entre les différentes fréquences d'entraînement. Cela peut s'expliquer par le fait qu'au début tous les modèles sont trop peu entraînés et connaissent trop peu d'intentions pour être performants et qu'ils s'appuient soit sur des words *embeddings* qui encodent déjà une notion de sémantique soit sur des features d'analyse sémantique.

2.4.8 Effet de la fréquence de retour utilisateur

On s'intéresse maintenant à un relâchement partiel de l'hypothèse formulée en Partie 2.1.2 quant à la nécessité de retour de l'utilisateur pour permettre l'apprentissage d'AIDME. La Figure 2.9 présente les résultats des simulations effectuées pour comparer un assistant qui aurait reçu un retour de l'utilisateur après chaque interaction et un assistant qui n'aurait reçu un retour que la moitié du temps (en pratique l'oracle a une probabilité 0.5 de fournir un retour). On observe sans surprise que l'assistant qui reçoit un retour à chaque interaction est plus performant tout au long de la simulation tant sur la détection des intentions que sur l'interprétation complète des requêtes. On peut noter cependant que la baisse de performance est moins importante que ce que l'on pouvait craindre.

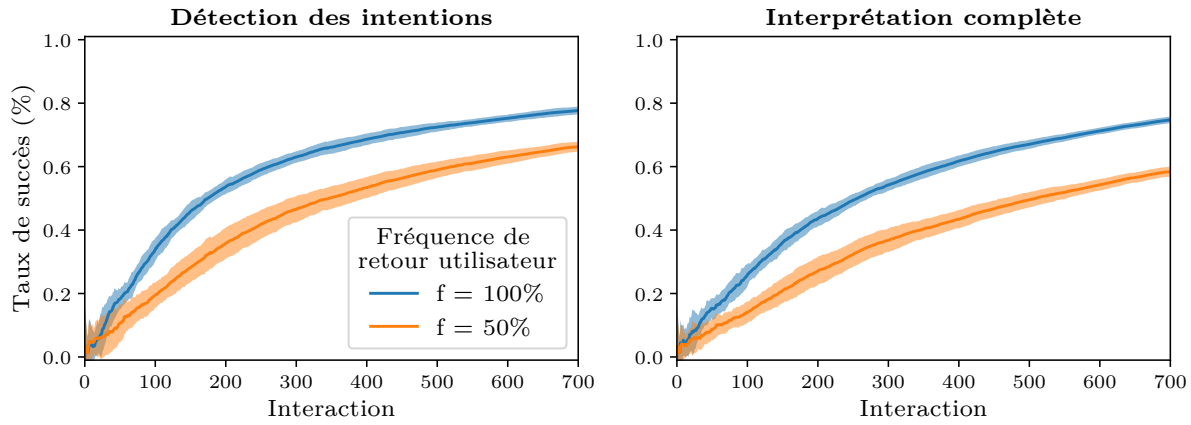


Figure 2.9 – Effet d'une baisse du retour de l'utilisateur

2.5 Intégration avec un assistant digital

Nous avons intégré AidMe avec l'assistant Jarvis proposé par Delgrange et al. (2020). Cet assistant apprend à partir de démonstrations effectuées sur une interface graphique et des instructions en langage naturel. L'interaction avec cet assistant se fait via un chatbot dans un navigateur ou sur un smartphone. Dans son fonctionnement standard, l'assistant apprend la correspondance entre une phrase en langage naturel et une séquence d'actions sur l'interface graphique (la procédure). Par exemple si l'utilisateur demande à l'assistant « Envoie un mail à Alice », le système cherche la procédure correspondante. S'il s'agit d'une procédure connue employant un pattern connu, le système extrait automatiquement Alice comme étant l'argument de la procédure et la réalise. Sinon l'assistant demande à l'utilisateur une démonstration sur l'interface graphique ou une explication de la manière dont réaliser la procédure.

L'utilisateur est forcé de retenir la forme verbale exacte avec laquelle il a enseigné une procédure pour pouvoir la réutiliser. S'il emploie une forme verbale même légèrement différente, le système n'est pas capable de faire le lien avec la forme connue et se comporte comme si l'utilisateur lui demandait quelque chose de nouveau en requérant une démonstration ou une explication. Cela peut rendre le système difficile à l'emploi.

AidMe permet à l'assistant d'éviter environ 24% de ces situations en découvrant de manière pro-active de nouvelles formes verbales. En pratique, AidMe fournit à l'assistant une reformulation de la requête dans une forme qu'il comprend. L'assistant peut demander confirmation à l'utilisateur ou effectuer directement la requête comme illustré ci-dessous. AidMe apprend directement à partir des données collectées directement par l'assistant et s'adapte ainsi à l'utilisateur.

Utilisateur — J'aimerais que tu envoies un mail à Bob.

Assistant — Me demandez-vous : Envoie un mail à Bob ?

Utilisateur — Oui !

Sur le plan technique, le système JARVIS est implémenté en java et AIDME en python. La communication entre les deux systèmes se fait via une API REST qui permet à JARVIS d'envoyer une requête de l'utilisateur et l'ensemble des requêtes avec laquelle la comparer et à AIDME de répondre avec les scores de similarité sémantique correspondant. JARVIS peut aussi déclencher l'entraînement du modèle de similarité sémantique en fournissant à AIDME les données.

2.6 Discussion

2.6.1 Approche par comparaison : le pour et le contre

Fréquence d'entraînement – La méthode employée pour interpréter les requêtes de l'utilisateur peut être qualifiée d'*indirecte*. On ne cherche pas directement l'intention qui permette d'interpréter correctement une requête mais la requête connue la plus proche. Un système de NLU traditionnel, quant à lui, fonctionne comme un classifieur. Il doit s'entraîner à chaque fois qu'une nouvelle intention est trouvée pour permettre sa détection dans le futur. Dans le cas d'un usage en conditions réelles, cela signifie que l'on contrôle difficilement les moments où le modèle a besoin de s'entraîner. Le modèle s'entraîne plus souvent, il est donc moins souvent disponible. À l'inverse le modèle de similarité sémantique est agnostique des intentions, son ré-entraînement n'est pas nécessaire pour détecter une intention nouvellement acquise. L'entraînement du modèle permet une meilleure prise en compte du langage de l'utilisateur. Il peut être effectué à tout moment, et notamment en dehors des moments d'utilisation. L'approche par comparaison permet ainsi d'augmenter la disponibilité du modèle par rapport à une approche directe tout en garantissant une performance optimale.

Sample efficiency – Comme évoqué déjà lors de la comparaison entre AIDME et DFLEARNER, les méthodes par comparaison sont aussi bien plus efficaces en termes de besoins en données. Dans le cas d'AIDME, n phrases conduisent à un corpus n^2 paires de phrases. Pour un classifieur multi-classes, l'un des challenges est notamment de s'assurer d'un certain équilibre entre les classes dans les données d'entraînement pour éviter qu'une classe sous-représentée ne soit ignorée par le modèle. Ainsi lorsqu'une nouvelle classe est découverte, cette classe qui ne contient qu'un exemple sera considéré comme très peu probable par le modèle et risque d'être ignorée. Un nombre relativement important d'exemples de chaque classe sera ainsi requis avant que le modèle ne propose une performance satisfaisante sur cette classe d'intentions.

Complexité algorithmique – L'un des problèmes que posent les méthodes par comparaison est leur complexité algorithmique. L'Algorithme 2.1 a une complexité linéaire en le nombre de phrases connues ($\mathcal{O}(|\mathcal{S}|)$). Cela signifie que plus le nombre de requêtes connues augmente, plus AidMe est long à répondre. On peut cependant proposer des heuristiques permettant de limiter le nombre de comparaisons effectuées par AidMe : ne comparer la requête de l'utilisateur qu'à un nombre défini de requêtes par intentions connues par exemple. L'Algorithme 2.2 a une complexité qui augmente exponentiellement avec le nombre d'arguments de l'intention identifiée $\mathcal{O}(w^{|ARGS|})$ où w est le nombre de mots de la requête et $|ARGS|$ le nombre d'arguments du pattern. Ainsi pour une requête comportant 10 mots et 3 arguments conduit à effectuer 720 comparaisons. Même si le temps d'inférence est correct, cela pose des problèmes pour de longues requêtes ou des patterns comportant de nombreux arguments. Pour contrôler la longueur des requêtes, on peut s'appuyer sur la seconde méthode évoquée en Partie 2.2.2 et remplacer la requête la plus proche trouvée par la plus courte requête de la même classe d'intention ($\arg\min\{|s| \mid s \in \mathcal{I} \text{ et } s \sim s_0\}$). On n'a cependant aucun moyen simple de contrôler le nombre d'arguments.

Sensibilité aux erreurs – L'une des limites importantes de l'approche est sa sensibilité à l'introduction d'erreurs dans les données d'apprentissage. Si par erreur, deux requêtes sont identifiées comme liées à la même intention, il n'est pas possible en l'état de corriger par interaction cette erreur et la requête dont l'intention enregistrée est erronée sera toujours mal interprétée. En simulation, on peut supposer que le retour de l'utilisateur est parfait cependant en pratique un utilisateur peut se tromper et introduire ainsi des erreurs dans les connaissances d'AidMe. Pour pouvoir déployer une solution il est nécessaire d'imaginer un protocole de correction d'erreur. On souhaiterait que ce protocole soit automatique si possible.

2.6.2 Peut-on partager un même assistant entre plusieurs utilisateurs ?

Le principe même de ces assistants apprenants est de s'adapter au mieux à l'utilisateur. C'est dans cet esprit qu'a été développé AidMe. On est toutefois en droit de s'interroger si l'assistant d'un utilisateur peut bénéficier des connaissances ou des données acquises par l'assistant d'un autre utilisateur. Dans le cas général, cela semble difficile car deux utilisateurs peuvent exprimer de la même manière deux requêtes qui sont en réalité différentes du point de vue de l'exécution, notamment lorsque cela concerne les préférences ou les habitudes de l'utilisateur (ex : deux utilisateurs qui n'utiliseraient pas les mêmes logiciels de messagerie).

Dans le cas d'AidMe, il semble que l'on puisse tout de même mutualiser une partie des données collectées pour améliorer les performances globales. On pourrait notamment partager le modèle de similarité sémantique en considérant que la notion de similarité sémantique n'est pas fondamentalement modifiée par des divergences en termes de préférences individuelles. La fonction de similarité sémantique est bien plus dépendante des cas d'utilisation que des individus utilisateurs. Comme illustré par la Figure 2.10, on peut concevoir un système dans lequel plusieurs utilisateurs partagent le même modèle de similarité sémantique tout en conservant les spécificités propres à chacun. Un tel modèle nécessiterait d'être testé afin de valider ces hypothèses. On pourrait aussi envisager le partage d'un modèle de similarité sémantique appris sur les données communes des utilisateurs et *fine-tuner* sur les données spécifiques de chaque utilisateur.

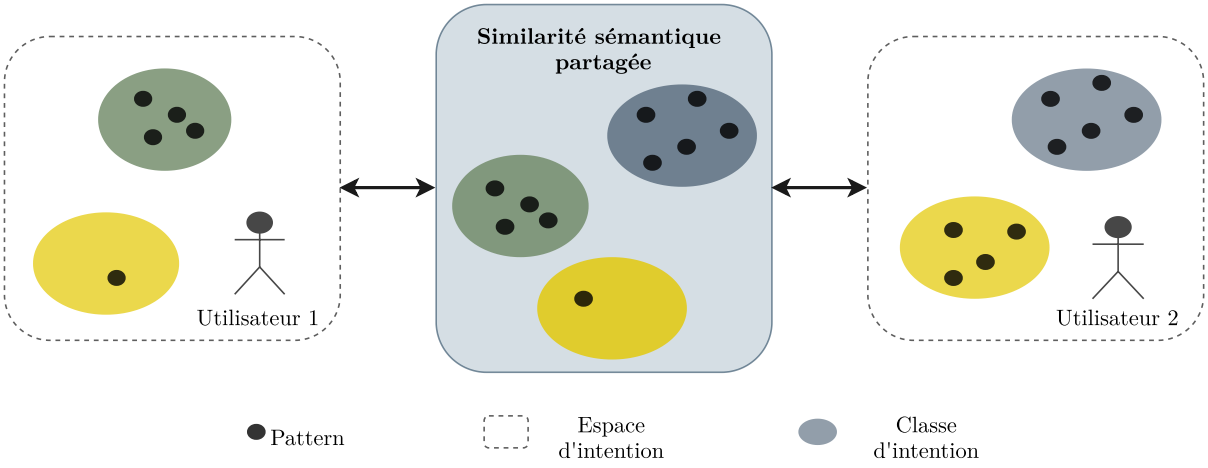


Figure 2.10 – Partage du modèle de similarité sémantique entre utilisateurs

2.6.3 Comment améliorer la détection des nouveaux patterns ?

L'une des principales limitations d'AidMe est la détection des nouveaux patterns. On évoque des pistes qui permettraient d'augmenter sa performance et de diminuer la complexité computationnelle :

Typage des arguments – Un premier axe d'amélioration d'AidMe est l'utilisation d'un typage des arguments de chaque requête qui permettrait de réduire l'espace de recherche des patterns à quelques candidats bien choisis. On entend par typage une information sur les caractéristiques des arguments des patterns comme une date, un nom, un lieu, une adresse mail, un nom d'application... Par exemple la requête « Trouve-moi les vols de Paris à Tokyo pour demain ? » possède trois arguments, Paris et Tokyo sont des lieux et demain indique une date. La majorité des systèmes de NLU emploie un typage des arguments et l'absence de typage dans notre environnement est l'une des raisons des

difficultés de DFLearner. En supposant que l'on sache déterminer le typage des éléments d'une requête parsée, la découverte du pattern sous-jacent devient bien plus facile car seuls certains mots de la requête peuvent jouer le rôle d'argument. On peut traiter ainsi le problème de combinatorialité des patterns lié à notre approche, réduisant la complexité l'Algorithme 2.2. La notion de typage pose cependant des questions difficiles dans le cadre d'un système apprenant par interaction :

— Les types sont-ils connus à l'avance ou sont-ils découverts par l'agent ? L'option la plus simple consisterait à définir des types standards suffisamment généraux pour couvrir toutes les possibilités mais suffisamment spécifiques pour que le gain soit visible.

— Comment déterminer le typage des éléments d'une requête parsée ?

— Est-il possible d'apprendre le typage des arguments ?

Analyse de la syntaxe – Un second axe d'amélioration consisterait à effectuer une analyse syntaxique des patterns candidats. Pour mémoire, dans l'Algorithme 2.2, étant donné le nombre d'arguments de l'intention cible, on construit l'ensemble $P_c = \{(p_c, args_c) \mid \lambda(p_c, args_c) = s\}$ des paires (patterns, arguments) possibles. On instancie ensuite les phrases $\lambda(p_0, args_c)$ à partir du pattern p_0 de référence et les arguments possibles de P_c. Il est très probable que la majorité des phrases formées soient syntaxiquement invalides. En vérifiant la syntaxe des phrases ainsi formées, on peut écarter une partie des patterns candidats p_c et ainsi ne tester que les patterns qui conduisent à des phrases syntaxiquement correctes. La principale limite d'une telle méthode est que notre système est destiné à être employé en interaction avec des utilisateurs dont la syntaxe des requêtes n'a pas de garantie d'exactitude.

Similarité sémantique dans l'espace des patterns – Enfin on peut raffiner la manière dont on calcule la similarité des patterns. La méthode de détection des nouveaux patterns et celle de détection de l'intention s'appuie sur une heuristique commune : en première approximation, on peut considérer qu'un élément inconnu appartient à la même classe que l'élément connu le plus proche relativement à une distance (ou pseudo-distance) donnée. Pour la détection des intentions, la pseudo-distance utilisée est la similarité sémantique. Elle permet d'estimer assez naturellement la proximité sémantique entre deux phrases. Dans l'espace des patterns, la notion de similarité sémantique est plus difficile à définir et surtout à estimer. En Partie 2.2.2, on estime la similarité sémantique entre deux patterns par la similarité sémantique contextuelle. En première approximation, cela permet de valider la méthodologie employée mais les limitations auxquelles on se heurte sont notamment dues à la difficulté d'estimer la similarité sémantique entre des

patterns. L'une des limites de la similarité sémantique contextuelle est sa dépendance aux arguments. Pour s'en affranchir, un moyen simple peut consister à proposer un nouvel estimateur de la similarité sémantique entre deux patterns en moyennant la similarité contextuelle sur un ensemble d'arguments :

$$Sim_{|ARGS}(p_1, p_2) = \frac{1}{|ARGS|} \sum_{args \in ARGS} Sim(\lambda(p_1, args), \lambda(p_2, args))$$

où $ARGS$ sont des ensembles d'arguments associés à p_1 et p_2. Le gain permis par cet estimateur serait à évaluer en fonction du temps de calcul induit supplémentaire. On peut aussi envisager de proposer un estimateur différent qui ne repose pas directement sur la fonction Sim.

2.6.4 Lien avec une approche développementale

Lors de la conception d'AidMe, nous nous sommes attachés à respecter les contraintes qu'impose une approche développementale. En effet, l'objectif est que les connaissances d'AidMe émergent de son expérience et de ses interactions avec l'utilisateur. Bien qu'on ne le démontre pas explicitement, il est clair qu'AidMe aura une trajectoire d'apprentissage différente selon l'historique des interactions avec l'utilisateur. À chaque nouvelle interaction, le système tente de positionner le nouvel élément par rapport à ce qu'il connaît déjà.

L'utilisation d'un modèle de langage pré-entraîné peut être considéré comme une concession à une approche purement développementale. Elle suppose l'accès inné à une forme de sémantique générale du langage. Cependant, AidMe n'est pas dépendant de ces modèles pré-entraîné et peut fonctionner sans si l'on se passe des modèles neuronaux (Partie 2.3.2). Cette concession permet néanmoins de rendre AidMe plus rapidement opérationnel en améliorant les performances du modèle de similarité sémantique.

Dans le cadre d'une application industrielle et avec l'objectif de rendre AidMe plus rapidement utilisable, on peut relâcher un peu plus les contraintes développementales sans trop renier la flexibilité. En effet, si l'on connaît le domaine dans lequel le système sera déployé, il est possible de pré-encoder un ensemble de patterns et d'intentions spécifiques.

2.6.5 Entre online clustering et apprentissage faiblement supervisé

Indépendamment du traitement du langage naturel ou des assistants, on peut formuler le problème de la détection d'intention comme un problème d'*online clustering*. À chaque nouvelle requête de l'utilisateur, il faut affecter la requête à une classe d'intention existante (un cluster existant) ou créer une nouvelle classe d'intention (un nouveau cluster). On

construit ainsi un ensemble de clusters. Formulé de la sorte, on peut s'interroger sur la pertinence d'utiliser des algorithmes existants d'online clustering. On se heurte à deux difficultés :

— Ces algorithmes requièrent de définir une notion de distance dans l'espace des objets à *clusteriser*. Dans notre cas, il s'agit de la fonction de similarité sémantique qui est apprise dans le cadre des interactions. On a vu qu'employer une fonction pré-apprise conduit à des résultats décevants.

— Le problème de détection des arguments et du pattern sous-jacent à la requête ne se formule pas aussi simplement comme un problème d'online clustering.

Enfin, contrairement à de l'apprentissage non supervisé, notre algorithme repose sur une supervision en ligne de l'utilisateur, permettant l'apprentissage de la fonction de similarité sémantique et de valider les clusters d'intention formés par AIDME. Il pourrait être intéressant d'étudier le comportement de ce type d'algorithme dans le cadre de tâches faiblement supervisées et où la sensibilité à l'erreur est moins critique.

2.7 Conclusion

Dans ces travaux, nous avons proposé une solution pour doter les assistants apprenants d'un outil d'interprétation du langage naturel qui s'adapte à l'utilisateur tout en généralisant. Celle-ci s'appuie sur la comparaison sémantique d'une nouvelle requête avec les requêtes connues. Les résultats de simulation démontrent les bénéfices que permet AIDME, en lui permettant notamment d'interpréter des patterns inconnus sans l'aide de l'utilisateur. Ces résultats nécessitent encore d'être validés par une campagne de test utilisateurs.

Cependant, si la solution proposée permet de gérer une certaine forme de variabilité langagière de l'utilisateur, on ne peut pas en conclure pour autant que l'assistant dispose d'une compréhension plus fine du langage. L'approche souffre des mêmes limites que les modèles de langages évoquées en Partie 1.3 : elle ne s'appuie que sur des sources textuelles. L'absence d'*embodiment* d'AIDME est à notre sens l'une des principales faiblesses structurelles de ce système. Cela tient à la séparation stricte des fonctions d'interprétation du langage naturel (Φ) et de la politique d'action (Ψ) sans interaction apprenante entre elles. Cela repose sur l'hypothèse forte qu'il serait possible d'apprendre à interpréter les requêtes de l'utilisateur sans qu'il soit nécessaire de comprendre leurs effets. En pratique, il est relativement facile d'éprouver les limites de ces systèmes. Cela rend notamment le système sensible aux erreurs que pourraient introduire l'utilisateur par inadvertance. Le fait de n'interpréter le langage qu'à travers les requêtes de l'utilisateur empêche le

système de saisir leurs implications réelles et rend difficile la conception d'un protocole de détection/correction des erreurs.

Dans la suite de ces travaux, on s'attaque au défaut de généralisation procédurale des assistants apprenants. On s'intéresse notamment à des agents de Deep RL capables d'exécuter des tâches formulées en langage naturel.

Chapitre 3

Le langage, outil d'apprentissage pour un agent autonome

Colorless green ideas sleep furiously.

— Noam Chomsky
Syntactic Structures, 1957

Sommaire

3.1 Introduction

Dans le chapitre précédent, nous avons proposé un module de NLU qui répond à certaines limites des assistants apprenants relatives à leurs capacités de généralisation linguistique. On s'attaque maintenant à la seconde limite identifiée : le défaut de généralisation procédurale. En effet, les assistants apprenants ne sont pas en mesure de transférer des compétences apprises entre des procédures similaires. Par exemple, après avoir appris à utiliser l'interface d'une plateforme de messagerie, l'agent proposé par (Delgrange et al., 2020) sera en difficulté si l'interface est mise à jour. Il ne pourra pas non plus exploiter ses connaissances pour utiliser une autre plateforme de messagerie concurrente. On souhaite donc proposer un algorithme d'apprentissage de procédures qui disposent d'une forme de généralisation procédurale. On se tourne pour cela vers les algorithmes de Deep RL (*Deep Reinforcement Learning*), domaine qui développe des algorithmes d'apprentissage par renforcement en utilisant les qualités du Deep Learning.

3.1.1 Motivation

Les motivations pour ce domaine sont multiples, ce paradigme d'apprentissage est tout d'abord parfaitement adapté à un apprentissage en continu (Tessler et al., 2016) et dans un contexte où les données d'apprentissage sont acquises dans la durée. De plus, de récents travaux suggèrent que le langage peut être utilisé comme un outil d'abstraction favorisant la généralisation (Narasimhan et al., 2018 ; Y. Jiang et al., 2019 ; Cideron et al., 2021). On souhaite ainsi explorer le couplage de l'apprentissage langage - politique d'action et l'utilisation du langage comme outil de généralisation.

En Deep RL, la notion de procédure est traduite par celle de politique d'action. On peut envisager plusieurs formes de généralisation de la politique d'action : à de nouveaux contextes ou à de nouvelles tâches. Pour un assistant apprenant, cela se traduit par la capacité à exécuter une procédure dans des contextes différents de ceux vus pendant l'apprentissage ou par la capacité à inférer de nouvelles procédures sur la base de celles déjà connues. Ces deux formes sont pertinentes pour un assistant apprenant et sont au cœur des préoccupations actuelles en Deep RL (Cobbe et al., 2019 ; Ruis et al., 2020).

Enfin, l'utilisation de Deep RL permet d'envisager un nouveau cas d'usage pour l'assistant apprenant Jarvis (Delgrange, 2018). Cet assistant est en mesure d'apprendre soit par démonstration soit par composition de tâches déjà connues. On a évoqué que l'apprentissage par démonstration pouvait être rébarbatif pour l'utilisateur et que l'une des stratégies consistait à organiser l'apprentissage de l'agent selon un curriculum qui permette de minimiser le nombre de démonstrations effectuées et maximiser la réutilisation de tâches connues. On peut envisager qu'un agent de Deep RL apprenne un ensemble de

procédures élémentaires qui seraient mises à la disposition de l'utilisateur et pourraient être composées pour apprendre des procédures de plus haut niveau. Ainsi l'utilisateur serait dispensé de devoir effectuer des démonstrations. On reviendra sur ce cas d'usage en Partie 4.8.

3.1.2 Contexte

Cependant, avant de s'attaquer à un véritable environnement numérique, on se tourne d'abord vers des environnements plus traditionnels en Deep RL, comme ceux des jeux vidéos ou de robotique simulée. En effet les environnements numériques, dans lesquels évoluent les assistants, présentent certaines difficultés spécifiques qui seront évoquées au Chapitre 4 et qui nécessitent un traitement particulier. Une part importante de la recherche en Deep RL s'appuie sur des environnements robotiques pour développer de nouvelles architectures. C'est donc un domaine privilégié pour explorer les problématiques de généralisation procédurale, quitte à écarter, dans un premier temps, certaines questions spécifiques aux assistants apprenants. On y reviendra au Chapitre 4 pour proposer une architecture Deep RL qui leur soit adaptée et qui exploite une partie des travaux présentés dans ce chapitre.

Ces travaux furent l'occasion d'une collaboration étroite avec l'équipe INRIA Flowers, et notamment avec Cédric Colas et Tristan Karch. Tous deux travaillent dans l'équipe de Pierre-Yves Oudeyer sur des questions d'apprentissage par renforcement pour des systèmes multi-tâches et autonomes.

3.1.3 Organisation du chapitre

On débute ce chapitre par rappeler les principes de l'apprentissage par renforcement (Partie 3.2). On reformule ensuite le problème de la généralisation procédurale dans le contexte d'agents de Deep RL apprenant par interaction et les liens avec la notion de généralisation systématique (Partie 3.3). Notre contribution s'articule en deux temps. En Partie 3.5, on présente l'algorithme LE2 qui montre la possibilité d'utiliser le langage comme unique source de supervision d'un agent de Deep RL autonome. En Parties 3.6 et 3.7 on propose IMAGINE qui démontre qu'apprendre à représenter des tâches exprimées en langage naturel, favorise la généralisation et permet l'utilisation de mécanisme d'exploration créative de l'environnement.

3.2 Apprentissage par renforcement

Dans cette partie, on rappelle quelques éléments fondamentaux de l'apprentissage par renforcement. On décrit aussi le principe des algorithmes DQN (*Deep Q-Network*) et DDPG (*Deep Deterministic Policy Gradients*) qui seront utilisés par la suite, ainsi que celui du *hindsight experience replay*. Enfin, on dresse l'état de l'art des systèmes d'apprentissage par renforcement qui utilise le langage comme source d'apprentissage.

3.2.1 Principes généraux

Cette partie est librement inspirée de l'ouvrage de référence de Sutton et Barto (2018).

Modélisation d'un problème d'apprentissage par renforcement

L'apprentissage par renforcement (RL) est un sous-domaine de l'apprentissage automatique. Les problèmes s'y expriment généralement de la manière suivante : un *agent* exécute sur un *environnement* une séquence d'actions et reçoit des *récompenses*. Celles-ci lui permettent d'évaluer sa réussite, d'apprendre de ses erreurs et de progresser dans la réalisation de la tâche. On représente classiquement ces interactions par le diagramme de la Figure 3.1. Un algorithme d'apprentissage par renforcement consiste à agir dans un environnement pour collecter les données nécessaires et optimiser la manière dont il choisit ses actions (la *politique d'action*). À la différence de l'apprentissage supervisé qui consiste à apprendre une association entre une donnée et un label, l'apprentissage par renforcement consiste à faire l'association entre un état de l'environnement et l'action la plus appropriée. L'apprentissage supervisé permet de répondre à la question « quoi », l'apprentissage par renforcement permet de répondre à la question « comment ».

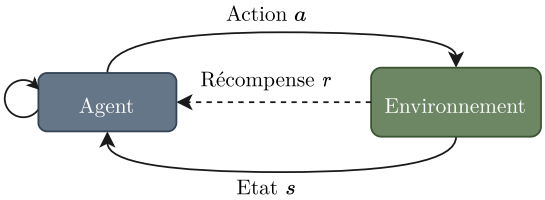


Figure 3.1 – Représentation schématique d'un problème RL

Mathématiquement, les problèmes d'apprentissage par renforcement sont modélisés par des processus décisionnels de Markov fini ou *finite* MDP (*Markov Decision Process*). À chaque instant t, l'agent prend connaissance de l'état $s_t \in \mathcal{S}$ de l'environnement et effectue une action $a_t \in \mathcal{A}$. À l'instant suivant $t+1$, il reçoit une récompense r_t et l'environnement se retrouve dans l'état s_{t+1}. La transition d'un état à un autre se fait selon la dynamique de l'environnement, représentée par une distribution $p(s_{t+1} \mid s_t, a_t)$. Les interactions avec l'environnement se font sur une base de temps discrète. Les performances

de l'agent sont mesurées par le gain à l'instant t : $G_t = \sum_{k>t} \gamma^(k-t) r_k$ où $\gamma \in [0,1]$ est un facteur d'actualisation (*discount factor*) qui permet de donner plus ou moins de poids aux récompenses futures en fonction du comportement souhaité. L'objectif de l'agent est d'apprendre la politique d'action π qui maximise l'espérance de son gain : $\max_\pi \mathbb{E}_\pi G_t$ où $\mathbb{E}_\pi$ désigne l'espérance calculée lorsque l'agent suit la politique π, c'est-à-dire $a \overset{\pi}{=} \pi(s)$ (a désigne ici l'action ou la distribution de probabilité sur l'espace d'action) et $s_{t+1} \sim p(\cdot \mid s_t, \pi(s_t))$.

Equations de Bellman et critère d'optimalité

À partir du gain, on peut définir la valeur d'un état V (*state-value function*) qui mesure les gains moyens que peut espérer un agent qui se trouve dans un état s et suit une politique d'action π. On peut aussi définir la *q-value* Q qui mesure les gains moyens que peut espérer un agent dans l'état s qui effectuerait l'action a dans le cadre de π. On peut écrire V, Q et la relation qui les unit :

$$V_\pi(s) = \mathbb{E}_\pi\left[G_t \mid s_t = s\right] \qquad Q_\pi(s,a) = \mathbb{E}_\pi\left[G_t \mid s_t = s, a_t = a\right]$$
$$V_\pi(s) = \mathbb{E}_\pi\left[Q_\pi(s, a_t)\right] = \sum_a \pi(a \mid s) Q_\pi(s,a)$$

La grande majorité des algorithmes s'appuient sur l'estimation de l'une de ces fonctions pour apprendre la politique d'action optimale. On peut écrire V ou Q sous une autre forme et faire apparaître l'*équation de Bellman* qui relie la valeur d'un état s à celle des états suivants :

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi\left[r_t + \gamma G_{t+1} \mid s_t = s\right] \\ V_\pi(s) &= \mathbb{E}_\pi\left[r_t + \gamma \mathbb{E}_\pi(G_{t+1} \mid s_{t+1}) \mid s_t = s\right] \\ V_\pi(s) &= \mathbb{E}_\pi\left[r_t + \gamma V_\pi(s_{t+1}) \mid s_t = s\right] \end{aligned} \tag{3.1}$$

On peut écrire la même équation pour Q et ainsi relier la q-value d'une paire état-action aux suivantes :

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[r_t + \gamma Q_\pi(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a)\right] \tag{3.2}$$

Pour un MDP fini, on peut définir les state-value et q-value optimales par $V^*(s) = \max_\pi V_\pi(s)$ et $Q^*(s,a) = \max_\pi Q_\pi(s,a)$. On peut montrer qu'il existe au moins une politique optimale π^* telle que $V^* \overset{\pi}{=} V_{\pi^*}$ et $Q^* = Q_{\pi^*}$. On peut remarquer que connaître V^* ou Q^* suffit à trouver une politique π^* et c'est pourquoi une grande partie des algorithmes cherchent à estimer ces quantités. *Le critère d'optimalité de Bellman* consiste à écrire les

équations (3.1) et (3.2) pour V^* et Q^* et à remarquer qu'elles ne dépendent plus d'une politique particulière :

$$V^*(s) = \mathbb{E}\left[r_t + \gamma V^*(s_{t+1}) \mid s_t = s)\right]$$
$$Q^*(s,a) = \mathbb{E}\left[r_t + \gamma \max_a Q^*(s_{t+1}, a) \mid s_t = s, a_t = a)\right]$$

Exploration vs Exploitation

L'un des premiers problèmes auxquels on se trouve confronté lors de la résolution d'un problème d'apprentissage par renforcement est le dilemme entre exploration et exploitation. En effet, pour récolter le plus de récompenses, on est tenté de choisir des actions qui ont déjà fait leurs preuves (*exploitation*), mais si l'on se cantonne à ces actions, on risque aussi de passer à côté d'actions encore plus fructueuses. Pour découvrir de telles actions, un agent doit s'autoriser à explorer son environnement en ne suivant pas toujours ce que la politique d'action apprise lui dicterait (*exploration*). La plus simple des stratégies d'exploration (ϵ-*greedy*) consiste à effectuer des actions aléatoires, avec probabilité ϵ, dans l'environnement. Par exemple, étant donné la politique d'action π apprise Q-learning $\mathcal{A}$ sur un espace d'action fini, la politique effectivement suivie est :

$$\mathbb{P}(a \mid s) = \frac{\epsilon}{|\mathcal{A}|} + (1-\epsilon)\pi(a \mid s)$$

3.2.2 Deep Q-learning

Q-learning

En RL, l'algorithme de Q-Learning (Watkins, 1989) est l'un des plus célèbres. Il consiste à estimer Q^* et à définir la politique d'action par $\pi(s) = \max_a Q(s,a)$, c'est-à-dire choisir l'action qui semble rapporter le plus de gains à long terme. Q^* est estimée par itération et mise à jour selon la règle suivante :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta_t$$

où $\delta_t = r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$ est appelée l'erreur de différence temporelle (*TD-erreur*) et $\alpha \in]0, 1]$ un hyperparamètre qui contrôle l'inertie de la mise à jour (0 pas de mise à jour, 1 pas de mémoire de l'ancienne valeur). On interprète δ_t comme l'erreur entre la valeur estimée de l'action a_t dans l'état s_t : $y_t = r_t + \gamma \max_a Q(s_{t+1}, a)$ après l'avoir effectuée, et sa valeur estimée avant sa réalisation : $Q(s_t, a_t)$. La règle de mise à jour consiste ainsi à faire évoluer $Q(s,a)$ pour qu'il se rapproche de la valeur cible y_t et ainsi minimiser δ_t.

Lorsque les espaces d'états $\mathcal{S}$ et d'actions $\mathcal{A}$ sont finis, on peut montrer que Q converge vers Q^* et que l'algorithme converge vers une politique optimale. L'algorithme de Q-Learning est dit *off-policy* car la mise à jour ne dépend pas de la politique elle-même mais uniquement de la valeur actuelle de Q. La dépendance à l'égard de la politique est indirecte et concerne uniquement les données collectées. Il est aussi *model-free*, car il ne nécessite pas de connaître la dynamique de l'environnement.

Deep Q-Network

En pratique, il est fréquent que l'espace d'états soit bien trop grand pour que l'on puisse espérer une convergence en un temps raisonnable. C'est là que l'utilisation d'estimateurs statistiques, et notamment de réseaux de neurones, pour approcher Q^* se révèle intéressante et permette même de s'affranchir du caractère fini de $\mathcal{S}$. Q^* est alors approximé par une fonction neuronale $\hat{Q} : \mathcal{S} \to [0,1]^{|\mathcal{A}|}$ qui estime la distribution de probabilité sur l'espace des actions étant donné un état s.

Cependant, l'utilisation de réseaux de neurones demande quelques adaptations. En effet, les données collectées par un système d'apprentissage par renforcement le sont *séquentiellement*. Elles sont donc très corrélées entre elles, et un réseau de neurones appris par les mises à jour successives d'un Q-learning est instable. Pour résoudre cette difficulté, Mnih et al. (2015) introduisent l'algorithme DQN (*Deep Q-Network*) et avec lui une série d'adaptations qui permettent d'améliorer la stabilité. L'une des innovations est l'introduction de l'*experience replay* : les interactions avec l'environnement sont stockées dans une mémoire sous la forme de *transitions* (s_t, a_t, s_{t+1}, r_t), qui décrivent l'action de l'agent et ses effets sur l'environnement. Le modèle $\hat{Q}$ n'est plus mise à jour après chaque interaction mais sur un batch de transitions sélectionnées dans la mémoire. On dit qu'elles sont *rejouées*. Ces dernières proviennent de trajectoires différentes et sont donc beaucoup moins corrélées, améliorant la stabilité de l'apprentissage. De plus, les transitions collectées sont potentiellement utilisées plusieurs fois pour mettre à jour le réseau de neurones, augmentant son efficacité en termes d'usage de données (*sample efficiency*).

Pour mettre à jour $\hat{Q}$, on minimise l'erreur moyenne $\frac{1}{N}\sum_i \delta_i^2$ où $\delta_i = (y_i - Q(s_i, a_i))$ est un équivalent de la TD-erreur. La valeur cible y_i est calculée, non pas à partir de Q mais, à partir d'un *target network* $\widetilde{Q}$: $y_i = r_i + \gamma \max_a \widetilde{Q}(s'_i, a)$. Le target network est une copie $\hat{Q}$ actualisée périodiquement mais significativement moins souvent que Q. Ainsi la valeur cible y_i ne dépend plus de $\hat{Q}$ et cela améliore la stabilité de l'apprentissage. Il s'agit de la seconde adaptation majeure de Mnih et al. On peut en donner l'interprétation intuitive suivante : une règle de mise à jour permet de rapprocher petit à petit un estimateur d'une valeur cible. Pour éviter que l'estimateur ne poursuive son ombre, la valeur cible ne doit pas (trop) dépendre de l'estimateur lui-même.

DQN est ainsi capable d'apprendre à jouer à une série de jeux Atari au même niveau que les meilleurs joueurs humains. Que le même algorithme soit capable d'apprendre une diversité de tâches sans adaptation spécifique constituait alors une prouesse. DQN a ouvert la voie au domaine du Deep RL et au développement d'algorithmes de Deep Learning pour l'apprentissage par renforcement.

3.2.3 Deep Deterministic Policy Gradients

L'un d'eux est l'algorithme DDPG (Deep Deterministic Policy Gradients, Lillicrap et al., 2016). Tout comme DQN, DDPG est off-policy et model-free, il est présenté par ses auteurs comme l'adaptation de DQN au cas où l'espace d'action $\mathcal{A}$ est continu. Il est notamment utile en robotique pour contrôler les mouvements d'un robot. DDPG est un algorithme dit *acteur-critique*. Il est composé de deux fonctions : un acteur (la politique d'action : $\pi : \mathcal{S} \to \mathcal{A}$) et un critique (qui mesure $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$). L'acteur sert à choisir l'action à effectuer sur l'environnement et le critique sert à évaluer ces actions et mettre à jour les deux fonctions. On peut noter que comme l'espace d'actions est continu, l'acteur fournit directement une action et non plus une distribution de probabilité sur l'espace d'action comme pour DQN.

La mise à jour de π et Q se fait de manière très similaire à DQN avec l'emploi de target networks $\widetilde{\pi}$ et $\widetilde{Q}$. La valeur cible est calculée à partir des target networks : $y_i = r_i + \gamma \widetilde{Q}(s'_i, \widetilde{\pi}(s'_i))$ et la quantité optimisée est $\frac{1}{N} \sum_i (y_i - Q(s_i, \pi(s_i))^2$.

Une autre différence avec DQN concerne la stratégie d'exploration. En effet, plutôt que d'adopter une stratégie ϵ-greedy, l'exploration se fait en ajoutant un bruit sur l'action choisie par l'acteur, typiquement un bruit gaussien.

3.2.4 Hindsight Experience Replay

Jusqu'à présent, on a évoqué des problèmes d'apprentissage par renforcement qui consistent à résoudre une tâche unique. Cependant, dans de nombreux cas, et notamment ceux que nous traiterons par la suite, l'agent doit apprendre à maîtriser un ensemble de tâches $\mathcal{G}$ réalisables dans l'environnement. Pour prendre en compte l'objectif $g \in \mathcal{G}$ poursuivi par l'agent, on écrit la politique sous la forme $\pi(s, g)$ (Schaul et al., 2015a). D'un point de vue mathématique, cela ne change rien et g peut être vu comme une partie de l'état de l'environnement qui reste fixe tout au long d'un épisode. Cela nous permet d'adapter facilement les algorithmes DQN et DDPG (ou tout autre algorithme off-policy). Cependant, cela ouvre une perspective intéressante sur la manière dont on rejoue les expériences. Pour un problème multi-objectifs, les transitions sont stockées sous la forme $(s_t, g, a_t, r(g_t), s_{t+1})$. Les récompenses obtenues dépendent de l'objectif g_t poursuivi

au moment de l'épisode. L'agent peut cependant réévaluer les transitions collectées en prétendant qu'il poursuivait un autre but pour lequel l'épisode aurait conduit à obtenir une récompense positive. Si l'agent s'aperçoit que pour g', $r(g') = 1$, l'agent peut imaginer une nouvelle transition $(s_t, g', a_t, r(g') = 1, s_{t+1})$. Ainsi, des transitions collectées dans un contexte particulier peuvent être rétrospectivement exploitées pour progresser dans la réalisation d'un autre objectif que celui initialement visé. Dans un environnement où les récompenses sont rares et où les différents objectifs ne sont pas de difficulté équivalente, cela permet à l'agent de collecter plus de récompenses positives et donc de progresser plus rapidement. Cette idée a été introduite par Andrychowicz et al. (2017) sous le nom de *Hindsight Experience Replay* (HER) et son intérêt notamment démontré dans le cadre de tâches robotiques qui ne sont apprises que par les agents qui emploient cette technique.

3.2.5 Le langage naturel pour exprimer des tâches de Deep RL

L'intérêt pour le rôle du langage dans les problèmes de Deep RL est de plus en plus important au sein de la communauté (Luketina et al., 2019) : pour formuler les tâches que doivent apprendre des agents (Hermann et al., 2017 ; Bahdanau et al., 2019), pour naviguer dans des environnements comme les jeux textuels (Narasimhan et al., 2018 ; Côté et al., 2019), pour exploiter une connaissance abstraite (Shridhar et al., 2021), etc. Ces approches cherchent à ancrer la compréhension du langage directement sur les perceptions et la dynamique de l'environnement et à l'utiliser comme un outil d'abstraction qui favorise la généralisation (Hermann et al., 2017). Luketina et al. (2019) a proposé une revue des différentes approches à ce sujet et les distinguent en deux catégories : les problèmes *conditionnés par le langage*, où le langage est nécessaire pour résoudre le problème et une seconde approche, dite *assistée par le langage*, où le langage est une aide supplémentaire. Dans ces travaux, on s'intéresse surtout à la première catégorie et à la possibilité d'apprendre une politique d'action qui exécute des tâches formulées en langage naturel (*instruction-following agent*). Notre motivation est double : contrôler un agent par le langage et explorer comment les représentations langagières peuvent faciliter la généralisation d'un agent.

Apprendre à exécuter des instructions

La majorité des travaux de la catégorie conditionnée par le langage propose des agents qui apprennent à suivre une instruction. Celle-ci est fournie en langage naturel à l'agent au début de chaque épisode et l'agent reçoit une récompense à la fin de l'épisode en fonction de sa réussite. Les instructions ne donnent pas explicitement les étapes à accomplir pour la réaliser et l'agent doit donc apprendre le « comment faire » à partir du « quoi faire ».

L'idée de pouvoir interpréter une commande en langage naturel est relativement ancienne (Winograd, 1972) mais les approches nécessitaient alors de connaître une partie de la dynamique de l'environnement. R. K. Branavan et al. (2010) est l'un des premiers à proposer un agent apprenant un *mapping* entre des instructions de haut niveau et des commandes d'exécution sans exploiter une pré-connaissance de l'environnement. D. L. Chen et Mooney (2011) proposent un agent qui apprend à naviguer dans un environnement en suivant des instructions. L'utilisation de Deep Learning a permis de généraliser ces approches en offrant la possibilité de combiner les représentations langagières avec les éléments de l'environnement (états, objectif) dans un même modèle (Bahdanau et al., 2019 ; Chan et al., 2019 ; Co-Reyes et al., 2019 ; Goyal et al., 2019 ; Y. Jiang et al., 2019 ; Cideron et al., 2021). Pour R. Kaplan et al. (2017), l'utilisation d'instructions est un moyen de guider l'agent et lui permettre de jouer à Montezuma réputé très difficile, en lui fournissant des récompenses supplémentaires lorsqu'il accomplit une instruction.

Langage et fonction de récompense

Pour apprendre de ses erreurs, un agent doit avoir accès à une fonction de récompense qui l'informe de ses succès et lui indique la voie à suivre. Pour plusieurs raisons, cela peut représenter une véritable difficulté : (1) techniquement, il est parfois difficile d'écrire une fonction qui vérifie les succès de l'agent et cela biaise l'apprentissage en direction de la fonction de récompense et (2) conceptuellement, écrire une telle fonction requiert une compréhension fine du problème à résoudre. Cela représente de plus une charge à effectuer pour chaque nouvel environnement. Cela motive l'idée d'apprendre dynamiquement la fonction de récompense. Le langage est alors un outil privilégié, car il permet d'exprimer de manière compacte un ensemble de situations qui réalisent une même instruction. Les approches sont diverses. Fu et al. (2019) emploie une méthode d'apprentissage par renforcement inverse qui exploite un modèle de l'environnement. (Bahdanau et al., 2019), inspiré par des techniques d'*adversarial learning*, apprend à différencier les états finaux associés aux objectifs des états non finaux à partir d'un jeu de données préalablement collecté. Goyal et al. (2019) apprend une fonction de récompense qui mesure la pertinence d'une action dans le contexte d'une instruction et emploie cette fonction pour fournir des récompenses intermédiaires à l'agent. (Sumers et al., 2020) explore la possibilité d'apprendre une fonction de récompense à partir d'un retour linguistique plus réaliste que celui généralement employé dans les approches précédentes.

Langage et Hindsight replay

Le principe du Hindsight Experience Replay peut être adaptée à l'utilisation du langage pour exprimer les tâches réalisables. L'un des reproches fait à HER est qu'il n'est pas

toujours évident de trouver a posteriori une tâche qui corresponde à la trajectoire réalisée. Pour des tâches langagières, il faut re-labelliser les trajectoires à partir des instructions en langage naturel. Pour cela, Chan et al. (2019) et Y. Jiang et al. (2019) utilisent ainsi une fonction hard-codée tandis que Cideron et al. (2021) apprend un modèle génératif d'instructions. Avec IMAGINE, on explore une troisième voie en réévaluant les objectifs à partir de la fonction de récompense apprise.

3.3 Définition du cadre d'étude

3.3.1 Généralisation procédurale et généralisation systématique

Dans le cadre des assistants apprenants, on est intéressé par la notion de généralisation procédurale, c'est-à-dire la capacité à transférer la connaissance d'une procédure impliquant certains objets à des procédures similaires incluant des objets comparables. Cette notion est traduite par celle de généralisation systématique dans la communauté en Intelligence Artificielle et notamment en Deep RL (Bahdanau et al., 2018). La notion de *systématicité* a été introduite par Fodor et Pylyshyn (1988) pour décrire une propriété des représentations mentales de la pensée humaine :

> Le fait que les capacités cognitives exhibent toujours certaines symétries, de sorte que la capacité d'entretenir certaines pensées implique la capacité d'entretenir d'autres pensées apparentées par leur contenu sémantique.

Bruner (1983, Chapitre 2) considère que la systématisation est l'un des *dons cognitifs initiaux* qui permet à l'enfant l'apprentissage du langage. Il observait en effet que l'enfant exécutait méthodiquement des tâches très simples de toutes les manières qu'il pouvait.

Pour les systèmes artificiels, il s'agit d'un problème difficile, notamment pour les architectures de réseaux de neurones qui ne parviennent pas facilement à généraliser à de nouvelles compositions (Lake et Baroni, 2017). Bahdanau et al. (2018) a montré que la généralisation systématique d'un système de VQA est favorisée par une plus grande modularité. Hill et al. (2019) a montré que les agents de Deep RL guidés par des instructions sont capables de généraliser le sens de prédicats, couleurs et formes d'objets. Tout récemment, Ruis et al. (2020) a proposé un nouvel environnement, gSCAN, pour évaluer la généralisation systématique. Kuo et al. (2020) a montré qu'une architecture neuronale qui compose plusieurs réseaux de neurones de manière à refléter la compositionnalité du problème facilite la généralisation systématique.

On cherche ainsi à développer des agents capables de généralisation systématique afin de les adapter au contexte des assistants numériques apprenants et disposer ainsi d'une capacité de généralisation procédurale.

3.3.2 Contributions

Comme on l'a évoqué en Introduction, ces travaux ont été réalisés lors d'une collaboration avec l'équipe Flowers d'INRIA, et notamment Cédric Colas et Tristan Karch. Intéressés par le développement des agents autonomes, ils souhaitaient exploiter le langage pour dépasser certaines limites rencontrées quant à la représentation des buts par un agent de Deep RL. En effet, les agents autonomes développés jusqu'alors nécessitaient d'implémenter manuellement l'espace des tâches réalisables et la fonction de récompense associée. Au-delà de son caractère rébarbatif pour le développeur, cela conduit les agents autonomes à démarrer leur apprentissage avec une connaissance innée de l'espace des tâches et de la fonction de récompense. De notre côté, nous souhaitions développer des agents capables par des interactions en langage naturel d'apprendre à effectuer des tâches dans un environnement, sans avoir besoin de définir l'espace des tâches ni la fonction de récompense, afin de pouvoir adapter un tel agent au contexte des assistants apprenants. L'enjeu était donc que l'agent apprenne simultanément le *sens* des tâches qui lui sont confiées et la manière de les accomplir dans l'environnement. Inspirés notamment par l'apprentissage des enfants qui doivent à la fois apprendre quelles sont les tâches réalisables, comment les représenter, les évaluer et les réaliser, nous partagions le même constat qu'une approche développementale conduirait à développer des agents plus autonomes.

Le projet s'est déroulé en deux temps. Nous avons débuté en développant LE2 (Language Enhanced Explorer) (Partie 3.5). LE2 est un agent de Deep RL dont la supervision est effectuée non pas via une fonction de récompense classique mais par un partenaire social qui fournit à l'agent des descriptions en langage naturel des transformations dans l'environnement. L'agent utilise ces descriptions en langage naturel pour apprendre de manière supervisée sa propre fonction de récompense à partir de laquelle il peut entraîner sa politique d'action (Bahdanau et al., 2019 ; Fu et al., 2019). Ces travaux ont donné lieu à un article présenté au workshop ViGIL à NeurIPS 2019 (Lair, Colas et al., 2019).

Dans un second temps (Parties 3.6 et 3.7), nous avons proposé IMAGINE, un nouvel agent inspiré par LE2. Alors que LE2 bénéficiait d'un modèle de langage pré-appris à partir duquel il obtenait la représentation des tâches de l'environnement, IMAGINE apprend son propre modèle de langage et ainsi une représentation des tâches qui lui est propre. On propose alors un mécanisme d'exploration créative de l'environnement qui s'appuie sur les propriétés de ces représentations apprises. IMAGINE peut en effet imaginer de nouvelles tâches à réaliser dans l'environnement en recomposant les descriptions langagières connues. Cette idée, directement inspirée des théories de Vygotsky, fait du langage un outil d'exploration au service de l'agent. En effet, sous condition que la fonction de récompense apprise généralise suffisamment, il est possible à l'agent de s'auto-superviser sur des tâches ou des situations pour lesquelles, il n'a jamais eu de description du partenaire social . On

montre ainsi que l'agent est capable d'apprendre de nouvelles tâches et de généraliser des comportements appris à de nouveaux objets, faisant ainsi preuve d'une forme de généralisation systématique. Ces travaux ont été publiés et présentés lors de NeurIPS 2020 (Colas, Karch, Lair et al., 2020).

3.4 Apprentissage en interaction dans un monde ouvert

LE2 et IMAGINE fonctionnent selon un même paradigme. Tous deux évoluent dans des environnements ouverts où des tâches de natures et complexités différentes sont réalisables. Ils ne les connaissent pas à l'avance et les découvrent en explorant leur environnement. Ils n'ont pas non plus accès à une fonction de récompense externe comme dans le cadre classique mais apprennent leur propre fonction de récompense interne grâce à des interactions avec un partenaire social. On décrit maintenant la nature de ce partenaire social et le mode d'interaction qui sera commun à la suite de ces travaux.

3.4.1 Interaction avec un partenaire social

A la fin de chaque épisode, l'agent interagit avec un oracle, que l'on nomme partenaire social SP. Ce dernier, idéalement un véritable humain ou groupe d'humains mais en pratique plutôt un agent simulé, observe le début et la fin de chaque épisode et décrit par des phrases en langage naturel les changements observés. Par exemple, au cours d'un épisode où l'agent aurait attrapé un objet de l'environnement, le partenaire social pourra prononcer une phrase comme « *You grasped the red chair* ». Ces *descriptions* indiquent à l'agent que les transformations observées dans l'environnement font sens pour un observateur extérieur. Elles représentent des tâches pertinentes à réaliser et l'agent doit apprendre à les reproduire. L'ensemble des descriptions collectées représentent l'ensemble des tâches intéressantes du point de vue du partenaire social. Lorsque l'agent démarre dans l'environnement, il n'a connaissance d'aucune des tâches faisables et il les découvre au fur et à mesure de ses interactions. Ces descriptions jouent un rôle similaire aux *instructions* qu'apprennent à suivre certains systèmes déjà cités (Luketina et al., 2019) : elles représentent les tâches réalisables par l'agent. On insiste cependant sur la distinction entre instruction et description : les instructions sont données à priori à l'agent, il les connaît toutes et au début de chaque épisode, une instruction à réaliser lui est imposée. Dans notre cas, les descriptions sont inconnues de l'agent et c'est lui qui décide au début de chaque épisode de la tâche qu'il se fixe. Ainsi lorsque le partenaire social décrit les transformations de l'environnement, il ne connaît pas l'objectif initial de l'agent.

On s'inscrit ici dans la continuité de l'approche développementale employée jusqu'alors. En effet, on s'inspire de l'autonomie avec laquelle les enfants explorent spontanément dans leur environnement (Chu et Schulz, 2020). Cette tendance naturelle observée chez eux est due à des processus cognitifs de motivation intrinsèque (Gopnik et al., 1999 ; F. Kaplan et Oudeyer, 2007) qui les amène à une forme de curiosité (Kidd et Hayden, 2015). On laisse donc à l'agent le rôle de choisir au début de chaque épisode l'objectif qu'il va poursuivre. Cette forme d'interaction est aussi plus naturelle, un enfant reçoit le retour de ses proches a posteriori, sans que ces derniers ne sachent toujours ce que l'enfant souhaitait faire, et n'est pas constamment dirigé par un adulte (Bornstein et al., 1992 ; Tomasello, 1999).

Cette approche est directement inspirée du cadre IMGEP (Intrisically Motivated Goal Exploration Processes) dans lequel des agents sont capables de fixer et poursuivre leurs propres buts internes sans avoir pour autant besoin d'une fonction de récompense externe (Baranes et Oudeyer, 2013 ; Forestier et Oudeyer, 2016 ; Forestier et al., 2017 ; Colas et al., 2019a). Ces agents sont capables d'explorer efficacement et de manière autonome leur environnement.

3.4.2 Propriétés du partenaire social

Les descriptions permettent à l'agent de découvrir quelles sont les tâches intéressantes et réalisables dans l'environnement. À l'instar de Bahdanau et al. (2019), l'agent apprend une fonction de récompense à partir de laquelle est entraînée la politique d'action. Cette fonction estime si, étant donné une description et une transformation de l'état de l'environnement, la description est appropriée pour décrire cette transformation. À la différence de Bahdanau et al., l'agent n'a pas besoin de recourir à un jeu de données externes et ne comptent que sur les données collectées dans l'environnement. L'agent n'a pas non plus besoin d'avoir accès à un modèle de l'environnement comme Fu et al. (2019). Il suppose néanmoins certaines propriétés du partenaire social :

— *Précision* : les descriptions fournies par l'agent sont toujours supposées juste, le partenaire social ne fait pas d'erreur ;

— *Exhaustivité* : le partenaire social fournit toutes les descriptions adaptées à la transformation de environnement observée ;

— *Présence* : le partenaire social est toujours présent et fournit donc des descriptions après chacun des épisodes.

La première propriété apparaît raisonnable pour un système expérimental dans lequel on souhaite limiter les sources d'erreurs. Bahdanau et al. (2019) a été confronté à la présence de faux négatifs (c'est-à-dire d'exemple de paires (description, état) dans les données d'entraînement de la fonction de récompense) et propose une heuristique simple

pour contourner cette difficulté. Les deux propriétés suivantes sont cependant plus contraignantes, notamment dans le cas où l'on souhaiterait remplacer le partenaire social par un véritable humain. On étudiera par la suite la robustesse de l'agent lorsque l'on relaxe partiellement ces propriétés.

3.5 LE2 : Apprentissage d'une fonction de récompense

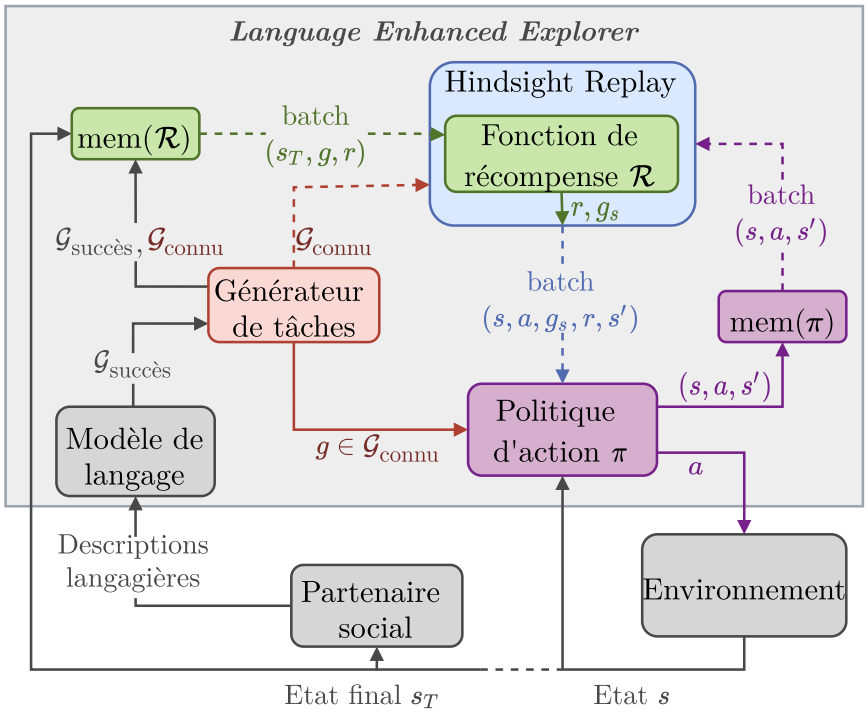


FIGURE 3.2 – Architecture du *Language Enhanced Explorer*

Dans cette partie, on décrit l'agent LE2, un agent Deep RL qui apprend sa propre fonction de récompense interne à partir des descriptions fournies par le partenaire social. La fonction de récompense apprend à déterminer si une phrase en langage naturel décrit correctement une observation de l'environnement. La politique d'action exploite ensuite ce signal pour s'entraîner.

3.5.1 Description de l'architecture

On représente l'architecture modulaire de LE2 en Figure 3.2. Les modules appris ou mis à jour par l'agent sont colorés, les autres sont gris et comprennent notamment les modules externes à l'agent (environnement, partenaire social). On commence par décrire sommairement le fonctionnement de l'agent avant de détailler certains aspects de l'architecture.

1. **Collecte de données de l'environnement**
   - *Durant un épisode :* Au début de chaque épisode, le générateur de tâches sélectionne une tâche objectif g à accomplir pendant l'épisode. L'agent récupère l'état s et exécute l'action a choisie par la politique π. Les transitions (s, a, s') sont enregistrées dans une mémoire dédiée à la politique $mem(\pi)$. Un épisode constitue une séquence de 50 actions sur l'environnement.
   - *Après un épisode :* Le partenaire social observe l'état final s_T de l'environnement et fournit l'ensemble des descriptions correspondantes en langage naturel. Chaque description correspond à une tâche réalisable dans l'environnement. Le modèle de langage transforme chaque description langagière en un vecteur numérique g représentant la tâche associée. L'ensemble de ces vecteurs $\mathcal{G}_{\text{succès}}$ alimente le générateur de tâches et permet de découvrir de nouvelles tâches potentielles. La mémoire dédiée à la fonction de récompense $mem(\mathcal{R})$ stocke s_T et l'ensemble des tâches réalisées ($\mathcal{G}_{\text{succès}}$) et par déduction l'ensemble des tâches non réalisées ($\mathcal{G}_{\text{connu}} \setminus \mathcal{G}_{\text{succès}}$) durant l'épisode.
2. **Mise à jour de la fonction de récompense $\mathcal{R}$, de la politique π du générateur de tâches et du mécanisme de replay :**
   - *Fonction de récompense* : la fonction de récompense est apprise de manière supervisée à partir de batchs (s, g, r) provenant de $mem(\mathcal{R})$.
   - *Politique d'action* : la politique d'action est apprise à partir des batchs (s, g_s, a, r, s') fournis d'une part par $mem(\pi)$ et d'autre part par le module de Hindsight Replay et la fonction de récompense.
   - *Générateur de tâches* : Au début de chaque épisode, une tâche est choisie de manière à optimiser la collecte de données et le retour du partenaire social. L'agent maintient donc un modèle interne des retours qu'il obtient après avoir choisi chacune des tâches.
   - *Mécanisme de replay* : La politique d'action est apprise par une variante du Hindsight Replay (Partie 3.2.4). Les transitions sont rejouées de manière à se concentrer sur les tâches pour lesquelles l'agent progresse le plus. Il s'auto-évalue donc périodiquement pour maintenir un modèle interne de son progrès.

Représentation de l'état de l'environnement s – L'état de l'environnement n'est pas un état au sens classique où il représenterait l'état courant observable de l'environnement. Il s'agit plutôt d'un *état augmenté* de la transformation de l'environnement depuis le début de l'épisode. Si on dénote par o_t l'observation courante de l'environnement et o_0, l'observation de l'environnement au début de l'épisode ($t = 0$), on représente l'état

courant par : $s_t = (o_t, \Delta o_t = o_t - o_0)$. L'état courant permet donc à l'agent de connaître les changements qui se sont produits depuis le début de l'épisode.

En effet, pour certaines tâches, dites *relatives*, la condition de réalisation nécessite de connaître l'état de l'environnement au début de l'épisode. Par exemple, pour savoir si l'agent a décalé un objet, il faut connaître les positions initiales et finales de l'objet. Cette représentation $(o_t, \Delta o_t)$ est donc nécessaire à la fois pour que le partenaire social puisse déterminer si une telle transformation a eu lieu, et pour que la fonction de récompense soit en mesure d'apprendre à discriminer ce genre de tâches.

D'autre part, bien que ne soit pas une motivation première, cela rejoint l'idée défendue par Nelson (Nelson, 1977 ; Nelson, 1996) et expliquée en Partie 1.5.3 que l'émergence des concepts se fait sur des bases fonctionnelles et non perceptuelles. La manière d'encoder l'état contient en effet une transformation fonctionnelle de l'espace. Cette idée n'a cependant pas été testée et mériterait une étude plus approfondie, par exemple en comparant l'acquisition des tâches absolues et relatives.

L'accès à l'état final suffit donc au partenaire social pour déterminer les transformations qui décrivent la différence entre les observations finale et initiale.

Modèle de langage

L'agent n'a pas accès à un espace des tâches et doit construire le sien. Le modèle de langage permet à l'agent de convertir les descriptions langagières découvertes au fur et à mesure en vecteurs numériques de taille fixe. Dans LE2, le modèle de langage est relativement simple et consiste à moyenner l'*embedding* des mots de la description, de manière assez similaire à ce qui était fait pour AIDME. On emploie un *embedding* de *glove* pré-entraîné (Pennington et al., 2014). Pour LE2, l'espace des tâches n'est donc pas appris, il est simplement découvert. Plus généralement, le modèle de langage est une fonction $\mathcal{L} : \mathcal{D} \to \mathcal{G}$ où $\mathcal{D}$ est l'espace des descriptions que peut prononcer le partenaire social, et $\mathcal{G}$ l'espace des vecteurs de tâches.

Fonction de récompense conditionnée par une tâche

La fonction de récompense $\mathcal{R}$ est apprise en parallèle de la politique d'action à partir des données collectées par cette dernière. Elle détermine si, étant donné un état de l'environnement s et une tâche g (représentant une description), g est bien réalisé dans s. On dit ainsi que $\mathcal{R}$ est conditionnée par une tâche (*goal-conditioned*) :

$$\mathcal{R} : \mathcal{S} \times \mathcal{G} \to \{0, 1\}$$

Données d'apprentissage – À la fin de chaque épisode, le partenaire social fournit à l'agent l'ensemble des descriptions qui sont réalisées. Ces descriptions, transcrites en vecteur de tâches par le modèle de langage $\mathcal{L}$, sont associées avec l'état final et constituent un ensemble d'exemples *positifs* : $\{(s_T, g, r = 1) \mid g \in \mathcal{G}_{\text{succès}}\}$. À partir de l'ensemble des tâches découvertes jusqu'à présent, l'agent peut aussi en déduire l'ensemble des tâches qui n'ont *pas* été réalisées durant l'épisode : $\mathcal{G}_{\text{échec}} = \mathcal{G}_{\text{connu}} \setminus \mathcal{G}_{\text{succès}}$. Il en construit un ensemble d'exemples *négatifs* : $\{(s_T, g, r = 0) \mid g \in \mathcal{G}_{\text{échec}}\}$. Tous ces exemples (s_T, g, r) collectées sont stockés dans $mem(\pi)$ à partir desquels $\mathcal{R}$ est apprise de manière supervisée.

Apprentissage – L'apprentissage de $\mathcal{R}$ est un problème de classification binaire. N'importe quel classifieur binaire peut théoriquement être utilisé. Nous en avons testé plusieurs et choisi un modèle de Random Forest. Il présente l'avantage d'être très rapide à entraîner et prédire et lors de nos études préliminaires, il possédait la courbe d'apprentissage la plus rapide. Ce dernier point est particulièrement important, la politique d'action étant apprise en parallèle, il est essentiel que la fonction de récompense atteigne rapidement un niveau de performance suffisante pour permettre à la politique de progresser. Le modèle est mis à jour périodiquement.

Les données collectées dans $mem(\pi)$ présentent deux inconvénients. Tout d'abord, certaines tâches sont sur-représentées par rapport à d'autres. En effet, l'agent découvre naturellement les tâches les plus faciles avant les plus difficiles et dispose donc de beaucoup plus d'exemples pour les premières. Le risque en utilisant un batch de données sélectionnées de manière aléatoire dans $mem(\pi)$ est que la fonction de récompense ignore les tâches sous-représentées. Pour forcer la fonction de récompense à apprendre à discriminer toutes les tâches, la représentation des tâches découvertes dans un batch de données d'entraînement est équilibrée.

Le second inconvénient est la distribution d'exemples positifs et négatifs. Il est en effet bien plus probable que l'agent ne réussisse pas une tâche donnée et la proportion d'exemples négatifs peut atteindre 95% pour certaines tâches, notamment les plus difficiles. Pour éviter que le modèle ne soit trop pessimiste (trop de faux négatifs), les batchs de données d'entraînement sont construits pour posséder au moins 20% d'exemples positifs. Cette proportion a été déterminée comme celle qui optimise le F_1 score (voir Figure 2.4).

Curriculum et mécanismes de motivation intrinsèque

LE2 appartient à la famille des IMGEP (Forestier et al., 2017) et est inspiré de CURIOUS (Colas et al., 2019a). LE2 s'appuie en effet sur deux mécanismes de motivation intrinsèque pour choisir la tâche qu'il se fixe au début d'un épisode et les transitions qui seront rejouées pour optimiser la politique. Ces mécanismes induisent un apprentissage par

curriculum, c'est-à-dire une trajectoire d'apprentissage. Ils permettent à l'agent d'explorer son environnement plus efficacement et d'ordonner la manière dont il apprend les différentes tâches en se focalisant sur celles où ils progressent.

Générateur de tâches – LE2 débute sans connaître les tâches réalisables dans l'environnement. Il les découvre au fur et à mesure lorsqu'il reçoit une nouvelle description du partenaire social. Pour réussir certaines d'entre elles, il en doit réussir d'autres plus simples. LE2 découvre d'abord les tâches les plus faciles et cela le conduit naturellement à découvrir les plus complexes. La manière dont l'agent explore son environnement joue ainsi un rôle essentiel dans sa trajectoire d'apprentissage. Pour améliorer l'efficacité de son exploration, l'agent choisit des tâches avec l'espoir qu'elles le conduiront à obtenir des descriptions du partenaire social qu'il n'obtient que rarement. L'agent tente ainsi d'optimiser l'utilité des données collectées et ses interactions avec le partenaire social .

Plus concrètement, on définit la *rareté* d'une tâche $\eta(g)$ par l'inverse du nombre de descriptions correspondantes obtenu. Plus une tâche est rare, moins elle a été réussie, moins elle est connue et plus il est difficile de l'apprendre. La qualité d'une trajectoire est la somme des raretés des tâches réussies pendant un épisode $q = \sum_{g \in \mathcal{G}_{\text{connu}}} \eta(g)$. Ainsi, une trajectoire de qualité conduit à obtenir des descriptions rarement obtenues. L'agent cherche à sélectionner une tâche qui conduira à une trajectoire de qualité. On modélise ce problème comme un problème de bandit manchot : chaque bandit est une tâche découverte dont la valeur associée est la qualité espérée d'une trajectoire générée par $\pi(\cdot, g)$:

$$v_\pi(g) = \mathbb{E}_\pi(q \mid g) = \sum_{g' \in \mathcal{G}_{\text{connu}}} p(g' \in \mathcal{G}_{\text{succès}} \mid g, \pi)\ \eta(g)$$

où $p(g' \in \mathcal{G}_{\text{succès}} \mid g, \pi)$, la probabilité de réussir g' en visant g sous la politique π, est simplement estimé par la fréquence empirique de cet événement sur les derniers épisodes. Pour maintenir un certain niveau d'exploration, on choisit finalement la prochaine tâche g à viser en adoptant une stratégie ϵ-*greedy* :

$$\mathbb{P}_\pi(g) = \frac{\epsilon}{|\mathcal{G}_{\text{connu}}|} + (1-\epsilon)\frac{v_\pi(g)}{\sum_{g' \in \mathcal{G}_{\text{connu}}} v_\pi(g')}$$

Hindsight Experience Replay – L'apprentissage de la politique d'action s'appuie sur la technique de Hindsight Experience Replay. Comme on l'a expliqué en Partie 3.2.4, cela permet de rejouer une transition en remplaçant l'objectif initialement visé par un autre objectif sur lequel on souhaite s'entraîner. Inspiré par Colas et al. (2019a), l'agent choisit intelligemment les objectifs qu'il substitue en privilégiant ceux pour lesquels il progresse le plus. L'agent s'auto-évalue périodiquement et mesure sa compétence SR_k. Il peut ainsi

obtenir une mesure de son progrès en apprentissage récent $LP_k(g) = |SR_k(g) - SR_{k-1}(g)|$ pour chacune des tâches découvertes. Au moment de choisir quelle tâche il ajoute à une transition (s, a, s'), il choisit g selon la distribution de probabilité suivante :

$$\mathbb{P}_\pi(g) = \frac{\epsilon}{|\mathcal{G}_{\text{connu}}|} + (1-\epsilon)\frac{LP(g)}{\sum\limits_{g' \in \mathcal{G}_{\text{connu}}} LP(g')}$$

Politique d'action conditionnée par une tâche

LE2 s'appuie sur l'algorithme DDPG détaillé en Partie 3.2.3. Il s'agit donc d'un agent acteur-critique. À l'instar de la fonction de récompense, l'acteur et le critique sont conditionnés par la tâche que l'agent cherche à accomplir (*goal-conditioned policy*) (Schaul et al., 2015a). La politique prend ainsi la forme suivante : $\pi : \mathcal{S} \times \mathcal{G} \to \mathcal{A}$.

La politique et le critique sont mis à jour de la manière suivante. On sélectionne un batch de transition (s, a, s') de $mem(\pi)$. Pour chaque transition, le module Hindsight Replay augmenté du mécanisme de motivation intrinsèque choisit une tâche g_s à substituer à la tâche originale et sur laquelle s'entraîner. Il calcule la récompense associée $\mathcal{R}(s, g_s)$ et construit un batch $(s, a, s', g_s, \mathcal{R}(s, g_s))$ de transitions complètes à partir desquels mettre à jour la politique et le critique.

3.5.2 L'environnement *ArmToolsToys*

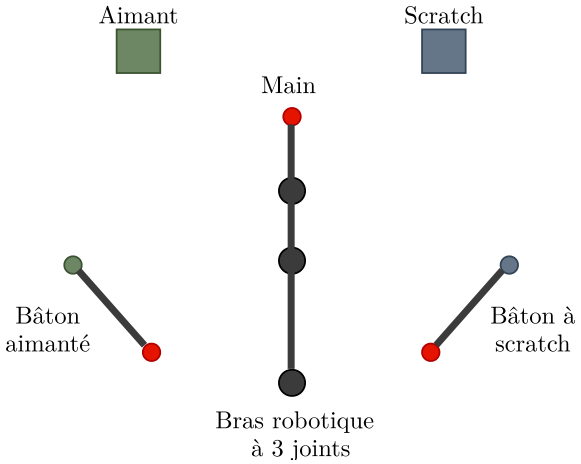


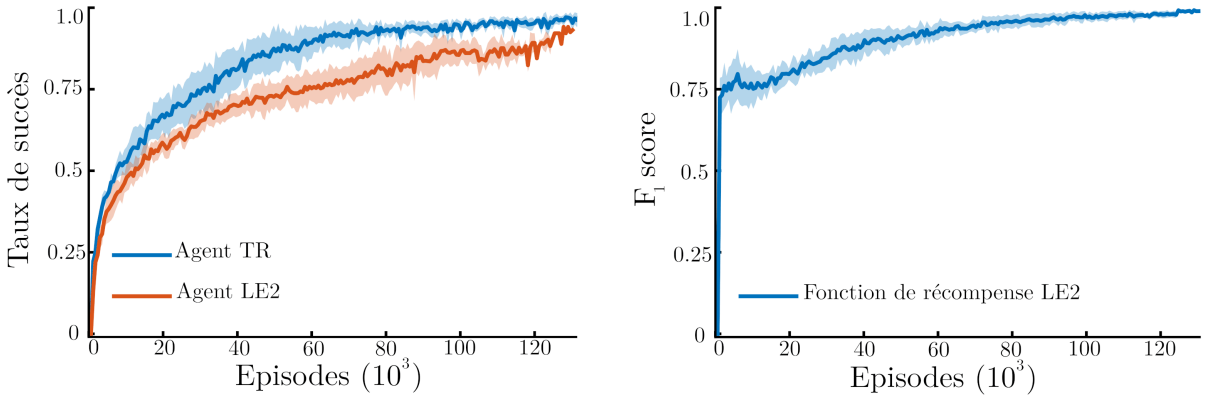
FIGURE 3.3 – Environnement *ArmToolsToys*.

Description – On évalue LE2 dans l'environnement *ArmToolsToys*, il s'agit d'un environnement en 2D de robotique simulée adapté de Forestier et Oudeyer (2016). Illustré en Figure 3.3, l'agent contrôle un bras robotique à 3 joints au bout duquel il dispose d'une main. Deux objets sont présents devant le bras : un cube aimanté et un cube à scratch. Pour manipuler ces cubes, l'agent ne peut pas utiliser directement sa main mais l'outil adapté devant lui : le bâton aimanté pour le cube aimanté et le bâton à scratch pour le cube à scratch. Le partenaire social définit 51 descriptions/tâches réalisables dans l'environnement qui sont présentées en Annexe B. Ce type d'environnement permet des tâches de difficulté variées : simples comme « *shift the hand to the right* » qui nécessite seulement d'apprendre à contrôler le bras aux plus difficiles comme « *move*

the magnet to the bottom right area » qui nécessite de contrôler le bras, d'apprendre à utiliser le bon outil et d'acquérir une notion de signification spatiale.

Représentation de l'état et de l'action – L'observation o_t de l'environnement est représentée par un vecteur à 17 dimensions contenant : la position angulaire des joints du bras (3), la position de la main (2), des poignées des outils (4), du bout des outils (4) et des cubes (4). L'état $s_t = (o_t, \Delta o_t)$ encode ainsi à la fois la position actuelle ainsi des objets ainsi que les déplacements depuis le début de l'épisode. L'agent contrôle la vitesse angulaire des joints ainsi que l'état de préhension de la main pour saisir ou lâcher. Une action est un vecteur à 4 dimensions.

3.5.3 Expériences et résultats



(a) Taux de succès moyenné sur les 51 tâches (moyenne / écart-type).

(b) F_1 score moyenné sur les tâches découvertes (moyenne / écart-type).

Figure 3.4 – Évaluation de l'agent LE2

Performances de π et $\mathcal{R}$ – On compare les performances de LE2 avec un agent (agent TR pour *True Reward*) qui aurait accès à la véritable fonction de récompense. L'objectif est de voir comment l'apprentissage de la fonction de récompense modifie les performances de l'agent. Celles-ci sont mesurées tous les 600 épisodes en moyennant le taux de succès de l'agent sur les 51 tâches possibles. On évalue aussi les performances de la fonction de récompense en mesurant le F_1 score entre les récompenses prédites et les récompenses réelles sur un ensemble de transitions récemment collectées par l'agent. On monitore ainsi la pertinence de la fonction de récompense au cours des simulations. Ces métriques, moyennées sur 10 simulations différentes, sont présentées en Figure 3.4. On observe comme l'on pouvait s'y attendre que LE2 apprend avec un léger retard par rapport à l'agent TR mais qu'in fine LE2 parvient à un niveau de performance quasi identique à celui de l'agent TR. Les débuts de l'apprentissage de la fonction de récompense sont très rapides

et cela permet à la politique d'action de démarrer rapidement ses progrès. En revanche, pour atteindre un F_1 score presque parfait, la fonction de récompense a besoin de près de 80 000 épisodes et cela explique le retard que prend la politique d'action sur l'agent TR.

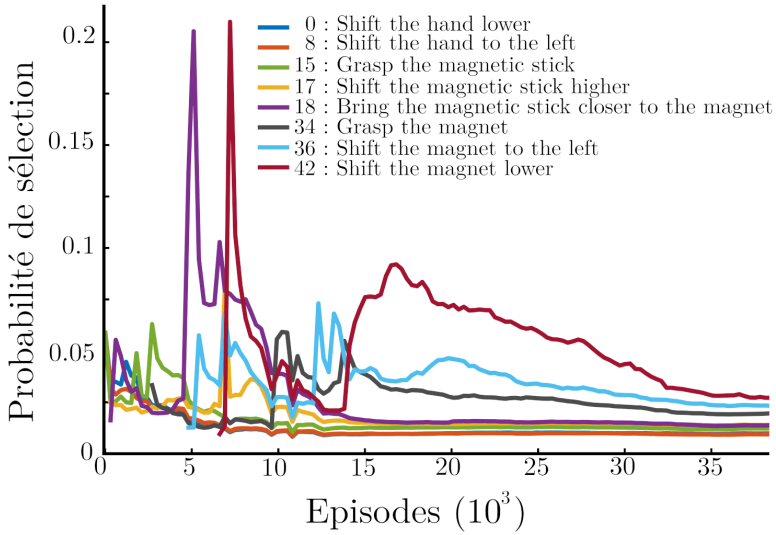


FIGURE 3.5 – Évolution de la probabilité de sélection pour un ensemble de tâches durant une simulation.

Curriculum – On peut observer l'apprentissage par curriculum en regardant l'évolution de la distribution de probabilité de sélection des tâches par le générateur. En Figure 3.5, on représente ainsi l'évolution de la probabilité de sélection pour quelques tâches. On peut voir ainsi l'ordre dans laquelle l'agent les apprend. Parmi les descriptions définies, certaines ne sont pas vraiment intéressantes pour elles-mêmes mais permettent de guider l'agent. Ainsi, apprendre à « rapprocher l'outil aimanté de l'aimant » permet au partenaire social d'orienter l'attention de l'agent pour qu'il apprenne que combiner l'outil aimanté et l'aimant conduit à des interactions intéressantes. Après s'être concentré sur cette tâche intermédiaire, on observe que l'agent se concentre naturellement et successivement sur les tâches qui consistent à attraper le cube aimanté puis le déplacer.

Délai d'apprentissage – On peut visualiser le délai entre la découverte d'une tâche, l'apprentissage de la fonction de récompense et celle de la politique d'action relative à la tâche. On l'illustre en Figure 3.6 où l'on trace, pour la tâche *Move the magnet to the bottom right area*, l'instant de sa découverte, le F_1 score de la fonction de récompense et le taux de succès de la politique d'action relative à cette tâche. On observe que le délai avant que la fonction de récompense ne démarre son apprentissage est relativement long et cela s'explique notamment par la stratégie de l'agent dans son choix des tâches. On

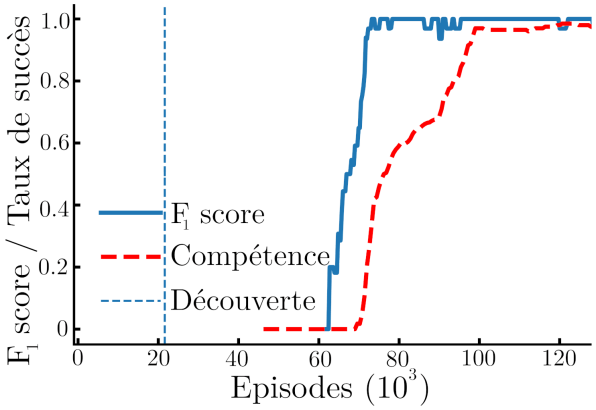


Figure 3.6 – Évolution des métriques pour une tâche spécifique.
Move the magnet to the bottom right area

constate aussi qu'une fois la fonction de récompense apprise, celle de la politique d'action est relativement rapide.

3.5.4 Conclusion

Contributions – Le développement de LE2 a permis d'apporter les contributions suivantes :

— Il est possible d'apprendre la fonction de récompense à partir des données collectées par la politique d'action et des descriptions du partenaire social. La fonction de récompense permet d'apprendre à lier le sens du langage directement sur les observations de l'environnement. Elle apprend le « quoi » que la politique d'action peut exploiter pour apprendre le « comment ».

— Les descriptions du partenaire social peuvent servir de tâches intermédiaires et guider l'agent à collecter des données pertinentes pour les tâches les plus difficiles. À noter que, contrairement à (R. Kaplan et al., 2017), cette aide est passive puisque c'est l'agent lui-même qui, via les mécanismes de motivation intrinsèque, se fixe ses propres objectifs.

Capacité de généralisation limitée – Cet agent est capable de généraliser les tâches apprises aux différentes configurations de l'environnement. Cependant, il n'est a priori pas en mesure de généraliser systématiquement. En effet, il est peu probable que l'apprentissage effectué dans le cadre d'*ArmToolsToys* soit transférable à un environnement similaire dans lequel on aurait introduit de nouveaux objets et outils.

En effet, l'utilisation d'un modèle de langage pré-appris donne à l'agent une notion du sens de chaque mot. Le rôle de la fonction de récompense est alors d'apprendre un mapping entre le sens d'une phrase et un changement dans l'état de l'environnement. La

fonction de récompense parvient à guider correctement l'agent, mais il est peu probable qu'elle généralise vraiment. La diversité des configurations dans *ArmToolsToys* ne permet d'ailleurs pas de le tester.

Du point de vue de l'agent, les différentes tâches sont indépendantes les unes des autres. Les tâches plus simples permettent certes d'apprendre les plus compliquées grâce aux mécanismes de motivation intrinsèque, mais il n'y a pas de transfert *horizontal* entre deux tâches similaires (comme *move the magnet* et *move the scratch*).

Dans la suite de ce chapitre, on propose donc d'apprendre le modèle de langage et on montre que les représentations apprises permettent de disposer d'un agent qui généralise de façon systématique.

3.6 IMAGINE : Exploration créative par l'imagination

3.6.1 Les bénéfices de l'imagination pour un agent apprenant

Tout comme LE2, IMAGINE découvre les tâches réalisables dans l'environnement au gré de ses interactions avec le partenaire social et apprend une fonction de récompense interne. Cependant, IMAGINE apprend aussi à représenter les tâches à travers son propre modèle de langage. Les bénéfices de cet apprentissage sont doubles. L'agent généralise plus facilement entre des tâches similaires et peut envisager une nouvelle forme d'exploration et d'apprentissage autonome, proche de ce que représente l'imagination pour un humain. En effet, l'agent peut exploiter certaines propriétés de compositionnalité du langage pour imaginer de nouvelles descriptions à partir de celles qu'il connaît déjà. Ces dernières, traduites dans l'espace des tâches par le modèle de langage appris, ouvrent la voie à une exploration créative de l'environnement et à la découverte de nouvelles tâches. Enfin, la fonction de récompense interne permet de les apprendre en totale autonomie. Dans cette partie, on décrit l'architecture d'un tel agent.

Imaginer pour explorer

L'exploration de l'environnement est en effet cruciale pour un système d'apprentissage par renforcement. La stratégie *ϵ-greedy* est la méthode la plus simple et aussi la plus employée pour assurer un niveau minimum d'exploration. On peut néanmoins envisager une exploration plus *active* de l'environnement. Les agents multi-objectifs peuvent, par exemple, orienter l'exploration en fonction du but poursuivi. L'heuristique de motivation intrinsèque employée pour LE2 pousse ainsi l'agent à privilégier les tâches qui le conduisent à des situations rarement rencontrées. Certains travaux proposent d'apprendre la représentation des tâches, puis de générer de nouvelles tâches dans cet espace (Laversanne-Finot et al.,

2018 ; Nair et al., 2018 ; Nair et al., 2019 ; Pong et al., 2020). L'une des limites de ces approches est que l'espace des tâches considéré est souvent l'espace d'états lui-même. Une tâche consiste ainsi à mettre l'environnement dans un état particulier. Dans ce contexte, les modèles génératifs de tâches (dont d'état) proposés permettent certes à l'agent d'explorer des tâches qu'il n'aurait jamais tentées sinon, mais le maintiennent dans la distribution de tâches déjà rencontrées, limitant *de facto* le potentiel exploratoire de ces tâches.

IMAGINE adopte une stratégie similaire mais puisque, plutôt que de générer des tâches, il génère des descriptions. La génération dans l'espace des descriptions permet, en exploitant certaines propriétés de composition du langage, de s'affranchir de la distribution des descriptions connues. L'expression de Noam. Chomsky (1957) « *Colorless green ideas sleep furiously* » illustre les possibilités créatives du langage. Syntaxiquement correcte mais sémantiquement étrange, elle s'appuie sur la combinaison de mots qui séparément sont bien connus, et qui, ensemble, expriment une idée nouvelle et certainement jamais rencontrée. C'est sur ce même principe qu'IMAGINE emploie le langage pour imaginer de nouvelles descriptions (et donc des tâches) : recombiner des phrases pour exprimer des idées nouvelles et conduire l'agent à une exploration créative.

Cette idée s'inspire directement de la vision vygotskienne du langage développée en Partie 1.5.4. L'enfant exploite en effet une forme de parole égocentrique comme un outil cognitif pour résoudre des tâches. Le langage lui permet de faire preuve d'une créativité unique et Vygotsky (1978, Chapitre 2) observe que son utilisation augmente avec la difficulté de la tâche.

Imaginer pour apprendre

En plus de permettre à l'agent d'explorer son environnement d'une manière unique, l'agent peut aussi apprendre à effectuer les nouvelles tâches imaginées sans avoir besoin du partenaire social. En effet, la fonction de récompense permet de superviser l'apprentissage de la politique d'action en calculant une récompense pour les tâches imaginées même si elles n'ont pas été vues pendant l'entraînement. Cependant, pour que le signal de la fonction de récompense soit exploitable par la politique d'action, celle-ci doit posséder certaines propriétés de *généralisation systématique*.

Pour un système artificiel, la généralisation systématique signifie pouvoir transférer sa compétence entre des objets similaires. Dans *ArmToolsToys* par exemple, il s'agit de généraliser la signification de la description *shift the magnet higher* à partir des descriptions *shift the magnet lower* et *shift the hand higher*.

L'une des conditions qui semblent favoriser la généralisation systématique est l'existence d'une structure inhérente à l'environnement et/ou au problème (Hill et al., 2019 ; Kuo et al., 2020 ; Ruis et al., 2020). Pour IMAGINE, cette structure se trouve d'une part dans le

langage dont les propriétés de compositionnalité permettent d'envisager un généralisation systématique telle que l'entendent Fodor et Pylyshyn. Elle se trouve d'autre part dans la structure de l'environnement. On adopte une représentation dite *single-slot object file* (Green et Quilty-Dunn, 2017). Chaque objet est ainsi représenté par un vecteur d'état propre dont les attributs sont similaires entre les objets. On propose aussi une architecture modulaire, adaptée à cette représentation centrée autour des objets, et basée sur les DEEP SET (Zaheer et al., 2017). On démontre que cette représentation et cette architecture conduisent à observer une forme de généralisation systématique aussi bien au niveau de la fonction de récompense que de la politique.

Description du fonctionnement d'IMAGINE

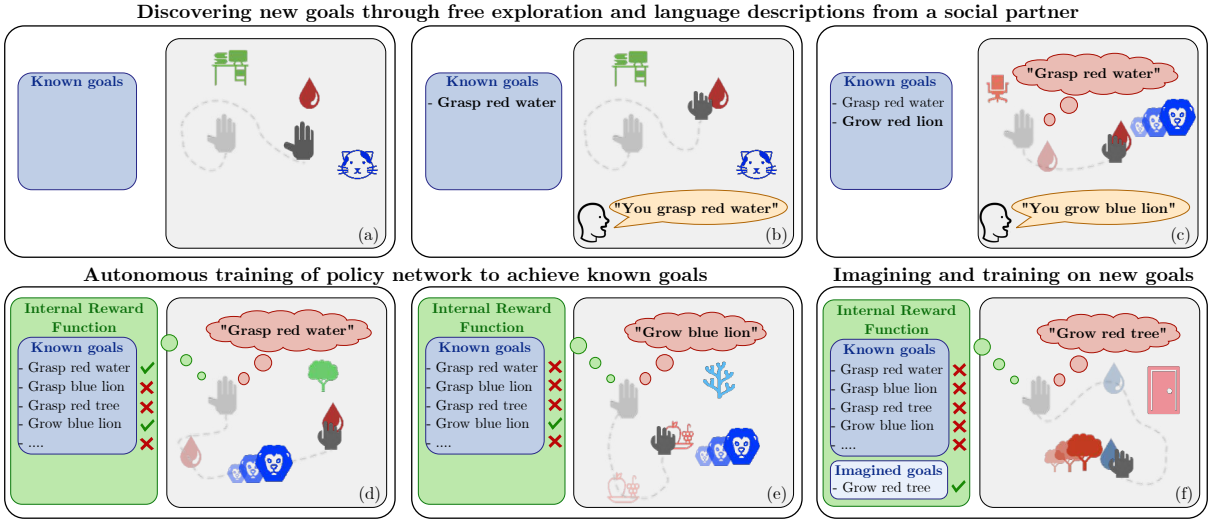


FIGURE 3.7 – Description d'IMAGINE. (Colas et al., 2020) L'agent (la main) peut interagir avec des objets dans un environnement procéduralement généré. Un partenaire social fournit une supervision en donnant des phrases descriptives des réussites de l'agent.

IMAGINE fonctionne selon un paradigme très proche de LE2. Cependant, au début de chaque épisode IMAGINE peut choisir de se fixer une tâche connue ou d'en imaginer une nouvelle. On évalue IMAGINE dans un nouvel environnement *Playground* que l'on décrit dans la section suivante. Il permet de disposer d'une plus grande variété d'objets partageant certaines caractéristiques communes et par lesquels on peut espérer observer une généralisation systématique. La Figure 3.7 illustre son fonctionnement : après une phase d'exploration et d'acquisition de descriptions langagières auprès du partenaire social (figures a, b, c), l'agent peut s'entraîner en autonomie sur les tâches connues ou imaginées grâce à sa fonction de récompense interne (figures d, e, f).

3.6.2 L'environnement *Playground*

L'environnement *ArmToolsToys* n'est pas tout à fait adapté à l'étude des processus d'imagination. En effet, il ne dispose que de deux objets et deux outils, et ne présentent pas d'autre variabilité que leurs positions initiales. Nous avons développé un nouvel environnement *Playground* que nous décrivons ci-dessous. Ce dernier propose une variété d'objets aux caractéristiques différentes mais dont les similarités permettent de mesurer la capacité de généralisation de l'agent.

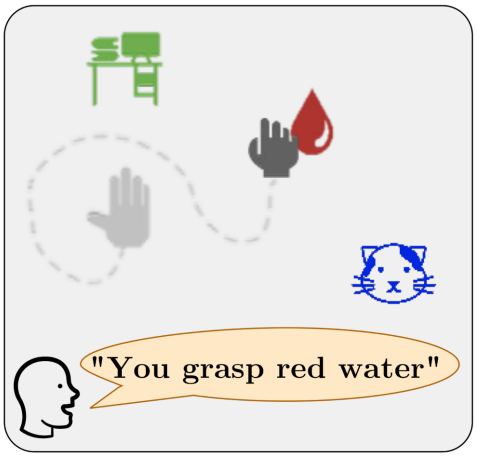


FIGURE 3.8 – Un environnement *Playground*

Description – *Playground* est en environnement 2D généré de manière procédurale dans lequel un agent peut se déplacer, ce dernier est doté d'une main pour interagir avec les objets présents. Chaque environnement généré contient 3 des 32 types d'objets (chien, cactus, sofa, etc.). Ces derniers sont répartis en différentes catégories : animal, plante, être vivant, meuble et provisions, illustrées en Figure 3.9. La catégorie être vivant est divisée en deux sous-catégories animal et plante. Chaque objet est généré avec ses propres caractéristiques de couleur et de taille et positionné aléatoirement dans l'environnement. Une dynamique intéressante de *Playground* est la possibilité de faire grandir certains objets. Les animaux grandissent lorsqu'on leur apporte de l'eau ou de la nourriture, les plantes grandissent uniquement avec de l'eau et les meubles ne peuvent évidemment pas grandir. La Figure 3.8 illustre un environnement issu de *Playground* contenant un meuble vert, de l'eau rouge et un chat bleu.

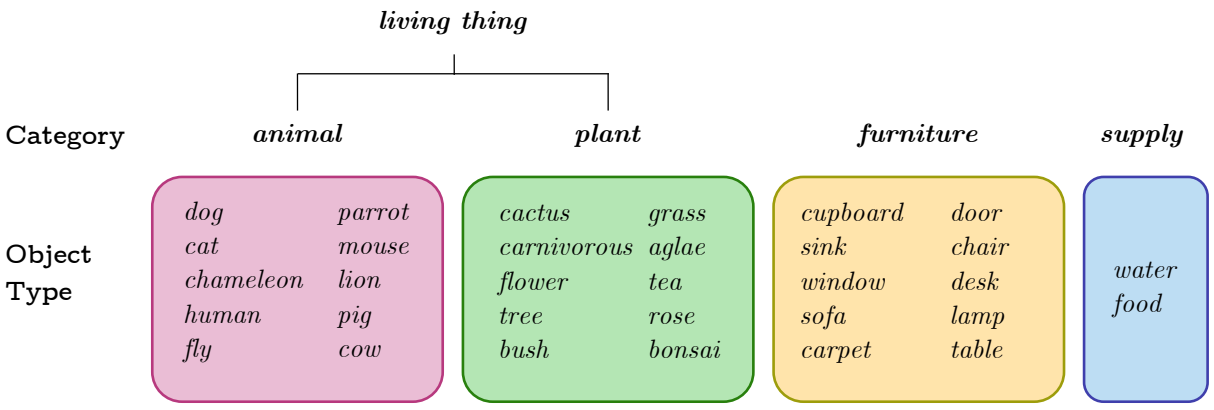


FIGURE 3.9 – Les différents types d'objets et catégories dans *Playground*

Représentation de l'état et de l'action – Tout comme pour LE2 dans *ArmToolsToys*, on différencie, l'*état* et l'*observation*. Une observation de l'environnement est représentée par un ensemble de vecteurs décrivant l'agent et les objets présents. Le vecteur d'observation d'un objet est de taille 39 et contient son type (32), sa position (2) sa couleur RGB (3), sa taille (1) et s'il est tenu par l'agent ou non (1). L'observation représentant l'agent est de taille 3 et contient la position (2) ainsi que la position fermée ou ouverte de la main (1). Il faut noter que la catégorie d'un objet n'est pas encodée explicitement dans la description d'un objet. Le vecteur d'état pour l'objet k est toujours la concaténation de l'observation courante o_t^k et la différence avec l'observation au début de l'épisode : $s_t^k = (o_t^k, \Delta o_t^k)$ (voir Partie 3.5.1). L'agent peut se déplacer par une translation continue et bornée dans l'espace, il peut aussi saisir et relâcher des objets. L'espace d'action est donc continue et de dimension 3 (translation : 2, statut de la main : 1 booléen).

Grammaire – Les descriptions des tâches réalisables dans *Playground* s'articule autour de 3 verbes d'actions : se déplacer vers un endroit (GO), attraper un objet (GRASP) ou faire grandir un objet (GROW). La grammaire présentée en Annexe C.1 permet d'exprimer 256 descriptions différentes, correspondants aux tâches réalisables dans l'environnement. On donne quelques exemples représentatifs en Tableau 3.1. Un objet peut être désigné par sa couleur uniquement (*blue thing*), son type d'objet ou de catégorie (*any cat* ou *any animal*), par sa couleur et son type d'objet/catégorie (*red cat* ou *red animal*). Comme on l'a expliqué, une catégorie regroupe plusieurs types d'objets mais n'est pas encodé directement dans le vecteur d'état et c'est donc à l'agent d'apprendre à regrouper les objets par catégorie. Cette grammaire relativement simple nous permet d'étudier finement les effets de l'imagination sur l'exploration et la généralisation. Le fait de privilégier la simplicité du langage n'est cependant pas incohérent avec la simplicité des premières interactions langagières d'un parent avec son enfant (Mintz, 2003).

Prédicat	**Exemples**
GO	go left
GRASP ANY	grasp any red thing
GRASP	grasp red cat
	grasp any plan
GROW ANY	grow any blue thing
GROW	grow green animal
	grow any tree

TABLE 3.1 – Exemples de descriptions réalisables dans *Playground*

Rôle du partenaire social

Comme pour LE2, le rôle du partenaire social est d'observer le début et la fin des épisodes et de fournir les descriptions langagières appropriées. Dans *Playground*, il a

aussi le rôle de « préparer » l'environnement et d'offrir à l'agent les bonnes opportunités d'apprentissage. Ainsi une fois que l'agent a choisi la tâche qu'il va poursuivre pendant l'épisode, le partenaire social organise la scène en choisissant les objets permettant de la réaliser ainsi que des objets distracteurs. Ce rôle s'apparente à celui que jouent les parents lorsqu'ils donnent à l'enfant les jouets dont il a besoin pour réaliser ce qu'il souhaite. Il s'inscrit parfaitement dans la théorie de l'échafaudage développée en Partie 1.5.5 où l'enjeu est de contrôler la difficulté de la tâche que cherche à résoudre l'agent : ni trop difficile en s'assurant que l'agent dispose des bons objets, ni trop facile en ajoutant des distracteurs.

On aurait pu envisager de générer d'abord un environnement et que l'agent apprenne à générer une tâche réalisable dans l'environnement. Outre la difficulté supplémentaire que cela représente, accorder un rôle plus actif au partenaire social permet d'envisager une nouvelle forme d'interaction.

Tâches d'entraînement et de test

L'ensemble des descriptions réalisables est partitionné en deux ensembles : $\mathcal{D}^{\text{train}}$ et $\mathcal{D}^{\text{test}}$. Le premier désigne l'ensemble des descriptions que le partenaire social est autorisé à prononcer et correspond donc aux tâches que l'agent peut découvrir au cours de leurs interactions. Au contraire les descriptions de $\mathcal{D}^{\text{test}}$ ne sont jamais prononcées par le partenaire social. Ces tâches servent à tester les capacités de généralisation de l'agent.

Différents types de descriptions test – La modularité de *Playground* et sa grammaire simple nous permettent de définir les descriptions qui seront utilisées pour tester la généralisation de l'agent. On définit 5 types de descriptions tests qui sont retirées de $\mathcal{D}^{\text{train}}$ pour former $\mathcal{D}^{\text{test}}$. Chacun de ces types permet d'évaluer un mode particulier de généralisation :

Type 1. *Association Objet-Attribut* : Il s'agit de la capacité à associer un attribut et un objet qui n'ont jamais été vus ensemble. On retire ainsi de $\mathcal{D}^{\text{train}}$ les 5 descriptions impliquant les combinaisons suivantes *{blue door, red tree, green dog}*.

Type 2. *Identification par l'attribut* : Il s'agit de la capacité à identifier un objet par son attribut. On retire de $\mathcal{D}^{\text{train}}$ les 8 descriptions avec le mot *flower*. Pour interpréter *grasp red flower*, l'agent doit identifier l'objet *flower* uniquement par sa couleur.

Type 3. *Généralisation Prédicat-Catégorie* : Il s'agit de la capacité à comprendre la combinaison d'un prédicat avec un nom de catégorie dont l'association n'a jamais été rencontrée par l'agent. Comme expliqué en Partie 3.6.2, une catégorie d'objets n'est pas encodée explicitement mais peut être inférée par l'agent de son expérience. On retire de $\mathcal{D}^{\text{train}}$ les 4 descriptions de type *grasp animal*.

Type 4. *Généralisation Prédicat-Object* : Similaire au type 3 mais a priori plus simple, il s'agit de la capacité à comprendre la combinaison d'un prédicat et d'un objet dont l'association n'a jamais été rencontrée par l'agent. On retire de $\mathcal{D}^{\text{train}}$ les 4 descriptions impliquant *grasp* et *fly*.

Type 5. *Généralisation sur la dynamique des prédicats* : Il s'agit de la capacité à généraliser le sens d'un prédicat à une autre catégorie d'objets pour laquelle la dynamique du prédicat est différente. On retire de $\mathcal{D}^{\text{train}}$ les 48 descriptions qui impliquent *grow* et un objet de la catégorie *plant* ainsi que les noms de catégories *plant* et *living thing*. La dynamique de l'environnement est différente pour les plantes et les animaux et l'on souhaite voir comment IMAGINE s'adapte. En effet, sans imagination, l'agent n'a aucune chance de réaliser qu'il est préférable d'amener uniquement de l'eau et non de la nourriture aux plantes.

Chacun des types de test permet d'évaluer si l'agent arrive à rapprocher des éléments vus séparément. Certains sont plus difficiles que d'autre à acquérir. Le Type 2, par exemple, nécessite de pouvoir identifier l'objet *flower* uniquement à partir de sa couleur ou par élimination des autres objets. Les tests de Type 5 est plus complexe que le Type 4 car la dynamique du prédicat *grow* change, pas celle de *grasp*. L'ensemble de ces descriptions de $\mathcal{D}^{\text{train}}$ et des différents types de test sont disponibles en Tableaux C.1 et C.2.

3.6.3 Architecture d'IMAGINE

On décrit maintenant l'architecture de l'agent IMAGINE. On la représente en Figure 3.10 sous un format similaire à LE2 (voir Figure 3.2, page 88). On peut en souligner les principales différences :

— Le générateur de descriptions collecte les descriptions obtenues lors des interactions avec le partenaire social et en imagine de nouvelles à partir de l'heuristique décrite en Partie 3.6.4. Durant un épisode, la politique d'action peut donc être conditionnée par une tâche connue ou imaginée

— Le modèle de langage $\mathcal{L}$ est maintenant appris par rétro-propagation du gradient de l'erreur de la fonction de récompense (*end-to-end*). On utilise un réseau récurrent LSTM (Hochreiter et Schmidhuber, 1997). Il permet de traduire les descriptions langagières ($\mathcal{D}$) dans un l'espace des tâches $\mathcal{G} \subset \mathbb{R}^{100}$.

— Le modèle de langage intervient après le générateur de descriptions. En effet, la représentation des tâches évolue avec l'apprentissage de $\mathcal{L}$. La représentation courante des descriptions est utilisée pour conditionner la politique d'action pendant un épisode ainsi que pour l'apprentissage de la fonction de récompense et l'utilisation du Hindsight Replay.

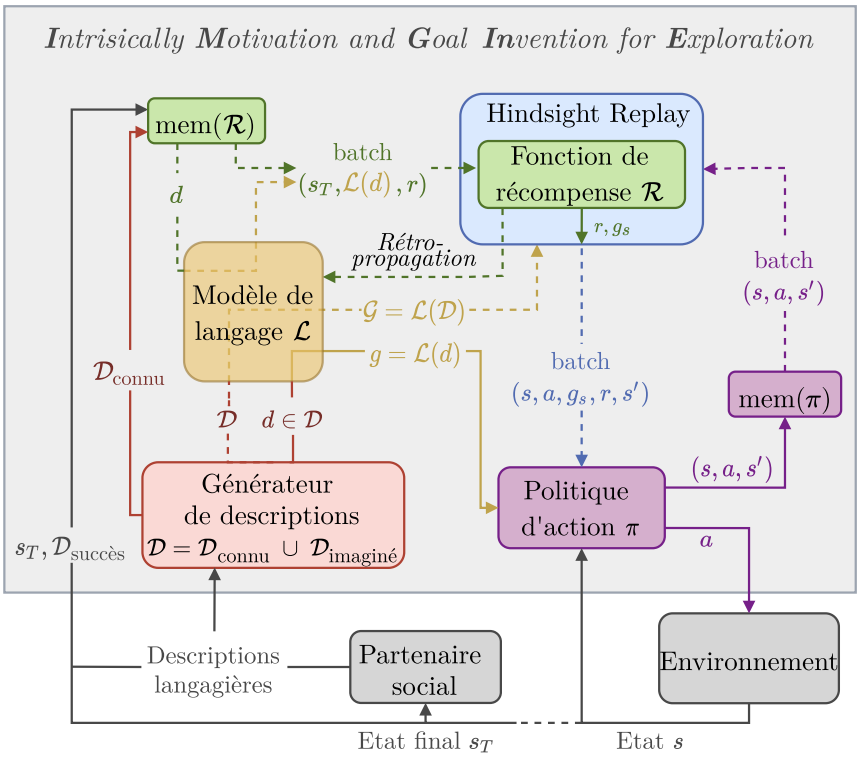


FIGURE 3.10 – Architecture d'IMAGINE. Les lignes pointillées indiquent un entraînement ou une mise à jour.

— Le Hindsight Replay se fait sur les tâches connues et imaginées. Cela permet à l'agent de s'entraîner sur des tâches imaginées. Cela nécessite toutefois que la fonction de récompense généralise correctement sur ces tâches.

Architecture modulaire centrée sur les objets

Une différence importante avec LE2, qui n'apparaît pas sur la Figure 3.10, est l'utilisation d'une architecture modulaire pour la fonction de récompense, la politique et le critique. Celle-ci est essentielle pour exploiter la représentation commune de l'état des objets et permettre un transfert efficace de compétence entre ceux-ci. C'est l'architecture modulaire qui permet la généralisation systématique évoquée en Partie 3.6.1.

Cette dernière s'appuie sur des DEEP SET (Zaheer et al., 2017) et sur un mécanisme attentionnel pour sélectionner les éléments de l'environnement pertinents (Chaplot et al., 2018). Les DEEP SET sont des modèles de réseaux de neurones s'appliquant sur un ensemble d'objets de même nature, et non un seul objet comme c'est le cas classiquement. Ils sont particulièrement adaptés pour traiter les objets dans *Playground* dont l'état est représenté selon un formalisme commun.

L'idée générale de notre architecture modulaire est la suivante. La représentation de la tâche $g = \mathcal{L}(d)$ est projetée dans un vecteur $A^{att}(g)$. Ce masque attentionnel est ensuite

appliqué à la concaténation de l'état s^k de chaque objet k avec l'état de l'agent s^a ($\mathbf{s}^k = [s^k, s^a]$) par un produit de Hadamard (*gated-attention*) : $A^{att}(g) \odot \mathbf{s}^k = A^{att}(g) \odot [s^k, s^a]$. Puis, les représentations obtenues sont agrégées par une fonction préservant *l'invariance par permutation*. En effet, le résultat obtenu ne doit pas dépendre d'un quelconque ordre des objets. Enfin, cette représentation latente de l'état de l'environnement conditionné par la tâche est projetée dans l'espace voulu (récompense, action, Q-value). À noter que la projection attentionnelle est apprise et différente pour chacune des fonctions ($\mathcal{R}$, π, Q).

Fonction de récompense – Plus précisément, la fonction de récompense prend la forme suivante. $A^{att}_{\mathcal{R}}(g) = \boldsymbol{\alpha}^g$ est le vecteur attentionnel, $\text{NN}^{\mathcal{R}}$ est un réseau partagé qui calcule une fonction de récompense spécifique à chaque objet. La récompense binaire est finalement calculée via une fonction OR valant 1 si l'une des récompenses spécifiques est supérieure à 0.5, et 0 sinon. Une telle fonction n'étant pas différentiable, elle est en pratique approximée par NN^{OR}, un réseau pré-entraîné, pour permettre un entraînement de bout en bout par *backpropagation*. À noter que $\boldsymbol{\alpha}^g$ dépendant de g et donc de $\mathcal{L}$, cela permet d'entraîner le modèle de langage en même temps que $\mathcal{R}$.

$$\mathcal{R}(\mathbf{s}, g) \;=\; \text{NN}^{\text{OR}}([\text{NN}^{\mathcal{R}}(\mathbf{s}^k \odot \boldsymbol{\alpha}^g)]_{k\in[1..N]})$$

Politique d'action et critique – La politique d'action et le critique prennent la forme suivante. Les masques attentionnels sont notés $A^{att}_{\pi}(g) = \boldsymbol{\beta}^g$ et $A^{att}_{Q}(g) = \boldsymbol{\gamma}^g$. L'agrégation est la composition d'un réseau partagé et d'une somme sur tous les objets. Comme pour LE2, π et Q sont entraînés à partir de DDPG (Lillicrap et al., 2016) selon les équations suivantes :

$$\pi(\mathbf{s}, g) \;=\; \text{NN}^{\text{a}}\Big(\sum_{k\in[1..N]} \text{NN}^{\pi}(\mathbf{s}^k \odot \boldsymbol{\beta}^g)\Big) \qquad Q(\mathbf{s}, \mathbf{a}, g) \;=\; \text{NN}^{\text{a-v}}\Big(\sum_{k\in[1..N]} \text{NN}^{Q}([\mathbf{s}^k,\ \mathbf{a}] \odot \boldsymbol{\gamma}^g)\Big).$$

Les architectures neuronales de la politique et de la fonction de récompense sont représentées en Figure 3.11. La fonction de récompense est représentée en vert sur la droite et la politique en violet à gauche. Le modèle de langage est en jaune et les mécanismes attentionnels en bleu.

Générateur de descriptions – Comme pour LE2, le générateur de descriptions sélectionne une description à réaliser pour l'épisode. En revanche, on n'utilise pas les mécanismes de motivation intrinsèque, car ceux-ci n'ont pas montré d'amélioration significative de la performance. Au début de chaque épisode, une description est choisie aléatoirement et avec la même probabilité parmi l'ensemble des descriptions connues $\mathcal{D}_{\text{connu}}$ ou imaginées $\mathcal{D}_{\text{imaginé}}$, ou seulement $\mathcal{D}_{\text{connu}}$ si l'imagination est désactivée. À la fin de chaque épisode, à partir

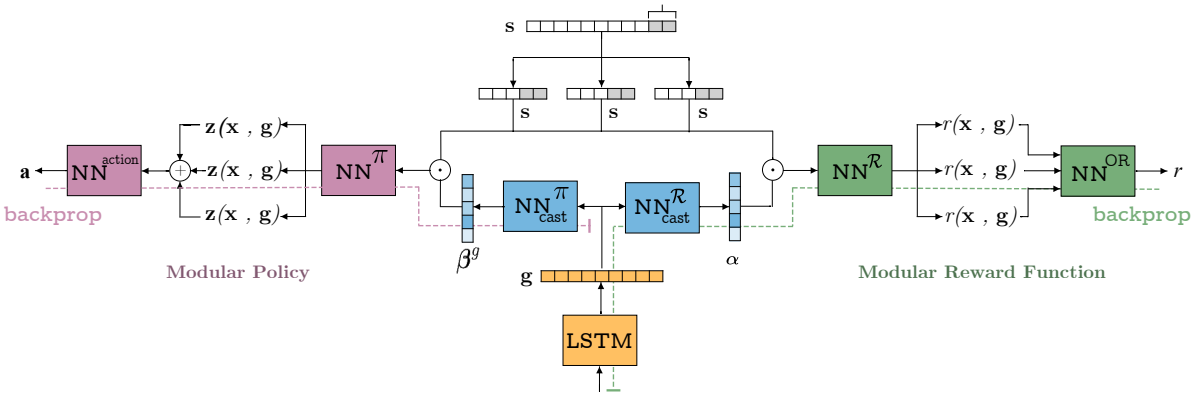


Figure 3.11 – Diagramme de l'architecture modulaire neuronale de la politique et de la fonction de récompense. (Colas et al., 2020)

des descriptions du partenaire social, le générateur de descriptions détermine les nouvelles descriptions $\mathcal{D}_{\text{n}}$ et met à jour l'ensemble des descriptions connues $\mathcal{D}_{\text{n}} \leftarrow \mathcal{D}_{\text{succès}} \setminus \mathcal{D}_{\text{connu}}$ puis $\mathcal{D}_{\text{connu}} \leftarrow \mathcal{D}_{\text{connu}} \cup \mathcal{D}_{\text{succès}}$. Puis le générateur de tâches déterminent l'ensemble des descriptions imaginables $\mathcal{D}_{\text{imaginé}}$à partir de l'heuristique décrite dans la section suivante et l'Algorithme 3.1.

3.6.4 Mécanisme d'imagination

Il nous reste maintenant à décrire la manière dont les descriptions sont imaginées. Le mécanisme d'imagination employé est directement inspiré du formalisme des grammaires de construction rapidement développées en Partie 1.5.1 et sur la capacité des enfants à exploiter les premières constructions apprises pour générer de nouvelles formes. On considère ainsi une grammaire de construction très simple dans laquelle les différentes descriptions acquises par interaction avec le partenaire social sont issues de construction que l'agent essaie d'inférer en substituant des mots par d'autres mots connus. L'utilisation de grammaires de construction dans des modèles computationnelles a montré qu'elles permettaient de généraliser entre des constructions (Dominey et Boucher, 2005 ; Hinaut et Dominey, 2013). On s'inspire donc de ce cadre pour construire notre heuristique d'imagination.

Description de l'heuristique

L'heuristique s'appuie sur la notion de *mots équivalents*, c'est-à-dire de mots qui jouent des rôles similaires dans des constructions. Deux mots sont dits équivalents s'ils apparaissent dans deux descriptions qui ne diffèrent que d'un mot. Par exemple les descriptions *grasp red lion* et *grow red lion* ne diffèrent que par le mot *grasp*/*grow*. On

Algorithme 3.1 : Imagination de nouvelles descriptions

Data : Ensemble des descriptions connues et déjà imaginées $\mathcal{D}_{\text{connu}}$ et $\mathcal{D}_{\text{imaginé}}$
Data : Liste des classes de mots équivalents $\mathcal{W}^{eq}$
Data : Ensemble des constructions $\mathcal{C}$
Input : Nouvelles descriptions $\mathcal{D}_{\text{n}} \subset \mathcal{D}_{\text{connu}}$
Output : $\mathcal{D}_{\text{imaginé}}$, $\mathcal{W}^{eq}$, $\mathcal{C}$

```
# Calcul des classes de mots équivalents
for d_n ∈ D_n do
    nouvelle_construction = True;
    for d_c ∈ D_connu \ {d_n} do
        if distance d'édition = 1 then
            nouvelle_construction = False ;
            w_n, w_c ← mots équivalents;
            nouvelle_classe = True;
            for s_eq ∈ W^eq do
                if w_n ∈ s_eq ou w_c ∈ s_eq then
                    nouvelle_classe = False;
                    s ← s ∪ {w_c, w_n};
            if nouvelle_classe then W^eq.append({w_c, w_n});
    if nouvelle_construction then C ← d_n;
# Imagination des nouvelles descriptions
for d_c ∈ C do
    for w ∈ d_c do
        for s_eq ∈ W^eq do
            if w ∈ s_eq then
                for w_eq ∈ s_eq do
                    d_im ← d_c.replace(w, w_eq);
                    if d_im ∉ D_connu then D_imaginé ← D_imaginé ∪ {d_im};
return D_imaginé, W^eq, C
```

peut ainsi définir des classes d'équivalence de mots et substituer dans des descriptions connues des mots par d'autres appartenant à la même classe. À partir de la description *grasp green tree*, on peut ainsi imaginer la description *grow green tree*. Certaines des descriptions imaginées par cette heuristique ne signifient rien (*go bottom top*), d'autres sont correctes mais pas forcément réalisables dans l'environnement (*grow red lamp*). Le choix d'une grammaire simple nous permet d'employer ce type d'heuristique pour explorer l'utilisation du langage comme un outil d'exploration et de généralisation.

L'Algorithme 3.1 décrit cette heuristique sous forme de pseudocode en deux temps : (1) l'algorithme détermine les classes de mots équivalents $\mathcal{W}^{eq}$ et les constructions $\mathcal{C}$; (2) l'algorithme imagine de nouvelles descriptions en substituant dans les constructions un mot équivalent. La distance d'édition entre deux mots mesure le nombre de mots dont

deux chaînes de caractères diffèrent (similaire à la distance de Levenshtein mais au niveau du mot). On compare les nouvelles descriptions collectées après un épisode à l'ensemble des descriptions connues, pour mettre à jour $\mathcal{W}^{eq}$ et $\mathcal{C}$. L'ensemble des descriptions imaginées $\mathcal{D}_{\text{imaginé}}$ est reconstruite à partir de $\mathcal{W}^{eq}$ et $\mathcal{C}$.

La Figure 3.12 est un diagramme de Venn de l'espace des descriptions. On voit que certaines descriptions imaginables ne sont pas réalisables et que certaines descriptions de l'ensemble de test ne sont pas imaginables (celles de Type 2). Il faut noter aussi que toutes les descriptions de $\mathcal{D}^{\text{train}}$ sont imaginables et bien que les descriptions connues soient filtrées de $\mathcal{D}_{\text{imaginé}}$, il peut arriver que certaines descriptions de $\mathcal{D}^{\text{train}}$ soient imaginées avant d'être découvertes.

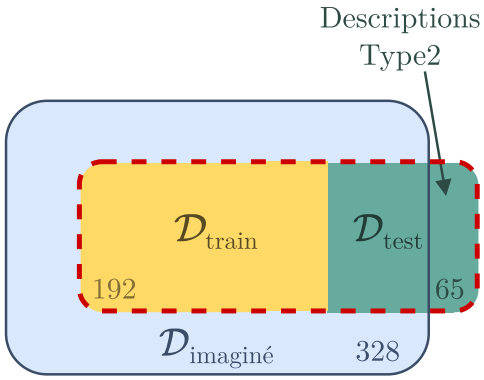


Figure 3.12 – Diagramme de Venn de l'espace des descriptions $\mathcal{D}$

3.7 Imagine : Expériences et résultats

On évalue maintenant les performances d'Imagine en termes d'apprentissage de la fonction de récompense, de la politique d'action. On évalue la capacité de généralisation systématique ainsi que les effets de l'imagination sur l'apprentissage et l'exploration. Les résultats présentés ici sont une sélection de ceux présentés dans Colas et al. (2020).

3.7.1 Métriques étudiées

On mesure les performances d'Imagine à partir de trois métriques :

La performance de la politique sur $\mathcal{D}^{\text{test}}$: le taux de succès moyen $\overline{\texttt{SR}}$ sur 30 épisodes. Cela permet de mesurer la généralisation systématique de l'agent sur des tâches pour lesquelles il n'a pas reçu de supervision du partenaire social. On observe deux formes de généralisation :

— *zéro-shot* : il s'agit d'une généralisation naturelle à de nouvelles descriptions sans qu'un entraînement spécifique ne soit nécessaire. On peut observer ce type de généralisation pour la politique et la fonction de récompense.

— *n-shot* : il s'agit d'une forme de généralisation permise par le processus d'imagination. L'agent est capable de *fine-tuner* sa politique d'action sur de nouvelles descriptions, à partir de sa fonction de récompense interne, et donc sans avoir recours à une supervision externe. On observe ce type de généralisation pour la politique d'action. Cette forme de généralisation est nécessaire pour réussir la généralisation de Type 5.

L'exploration en direction d'objectifs (*goal-oriented exploration*) : on désigne par cette expression le fait de mesurer si le comportement exploratoire de l'agent est interprétable de l'extérieur par un humain, c'est-à-dire si un observateur pourrait attribuer une intentionnalité à l'agent. On mesure cette métrique relativement à un ensemble de pseudo-tâches bien choisie. Il peut s'agir de $\mathcal{D}^{\text{test}}$ ou d'un autre ensemble définie manuellement. Plus concrètement, étant donné un ensemble de descriptions $\mathcal{D}^{\text{int}}$, on définit le *nombre d'interactions intéressantes* I2C par le nombre de fois où sur les 600 derniers épisodes, l'agent a réussi les descriptions de $\mathcal{D}^{\text{int}}$, indépendamment de l'objectif visé pendant l'épisode :

$$\text{I2C}(\mathcal{D}^{\text{int}}) = \sum_{i \in \mathcal{D}^{\text{int}}} \sum_{t=1}^{600} \delta_{i,t} \qquad \text{où } \delta_{i,t} = 1 \text{ si } i \text{ est réalisée pendant l'épisode } t.$$

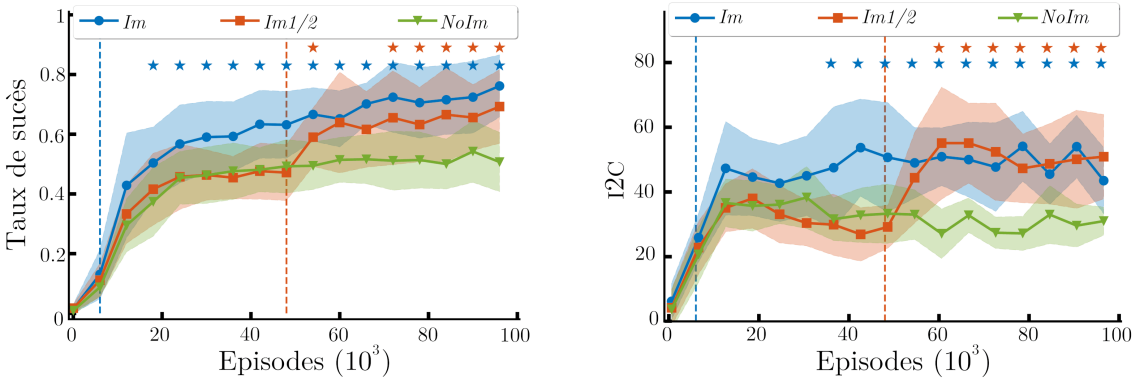
On évalue I2C sur $\mathcal{D}^{\text{test}}$ ainsi que sur un ensemble $\mathcal{D}^{\text{extra}}$ contenant certaines descriptions non réalisables dans l'environnement mais intéressantes en termes d'exploration comme le fait d'amener de l'eau ou de la nourriture à un meuble. Cela démontre une forme de généralisation de la dynamique avec laquelle les animaux et les plantes grandissent.

La performance de la politique sur $\mathcal{D}^{\text{train}}$: *Playground* étant généré procéduralement, cela permet de mesurer la généralisation de la politique à de nouveaux états pour les tâches découvertes en interaction avec le partenaire social. En pratique, on observe qu'IMAGINE généralise très bien sur $\mathcal{D}^{\text{train}}$ et donc à de nouveaux états $\overline{\texttt{SR}} = 0.95 \pm 0.05$. C'est pourquoi, dans la suite on se concentre essentiellement sur les performances sur $\mathcal{D}^{\text{test}}$.

On calcule la moyenne et l'écart-type sur une moyenne de 10 simulations. Les différences statistiques significatives entre deux conditions, marquées par une étoile, sont calculées à partir d'un t-test de Welch à deux queues de niveau $\alpha = 0.05$ avec l'hypothèse nulle $\mathcal{H}_0 : \mu_1 = \mu_2$ (Colas et al., 2019b).

3.7.2 Effets de l'imagination sur la généralisation et l'exploration

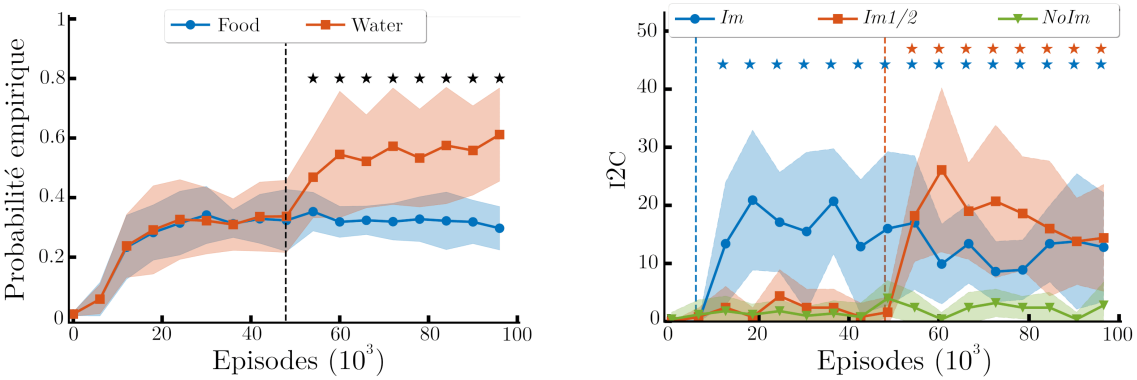
On compare les gains permis par l'imagination sur $\mathcal{D}^{\text{test}}$ dans trois conditions différentes : pas d'imagination *NoIm*, imagination dès le début (*Im*, après l'épisode $6 \cdot 10^3$), imagination à partir de la moitié de la simulation (*Im1/2*, après l'épisode $48 \cdot 10^3$. On observe en Figure 3.13 que les effets de l'imagination se manifestent dès que l'imagination est autorisée.



(a) Taux de succès $\overline{\texttt{SR}}$ moyen sur $\mathcal{D}^{\text{test}}$

(b) I2C calculée sur $\mathcal{D}^{\text{test}}$

FIGURE 3.13 – Effets de l'imagination sur la généralisation et l'exploration. La ligne verticale indique le moment où l'imagination est activée. Les étoiles indiquent une différence significative avec la condition sans imagination.



(a) Probabilité que l'agent apporte eau ou nourriture à une plante pour la faire grandir

(b) I2C calculée sur $\mathcal{D}^{\text{extra}}$

FIGURE 3.14 – Illustration du phénomène d'*adaptation comportementale*. Effets de l'imagination sur la généralisation et le comportement exploratoire de l'agent.

La Figure 3.13a montre que les gains en termes de généralisation sont d'autant plus visibles qu'elle commence tôt dans la simulation.

Adaptation comportementale

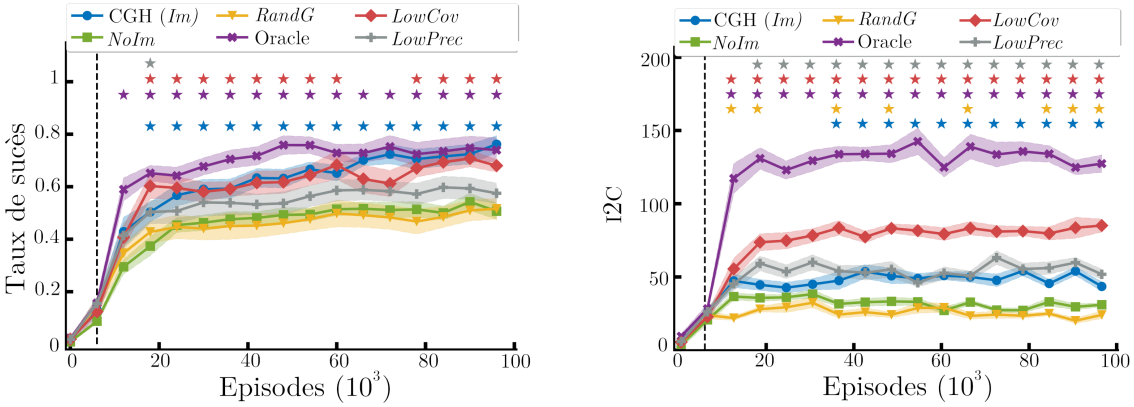
On utilise le terme d'*adaptation comportementale* pour caractériser le phénomène par lequel l'imagination modifie l'apprentissage et le comportement exploratoire de l'agent.

Généralisation n-shot – L'imagination modifie de manière très intéressante le comportement de l'agent à l'égard de certaines tâches. Une partie des descriptions réalisables dans l'environnement consiste à faire grandir un animal ou une plante. Celles relatives aux animaux sont toutes dans $\mathcal{D}^{\text{train}}$, tandis que celles relatives aux plantes sont dans $\mathcal{D}^{\text{test}}$

(généralisation de Type 5). La Figure 3.14a représente la probabilité empirique que l'agent apporte eau ou nourriture à une plante lorsqu'on lui demande de la faire grandir avant et après avoir autorisé l'agent à imaginer. Sans imagination, on observe que lorsque l'on demande à l'agent de faire grandir une plante, il apporte avec une probabilité similaire de l'eau ou de la nourriture à la plante. C'est déjà le signe d'une généralisation *zéro-shot*. En effet, l'agent transfère ce qu'il a appris sur les animaux vers les plantes, prouvant qu'il a compris la dynamique du prédicat *grow* et qu'il sait repérer les objets plantes. Plus intéressant encore, lorsque l'agent commence à imaginer, il se met à apporter bien plus souvent de l'eau que de la nourriture. Cela montre que la fonction de récompense a correctement généralisé le prédicat *grow* et qu'elle peut guider la politique d'action pour qu'elle réalise que l'eau est plus adaptée. On assiste alors à une généralisation *n-shot* de la politique d'action.

Comportement exploratoire – On observe aussi en Figure 3.14b, une modification du comportement exploratoire. On calcule I2C sur $\mathcal{D}^{\text{extra}}$ qui contient des objectifs non réalisables comme le fait d'apporter de l'eau ou de la nourriture à un meuble. On observe que cette situation se produit significativement plus souvent lorsqu'un agent est autorisé à explorer. Bien qu'irréalisables, observer l'agent tenter de réaliser ces tâches démontre une attitude exploratoire systématique similaire à celle que pourrait adopter un enfant (Bruner, 1983).

3.7.3 Les propriétés importantes du mécanisme d'imagination



(a) Taux de succès $\overline{\text{SR}}$ moyen sur $\mathcal{D}^{\text{test}}$ **(b)** I2C calculée sur $\mathcal{D}^{\text{test}}$

FIGURE 3.15 – Propriétés des variantes d'imagination. Les différences statistiques sont exprimées par rapport à la condition *NoIm*. Pour plus de lisibilité, on utilise l'erreur-type plutôt l'écart-type.

Dans cette partie on s'interroge sur les propriétés qui font de l'heuristique employée, un moyen efficace d'imaginer et de conduire aux résultats observés en termes de généralisation et d'exploration. On caractérise l'heuristique par deux propriétés :

1. *La couverture* : la proportion des descriptions de test qui sont imaginables ;
2. *La précision* : la proportion des tâches imaginables qui sont réalisables.

On compare alors notre CGH (*Construction Grammar Heuristic*) aux variantes suivantes :

— *Oracle* : Les phrases imaginées sont exactement celles de l'ensemble $\mathcal{D}^{\text{test}}$;

— *RandG* : Chaque phrase imaginée est remplacée par une description aléatoire générée à partir des mots de $\mathcal{D}^{\text{train}}$;

— *LowCov* : variante de CGH obtenue en filtrant la moitié des descriptions imaginées pour faire baisser la couverture et maintenir la précision. Le filtrage est aléatoire et donc différent pour chaque simulation ;

— *LowPrec* : variante de CGH obtenue en générant, pour chaque phrase imaginée, une description aléatoire afin de faire baisser la précision et maintenir la couverture.

Les propriétés de couverture et précision des différentes variantes, calculées à partir de $\mathcal{D}^{\text{train}}$, sont rapportées en Tableau 3.2.

Variante	Couverture	Précision
CGH	0.87	0.45
Oracle	1	1
LowCov	0.44	0.45
LowPre	0.87	0.30
RandG	≈0	≈0

Table 3.2 – Couverture et précision des différentes variantes de CGH

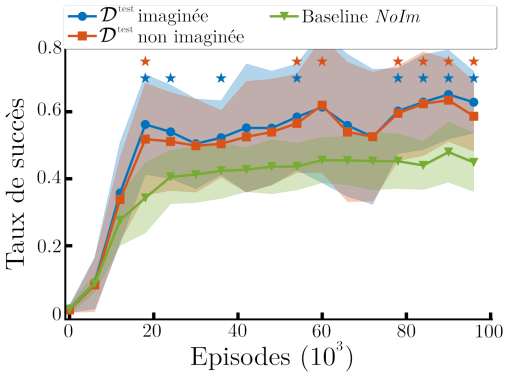


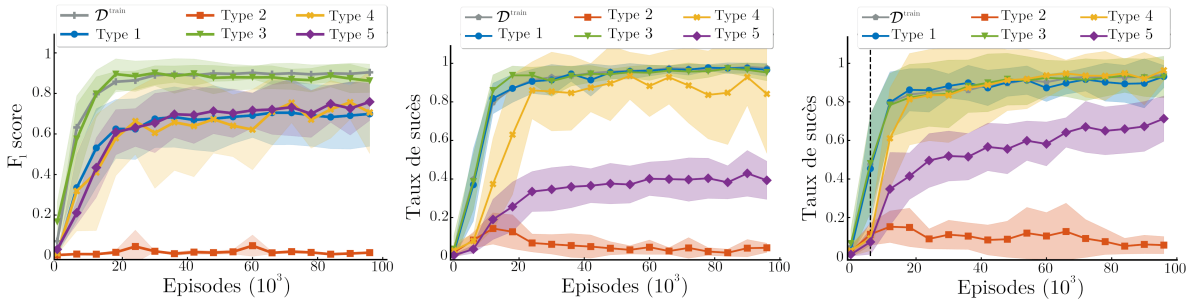
Figure 3.16 – Généralisation de *LowCov* pour les descriptions imaginées et non imaginées.

Couverture vs Précision – On observe en Figure 3.15a que les performances de CGH sont du même niveau que l'*Oracle* avec cependant un petit délai, celles de *LowCov* s'approchent de CGH et sont en tout significativement meilleures que *NoIm*. Cependant, *LowPrec* et *RandG* ne montrent pas d'amélioration significative par rapport à *NoIm*. Cela semble indiquer que la précision avec laquelle l'heuristique produit des descriptions est la propriété clé. On observe en Figure 3.15b que le gain en exploration est préservé en diminuant la précision et, de manière surprenante, amplifiée par une baisse de la

couverture. Les performances de *RandG*, similaires à celles de *NoIm*, montrent que les effets observés par l'imagination ne sont pas des artefacts et que c'est l'utilisation du langage comme outil de représentation qui conduit aux résultats observés.

Généralisation *zéro-shot* versus *n-shot* – Le fait que *LowCov* montre des performances quasi-similaires à CGH alors que cette variante voit moitié moins de descriptions de $\mathcal{D}^{\text{test}}$ semble indiquer que la performance sur les descriptions non imaginées peut bénéficier de l'entraînement sur celles imaginées. On observe en effet en Figure 3.16 que la performance de *LowCov* est équivalente sur les descriptions imaginées et non imaginées tout en étant significativement meilleure que la condition *NoIm*. Cela indique que la généralisation *n-shot* sur certaines descriptions permet une généralisation *zéro-shot* sur des descriptions non imaginées. L'imagination permet donc de progresser sur l'ensemble des descriptions de $\mathcal{D}^{\text{test}}$ et pas uniquement sur celles imaginées.

3.7.4 Généralisation sur les 5 types de descriptions tests



(a) Fonction de récompense **(b)** Politique d'action sans imagination **(c)** Politique d'action avec imagination

FIGURE 3.17 – Généralisation par types de test pour la fonction de récompense et la politique. Les deux graphiques de gauche permettent de mesurer la généralisation *zéro-shot* et celui de droite la généralisation *n-shot*

En Partie 3.6.2, nous avons détaillé les 5 types de descriptions tests permettant d'évaluer différentes facettes des propriétés de généralisation de l'agent. Dans cette partie, on évalue les performances d'IMAGINE sur chacun de ces types. On a défini en Partie 3.7.1 deux formes de généralisation systématique : *zéro-shot* et *n-shot*. La politique d'action et la fonction de récompense interne sont capables de la première forme lorsqu'elles réussissent à interpréter correctement les descriptions de $\mathcal{D}^{\text{test}}$ du premier coup (sans entraînement spécifique, i.e. sans imagination pour la politique). Lorsque la fonction de récompense est en mesure de généraliser en *zéro-shot* pour certaines descriptions, l'agent peut alors entraîner la politique d'action en autonomie, c'est-à-dire sans le partenaire social. On dit alors qu'elle généralise en *n-shot*.

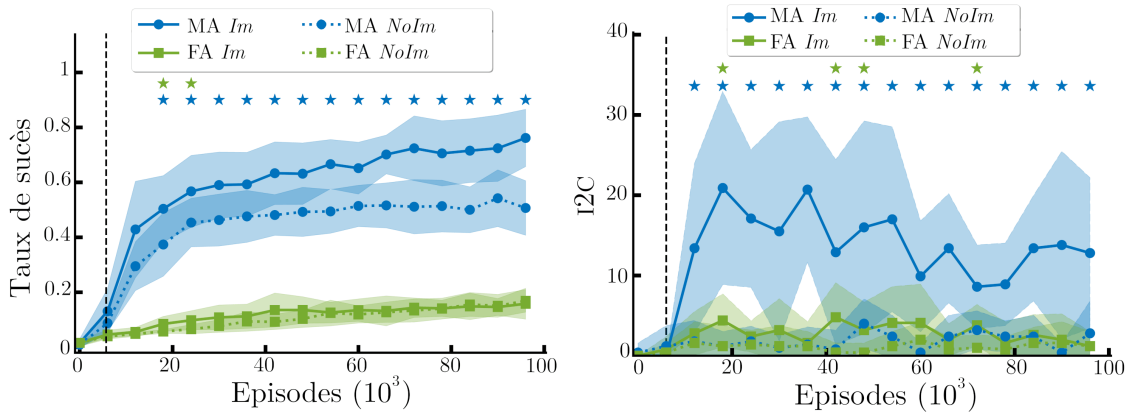
Pour différencier les performances en *zéro-shot* et *n-shot* de la politique d'action, on mesure $\overline{\texttt{SR}}$ dans les conditions *NoIm* et *Im*. Pour évaluer la généralisation de la fonction de récompense, on collecte, à l'aide d'une politique d'action déjà entraînée, un ensemble de données sur $\mathcal{D}^{\text{test}}$ et on monitore le F_1 score au cours des simulations. Cette méthode d'évaluation est différente de celle employée pour LE2 où l'on testait uniquement la généralisation sur de nouveaux états *pendant* le co-entraînement.

Les résultats sont présentés en Figure 3.17. On observe en Figure 3.17a que $\mathcal{R}$ parvient à correctement généraliser en *zéro-shot* sur les Types 1, 3, 4 et 5 mais pas sur le Type 2. Cela montre un bon niveau de généralisation systématique nécessaire à ce que l'imagination produise ses effets. De même, la politique d'action généralise facilement en zéro-shot (Figure 3.17b) sur les Types 1, 3, 4, à moitié sur le Type 5 et pas du tout sur le Type 2. En *n-shot* (Figure 3.17c), la généralisation est bien meilleure sur le Type 5 et montre l'effet d'adaptation comportementale décrit plus haut.

Les Types 1, 3 et 4 sont de difficulté similaire et consistent à associer deux notions vues séparément (attribut-objet, prédicat-objet/catégorie). La généralisation de Type 5 est plus difficile, car elle requiert de s'adapter à un changement de dynamique. Sans *fine-tuning* et sans imagination, il n'y a pas de raison que la politique d'action réalise que la nourriture ne fait pas grandir une plante. Enfin, le Type 2 résiste à la fois à la fonction de récompense et à la politique d'action. Pour résoudre le Type 2, l'agent doit identifier l'objet *flower* soit par sa couleur quand elle est mentionnée, soit en le déduisant des autres objets connus qu'il perçoit (*grasp any flower*). Cela semble indiquer que l'agent favorise le nom de l'objet pour l'identifier plus que la couleur. D'un point de vue statistique, ça n'est pas étonnant, il existe 32 objets pour seulement 3 couleurs, il est donc plus efficace de privilégier l'objet que la couleur. Cette dernière peut en revanche servir à départager deux objets ou à identifier l'objet lorsque son type ou sa catégorie n'est pas mentionné (*thing*). La probabilité pour que deux objets identiques de couleur différente soient dans la même scène est cependant relativement faible et cela peut expliquer que l'agent n'ait pas développé cette compétence.

3.7.5 Pourquoi l'architecture modulaire est essentielle

Dans cette section, on évalue les effets des choix architecturaux et notamment celui des DEEP SET (Zaheer et al., 2017). On compare ainsi notre architecture modulaire à une architecture dite « plate » où l'état de l'environnement est une concaténation des vecteurs d'état des objets et de l'agent $s_F = [s_{o_1}, s_{o_2}, s_{o_3}, s_a]$. Chaque objet n'est pas traité séparément par les réseaux $\text{NN}^{\mathcal{R}}$ et NN^{π}, mais en entier directement : $\text{NN}\left(s_F \odot A^{att}(g)\right)$ On compare les effets des architectures *plate* (FA) et modulaire (MA) sur la fonction de récompense et la politique.



(a) Taux de succès $\overline{\texttt{SR}}$ moyen sur $\mathcal{D}^{\text{test}}$

(b) I2C calculée sur $\mathcal{D}^{\text{extra}}$

FIGURE 3.18 – Comparaison des architectures MA et FA. Les étoiles indiquent une différence significative entre CGH et la condition *NoIm* correspondante.

Fonction de récompense : FA vs MA– L'évaluation de la fonction de récompense est réalisée de manière supervisée et indépendamment de la politique d'action à partir d'un ensemble de $50{\cdot}10^3$ trajectoires collectées à l'aide d'une politique pré-entraînée. Les fonctions de récompense sont entraînées sur les états finaux et testées sur un sous-ensemble des états de ces trajectoires. On observe dans le Tableau 3.3 que l'architecture modulaire démontre de bien meilleures capacités de généralisation que la *plate* tant sur $\mathcal{D}^{\text{train}}$ que $\mathcal{D}^{\text{test}}$. Il est aussi peu probable que la fonction de récompense FA permette d'entraîner correctement une politique d'action.

		MA	FA
$\mathcal{R}$	$F1_{\text{train}}$	0.98 ± 0.02	0.60 ± 0.10
	$F1_{\text{test}}$	0.64 ± 0.22	0.22 ± 0.05
π	$\overline{\texttt{SR}}_{\text{train}}$	0.95 ± 0.05	0.40 ± 0.13
	$\overline{\texttt{SR}}_{\text{test}}$	0.76 ± 0.10	0.16 ± 0.06

TABLE 3.3 – Comparaison des architectures *plate* et modulaire.

Politique d'action : FA vs MA– Pour comparer les effets des deux architectures sur la politique d'action, on emploie la même architecture MA pour la fonction de récompense. Le Tableau 3.3 montre que la politique d'action FA ne généralise que très peu sur les descriptions test. La Figure 3.18 permet de comparer les effets de l'imagination sur la généralisation et l'exploration. On observe très nettement que l'architecture *plate* ne bénéficie de l'imagination dans aucune des deux métriques. Cela démontre l'absence de généralisation systématique de cette architecture et confirme que le choix d'une architecture modulaire est cruciale.

3.7.6 Un partenaire social plus réaliste

Le partenaire social permet de simuler l'interaction entre un agent et un humain qui superviserait son apprentissage. Pour mieux contrôler la simulation, nous nous sommes appuyés sur les hypothèses détaillées en Partie 3.4.2 : précision, exhaustivité, présence. Avec l'idée qu'il serait intéressant d'adapter cette forme d'interaction à un environnement réel, on évalue la robustesse de l'agent si l'on assouplit les hypothèses de présence et d'exhaustivité. On teste les conditions suivantes :

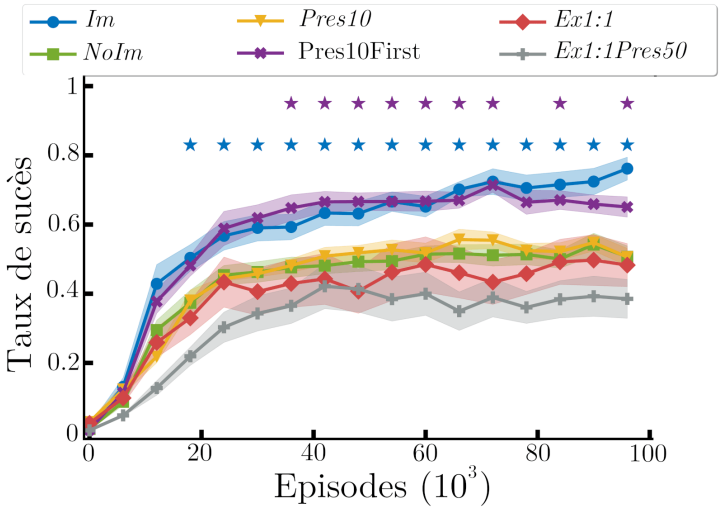


FIGURE 3.19 – Influence du partenaire social sur la généralisation

— ***Pres10*** : le partenaire social est présent 10% des épisodes ;

— ***Pres10First*** : le partenaire social est toujours présent sur le premier dixième de la simulation et absent ensuite ;

— ***Ex1:1*** : le partenaire social ne fournit qu'une description positive et une négative par épisode ;

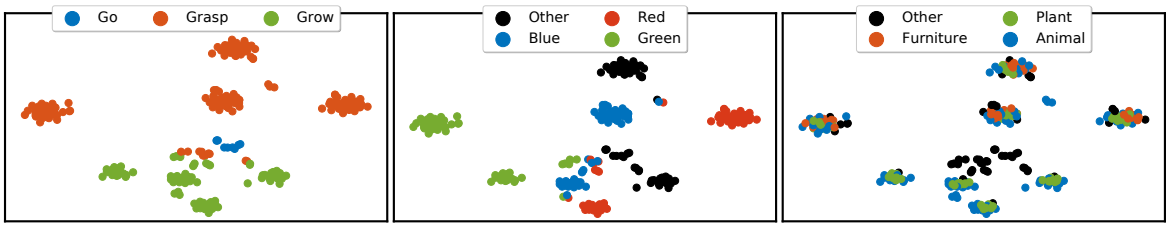
— ***Ex1:1Pres50*** : le partenaire social ne fournit qu'une description positive et une négative dans 50% des épisodes.

On mesure le taux de succès moyen de ces différentes variantes et on les compare avec deux conditions contrôles : ***Im*** et ***NoIm***. La Figure 3.19 montre que l'imagination permet de compenser l'absence du partenaire social : *Pres10* atteint un niveau de performance comparable à *NoIm*. Cependant, la même quantité de supervision est bien plus efficace lorsqu'elle a lieu au début de l'apprentissage (*Pres10* vs *Pres10First*) et *Pres10First* atteint un niveau quasi-similaire à *Im*. Bien qu'ici la stratégie de supervision du partenaire social soit extrêmement simple, on peut dresser le parallèle avec la théorie de l'échafaudage de Bruner (voir Partie 1.5.5) qui décrit que l'enfant a besoin de moins en mois de supervision de la part de ses parents/enseignants. L'agent démontre aussi une certaine robustesse lorsque c'est l'hypothèse d'exhaustivité qui est relâchée. En effet, l'imagination compense quasiment cette perte de supervision.

3.8 Discussion

3.8.1 Visualisation des représentations des tâches

Pour visualiser l'apprentissage du modèle de langage, on effectue une transformation t-SNE (Van Der Maaten et Hinton, 2008) à partir des représentations apprises des descriptions que l'on représente en Figure 3.20. On observe que les descriptions sont organisées par prédicats et par couleurs mais pas par catégorie ou nom d'objets.



(a) aaa **(b)** aaa **(c)** aaa

Figure 3.20 – Projection t-SNE de l'espace des tâches. La même projection t-SNE est représentée avec différents codes couleurs pour représenter les tâches en fonction de leur (a) prédicat, (b) de l'éventuel couleur ou (c) de la catégorie d'objets.

3.8.2 Mécanisme attentionnel

Pour filtrer l'information pertinente de l'état de chaque objet, l'architecture modulaire utilise un mécanisme de porte attentionnelle (*gated-attention*, Chaplot et al., 2018). Un masque attentionnel est appris par la fonction de récompense $\boldsymbol{\alpha}^g$ et la politique $\boldsymbol{\beta}^g$. Ce vecteur ne dépend pas de l'objet mais uniquement de la description de la tâche g. Le masque attentionnel filtre l'information du vecteur d'état de chaque objet de la scène par un produit de Hadamard (terme-à-terme). Cela permet au modèle de *sélectionner* l'information pertinente à passer au réseau suivant.

Ce mécanisme présente l'avantage d'être très facilement interprétable. On peut en effet visualiser ces masques en Figure 3.21 calculés à la fin de l'entraînement pour certaines tâches. On souligne les deux points suivants :

Sélection optimale de l'information : Le mécanisme attentionnel permet à l'agent de sélectionner les informations minimales nécessaires. On observe ainsi que lorsqu'aucune couleur n'apparaît dans la description, le masque ne prête pas attention aux informations de couleurs. De même, à première vue, les vecteurs $\boldsymbol{\alpha}^g$ et $\boldsymbol{\beta}^g$ se ressemblent beaucoup. On peut noter cependant certaines différences intéressantes. Pour la description *Go top*, la fonction de récompense ne se concentre que sur la position de l'agent alors que la politique

va prendre en compte la position et le delta entre la position actuelle et celle au début de l'épisode. Aussi, pour les descriptions avec le prédicat *grow*, la position de l'objet ne va être représentée que pour la politique d'action.

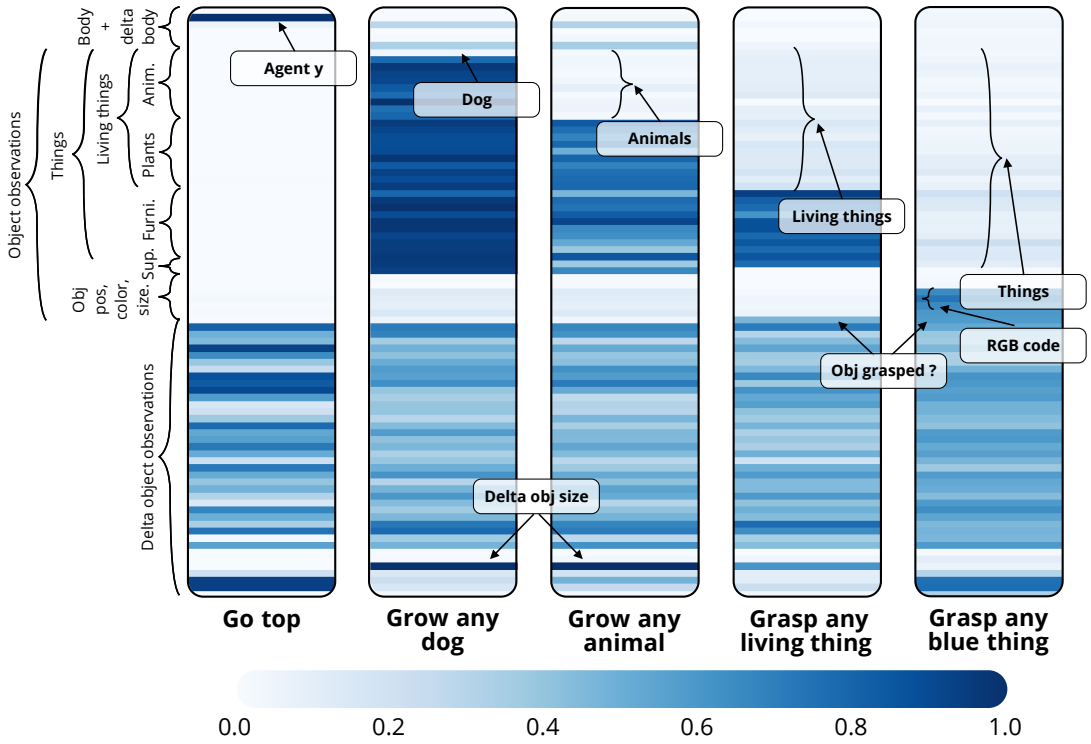
Rôle de l'attention dans la généralisation systématique – On peut observer que le masque attentionnel met en avant les objets qui ne sont pas concernés par la description. Cela semble contre-intuitif. Pourtant, cela permet tout autant de filtrer les objets pertinents. En effet, le produit de Hadamard entre le masque et le vecteur d'état va rester proche de zéro pour l'objet concerné par la description et proche de 1 pour les autres. Cela permet bien au modèle de sélectionner l'objet pertinent. C'est l'utilisation d'un encodage *one-hot* qui crée cet effet étrange.

On peut aller plus loin dans l'interprétation et montrer qu'en réalité cette méthode est la plus efficace et favorise la généralisation systématique. En effet, en adoptant ce mécanisme, le modèle fait la chose suivante : si l'objet correspond à celui de la description, il a trouvé le bon objet. Cette information est directement portée par le fait que toutes les valeurs décrivant la nature de l'objet sont proches de 0. L'agent peut alors raisonner sur l'objet indépendamment de sa nature. À l'inverse si l'objet ne correspond pas à celui de la description, il conserve l'information qu'il est singulier au regard la tâche actuelle et qu'il doit être traité de manière particulière. Cette information est portée par le fait qu'une des valeurs décrivant la nature de l'objet est non nulle et proche de 1. Il est alors toujours pertinent pour l'agent de conserver l'information au sujet de la nature de cet objet. Si le masque attentionnel était inversé, cela conduirait l'agent à traiter le bon objet comme s'il était particulier alors que le même traitement pourrait s'appliquer indépendamment de sa nature.

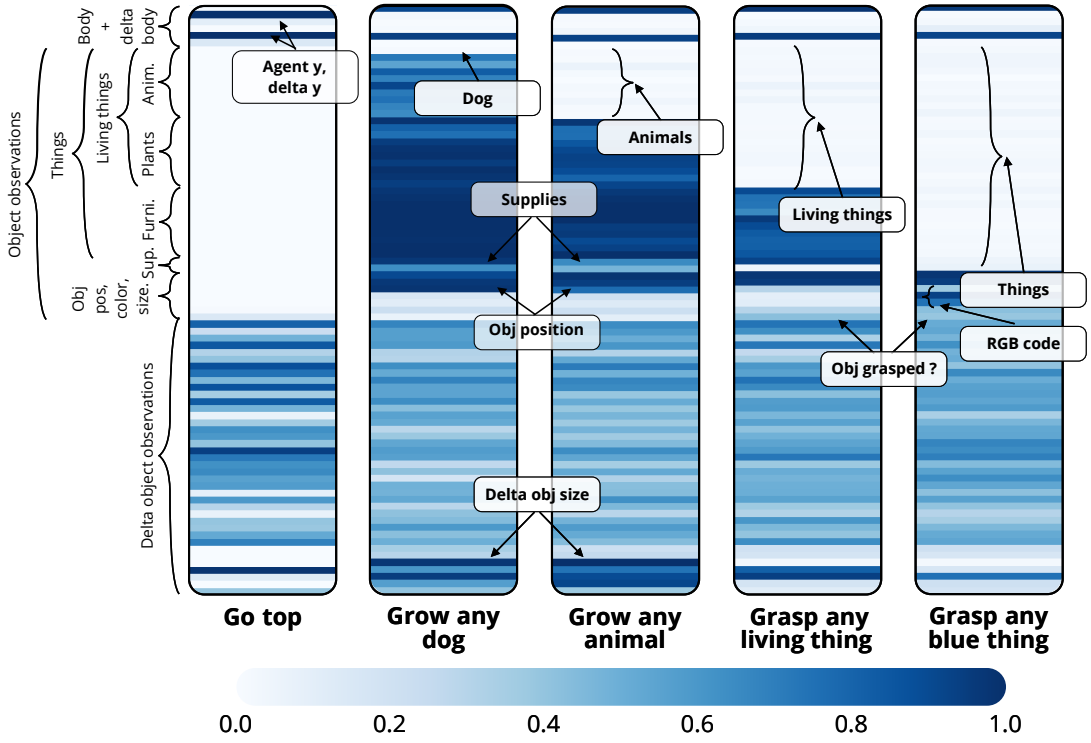
Le fait de traiter le bon objet (quasi-)indépendamment de l'objet dont il s'agit est donc un facteur de généralisation systématique. Les simulations que nous avons réalisées sans mécanisme attentionnel ont d'ailleurs conduit à des performances médiocres de l'agent.

3.8.3 Types de généralisation et ZPD

On peut donner une interprétation de la réussite de l'agent sur les différents types de descriptions test proposées en Partie 3.6.2 et évaluées en Partie 3.7.4 en termes de Zone Proximale de Développement (voir Partie 1.5.5). En effet, pour un enfant, Vygotsky (1978) définit la ZPD comme une mesure du potentiel d'apprentissage de l'enfant à un instant donné. Transposé pour l'agent, on la définit comme la zone dans laquelle il est en mesure d'apprendre en autonomie. Cela correspond en fait aux tâches sur lesquelles l'agent généralise en *n-shot*. D'après les résultats obtenus en Partie 3.7.4, une bonne approximation de cette zone est directement obtenue en déterminant les tâches sur lesquelles la fonction



(a) Fonction de récompense $\boldsymbol{\alpha}^g$



(b) Politique d'action $\boldsymbol{\beta}^g$

Figure 3.21 – Représentation du masque attentionnel.

de récompense est capable de généraliser en *zéro-shot* mais pas la politique d'action. En effet, les tâches pour lesquelles la fonction de récompense généralise mais pas la politique d'action sont celles pour lesquelles la politique d'action peut être entraînée sans supervision du partenaire social. Celles sur lesquelles la politique d'action généralise déjà n'appartiennent pas à la ZPD puisque même si l'agent ne les a jamais rencontrées, il les maîtrise déjà. Elles ne représentent donc pas de potentiel d'apprentissage. Ainsi la ZPD de l'agent correspond peu ou prou à la différence de généralisation entre la fonction de récompense et la politique d'action.

Cela permet d'envisager une nouvelle forme d'apprentissage en interaction plus dirigée. Le partenaire social pourrait orienter le curriculum d'apprentissage de l'agent afin de le maintenir dans la ZPD. On quitte alors le domaine d'apprentissage autonome où c'est l'agent lui-même qui se fixe ses propres tâches pour laisser le partenaire social guider l'agent.

3.8.4 Vers un partenaire social plus humain

Vers un langage plus réaliste – Les défis pour pouvoir remplacer le partenaire social par un véritable humain sont encore nombreux. La grammaire employée dans *Playground* a volontairement été choisie simple afin de pouvoir étudier les effets de l'imagination et de l'architecture modulaire sur la généralisation systématique. Une première direction consiste à complexifier la grammaire employée pour se rapprocher d'un langage vraiment naturel. L'utilisation de modèle de langage pré-entraîné serait intéressante pour donner à l'agent une notion de la sémantique des mots mais aussi pour lui permettre de gérer un vocabulaire plus varié. Un tel modèle pourra être fine-tuné de manière assez similaire à la manière dont le modèle de langage est appris dans IMAGINE. On pourrait aussi envisager d'utiliser AIDME dont le rôle est de convertir une requête de l'utilisateur exprimée de manière générale vers une construction connue de l'agent. Les performances impressionnantes de GPT-3 (Radford et al., 2019) pourraient aussi permettre de réaliser ce genre d'application.

Vers une supervision plus efficace – La stratégie de supervision employée est un frein à une supervision humaine. Il est en effet difficilement concevable qu'un humain observe pendant des milliers d'épisodes l'agent réaliser ces tâches et lui fournissent à chaque fois toutes les bonnes descriptions. Il faut donc penser l'interaction entre l'humain et l'agent de façon à ce qu'elle soit la moins contraignante possible tout en garantissant une trajectoire d'apprentissage efficace. L'étude de la Partie 3.7.6 a montré que la stratégie de supervision employée était largement sous-optimale. L'agent fait quasiment aussi bien avec seulement 10% de feedback à condition de concentrer ce dernier au début de la simulation. Cette observation est cohérente avec la théorie de l'échafaudage de Bruner

(Wood et al., 1976). Diminuer la quantité de feedback nécessaire, c'est alléger la charge la supervision et transférer la responsabilité de l'apprentissage sur l'agent.

L'enjeu pour entraîner la fonction de récompense est de collecter des exemples négatifs. Nous avons pris le parti de fournir à l'agent toutes les descriptions positives afin qu'il puisse en déduire les descriptions négatives. Confronté à la même problématique, Bahdanau et al. (2019) utilise un ensemble de données collectées par un oracle pour fournir les exemples positifs et exploite les trajectoires collectées par l'agent comme exemples négatifs. Cependant au fur et à mesure de l'apprentissage, il est de plus en plus probable que ces dernières soient des exemples positifs et cela conduit la fonction de récompense à s'entraîner sur un ensemble de données contenant des faux négatifs. Pour y remédier, Bahdanau et al. (2019) emploie la fonction de récompense pour contrôler si les exemples collectés sont vraiment négatifs, selon une approche assimilable à de l'apprentissage semi-supervisé. On peut envisager de combiner nos deux approches : la supervision du partenaire social étant cruciale au début de l'apprentissage, le partenaire social fournirait l'ensemble des descriptions positives pendant une première phrase de l'apprentissage de l'agent. Au fur et à mesure, il diminue la quantité de descriptions fournies et l'agent peut compter sur la fonction de récompense pour bootstraper des exemples négatifs à partir des trajectoires collectées. Enfin, tout comme pour AIDME, il est envisageable de mutualiser l'apprentissage d'un tel agent à partir de plusieurs superviseurs.

Optimiser la supervision – De manière plus générale, il serait intéressant de travailler sur l'optimisation de la supervision : fournir la supervision optimale au bon moment. On se place alors du point de vue de l'agent et on souhaite faire en sorte que le gain marginal obtenu par chaque feedback (les descriptions) soit maximal. Si la supervision de l'agent est aussi importante au début de l'apprentissage, c'est certainement parce que c'est à ce moment qu'il découvre les tâches réalisables dans l'environnement. On fait l'hypothèse (qui reste encore à démontrer) que c'est la raison pour laquelle la même supervision obtenue sur toute la durée de l'apprentissage ou au début conduit à des résultats si différents (voir Partie 3.7.6). Ainsi, une première stratégie à expérimenter serait de concentrer la supervision des tâches au moment où elles sont découvertes. Cela peut être fait dans le cadre des agents autonomes et requiert simplement d'adapter la stratégie du partenaire social, notamment en retenant le moment où l'agent découvre une tâche pour la première fois. De manière indirecte, cela aura pour conséquence de réduire la quantité de supervision nécessaire et donc la charge d'un éventuel partenaire social humain.

3.9 Conclusion

Dans ce chapitre, notre objectif était de concevoir des agents capables d'apprendre à effectuer des tâches exprimées en langage naturel tout en disposant de capacité de généralisation procédurale. Dans le contexte des agents de Deep RL exploitant le langage, on a traduit cette notion par celle de généralisation systématique (Fodor et Pylyshyn, 1988). On s'est placé dans le cadre d'un système apprenant par interaction avec un partenaire social fournissant une supervision langagière. Les résultats obtenus avec LE2 et IMAGINE montrent les propriétés de généralisation recherchées sont permises par certaines propriétés de l'agent et de l'environnement :

— L'environnement doit posséder une certaine structure dans sa dynamique qui soit propice à cette forme de généralisation. Cette structure doit être reflétée dans la manière dont l'espace d'état est encodé.

— L'apprentissage de la fonction de récompense permet d'apprendre à relier la sémantique du langage à des observations de l'environnement. Cette dernière suffit à entraîner la politique d'action et peut même être employé pour superviser l'apprentissage de tâches sur lesquels la fonction de récompense n'aurait pas été entraînée.

— Apprendre à représenter les tâches permet de disposer de représentations qui sont généralisables à de nouvelles tâches. L'utilisation du langage comme outil d'abstraction pour formuler les tâches permet d'apprendre des représentations qui généralisent en dehors de la distribution des formulations rencontrées. On peut alors assister à la généralisation systématique de l'agent.

— L'utilisation d'une architecture qui reflète la structure de l'environnement et des tâches est aussi essentielle pour observer cette forme de généralisation (Partie 3.7.5). Ces résultats sont alignés avec ceux de Kuo et al. (2020).

Les résultats en termes de généralisation systématique obtenue démontrent le potentiel et l'intérêt des approches développementales inspirées par les Sciences Cognitives, et cela constitue une contribution majeure de ce chapitre. En effet, la technique d'exploration imaginative est directement inspirée des travaux de Vygotsky présentés en Partie 1.5.4 qui insiste sur le rôle du langage pas seulement pour communiquer mais comme un outil. L'apprentissage du modèle de langage à partir des changements d'état de l'environnement est inspirée des travaux de Nelson et Mandler sur la conceptualisation fonctionnelle (Partie 1.5.3). Ces résultats peuvent aussi être interprétés à l'aune des outils de la ZPD développée par Vygotsky (Partie 3.8.3).

Ce chapitre nous a montré qu'il est tout à fait possible de concevoir des agents qui apprennent à lier les formes langagières sur l'environnement qui les entoure, de généraliser et de transférer les compétences entre les objets. On cherche maintenant à

exploiter ces résultats dans le cas des assistants apprenants. Ces derniers évoluent dans des environnements numériques aux difficultés toutes spécifiques.

Code et vidéo. Tout comme pour AIDME, le code a été mis à disposition en open source : `Environnement Playground`[1] et `Code d'IMAGINE`[2]. À des fins de pédagogie, un site web avec des vidéos de démonstration est aussi disponible : https://sites.google.com/view/imagine-drl

[1] https://github.com/flowersteam/playground_env
[2] https://github.com/flowersteam/Imagine

Chapitre 4

Agent apprenant dans un environnement domotique réaliste

It's supposed to be automatic, but actually you have to push this button.

— John Brunner
Stand on Zanzibar, 1968

Sommaire

4.1 Introduction et contexte

4.1.1 Environnement physique vs. numérique

L'agent IMAGINE détaillé dans le chapitre précédent possède des propriétés de généralisation systématique qui peuvent être assimilées à une forme de généralisation procédurale. Dans ce chapitre, on souhaite ainsi adapter ces résultats encourageants à un environnement domotique réaliste dans lequel un assistant numérique peut apprendre à utiliser un ensemble d'objets connectés. Comme brièvement évoqué, les environnements numériques posent des défis particuliers. En effet, pour IMAGINE, la généralisation systématique est notamment facilitée par la structure inhérente aux espaces d'états et d'actions (Hill et al., 2019 ; Ruis et al., 2020) et au choix de l'architecture neuronale de l'agent qui reflète cette structure (Kuo et al., 2020). Nous avons ainsi employé pour IMAGINE, une représentation de l'état des objets dite *single-slot object file* (Green et Quilty-Dunn, 2017) et des DEEP SET (Zaheer et al., 2017).

Dans les environnements de robotique simulée, comme *Playground* ou *ArmToolsToys*, la représentation de l'état de l'environnement et des actions réalisables est relativement naturelle. L'état est classiquement représenté par la position des objets, celle du robot ainsi que de leurs caractéristiques pertinentes (couleurs, taille, position du *gripper*, etc.). L'action, quant à elle, est représentée par les mouvements que le robot peut effectuer (translation, vitesse des joints, action de préhension, etc.). À l'inverse, dans un environnement numérique, la représentation des objets n'est pas évidente. Dans un environnement physique, les actions réalisables sont directement déterminées par les capacités physiques du robot (déplacement, préhension, etc.). Dans un environnement numérique, ce sont les objets numériques (site Web, objet connecté, API (*Application Programming Interface*), etc.) qui déterminent les actions et chacun possède sa propre interface.

Dans un environnement domotique, les objets connectés sont généralement contrôlables via une API proposée par le fabricant et qui permet d'en observer l'état et d'exécuter des actions. Pour un utilisateur non spécialiste, le fabricant met souvent à disposition une application (Web ou smartphone) ou rend son appareil compatible avec les assistants vocaux comme Amazon Alexa, Siri ou Google Assistant. Les possibilités d'interaction sont donc définies à l'avance et cela limite l'interopérabilité entre les objets de différents fabricants. Il est par exemple difficile pour un assistant vocal d'exécuter des actions comme : « Règle tous les radiateurs sur la température de celui du salon. ». Cela nécessite pourtant simplement de récupérer une information sur l'un des objets et de la propager aux autres.

4.1.2 Les plateformes domotiques

Cette absence d'interopérabilité entre des objets et services de constructeurs différents est rendue difficile par le fait qu'aucun standard d'API n'a encore véritablement émergé. Chacun dispose ainsi de sa propre API et reflète directement les fonctionnalités proposées par l'objet. Le fait de disposer d'un standard domotique pour développer un agent apprenant aurait été idéal pour le développement d'un agent apprenant notamment pour disposer d'un espace d'action bien formé.

Pour y remédier, on se tourne vers des plateformes conçues pour agréger un ensemble d'objets connectés sur un réseau et les contrôler à travers une interface commune. Ces plateformes, souvent *open source* et soutenues par une communauté d'utilisateurs, proposent ainsi une certaine interopérabilité. Pour rendre un objet connecté compatible avec la plateforme, un adaptateur est développé par la communauté afin de correspondre aux standards de la plateforme. En s'appuyant sur l'interface proposée par ces plateformes comme standard d'interaction avec les objets, on peut alors modéliser correctement un espace d'action utilisable par un agent de Deep RL. De plus, cela permet d'exploiter la structure de ces interfaces comme support nécessaire à la généralisation systématique que l'on souhaite obtenir (Hill et al., 2019 ; Kuo et al., 2020).

Nous avons choisi de travailler avec la plateforme OpenHAB [1], qui présente l'avantage d'être *open source*, de disposer d'une communauté active et d'être utilisée par Cloud Temple. Notre objectif est donc d'exploiter l'API d'OpenHAB pour développer un agent capable d'apprendre à contrôler différents objets connectés en fonction d'instructions de l'utilisateur via cette plateforme. Nous nous positionnons dans un environnement réaliste qui doit nous permettre de déployer l'agent en conditions réelles.

4.1.3 Organisation du chapitre

Dans ce chapitre, on revient sur les modes d'interaction possibles entre un agent de Deep RL et une interface numérique (Partie 4.2). On décrit ensuite sommairement la plateforme OpenHAB, ainsi que le simulateur que nous avons développé, qui permet d'émuler une interface OpenHAB et un ensemble d'objets connectés (Partie 4.3). Ce simulateur est unique dans le domaine, il est entièrement paramétrable et doit permettre d'investiguer comment des agents peuvent apprendre à manipuler une API sous la forme d'un espace d'action semi-structuré. À partir de ce simulateur, on définit plus précisément l'environnement dans lequel seront effectuées les expérimentations (Partie 4.4). Nous proposons enfin un agent, ILEAD, adapté à l'interface d'OpenHAB (Partie 4.5) et nous présentons une série de résultats préliminaires que nous avons obtenus (Partie 4.6). Nous

[1] https://www.openhab.org/

terminons en proposant une architecture permettant d'intégrer AIDME et ILEAD avec l'agent JARVIS (Delgrange, 2018) déjà développé par Cloud Temple (Partie 4.8).

4.2 Interaction avec un objet numérique dans un monde ouvert

Apprendre à interagir avec des services numériques est un problème difficile pour les systèmes artificiels (Shi et al., 2017) et la question de l'interface est importante puisqu'elle conditionne la perception de l'agent et sa capacité d'action. Cet enjeu dépasse cependant celui des agents artificiels apprenants. Pour un utilisateur humain, l'écran et l'interface graphique permettent de standardiser l'observation de l'état numérique d'un ordinateur. Le clavier et la souris sont les effecteurs de l'humain pour agir sur cet environnement. Les modes d'interactions entre un humain et un système numérique font l'objet de recherches pour la rendre aussi fluide et intuitive que possible (Lee et al., 2012). L'utilisation d'une interface graphique pour un humain permet, par exemple, de simuler un espace physique dans lequel l'utilisateur a déjà des repères. La souris représente sa main et le clavier son stylo. Les termes employés pour décrire cet espace numérique sont d'ailleurs évocateurs : le bureau, les dossiers, déplacer un fichier, etc. Le parallèle avec l'espace physique rend intuitif l'interaction avec ces systèmes, cette idée est largement exploitée dans tous les systèmes tactiles de type smartphone, tablette, etc. Pour un agent numérique apprenant, on peut distinguer deux formes d'interaction : par interface graphique ou par API. Notons qu'on s'intéresse ici aux environnements dit ouverts, ou *open-ended*, c'est-à-dire qui sont par définition multi-tâches et proposent un espace d'interaction très riche (Stanley et al., 2019).

4.2.1 Interaction par interface graphique

Les environnements *World of Bits* ont été ainsi proposés pour entraîner des agents artificiels à effectuer des tâches sur des environnements Web simulés ou réalistes (Shi et al., 2017 ; Liu et al., 2018 ; Gur et al., 2019 ; Jia et al., 2019 ; Merkle et Philipp, 2019). L'agent contrôle la souris et un pseudo-clavier et observe l'écran ainsi que le DOM (*Document Object Model*) de la page Web. (Le DOM d'une page Web est une représentation standard sous forme d'arbre d'une page HTML.) Le clavier est simplifié et n'est pas constitué de lettres, mais directement de mots ainsi que de raccourcis. Cela permet d'éviter les combinaisons de touches malencontreuses, mais aussi de simplifier la tâche de l'agent qui n'a pas besoin d'apprendre à taper un mot et de lui permettre d'utiliser les raccourcis comme le copier-coller. Y. Li et al. (2020) propose un environnement similaire

pour un écran de smartphone qui s'appuie sur l'image visible par l'utilisateur et une structure similaire au DOM d'une page Web.

La principale limite de cette approche est que la page d'un site Web change régulièrement et il n'est pas évident pour un agent de s'adapter efficacement à ces évolutions rapides. C'est pourquoi les approches précédentes s'appuient notamment sur le DOM qui permettent d'ancrer l'apprentissage sur la structure de la page. De manière générale, la capacité à interpréter automatiquement l'écran d'une interface graphique est un sujet de recherche actif (Mazumder et Riva, 2020 ; T. J.-J. Li et al., 2021).

4.2.2 Interaction par API

Une autre manière d'interagir avec un service numérique est d'utiliser directement l'API exposée par ce dernier. Le terme d'API est relativement général et désigne un ensemble de fonctions informatiques qui permettent de *requêter* des informations ou d'exécuter des commandes sur le système. Les développeurs s'en servent fréquemment pour interagir avec des systèmes informatiques, mais nous avons trouvé relativement peu d'exemples de systèmes artificiels qui apprenne à utiliser une API pour interagir avec un environnement de services.

Les agents conversationnels orientés tâches sont capables de formuler des requêtes à une API pour récupérer une information en vue de répondre aux demandes des utilisateurs. Cependant, il s'agit en général de formuler une requête SQL sur une base de données donc dans un environnement très contraint (Z. Zhang et al., 2020). Pour des agents de Deep RL, on est confronté aux problématiques suivantes : une observabilité réduite et un espace d'action mal défini.

Observabilité d'une API – En effet, l'API d'un service n'offre généralement pas d'état observable de l'environnement. Pour avoir accès aux ressources mises à disposition par l'API, il est souvent nécessaire de faire des requêtes, par exemple une requête `GET` pour les API REST (*Representational state transfer*) souvent utilisé pour interagir avec les services Web. Dans le cas d'OpenHAB que nous utiliserons par la suite, la plateforme présente l'avantage de maintenir un état courant des différents objets qui permet de disposer d'un état observable à tout moment.

Définition d'un espace d'action – Les actions exécutables sur une API sont simplement un ensemble de fonctions appelables avec d'éventuels paramètres. Dans un environnement ouvert comme un environnement domotique, chaque objet expose des fonctions qui lui sont propres et la définition d'un espace d'action est donc difficile dans

le cas général. À nouveau, l'utilisation d'OpenHAB permet de disposer d'une interface structurée et d'un espace d'action bien défini quels que soient les objets manipulés.

Asymétrie des perceptions – Enfin, dans le cadre d'un apprentissage en interaction on est confronté à une asymétrie entre ce qui est perceptible et / ou le mode de perception de l'environnement entre l'agent et le partenaire social. Dans le cadre de la domotique, l'agent est dans un environnement numérique et l'utilisateur dans l'environnement physique. Les perceptions entre les deux environnements ne sont pas nécessairement alignées. Le niveau de luminosité de l'environnement peut ainsi être représenté par une valeur numérique dans l'environnement numérique. Une faible augmentation ou une diminution de cette valeur peut passer inaperçu pour un utilisateur dans l'environnement physique.

Pour un agent artificiel, l'utilisation de ces interfaces est moins simple, car il ne dispose pas des repères du monde physique et sera donc en difficulté pour acquérir un sens physique de ses actions. On peut toutefois tenter de se ramener dans un formalisme similaire à celui de la robotique simulée. Cela implique de définir une interface suffisamment générale pour rendre les observations et les actions de l'agent indépendantes des objets numériques et de leurs fonctionnalités. C'est ce que nous permet de faire OpenHAB.

4.3 L'environnement : OpenHAB et le simulateur

Nous avons conçu un simulateur émulant une interface OpenHAB et assurant une parfaite compatibilité avec la plateforme. Cela doit permettre à un agent d'être compatible avec de nombreux objets sur OpenHAB et surtout d'envisager le déploiement d'un agent en conditions réelles après une phase de test dans l'environnement de simulation. On décrit dans cette partie le fonctionnement d'OpenHAB ainsi que le simulateur.

4.3.1 Description de la plateforme OpenHAB

Les concepts : Binding, Thing, Channel, Item – OpenHAB est une plateforme qui propose d'agréger un ensemble d'objets connectés sur un réseau et de pouvoir interagir avec eux depuis une même interface. Chaque objet connecté communique avec la plateforme grâce à un Binding, il s'agit de l'interface entre l'API du fabricant et celle d'OpenHAB. Un objet est représenté par un Thing, il s'agit d'une représentation modulaire qui contient notamment l'adresse physique de l'objet ainsi que ses fonctionnalités. Ces dernières sont représentées par les Channels, une structure de données qui contient :

1. *Nom* pour l'identifier de manière unique ;
2. *Description*, fournie, détaillant quels sont les effets réalisables par ce Channel ;

3. *Type*, il correspond à l'un des 12 types d'ITEMs autorisés dans OpenHAB. Un ITEM spécifie le type d'information qui est porté par le CHANNEL ainsi que les actions exécutables. Le Tableau 4.1 représente l'ensemble des ITEMs définit dans OpenHAB ;
4. *Mode lecture-écriture* : un CHANNEL peut être en mode lecture, écriture ou lecture-écriture, différenciant ainsi ceux qui portent de l'information sur l'état de l'objet (lecture), ceux qui permettent d'agir (écriture) et mixte (lecture/écriture).
5. *Valeur* (si CHANNEL en mode lecture). Il peut s'agir d'un booléen (par exemple pour le Switch), d'une valeur numérique (comme pour le Dimmer), d'un triplet de valeurs (comme pour le Color qui encode la couleur au format HSB) et enfin une chaîne de caractères (pour l'ITEM String).

Pour (un peu) plus de détails, ces différents concepts sont définis sur la page Web de la plateforme [2].

Les actions dans OpenHAB – On peut définir l'ensemble des fonctions sur OpenHAB comme l'ensemble des méthodes exécutables sur les ITEMs (voir Tableau 4.1). Elles ne sont pas toutes exécutables sur tous les ITEMs. Cela représente 13 actions au total qui prennent chacune des paramètres qui leur sont spécifiques. Ainsi, la fonction OnOff d'un Dimmer prend un paramètre spécifique ON ou OFF tandis que Percent de Dimmer prend comme argument l'intensité du Dimmer à modifier. Les différents types de paramètres sont les suivants :

1. Des paramètres spécifiques définis par OpenHAB (OnOff, IncreaseDecrease, Next-Previous, RewindFastforward) ;
2. Une valeur numérique (Percent, Decimal) ;
3. Un code couleur (HSB) ;
4. Une localisation (Point) ;
5. Une chaîne de caractères (String).

Exemple – On peut donner l'exemple d'une enceinte connectée (voir Figure 4.1b pour un diagramme tiré du simulateur). Le type d'ITEM Player peut être employé par un CHANNEL dédié pour contrôler l'état de lecture de l'objet. Les fonctions proposées sont alors des fonctions de lecture PlayPause, de passage au titre suivant NextPrevious et d'avance rapide RewindFastForward. Pour lancer la lecture, on peut envoyer via OpenHAB la commande PlayPause(PLAY) sur ce CHANNEL.

[2] https://www.openhab.org/docs/concepts/

En résumé – Autrement dit, les ITEMs constituent une brique de base qui permet d'encoder l'état d'un objet et d'exposer les actions disponibles. Les CHANNELs fournissent une sur-couche sémantique aux ITEMs en indiquant la fonctionnalité réelle de l'ITEM vis-à-vis d'un THING.

OpenHAB est compatible avec un nombre d'objets impressionnants. Chacun est décrit dans la documentation de la plateforme. Lorsque l'on connecte un objet au réseau sur lequel la plateforme est installée, celui-ci peut être détecté et ses informations récupérées automatiquement (CHANNELs, descriptions, fonctions associées, etc.).

Item	**Description**	**Fonctions associées**
Color	Color information (RGB)	OnOff, IncreaseDecrease, Percent, HSB
Contact	Status of contacts, e.g. door/window contacts. Does not accept commands, only status updates.	OpenClosed
DateTime	Stores date and time	-
Dimmer	Percentage value for dimmers	OnOff, IncreaseDecrease, Percent
Group	Item to nest other items / collect them in groups	-
Image	Binary data of an image	-
Location	GPS coordinates	Point
Number	Values in number format	Decimal
Player	Allows control of players (e.g. audio players)	PlayPause, NextPrevious, RewindFastforward
Rollershutter	Roller shutter Item, typically used for blinds	UpDown, StopMove, Percent
String	Stores texts	String
Switch	Switch Item, used for anything that needs to be switched ON and OFF	OnOff

TABLE 4.1 – Les différents types d'ITEMs disponibles sur OpenHAB, adapté de https://www.openhab.org/docs/configuration/items.html#type

4.3.2 Description du simulateur

Intérêt d'un simulateur

Afin de pouvoir tester l'apprentissage d'agents de Deep RL dans un environnement domotique, nous avons développé un simulateur d'objets connectés compatibles avec l'interface OpenHAB. Le simulateur a été conçu avec un double objectif :

1. Permettre l'étude et le développement d'agents de Deep RL adapté à la manipulation d'objets connectés dans un formalisme proche de ce qui pourrait être fait en conditions réelles. Le simulateur implémente donc déjà un ensemble d'objets connectés, mais il est relativement facile d'en ajouter de nouveaux pour complexifier encore l'environnement. Le simulateur dispose aussi d'une série d'options permettant d'être utilisé en mode *simplifié*.
2. Pré-entraîner un agent de Deep RL en vue d'un déploiement en conditions réelles. Le simulateur a donc été conçu de façon à reproduire aussi fidèlement que possible le fonctionnement d'OpenHAB. Un agent pré-entraîné sur le simulateur est donc en mesure d'envoyer des commandes sur une interface OpenHAB moyennant un convertisseur développé par Cloud Temple.

Le simulateur sera prochainement disponible en *open source* sur Github. On décrit maintenant le fonctionnement du simulateur.

Channels, Things et Actions dans le simulateur

Channel – On peut représenter les Channels comme un objet composé d'attributs (nom, description, type et éventuellement valeur) ainsi que des méthodes qui permettent de modifier l'état du Channel et qui dépendent du type d'Item. On représente trois types de Channels en Figure 4.1a en fonction de leur mode lecture-écriture. Un channel en mode lecture (R) ne comporte pas de méthodes puisqu'il n'est associé à aucune action, le channel en mode écriture (W) ne dispose pas d'attributs valeur puisqu'il ne stocke pas d'information. Le type de l'attribut valeur d'un Channel peut prendre plusieurs formes en fonction de l'Item.

Thing – Un Thing est une collection de Channels. On représente en Figure 4.1b l'exemple d'une enceinte connectée avec trois Channels de types différents : un *Dimmer* pour contrôler le volume, un *Switch* pour mettre en marche et arrêter l'appareil et un *MediaPlayer* pour contrôler la lecture. Les méthodes d'un Channel sont spécifiques à leurs fonctionnalités. Un environnement OpenHAB est simplement une collection d'objets connectés.

Les actions dans le simulateur – La diversité des paramètres que peuvent prendre les fonctions des ITEMs est une difficulté pour modéliser l'espace d'action d'un agent de Deep RL. À des fins de simplification, on choisit de discrétiser les fonctions qui prennent des paramètres spécifiques. Par exemple un ITEM qui dispose de la fonction OnOff, est représenté dans le simulateur comme disposant des fonctions On et Off (On = OnOff(On) et Off = OnOff(Off)). Les autres types d'actions sont utilisables avec leurs propres paramètres. On ajoute à ces actions une action supplémentaire *DoNothing* qui permet à un agent apprenant de signaler qu'il ne souhaite pas interagir avec OpenHAB. Cela peut s'avérer utile lorsque l'instruction que l'on donne à l'agent est déjà réalisée dans l'environnement. À noter que la simplification mentionnée ici n'induit aucune perte fonctionnelle par rapport à OpenHAB.

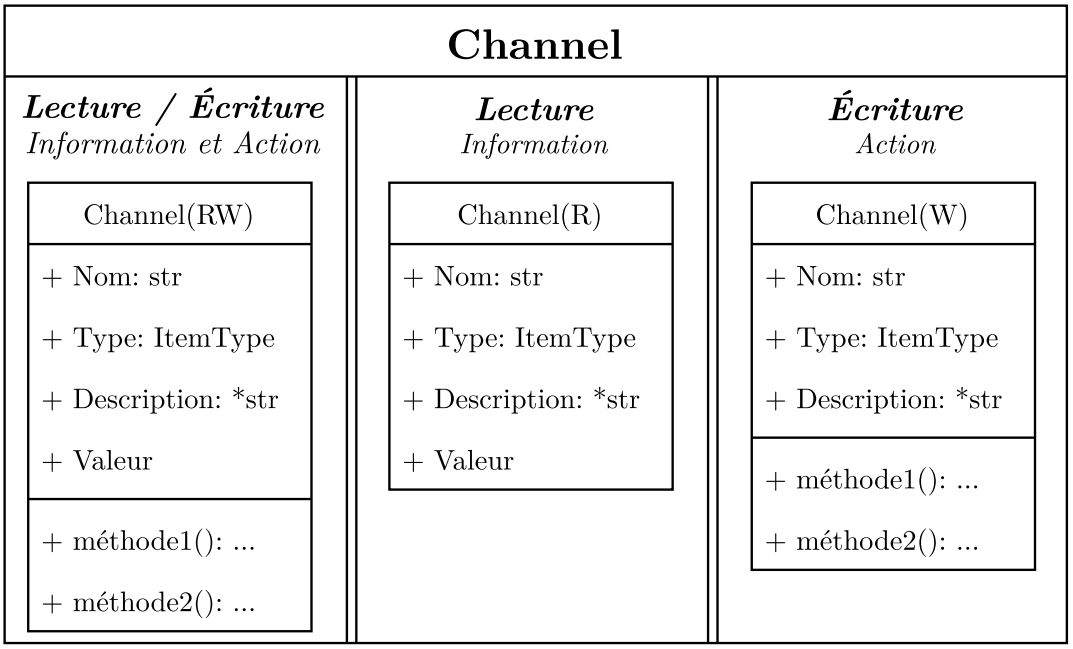
Les objets implémentés dans OpenHAB

La structure du simulateur permet d'implémenter n'importe quel objet compatible avec OpenHAB. En pratique et pour nos simulations, nous avons implémenté une série d'objets plus spécifiques à notre cas d'usage. Il s'agit d'objets de types télévisions, lumières, enceintes, volets, prises de courants et radiateurs. Cela permet par exemple de simuler le salon d'un utilisateur. Pour chaque type d'objets, il en existe plusieurs versions possédant plus ou moins de fonctionnalités et qui sont directement inspirés d'objets existants sur OpenHAB.

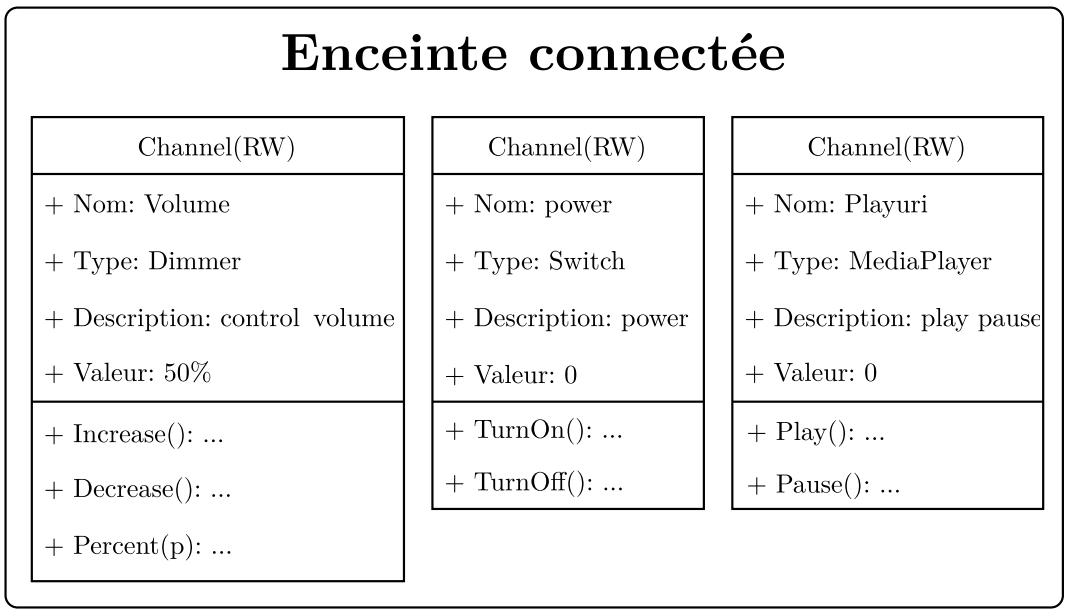
4.3.3 Le partenaire social dans le simulateur

L'un des enjeux dans la conception du simulateur est le partenaire social. En effet, pour chaque objet implémenté, il est nécessaire de fournir au partenaire social le moyen d'observer les changements qui se produirait dans l'espace physique lorsque l'agent interagit avec l'objet. Face à la pénibilité que peut représenter l'implémentation du partenaire social si elle doit être effectuée indépendamment pour chaque objet, nous avons implémenté une méthode d'évaluation progressive des CHANNELs qui permet de définir très rapidement de nouveaux objets et les fonctions du partenaire social associées.

En effet, bien qu'un ITEM ne contienne pas toute la sémantique d'un CHANNEL, les fonctionnalités représentées par de mêmes ITEMs possèdent certaines affordances. Un Dimmer permet de contrôler aussi bien le volume d'une enceinte connectée que la luminosité d'une ampoule ou la température d'un radiateur. Dans tous les cas, il s'agit de contrôler une *intensité* et on mesure son évolution de la même façon. Déterminer les modifications subies par un Dimmer revient : (1) à identifier si l'intensité a augmenté ou diminué ($\Delta x < 0$ ou > 0) et (2) à produire une phrase qui exprime cette modification



(a) Représentation d'un Channel dans OpenHAB : lecture, écriture, lecture-écriture



(b) Exemple d'un Thing dans OpenHAB : une enceinte connectée avec 3 Channels

Figure 4.1 – Représentations des Things et des Channels dans OpenHAB

et qui soit adaptée à l'objet (*tu as monté le son de l'enceinte* vs *tu as fait baisser la température du radiateur*). Seule la seconde partie est spécifique au CHANNEL, on peut ainsi produire à partir de (1) un label générique qui indique un changement d'état de l'ITEM et en (2) associer ce label à un description spécifique. Ainsi lorsque l'on définit un nouvel objet, il suffit d'associer les labels de changement d'état des CHANNELs aux descriptions en langage naturel.

Descriptions absolues et relatives – le partenaire social peut produire deux formes de descriptions : celles qui décrivent l'état de manière absolue ou relatif. Pour les premières, leur valeur de vérité est appréciable à chaque instant (« la télévision est allumée ») tandis que pour les secondes, elle s'apprécie entre deux états (« le volume de la télévision a augmenté »).

4.4 Définition de l'environnement à partir du simulateur

Le simulateur tel qu'on l'a décrit jusqu'à présent permet d'implémenter à peu près n'importe quel objet qui serait disponible sur OpenHAB, de pouvoir interagir avec eux d'une manière similaire à ce qui pourrait être fait sur OpenHAB et de simuler un partenaire social qui fournit des descriptions en langage naturel de l'état courant de l'environnement physique ainsi que des changements observés entre deux instants. À partir du simulateur, on peut implémenter un environnement spécifiquement conçu pour interagir avec des agents de Deep RL. L'environnement joue le rôle d'interface entre le simulateur (ou OpenHAB directement) et l'agent.

Il nous reste donc à définir plus précisément cette interface et notamment comment on interagit avec le simulateur (espace d'action), quel est l'état observable à chaque instant pour l'agent et la manière dont sont structurés les épisodes dans l'environnement. On commence par définir le mode d'interaction.

4.4.1 Définition de l'espace d'actions

A l'inverse d'un environnement de robotique simulée où l'espace d'actions est directement liée aux capacités locomotrices du robot (déplacement, saisie d'objets, etc.), dans un environnement domotique numérique, ce sont les objets eux-mêmes qui définissent l'espace d'actions. La particularité de cet environnement est donc que l'ensemble des actions réalisables n'est pas défini extérieurement, il n'est défini qu'au moment où l'en prend

connaissance des objets présents. Si des objets sont installés ou retirés de l'environnement, l'espace d'actions évolue en conséquence.

À notre connaissance, il s'agit d'une situation relativement peu explorée en Deep RL. L'interface d'OpenHAB impose des contraintes sur la manière dont une action doit être formulée, mais on peut envisager différents paradigmes d'interaction tant qu'ils restent compatibles avec OpenHAB.

Discrétisation des paramètres des actions

Certaines des actions dans OpenHAB nécessitent le choix d'un paramètre dans des espaces variés : nombre entre 0 et 100 pour un Dimmer, code couleur HSB pour Color, une chaîne de caractère pour String, etc. La variabilité que peuvent prendre ces paramètres est une difficulté pour l'agent apprenant. On fait donc le choix de discrétiser ces paramètres. L'environnement propose ainsi de discrétiser, selon différents niveaux ou catégories, les paramètres de certaines des actions.

Ainsi, la fonction percent d'un Dimmer peut être utilisée sous la forme Percent(x) où x est soit un entier entre 0 et 100 (mode normal), soit un label choisi par l'utilisateur et qui représente un niveau d'intensité, par exemple *high*, *average* ou *low*. Chaque label est associé à une valeur d'intensité. Le nombre de niveau d'intensité est paramétrable ainsi que les labels. C'est ce mode que l'on utilisera par la suite et on considère donc que les paramètres des fonctions sont toujours discrétisés. On opère de même pour l'item Color et String.

Le cas particulier des Channels de type String – Le type String présente certaines difficultés particulières pour un agent de Deep RL. En effet, ce type d'Item est parfois employé par les fabricants d'objets comme un type générique qui peut porter n'importe quelle information sous la forme d'une chaîne de caractères. Un Channel de ce type peut ainsi contenir aussi bien une phrase ou un mot en langage naturel qu'une URL (*Uniform Ressource Locator*), une URI (*Uniform Ressource Identifier*) ou une chaîne de caractère spécifique à l'objet. Cela représente évidemment une difficulté pour l'agent qui doit choisir le paramètre d'une action mais aussi pour en représenter le sens. On choisit donc de discrétiser les valeurs que peuvent prendre un Channel de type String afin d'en limiter la variabilité. Dans les simulations que nous présentons dans ce chapitre, nous nous sommes limités aux numéros de chaînes de télévision.

Mode d'interaction directe vs séquentielle

On cherche donc un mode d'interaction avec l'environnement qui conduise à générer une action exécutable dans OpenHAB tel que définie dans Partie 4.3. On envisage deux

approches :

1. Une approche *directe* serait de parcourir l'ensemble des objets disponibles et leurs CHANNELs associés pour lister à chaque instant l'ensemble des actions possibles. On pourrait alors représenter toutes ces actions dans un même espace et choisir une action à effectuer parmi celles-ci.
2. Une approche *séquentielle* consiste à demander à l'agent d'effectuer des choix successifs qui déterminent l'action. L'agent doit d'abord choisir le THING sur lequel il souhaite agir, puis le CHANNEL, puis la méthode du CHANNEL et enfin d'éventuels paramètres de cette méthode.

L'approche séquentielle est inspirée de la manière dont AlphaStar traite l'espace d'action pour jouer à StarCraft II (Vinyals et al., 2019). AlphaStar utilise un espace d'actions structuré dans lequel, l'agent choisit séquentiellement l'action à réaliser (se déplacer, construire, etc.), les unités pour la réaliser, la cible de l'action, etc. Cette approche séquentielle, similaire à celle qu'un humain pourrait adopter pour jouer à un jeu, nous a paru adaptée à notre problématique. Dans StarCraft II, l'ensemble des actions possibles dépend des unités en jeu, dans OpenHAB, il dépend des objets sur le réseau. On parle aussi de décomposition autorégressive de l'action (Metz et al., 2017).

Avantages d'un mode d'interaction séquentiel

— ***Embodiment* numérique :** De plus, cette approche permet d'imiter un utilisateur qui naviguerait de menus en sous-menus sur une interface graphique pour exécuter des actions sur un logiciel (ex : Fichiers > Enregistrer Sous > ...). Il s'agit de fournir à l'agent une forme d'*embodiment* dont il est privé dans l'espace numérique. On conçoit donc cette approche séquentielle comme un moyen de se déplacer dans un espace numérique. On est ici inspiré par les travaux de (Hill et al., 2019) qui a notamment montré les bénéfices de l'*embodiment* sur des tâches de VQA.

— **Exploiter la structure de l'espace d'action :** Le choix du mode d'interaction avec l'environnement digital influe directement sur la représentation des actions, le type d'algorithme RL et d'architecture que l'on pourra utiliser. L'approche séquentielle présente l'avantage d'exploiter la structure inhérente à OpenHAB. Si l'on souhaite observer le phénomène de généralisation systématique observé pour IMAGINE (Partie 3.7.4), exploiter cette structure est essentiel (Kuo et al., 2020).

Cette approche est illustrée en Figure 4.2. On peut par exemple augmenter le son d'une enceinte en choisissant successivement le THING *enceinte*, le CHANNEL *Volume*, l'action *Percent* et le paramètre *VeryHigh*.

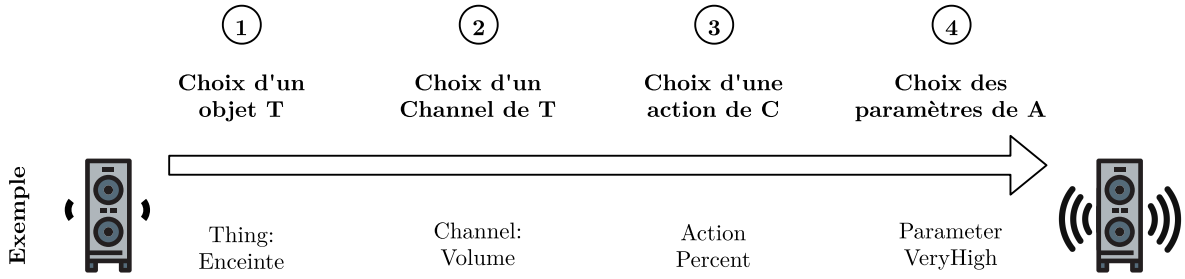


Figure 4.2 – Sélection séquentielle d'une action dans le simulateur OpenHAB

4.4.2 Espace d'état et focus attentionnel

L'état observable de l'environnement peut être défini de manière générale comme l'ensemble des Channels en mode lecture ou lecture-écriture des objets de l'environnement. Cependant, pour tenir compte du mode d'interaction séquentiel avec l'environnement, on fait évoluer l'état au cours d'un cycle de choix d'action. Lorsque l'agent doit choisir sur quel objet il souhaite interagir, l'environnement lui propose d'observer tous les objets présents. Une fois l'objet choisi, l'état de l'environnement visible par l'agent est restreint aux Channels dudit objet. Une fois le Channel choisi, l'agent ne voit plus que l'état du Channel en question. Cela permet à l'agent de concentrer naturellement son attention sur les éléments pertinents à l'action qu'il cherche à réaliser. On appelle ce mode d'évolution de l'état observable un *focus attentionnel*. L'état visible par l'agent à chaque étape est représenté en Figure 4.3.

Ce focus attentionnel participe à la création d'une forme d'embodiment numérique que l'on a évoqué précédemment. On peut l'interpréter comme un zoom progressif de l'agent similaire à un humain qui concentrerait son attention sur une petite partie de son environnement lorsqu'il réalise une tâche.

4.4.3 Structure d'un épisode dans l'environnement numérique

Cela nous conduit aussi à préciser la manière dont on interagit avec l'environnement. Une action pour notre agent Ilead consiste à faire un choix parmi un ensemble d'options proposées par l'environnement. On illustre ce comportement sur la Figure 4.3. L'agent commence par choisir parmi les objets proposés par l'environnement numérique celui sur lequel il souhaite agir étant donné une tâche en langage naturel et l'état de l'environnement. Puis, l'environnement répond à l'agent en fournissant l'ensemble des Channels en mode écriture de l'objet choisi. De même,l'agent choisit l'action qu'il souhaite effectuer en fonction du Channel qu'il aura choisi et des paramètres associés. L'action complète est alors exécutée sur le simulateur d'OpenHAB et le partenaire social (ou l'utilisateur) peut percevoir son effet dans l'environnement physique. On constate ainsi l'asymétrie entre

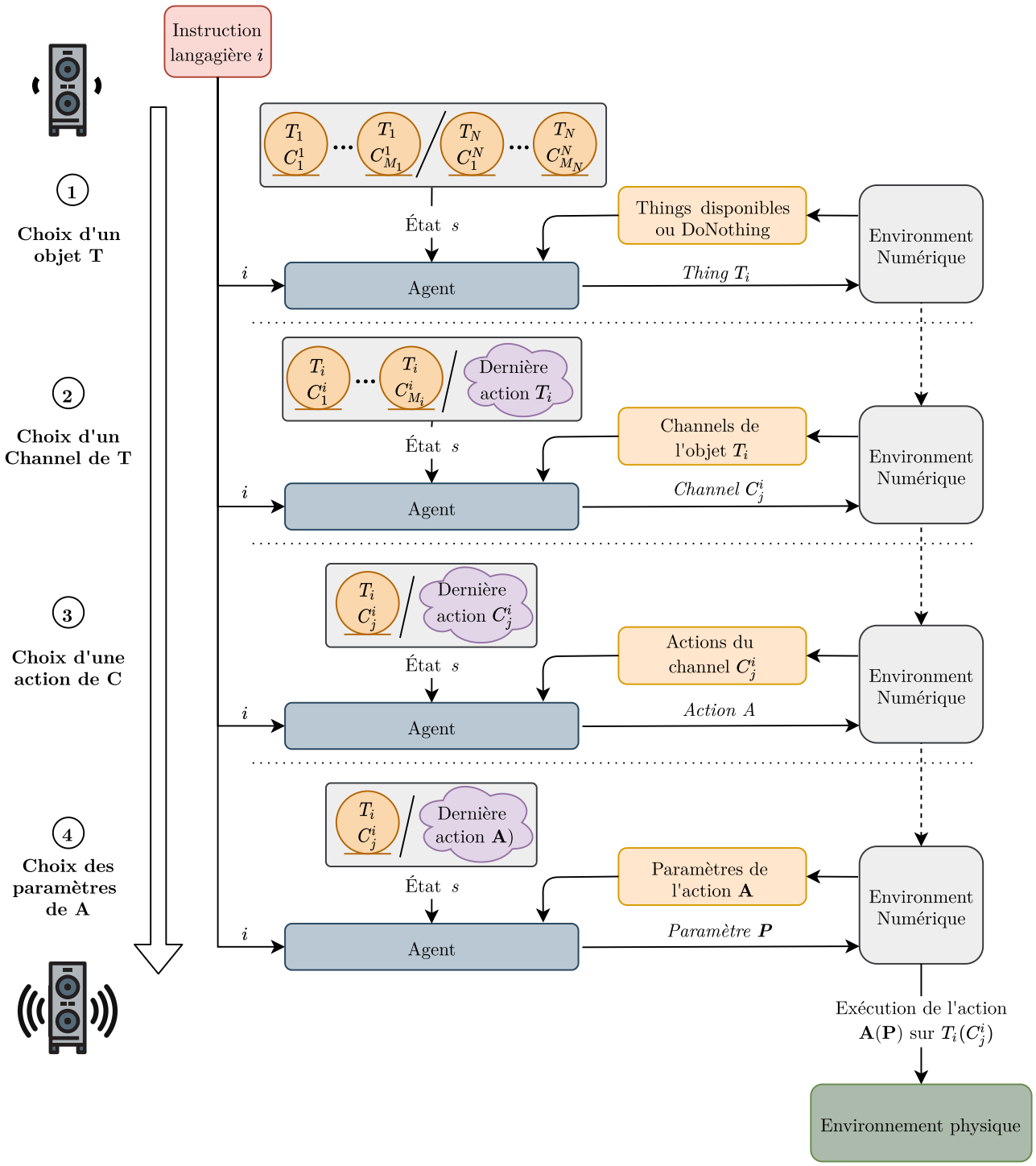


Figure 4.3 – Représentation de l'état de l'environnement comme une collection de Channels

l'environnement numérique de l'agent et l'environnement physique de son superviseur. Ce dernier n'observe les effets des actions de l'agent que toutes les 4 actions.

Une fois que l'agent a exécuté une action complète sur l'environnement, l'environnement numérique propose à nouveau à l'agent de choisir un objet sur lequel effectuer son action suivante. L'agent a aussi le choix d'interrompre l'épisode en choisissant de ne rien faire (action *DoNothing*). Pour éviter la situation où l'agent continue à exécuter des actions indéfiniment, on limite un épisode à deux cycles (i.e. deux actions exécutées sur OpenHAB).

Stratégie de supervision du partenaire social – La stratégie de supervision du partenaire social est entièrement paramétrable par l'utilisateur de l'environnement. Dans nos simulations, nous nous sommes placés dans un cadre similaire à celui employé pour IMAGINE. le partenaire social intervient donc lorsqu'il observe un changement dans l'espace physique qui a une signification particulière pour lui.

4.5 L'agent ILEAD

On peut maintenant décrire notre agent ILEAD (Interactive LEarning Agent for Domotic environment) et qui s'appuie sur ces représentations.

4.5.1 Architecture de l'agent

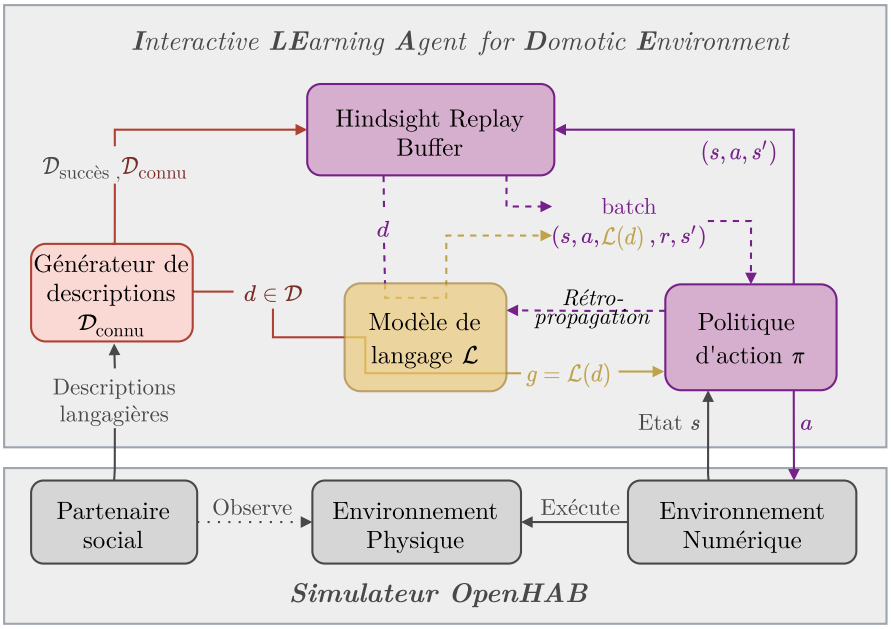


FIGURE 4.4 – Architecture de ILEAD

Quelques différences essentielles avec IMAGINE et LE2

On représente en Figure 4.4 l'architecture d'ILEAD dans un format similaire à celui employé pour LE2 et IMAGINE. On peut en souligner les principales différences par rapport à IMAGINE :

— *L'environnement sur lequel l'agent agit est différent de celui observé par le partenaire social.* En effet l'agent agit sur l'environnement numérique et le partenaire social observe l'environnement physique. En raison de cette asymétrie entre l'agent et le partenaire social, l'agent ne dispose pas de supervision pour les transitions intermédiaires qui le conduisent à réaliser une action. Seule la dernière transition, celle où l'action est exécutée sur OpenHAB peut occasionner un changement d'état de l'environnement physique et permettre de récupérer un retour de l'utilisateur sous la forme d'un retour en langage naturel. À noter que certaines actions peuvent n'avoir aucun effet sur l'environnement physique comme allumer un objet qui le serait déjà.

— *L'architecture ne dispose pas de fonction de récompense.* C'est une conséquence directe de la remarque précédente. Dans *Playground* on pouvait apprendre une fonction de récompense pour superviser a posteriori les transitions collectées pendant un épisode (transitions pour lesquelles le partenaire social n'a pas fourni de supervision, car il n'intervient qu'à la fin d'un épisode). Dans OpenHAB, on ne dispose pas de moyen simple pour superviser les transitions intermédiaires. On se place alors dans un contexte où le partenaire social n'intervient pas seulement à la fin d'un épisode mais à chaque changement d'état de l'environnement physique. Celui-ci possède les mêmes propriétés que pour LE2 et IMAGINE (voir Partie 3.4.2) et permet donc à l'agent de savoir quelles tâches ont été accomplies et d'inférer celles qui ne l'ont pas été. L'agent dispose ainsi des retours positifs et négatifs pour apprendre la politique d'action. Une fonction de récompense s'avère donc inutile sauf à pallier l'absence du partenaire social.

— *Le mécanisme de Replay est légèrement différent.* À nouveau, cela découle du point précédent. Pour rappel, le principe du Hindsight Replay (Andrychowicz et al., 2017) est de rejouer des transitions en substituant le but originalement visé lorsque la transition a été collectée par un autre. Alors que dans IMAGINE et LE2, la substitution de tâches étaient effectuées au moment de rejouer les transitions pour l'apprentissage de la politique d'action, pour ILEAD il s'effectue au moment où le retour du partenaire social est collecté. Ainsi, on ne substitue une tâche par une autre que dans les transitions qui conduisent à un changement d'état dans l'environnement physique.

Description générale

On revient maintenant sur les différents modules. Le rôle du générateur de descriptions est de collecter les descriptions qui proviennent du partenaire social et de fournir à la politique d'action un objectif au début de chaque épisode. Le modèle de langage est similaire à celui employé pour IMAGINE, il s'agit d'une architecture LSTM (Hochreiter et Schmidhuber, 1997). À la différence d'IMAGINE, le modèle est appris directement avec la politique d'action. On détaille enfin l'architecture de la politique d'action qui permet de s'adapter aux structures particulières des espaces d'état et d'action.

L'espace d'action étant discret, on n'utilise pas DDPG comme pour LE2 et IMAGINE mais une version de l'algorithme DQN (Mnih et al., 2015) évoqué en Partie 3.2.2. On implémente une version dite Double DQN avec Prioritized Experience Replay (Schaul et al., 2015b ; van Hasselt et al., 2016) qui corrige certains biais d'estimation (van Hasselt et al., 2016) et emploie une stratégie de Replay favorisant les transitions les plus pertinentes (Schaul et al., 2015b). On propose donc une architecture calculant la Q-valeur associée à une paire état-action. Cela passe par le calcul d'une représentation de l'état courant, que l'on appelle le *contexte* et d'une représentation pour chacune des actions possibles. L'architecture proposée permet à un même agent de s'adapter à la variabilité d'un environnement numérique et notamment le nombre d'objets et de CHANNELS, ainsi qu'à l'espace d'actions relativement complexe qui en découle.

4.5.2 Représentation du contexte

Représentation d'un CHANNEL

La représentation de l'état de l'environnement passe donc notamment par la représentation des CHANNELS. Afin d'être utilisé par un agent de Deep RL, il est nécessaire que l'on puisse représenter l'état d'un CHANNEL sous la forme d'un vecteur numérique de taille fixe. On choisit de représenter un CHANNEL par :

1. La description en langage naturel de l'objet auquel appartient le CHANNEL. Cette description est encodée dans un vecteur de taille fixe à l'aide d'un modèle de langage. En pratique, on utilise la moyenne des *embeddings* Glove (Pennington et al., 2014) des mots de la description. Cela permet d'identifier précisément à quel objet appartient le CHANNEL. Pour un usage en conditions réelles, cette description en langage naturel est fournie par l'utilisateur.
2. Le type d'ITEM : Cela permet de rapprocher naturellement des CHANNELS d'un même type qui, bien qu'ils représentent des fonctionnalités en apparence totalement différentes, sont similaires en termes de structures. De plus, le type de valeur portée un CHANNEL étant directement conditionné par le type d'ITEM (code couleur HSB

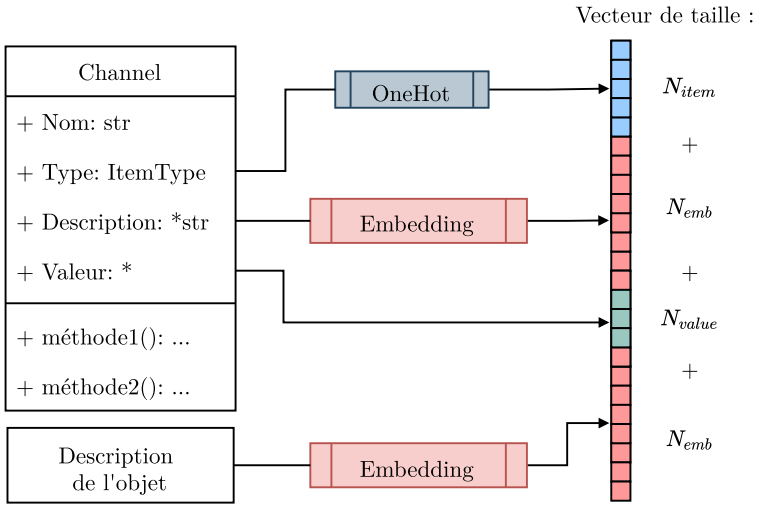


FIGURE 4.5 – Représentation de l'état d'un CHANNEL sous la forme d'un vecteur de taille standard

pour Color, coordonnées GPS pour Location), connaître le type d'ITEM permet d'inférer le type de valeurs que l'on visualise. Le nombre de types d'ITEMs étant connu, on l'encode en utilisant une représentation OneHot.

3. La description en langage naturel du CHANNEL : elle permet à la fois d'identifier le CHANNEL de manière unique au sein de l'objet et de porter une information sémantique sur ce que permet ce CHANNEL. Elle est encodée sur le même principe que la description de l'objet.

4. La valeur de l'état du CHANNEL : elle décrit l'état courant de l'objet. Standardiser la manière dont on représente cette valeur indépendamment du type d'ITEM n'est pas facile, notamment à cause du type String (voir ci-dessous). En effet, si on exclut ce type, on peut représenter toutes les valeurs que peuvent prendre les différents ITEMs par un vecteur de taille 3. L'état d'un Dimmer est ainsi représentée par $[x, 0, 0]$. Tout comme les paramètres des actions sont discrétisés, on choisit de discrétiser aussi les valeurs admissibles du type String. On peut alors les encoder à l'aide d'une représentation OneHot ou ordinale. On peut ainsi encoder les valeurs de tous les ITEMs par un vecteur d'une taille standard.

La représentation d'un CHANNEL est simplement la concaténation de ces dernières. Les descriptions de l'objet et du CHANNEL permettent d'encoder une forme d'identifiant unique, l'ITEM informe sur le type de valeur porté et la valeur encode l'état courant du CHANNEL. On illustre la représentation d'un CHANNEL en Figure 4.5. On note que cette représentation n'est pas apprise. Cela permet de s'appuyer sur des caches notamment pour le calcul des encodages des descriptions et d'accélérer l'apprentissage et l'inférence.

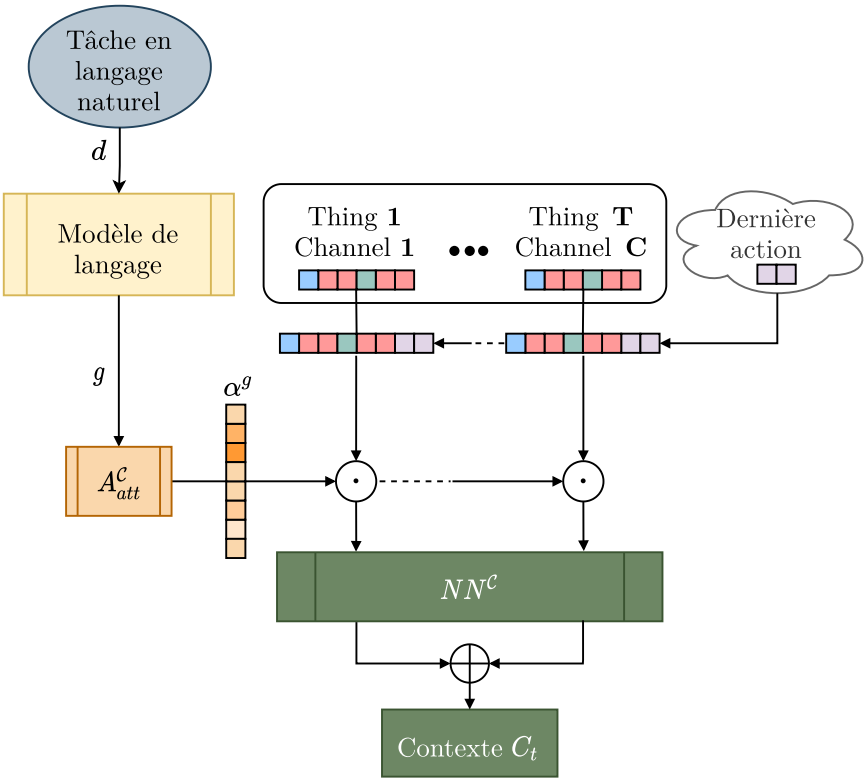


Figure 4.6 – Représentation du contexte agrégeant les différents Channels observables, la dernière action effectuée et la tâche en langage naturel

Représentation du contexte

On définit le contexte comme une représentation calculée à partir de l'état d'un ensemble de Channels. Ces derniers sont couplés avec la tâche, en langage naturel, poursuivi par l'agent et la représentation de la dernière action effectuée. C'est à partir de ce contexte que l'agent calcule la Q-valeur des paires contextes-actions. On calcule la représentation du contexte en utilisant une architecture modulaire similaire à celle détaillée pour Imagine en Partie 3.6.3, où le Channel joue le rôle de l'objet. La représentation du contexte est donc apprise.

La représentation c_i de chaque Channel observable ($\mathcal{C}_{obs}$) est concaténée avec celle de la dernière action effectuée $\mathbf{a}_{t-1}$ (voir Partie 4.5.3 pour son calcul). Puis un masque attentionnel $\boldsymbol{\alpha}^g$, calculé à partir de la représentation de la tâche $g = \mathcal{L}(d)$, est appliqué à chaque paire (Channel, action). Ces représentations sont ensuite projetées dans un espace de plus grande dimension à l'aide d'un réseau de neurones partagé $\text{NN}^{\mathcal{C}}$ avant d'être agrégées par une somme pour préserver l'invariance par permutation. On illustre l'architecture du calcul du contexte en Figure 4.6 :

$$C_t(s) = \sum_{\mathcal{C}_{obs}} \text{NN}^{\mathcal{C}} \left(\boldsymbol{\alpha}^g \odot [c_i, \mathbf{a}_{t-1}] \right)$$

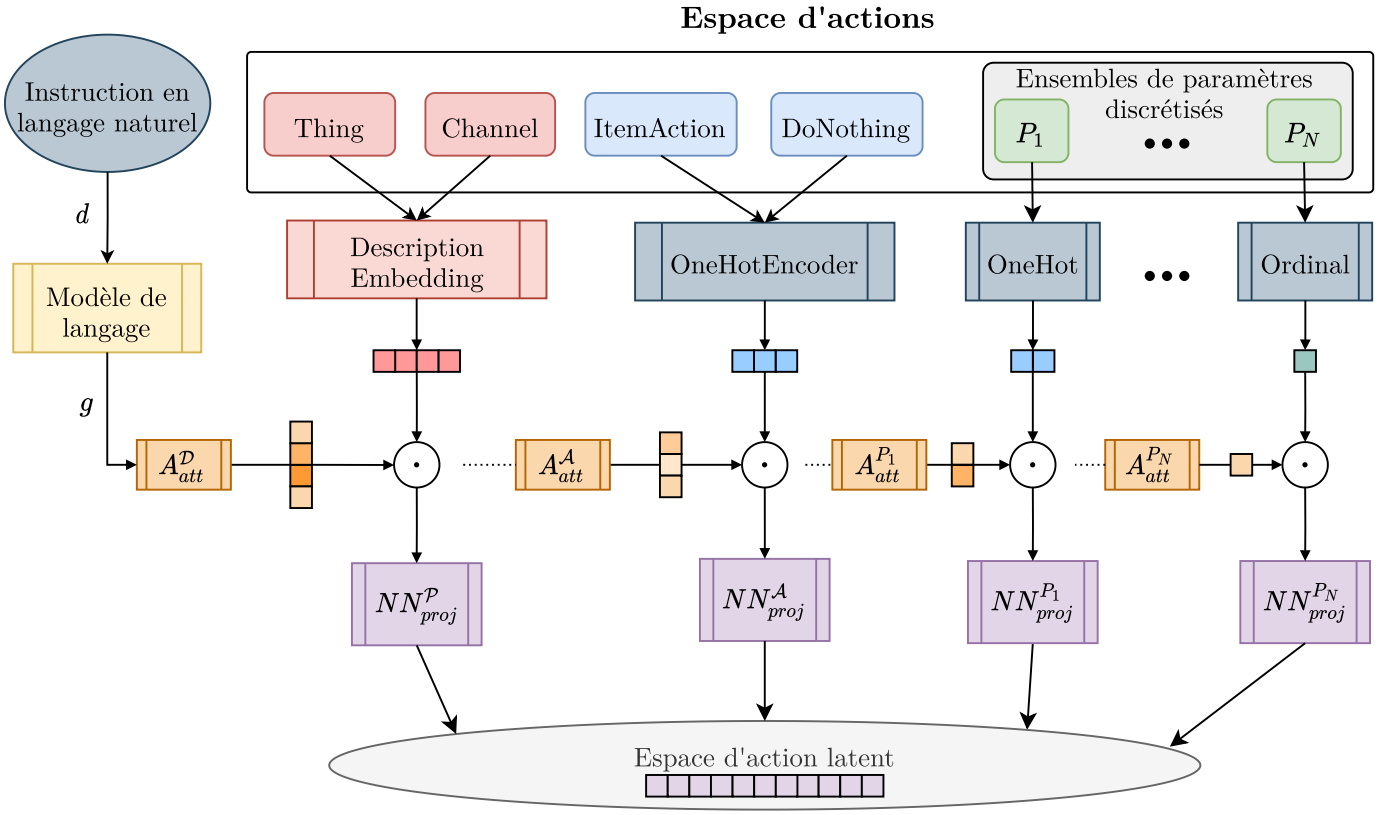


FIGURE 4.7 – Projection de l'espace d'action dans un espace commun

Remarque – Le choix d'inclure la dernière action effectuée dans le contexte est motivé par le constat suivant : une même collection de CHANNELs observables couplée à la même instruction en langage naturel ne renseigne pas suffisamment l'agent sur l'objet de sa décision. En effet, l'agent doit pouvoir différencier le cas où il n'y a qu'un seul objet observable dans l'environnement et où il doit sélectionner l'objet sur lequel il souhaite agir et le cas où il ne voit qu'un objet parce qu'il s'agit de celui déjà choisi. La représentation de la dernière action effectuée joue le rôle de mémoire permet à l'agent de « savoir où il en est ».

4.5.3 Représentation des actions

L'espace d'action manipulée par ILEAD est relativement complexe. En effet, les actions peuvent consister à sélectionner un objet, un CHANNEL, une fonction ou des paramètres. Chacune d'entre elles est représentable, mais il n'existe pas de représentation simple qui les rassemble. On apprend donc une représentation latente de toutes ces actions. C'est cette représentation qui, combinée avec le vecteur de contexte, permet de calculer la Q valeur associée à chaque action. On adopte différents modes de représentation selon la nature de l'action :

— Les actions qui consistent à choisir un objet ou un CHANNEL sont représentées par la description associée. Cela permet de gérer un espace d'action qui évolue notamment si de nouveaux objets sont introduits. Comme pour les CHANNELs, les descriptions sont encodées en moyennant les représentations Glove des mots qui la composent.

— Pour les actions types d'OpenHAB ainsi que celles ajoutées dans le simulateur (*DoNothing*), on utilise un encodage OneHot.

— Pour les paramètres des actions, on utilise un encodage OneHot ou ordinal suivant le type de paramètres. Cela est possible puisque l'on se place dans le cadre où l'on a discrétisé tous les paramètres des actions (voir Partie 4.4.1). Chaque type de paramètres est encodé séparément (les niveaux d'intensité d'un Dimmer, les chaînes de télévision, les couleurs, etc.). Cela permet à une même architecture de gérer un ensemble arbitraire de types de paramètres si besoin.

On pourrait directement projeter ces représentations a dans un espace de dimension commune à l'aide de réseaux de neurones dédiées. En pratique, on applique à nouveau un masque attentionnel (Chaplot et al., 2018) calculé à partir de la représentation de la tâche $g = \mathcal{L}(d)$. Nos expériences ont montré que cela permet à l'agent de gagner en performance. On projette donc g dans les différents sous-espaces d'encodage des descriptions, et on calcule le produit de Hadamard avec les actions : $A^i_{att}(g) \odot a^i$ où i désigne le type d'encodage. Enfin, on projette ces représentations dans un espace d'action commun à l'aide de réseaux de neurones dédiées. On illustre l'architecture du calcul du contexte en Figure 4.7 que l'on peut résumer par l'équation suivante :

$$\mathbf{a} = \mathrm{NN}^i_{proj}\left(A^i_{att}(g) \odot a^i\right)$$

4.5.4 Calcul des Q-valeurs

À partir des représentations du contexte et des actions détaillées ci-dessous, on calcule les Q-valeurs associées à chaque action. On concatène simplement le contexte C_t avec l'action a et on utilise un réseau de neurones totalement connecté avec une couche cachée. On adopte une stratégie d'exploration ϵ-greedy durant l'apprentissage. Finalement la politique d'action peut s'écrire :

$$\pi(s) = \arg\max_{\mathbf{a_j} \in \mathcal{A}} \mathrm{NN}^Q([C_t(s), \mathbf{a_j}])$$

Tous les réseaux de neurones sont entraînés par rétro-propagation du gradient à partir de l'erreur de DQN. On utilise une stratégie d'exploration ϵ-greedy.

4.6 Expériences et résultats

Dans cette partie, on évalue l'architecture proposée dans différents contextes de simulation. On étudie tout d'abord la capacité d'ILEAD à manipuler différents objets (Partie 4.6.1). On évalue ensuite ILEAD sur des ensembles d'objets (Partie 4.6.2). On poursuit en évaluant les capacités de généralisation systématique de l'agent (Partie 4.6.3.

Les résultats présentés ici sont encore préliminaires et ces travaux sont toujours en cours. Pour suivre l'apprentissage d'ILEAD sur un ensemble de tâches, on mesure périodiquement, tous les 100 épisodes, le taux de succès de l'agent pour chaque tâche moyennée sur 25 tentatives (dans 25 configurations de l'environnement générées de manière aléatoire). On réalise au minimum 10 simulations et on présente la moyenne et l'écart-type.

Dans les résultats présentés dans cette partie, on regroupe les tâches dans des ensembles cohérents et on calcule le taux de succès moyen sur cet ensemble. Dans ces simulations, une tâche n'implique qu'un seul objet, on peut ainsi calculer par exemple, le taux de succès moyen par objet. C'est cette mesure qui est présentée dans les Figures 4.8 à 4.10. Dans la Figure 4.11, on calcule un taux de succès global agrégé sur toutes les tâches de l'environnement. Dans la Partie 4.6.3, on regroupe toutes les tâches pour lesquelles l'agent reçoit une supervision langagière du partenaire social dans un ensemble *Train* et on rassemble les autres tâches (non supervisées par le partenaire social) au sein d'un ou plusieurs groupes. Ces dernières permettent de mesurer le niveau de généralisation systématique de l'agent dans différents environnements.

Remarque : Dans ces simulations, un agent est en mesure d'exécuter au plus deux actions dans un épisode, s'il choisit d'effectuer l'action *DoNothing*, cela interrompt automatiquement l'épisode. Au début de chaque épisode, l'environnement est positionné dans un état aléatoire.

4.6.1 Manipulation d'objets uniques

Les différents objets

On définit différents types d'objets inspirés de ceux existants sur OpenHAB :

— Des objets de type prises électriques (Plug). Ils sont constitués d'un unique CHANNEL de type Switch et peuvent uniquement être allumés ou éteints ;

— Des objets de types stores (Blinds) qui peuvent être ouverts ou fermés par un CHANNEL de type RollerShutter ;

— Divers objets de type lampes. On retrouve trois types de fonctionnalités pour les lampes : la variation de la luminosité (ITEM Dimmer), la variation de la tempé-

rature de la couleur (ITEM Dimmer), la variation de la couleur (ITEM Color). On teste trois types d'objets différents : BLight qui ne possède que le CHANNEL de variation d'intensité lumineuse, BTLight qui possède les variateurs d'intensité et de température et ColorLight qui possède les trois types de fonctionnalités ;

— Les objets de type Speaker qui disposent des fonctionnalités volume (ITEM Dimmer) et de contrôle de lecture (ITEM MediaPlayer) ;

— Les objets de type Télévision qui disposent des fonctionnalités de changement de chaîne (ITEM String), de volume (ITEM Dimmer) et de contrôle de lecture (ITEM MediaPlayer).

Chaque objet définit un ensemble de tâches qui lui sont propres et qui sont évaluables par le partenaire social. Plus un objet est complexe, plus il y a de tâches exécutables sur l'objet. On ne considère ici que des tâches qui n'impliquent qu'un seul objet à la fois et pas un ensemble d'objets (ex : « allume toutes les lumières »).

Statut on/off des objets et observabilité – De plus, on suppose que les objets sont toujours allumés (exceptés pour les objets de type Plug) et on ne laisse pas l'agent les éteindre. On est en effet confronté à la problématique suivante : quand un objet est éteint, il n'est pas évident de définir quel est l'état observable des CHANNELs. Faut-il garder en mémoire le dernier état disponible de l'agent ? Mais celui-ci ne risque-t'il pas de fausser la perception de l'agent ? Faut-il alors ne conserver que l'état du CHANNEL qui contrôle le caractère allumé ou éteint ? Mais, dans ce cas, comment identifier automatiquement ce CHANNEL ? En pratique, il est vrai que les objets sont souvent dotés d'un CHANNEL de type Switch qui contrôle l'état allumé/éteint. On choisit pour le moment de simplifier ce problème en supposant que les objets sont allumés. Si l'on suppose que l'on connaît le CHANNEL qui permet d'allumer l'objet, cette simplification revient à systématiquement exécuter l'action TurnOn préalablement à toute action.

Exemples de tâches pour BLight – Pour un objet BTLight, les tâches exécutables sont les celles liées à chacun des CHANNELs Dimmer de la luminosité et de la température, certaines indiquent un changement par rapport à l'état courant (*tâche relative*), les autres sont des descriptions de l'objectif à atteindre (*tâche absolue*).

— Increase <*name*> brightness (relative)

— Decrease <*name*> brightness (relative)

— Set <*name*> brightness to low (absolue)

— Set <*name*> brightness to average (absolue)

— Set <*name*> brightness to high (absolue)

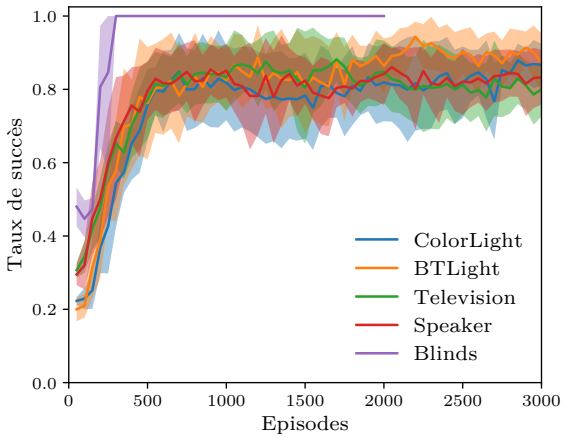


FIGURE 4.8 – Taux de succès d'ILEAD sur des objets uniques

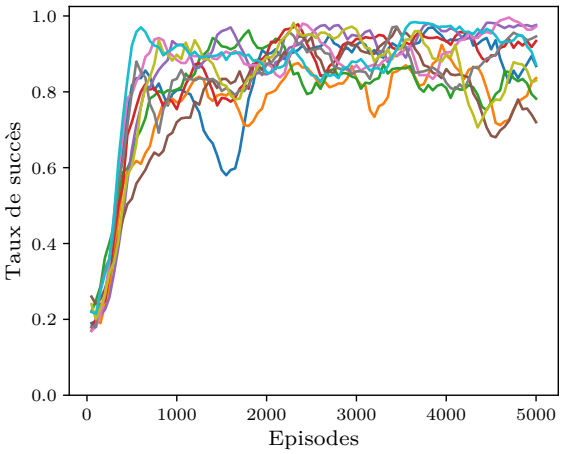


FIGURE 4.9 – Taux de succès de différents agents sur un objet BTLight

— Increase <*name*> temperature (relative)

— Decrease <*name*> temperature (relative)

— Set <*name*> temperature to low (absolue)

— Set <*name*> temperature to average (absolue)

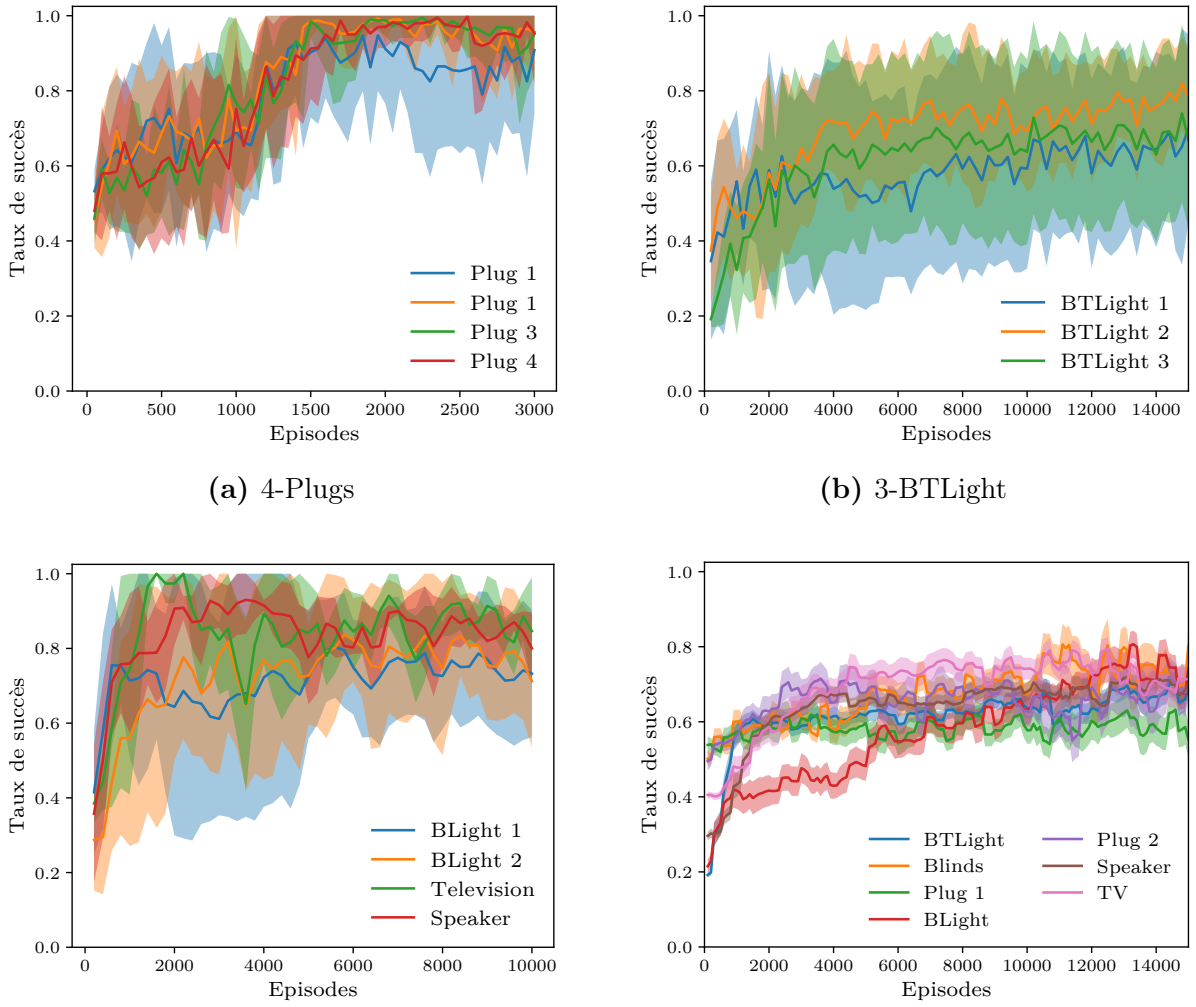
— Set <*name*> temperature to high (absolue)

Les tâches des autres objets sont présentés en Annexe D.

Résultats de simulation

On évalue la capacité d'ILEAD à manipuler chacun de ces objets dans des environnements qui ne contiennent qu'un objet à la fois. On présente les résultats en Figure 4.8. On observe que ILEAD parvient à maîtriser chacun des objets avec un excellent score sur tous les objets (90% en moyenne) en un millier d'épisodes en moyenne. Pour un objet de type Blinds, la performance atteint très vite 100% (quelques centaines d'épisodes), l'état de ces objets est cependant extrêmement simple.

Bien que les performances soient excellentes du point de vue d'un système de RL, cela semble insuffisant pour un usage en conditions réelles. Les systèmes de Deep RL sont relativement sensibles aux valeurs d'hyperparamètres et comme on le montre en Figure 4.9 pour un objet de type BTLight, certains agents parviennent à maintenir un niveau de performance entre 95% et 100% tandis que d'autres ont des performances bien plus erratiques. La seule différence entre ces agents est la *random seed* de la simulation. En conditions réelles, il serait ainsi nécessaire de choisir parmi une collection d'agents.



(a) 4-Plugs

(b) 3-BTLight

(c) Maison connectée facile

(d) Maison connectée difficile

Figure 4.10 – Taux de succès de ILEAD dans différents environnements contenant plusieurs objets. Pour la Figure 4.10d, on représente l'erreur-type et non l'écart-type pour plus de lisibilité.

4.6.2 Manipulation d'ensembles d'objets

On évalue ILEAD dans 4 configurations différentes. Dans les deux premières, l'agent doit manipuler plusieurs objets de même nature. Dans les deux suivants, l'environnement simule une maison connectée contenant plus ou moins d'objets différents. Les environnements sont les suivants :

— 4-Plugs : l'environnement contient simplement 4 Plugs qui peuvent chacun être allumé ou éteint ;

— 3-BTLight : l'environnement est constitué de 3 lampes de type BTLight ;

— HouseEasy : l'environnement simule un salon connecté dans lequel il y a une Télévision, un Speaker et deux Blights ;

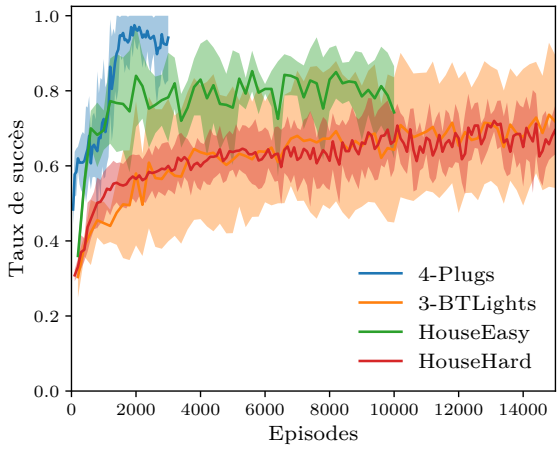


Figure 4.11 – Taux de succès moyen dans les environnements 4-Plugs, 3-BTLight, HouseEasy, et HouseHard

— HouseHard : l'environnement contient 7 objets : deux objets de type Plugs, un Blinds, une Blight, une BTLight, une Télévision et un Speaker.

On représente le taux de succès de ILEAD sur chacun des objets dans chacun des environnements en Figure 4.10 et le taux moyen sur chaque environnement en Figure 4.11.

On observe à nouveau une grande variabilité au sein d'un même environnement en fonction de la simulation. Il semble que cette variabilité augmente lorsque les objets présents sont similaires. Ainsi la variance est très importante dans l'environnement 3-BTLights (Figure 4.10b) et dans l'environnement HouseEasy, la variance pour les 2 BLights est plus grande pour que le Speaker et la Télévision (Figure 4.10c). On observe aussi en Figure 4.11 que la variance globale est bien plus importante pour 3-BTLight que HouseHard. Cela semble indiquer des difficultés de l'agent à manipuler des objets identiques qui ne diffèrent que par leurs noms (et donc leurs descriptions). Pour l'environnement 4-Plugs, on observe que sur les 10 simulations, 5 atteignent et maintiennent une performance maximale et 5 sont fluctuantes.

La comparaison entre les environnements HouseEasy et HouseHard (Figures 4.10c et 4.10d) montrent aussi que plus le nombre d'objets augmente, plus la performance diminue. Ainsi pour certains objets qui sont communs aux deux environnements, la performance atteinte est meilleure dans HouseEasy que HouseHard. On observe aussi dans ces environnements que la performance pour les objets plus simples n'est pas meilleure que pour les objets plus difficiles. On observe même l'inverse. Ainsi l'agent apprend d'abord à manipuler la Télévision, le Speaker ou BTLight avant les Plugs. Cela peut s'expliquer par la distribution des trajectoires collectées concernant chacun des objets. En effet, le Replay Buffer contient plus de transitions pour les objets les plus complexes, car les opportunités d'interaction sont plus nombreuses. La politique d'action est ainsi d'avantage entraînée

sur ces objets que sur les plus simples. On peut envisager de corriger cela à l'aide d'une heuristique quant au Replay (voir Partie 4.7).

Bien que les performances puissent certainement être améliorées, ces expériences valident que l'architecture est capable d'apprendre à manipuler différents objets de différentes natures en même temps.

4.6.3 Capacité de généralisation systématique

Cadre d'expériences

On souhaite évaluer maintenant la capacité généralisation procédurale de notre système, c'est-à-dire à transférer la compétence acquise sur certains objets à d'autres similaires. Pour cela, on définit un environnement dans lequel on restreint l'accessibilité de certains objets pendant l'entraînement. Ainsi, l'agent ne voit qu'une partie des objets et ne reçoit des descriptions de tâches que sur les objets visibles. Pendant les phases de tests, tous les objets sont visibles et certaines tâches assignées à l'agent sont donc relatives aux objets qu'il n'aura pas vu pendant l'entraînement. Ces tâches représentent les tâches de test qui permettent d'évaluer la capacité de généralisation.

On peut ainsi définir un ensemble de tâches d'entraînement (relatives aux objets visibles pendant l'entraînement, ensemble *Train* dans la Figure 4.12), et un ou plusieurs ensembles de tâches de tests (relatives aux objets non visibles pendant l'entraînement). Le taux de succès sur les tâches de tests nous renseignent sur les capacités de généralisation systématiques de l'agent. Il faut noter qu'en phase de tests, l'espace d'observation de l'environnement change de manière assez importante en raison de l'introduction des objets cachés et que cela représente une difficulté supplémentaire même pour les tâches sur lesquels il s'est entraîné.

Dans les expériences qui suivent, on masque aussi certains CHANNELs de certains objets. Les tâches liées aux CHANNELs visibles alimentent l'ensemble d'entraînement et celles relatives aux CHANNELs masqués l'ensemble de test.

Le type de généralisation testée dans ces simulations correspond à la généralisation *zero-shot* évoquée dans IMAGINE. On ne peut pas évaluer de généralisation *n-shot*, car l'agent ne dispose pas de fonction de récompense à partir de laquelle il peut s'entraîner en autonomie.

Choix automatique de l'objet – Les premières simulations que nous avons conduites dans ce cadre ont montré des performances très décevantes. Nous avons identifié que l'origine de ces difficultés proviennent d'un défaut de généralisation relative à l'étape du choix de l'objet. En effet, lorsque l'agent choisit le bon objet sur lequel agir, il réussit

très souvent à accomplir la tâche. Afin de confirmer ces observations, nous avons simplifié l'espace d'action en fournissant à l'agent l'objet sur lequel agir en fonction de la tâche objectif. Le mode d'interaction est toujours séquentiel, mais en 3 étapes et non 4. L'agent choisit ainsi seulement le CHANNEL, l'action puis le paramètre. Cette simplification est motivée par l'heuristique suivante : dans les tâches considérées jusqu'à présent, le nom de l'objet est toujours présent dans la description en langage naturel de la tâche. Il est simple d'identifier à première vue sur quel objet agir. On présente ci-dessous les simulations réalisées dans ce cadre.

Il faut noter que les difficultés de l'agent à identifier le bon objet sur lequel agir rejoignent les observations faites dans la section précédente et constituent l'une des voies d'amélioration de l'agent qui seront discutées plus tard.

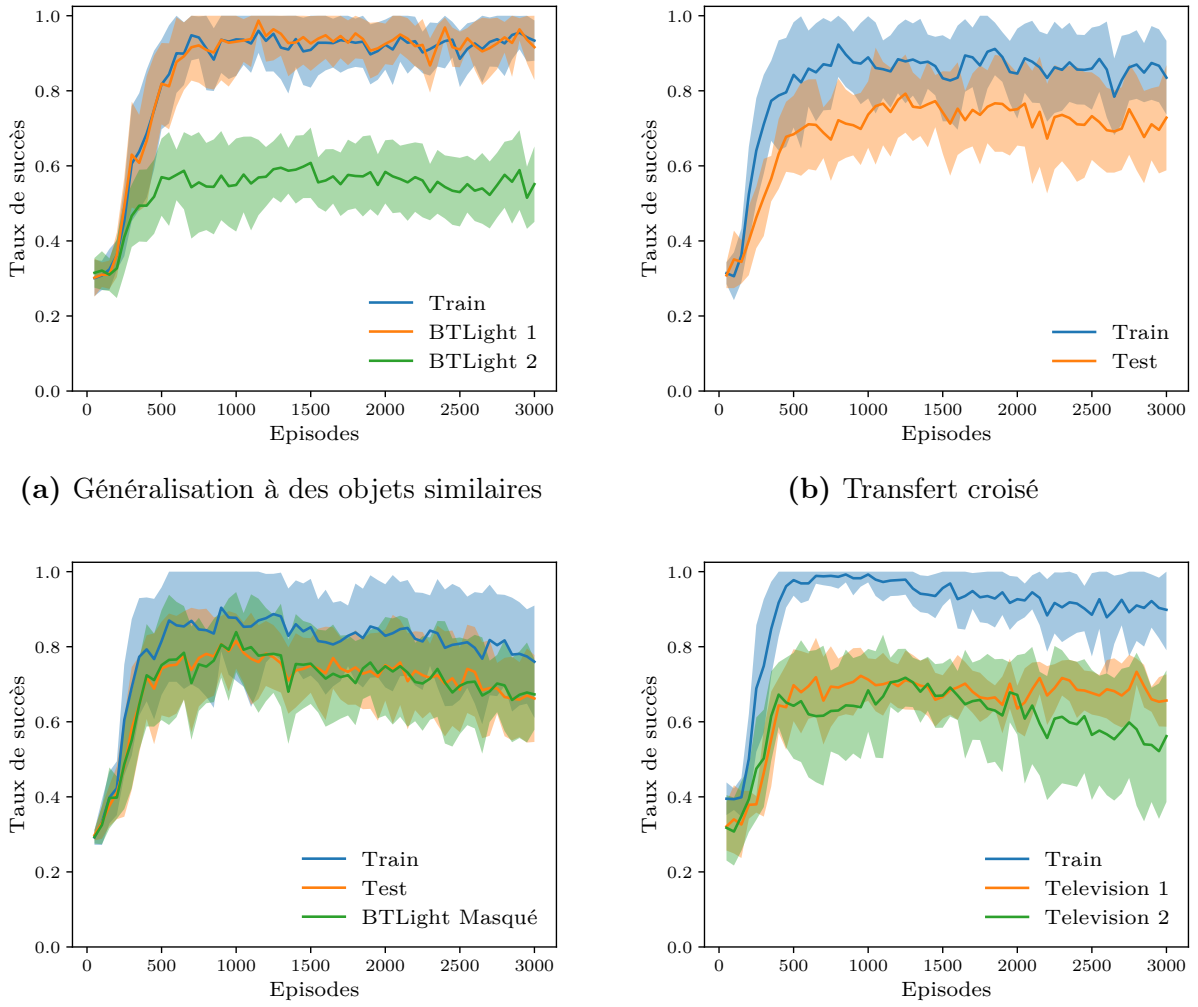
Résultats de simulation

On présente en Figure 4.12 les résultats de 4 types de simulation :

Généralisation à des objets similaires – (Figure 4.12a) L'environnement est composé de trois BTLights dont deux sont exactement identiques à l'exception de leurs noms (et de leurs descriptions) et la troisième diffère par son nom, description et par la description de ces CHANNELs. L'entraînement est réalisé sur une seule des BTLight. On teste ainsi la capacité à transférer les compétences acquises sur un objet à d'autres. On observe que les performances sur l'ensemble de test sont équivalentes à celles sur l'ensemble d'entraînement pour la BTLight identique, mais significativement inférieure pour la BTLight dont les descriptions sont différentes. Cela indique un défaut de généralisation relative à la sémantique des CHANNELs qui semble l'empêcher de distinguer entre les CHANNELs de luminosité et de température (représentées tous deux par un Dimmer).

Transfert croisé entre objets – (Figure 4.12b) L'environnement est composé de 2 BTLights. Pour chacune des BTLights, on masque l'un des CHANNELs : le CHANNEL luminosité pour l'une et le CHANNEL température pour l'autre. On teste ainsi la capacité de l'agent à transférer sa compétence entre objets. On observe que les performances sur l'ensemble de test sont légèrement moins bonnes que sur les tâches d'entraînement. Pour autant ILEAD généralise relativement bien sur ce type de tâches.

Transfert croisé et généralisation à un objet similaire – (Figure 4.12c) Il s'agit d'une combinaison des deux approches précédentes. L'environnement est composé de 3 BTLights. L'un est totalement masqué et pour les deux autres, on masque l'un des CHANNELs (luminosité pour l'un et température pour l'autre). On teste ainsi la capacité



(a) Généralisation à des objets similaires

(b) Transfert croisé

(c) Transfert croisé et généralisation à un objet similaire

(d) Généralisation à de nouveaux objets

FIGURE 4.12 – Évaluation de la généralisation systématique d'ILEAD dans différents contextes : généralisation à des objets similaires, transfert de compétences entre objets et généralisation à de nouveaux objets

de l'agent à combiner sa connaissance partielle d'objets pour manipuler un nouvel objet. En plus de représenter la performance de test, on représente aussi la performance pour la BTLight masquée. On observe que comme précédemment, la performance en test est quelques points en dessous de celle d'entraînement.

Généralisation à de nouveaux objets – (Figure 4.12d) L'environnement est composé de quatre objets : un Speaker simplifié, une Télévision simplifiée et deux Télévisions telles que présentées précédemment, ces dernières étant masquées. Les objets simplifiés sont des objets pour lesquels on a supprimé un ou plusieurs CHANNELs pour **toute** la simulation (différent du masquage). Le Speaker simplifié dispose uniquement de la fonctionnalité du

volume, la Télévision simplifiée ne permet pas de contrôle le volume. Les deux Télévisions sont masquées et pour les manipuler pendant les tests, l'agent doit donc combiner sa connaissance des deux objets simplifiés. Les CHANNELs de la première télévision sont exactement les mêmes (même description notamment) que ceux des objets simplifiés. Pour la seconde télévision, les CHANNELs sont de même type, mais les descriptions sont différentes.

Les performances de test sont cette fois significativement plus faibles que pour les autres simulations. L'agent est en mesure de recombiner ses compétences acquises sur les objets simplifiés, mais avec un certain niveau d'erreur. On observe que les performances sur la seconde Télévision sont légèrement plus faibles que sur la première, cela indique que l'agent parvient à transférer ses compétences malgré les changements dans les descriptions des CHANNELs. On note cependant que les performances pour la seconde Télévision souffrent d'une variabilité plus importante en fonction des simulations.

La différence entre la première et la seconde Télévision est bien moins importante qu'entre la première et la seconde BTLight de la première expérience présentée en Figure 4.12a. Cela s'interprète par le fait que les CHANNELs de la Télévision sont tous de types différents, ce qui les rend plus facilement distinguables, alors que pour la BTLight ils étaient identiques.

4.7 Discussion

Les résultats présentés ici sont encore préliminaires et l'algorithme nécessite encore quelques ajustements notamment pour stabiliser les performances, améliorer la capacité de généralisation systématique et conduire plus de tests quant au comportement d'ILEAD dans des environnements plus réalistes. Ils sont cependant très encourageants. En effet, ils viennent confirmer le potentiel de notre approche. En exploitant la structure d'OpenHAB, on peut définir des environnements suffisamment standardisés pour être manipulées par une même architecture. Les performances en termes de généralisation à de nouveaux objets nécessitent d'être eux aussi approfondies, mais démontrent la capacité de l'agent d'exploiter la sémantique associée au CHANNEL pour transférer sa connaissance entre différents objets.

Les environnements OpenHAB permettent une variabilité importante en termes de nombres d'objets et de tâches réalisables. C'est évidemment un atout pour développer des agents robustes à des conditions changeantes. Cependant, les algorithmes de Deep RL sont très sensibles aux hyperparamètres choisies (P. Henderson et al., 2017), ces derniers étant généralement choisis empiriquement en fonction de l'environnement utilisé. Il est donc difficile de trouver un jeu d'hyperparamètres optimal pour tous les contextes. C'est

notamment l'une des raisons qui a motivé l'emploi d'un algorithme relativement simple comme Double DQN avec Prioritized Experience Replay. Dans les simulations présentées dans ce chapitre, nous avons ainsi utilisé le même jeu d'hyperparamètres. Cela peut expliquer l'instabilité observée dans certains contextes, et parfois la baisse de performance.

Parmi les défis qui restent à adresser, on souhaiterait notamment pouvoir gérer automatiquement le caractère allumé ou éteint d'un objet et apprendre des tâches qui combinent plusieurs objets différents. On évoque ainsi des pistes à explorer et qui peuvent permettre d'étendre les usages d'ILEAD.

Supervision de l'objet choisi

On a constaté que la manipulation d'objets très ressemblants et la généralisation systématique quant au choix de l'objet semble constituer une difficulté importante pour ILEAD. Le choix d'une action est composé de 4 étapes : choix de l'objet, choix du CHANNEL, choix de l'action puis choix du paramètre (voir Figure 4.2). Ce n'est qu'à l'issue de cette dernière étape, une fois que l'action est éventuellement visible pour le partenaire social, que l'agent reçoit une récompense. On ne peut pas superviser indépendamment les choix séquentiels d'ILEAD, car ils sont internes à l'environnement numérique et l'utilisateur n'en a pas connaissance. On peut cependant envisager de fournir des récompenses supplémentaires à l'agent au moment du choix de l'objet. En effet, le choix de l'objet fait référence à quelque chose d'interprétable dans l'environnement physique. L'agent pourrait ainsi périodiquement demander confirmation au partenaire social si l'objet sur lequel il compte agir permet bien de réaliser la tâche souhaitée. On dispose ainsi de deux canaux de supervision différents et complémentaires.

Apprentissage de fonctions de récompenses internes

À la différence d'IMAGINE, ILEAD ne s'appuie pas sur une fonction de récompense interne, mais directement sur les descriptions fournies par le partenaire social. Pour autant, ILEAD pourrait apprendre une fonction de récompense pour être en mesure de s'entraîner en autonomie sans la présence du partenaire social. Comme pour IMAGINE, cette fonction serait en mesure de déterminer si dans un état donné, une tâche est réalisée. En entraînant le modèle de langage avec la fonction de récompense et non la politique d'action, on peut aussi espérer disposer d'une représentation des tâches qui soit stable plus rapidement. En effet, l'apprentissage de la fonction de récompense étant plus facile, on peut s'attendre à ce qu'elle soit plus rapide.

On a évoqué précédemment la possibilité d'interroger le partenaire social pour valider l'objet sur lequel l'agent souhaite agir. Il serait aussi possible d'apprendre une seconde fonction de récompense interne qui apprendrait à valider l'objet. Outre la possibilité de

s'entraîner en autonomie, en étant en mesure d'associer une tâche à l'objet visé, on pourrait proposer une heuristique de Replay qui, basée sur cette dernière, pourrait maintenir dans les données rejouées un certain équilibre entre les objets concernés et répondre ainsi à l'observation faire en Partie 4.6.2.

Utilisation de Graph Neural Network

La structure de l'environnement OpenHAB permet d'envisager de nouvelles manières de représenter l'environnement. En effet dans la représentation actuelle, l'agent traite chacun des CHANNELs indépendamment les uns des autres même au sein d'un objet. L'état de l'environnement peut assez naturellement être représenté sous la forme d'un graphe et l'on pourrait ainsi envisager l'utilisation de Graph Neural Networks qui prennent un place grandissante (Z. Wu et al., 2021). On pourrait ainsi disposer d'une représentation de l'environnement plus unifiée et envisager d'apprendre des tâches impliquant plusieurs objets à la fois.

Utilisation d'une interface graphique

L'une des idées qui a motivé le mode d'interaction séquentiel avec l'interface d'OpenHAB est la simulation d'une forme de navigation par sous-menus à laquelle les utilisateurs d'un logiciel sont familiers. Bien qu'on ne le démontre pas dans ces travaux, on considère que cela permet de créer une forme d'*embodiment numérique*. Cela permet aussi d'envisager un nouveau mode d'interaction avec l'utilisateur et d'envisager des démonstrations pour faciliter l'apprentissage de l'agent. En effet, la structure d'OpenHAB permet d'envisager la génération automatique d'une interface graphique très simple qui simulerait une navigation en sous-menus. L'agent pourrait alors solliciter un utilisateur en lui demandant d'effectuer la séquence de choix relative à une tâche et exploiter cette démonstration pour apprendre par imitation.

Apprentissage par renforcement hiérarchique

Les résultats obtenus sur la généralisation systématique semble indiquer que l'agent transfère bien sa compétence entre les CHANNELs. On peut envisager de modifier notre approche du problème en s'appuyant sur un modèle d'apprentissage par renforcement hiérarchique (Kulkarni et al., 2016 ; Pierrot et al., 2019). Dans ces systèmes, plusieurs politiques d'actions sont apprises à des échelles différentes. On pourrait ainsi concevoir une architecture dans laquelle un premier étage hiérarchique apprend à manipuler les CHANNELs, un second les objets et éventuellement un troisième qui apprend à effectuer

des tâches impliquant plusieurs objets. Y. Jiang et al. (2019) a notamment montré que le langage pouvait jouer le rôle d'abstraction entre les politiques de bas et haut niveaux.

4.8 Intégration de Jarvis, Ilead et AidMe

Dans cette dernière partie, on propose une architecture ainsi qu'un exemple d'utilisation qui intègre AidMe, Ilead et Jarvis (Delgrange et al., 2020). Jarvis dispose de deux modes d'apprentissage : un mode d'apprentissage par démonstration sur une interface graphique et un mode d'apprentissage par composition d'instructions connues. Dans cette intégration, on ne sollicite plus l'utilisateur pour effectuer des démonstrations et on ne s'appuie que sur le second mode d'apprentissage. C'est là qu'intervient Ilead qui fournit à Jarvis un ensemble d'instructions en langage naturel connues et que la politique d'action sait réaliser. Le rôle d'Ilead est donc d'apprendre un ensemble d'actions élémentaires que l'utilisateur peut ensuite composer pour apprendre à Jarvis des procédures de plus haut niveau. AidMe permet à l'utilisateur de s'exprimer aussi librement qu'il le souhaite en convertissant les procédures de plus haut niveau, mais aussi les instructions élémentaires dans une formulation connue de Jarvis et de Ilead. La Figure 4.13 représente l'intégration des trois agents selon deux étapes : un pré-entraînement en simulation (partie haute) puis l'usage en conditions réelles (partie basse).

4.8.1 Pré-entraînement d'Ilead en simulation

Dans l'idéal, on souhaiterait voir Ilead apprendre ces actions élémentaires directement au contact de l'utilisateur. Dans cette éventualité, Ilead pourrait ne pas se limiter à l'apprentissage d'actions élémentaires, mais à tout type d'action. Cependant, pour plusieurs raisons, cela ne nous semble pas réaliste.

Tout d'abord, les algorithmes de RL sont relativement lents dans leur apprentissage et comme on l'a vu dans la partie précédente, même pour un ensemble relativement simple d'objets, plusieurs centaines d'épisodes sont nécessaires pour manipuler avec suffisamment de précision les objets.

De plus, difficile d'imaginer qu'un utilisateur laisserait un système artificiel « jouer » avec ses objets connectés (dont il verrait les effets dans son salon) tout en acceptant de lui fournir un retour en langage naturel.

Enfin, on a observé que l'apprentissage d'Ilead présentait une certaine instabilité, les performances d'apprentissage varient en effet selon les simulations et certains agents ont une tendance à l'oubli, c'est-à-dire que leurs performances baissent après avoir atteint un très bon niveau.

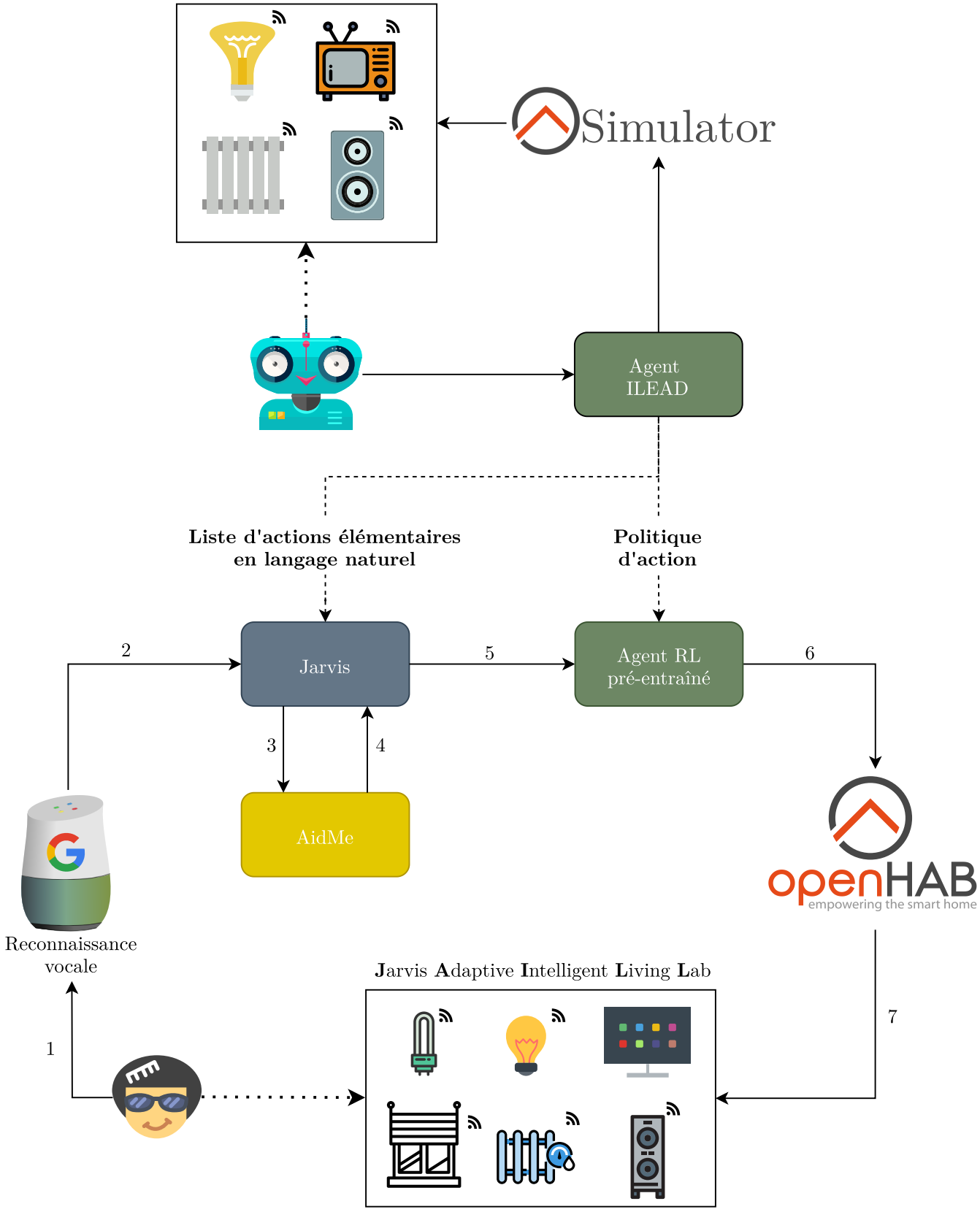


Figure 4.13 – Intégration Jarvis, ILead, AidMe– Entraînement en simulation de ILead puis usage en conditions réelles.

On propose donc de réaliser cet apprentissage en simulation grâce au simulateur et à le partenaire social sur un ensemble d'objets. Cela permettra notamment de sélectionner une politique d'action qui démontre une certaine stabilité dans les performances. Les instructions ainsi apprises sont ensuite transmises à Jarvis et constituent l'ensemble des actions élémentaires à sa disposition. On peut envisager deux types d'apprentissage :

— **Apprentissage sur une simulation des objets en conditions réelles** : Le plus simple si l'on connaît les véritables objets utilisés en conditions réelles est d'implémenter au sein du simulateur et d'effectuer directement l'apprentissage sur ces objets. Cela garantit un certain niveau de performance, mais se révèle contraignant à l'usage puisqu'il faut implémenter chaque nouvel objet.

— **Apprentissage sur des objets fictifs** : Une autre option consiste à implémenter dans le simulateur un ensemble d'objets suffisamment généraux et représentatifs sur lequel Ilead est entraîné. On exploite ensuite les capacités de généralisation systématique pour transférer les apprentissages en simulation aux objets en conditions réelles. Au vue des performances actuelles d'Ilead, on ne peut pas encore s'appuyer de manière robuste sur cette approche, mais elle présente l'avantage certain de pouvoir disposer d'agents qui s'adaptent naturellement à n'importe quelle configuration.

4.8.2 Composition d'instruction et usage en conditions réelles

Une fois que l'on dispose d'un agent Ilead qui a appris une série d'instructions relatives à des objets on peut intégrer la politique d'action à un agent en conditions réelles. Ilead fournit alors à Jarvis l'ensemble des phrases en langage naturel décrivant les tâches qu'il sait accomplir dans l'environnement. Ces phrases constituent pour Jarvis des actions élémentaires qu'il peut composer selon les instructions de l'utilisateur.

Lorsque l'utilisateur souhaite apprendre une nouvelle procédure à Jarvis, il décrit par des phrases en langage naturel les étapes à réaliser. Jarvis mémorise ces étapes, les décompose en étapes élémentaires et les transmets à Ilead pour exécution dans l'environnement. AidMe peut-être employé pour associer les phrases en langage naturel inconnues avec leurs équivalents connus par Jarvis. On représente cet usage dans la partie basse de la Figure 4.13.

4.8.3 Scénario d'usage

On détaille maintenant deux scénarios d'usage montrant l'intérêt que peut représenter l'intégration entre Ilead, Jarvis et AidMe. On considère un environnement représentant le salon d'un utilisateur composé d'une télévision, d'une enceinte, de stores, d'un luminaire

et d'un plafonnier. On suppose qu'Ilead a donc appris à maîtriser ces différents objets et fournit à Jarvis l'ensemble d'actions élémentaires suivantes :

- Éteins le luminaire.
- Allume le luminaire.
- Éteins le plafonnier.
- Allume le plafonnier.
- Ferme les stores.
- Ouvre les stores.
- Mets la chaîne 1 sur la télévision.
- Mets la chaîne 2 sur la télévision.
- Mets la chaîne 3 sur la télévision.
- Joue le film sur la télévision.
- Mets le film en pause sur la télévision.
- Mets le volume de l'enceinte au niveau bas.
- Mets le volume de l'enceinte au niveau moyen.
- Mets le volume de l'enceinte au niveau fort.
- Augmente le volume de l'enceinte.
- Baisse le volume de l'enceinte.

L'utilisateur souhaite d'abord éteindre toutes les lumières dans le salon :

Utilisateur — Éteins toutes les lumières du salon.
Assistant — Avec plaisir ! Pourriez-vous me détailler en quoi cela consiste ?
Utilisateur — Éteins le luminaire.
Assistant — Ok ! Et ensuite ?
Utilisateur — Éteins aussi le plafonnier.
Assistant — Voulez-vous dire : Éteins le plafonnier ?
Utilisateur — Oui.
Assistant — Ok ! Et ensuite ?
Utilisateur — C'est fini.
Assistant — C'est noté !

La première instruction employée correspond à une action élémentaire apprise par Ilead sur le luminaire. La seconde instruction diffère légèrement et AidMe peut la réinterpréter pour la mettre sous la forme exacte connue par Ilead *Éteins le plafonnier.* Jarvis transmets ces deux instructions normalisées à Ilead qui peut les exécuter. On peut noter que dans ce cas précis, il est probable qu'Ilead aurait pu gérer correctement le léger écart de forme verbale employé.

Maintenant, l'utilisateur souhaite visionner un film sur une chaîne particulière. Il peut décrire la manière dont il souhaite que l'assistant prépare son environnement et réutiliser notamment des procédures déjà apprises :

Utilisateur — Prépare le salon pour le film sur la 2.
Assistant — Avec plaisir ! Pourriez-vous me détailler en quoi cela consiste ?

Utilisateur — Ferme les stores.
Assistant — Ok ! Et ensuite ?
Utilisateur — Mets la chaîne 2 sur la télévision.
Assistant — Ok ! Et ensuite ?
Utilisateur — Mets le volume de l'enceinte au niveau moyen.
Assistant — Ok ! Et ensuite ?
Utilisateur — Éteins les lampes du salon.
Assistant — Voulez-vous dire : Éteins toutes les lampes du salon ?
Utilisateur — Oui.
Assistant — Ok ! Et ensuite ?
Utilisateur — C'est fini.
Assistant — C'est noté !

Les premières instructions de l'explication correspondent à des actions élémentaires connues par ILEAD. En revanche, la dernière instruction « Éteins les lampes du salon. » fait référence à la procédure précédemment apprise. Elle n'est cependant pas exprimée dans la forme exacte vue pendant l'apprentissage. C'est alors le rôle d'AIDME de la convertir sous la forme « Éteins toutes les lumières du salon. ». JARVIS peut alors l'interpréter et transmettre à ILEAD les instructions élémentaires correspondantes : [Éteins le luminaire., Éteins le plafonnier.].

4.8.4 Implémentation technique

L'intégration des trois agents JARVIS, ILEAD et AIDME n'a pas encore été réalisée techniquement dans le scénario évoqué ci-dessous. Cependant, nous avons déjà réalisé l'intégration de JARVIS et AIDME (voir Partie 2.5) en utilisant une API REST pour communiquer entre les deux modules.

Tôt dans le projet, nous avons vu l'opportunité d'intégrer un agent comme JARVIS avec un agent de Deep RL ayant appris à effectuer un ensemble d'actions élémentaires. Nous avions donc validé cette idée en dotant une version pré-entraînée de l'agent LE2 d'une API REST lui permettant d'être contrôlée par un système similaire à JARVIS. Lorsque JARVIS se connecte à LE2, il peut obtenir l'ensemble des actions élémentaires maîtrisées. Celles-ci sont envoyées à JARVIS sous la forme d'un couple (identifiant, instruction) où l'instruction est exprimée en langage naturel. JARVIS peut ensuite exécuter des commandes en envoyant simplement l'identifiant de l'instruction à LE2. Dans le contexte de l'environnement *ArmToolsToys*, on pouvait ainsi apprendre à LE2 à déplacer tous les objets au même endroit (le cube aimanté et le cube à scratch en haut à droite par exemple).

Aucun obstacle technique ne s'oppose donc à l'intégration des trois agents.

4.9 Conclusion

Dans ce chapitre, nous avons montré comment des algorithmes de Deep RL pouvaient être adaptés au cas d'usage des assistants apprenants par interaction. Cela nécessite tout d'abord de disposer d'un environnement dont la structure est exploitable par une architecture neuronale et qui soit adaptée à une généralisation systématique. Nous avons ainsi proposé un simulateur pour la plateforme OpenHAB qui permet d'inter-opérer des objets connectés à travers une interface commune. Le simulateur peut-être employé pour étudier la manière dont les agents de Deep RL se comporte dans ce type d'environnements. Il est entièrement configurable et sera mis à disposition de la communauté prochainement.

L'architecture proposée permet de gérer un espace d'observation avec un nombre d'objets variables et un espace d'actions mouvants. ILEAD est capable d'apprendre à manipuler différents objets ainsi que différentes combinaisons d'objets avec des performances encourageantes. De même,les performances en termes de généralisation systématique, bien qu'encore limitantes pour un usage en conditions réelles, sont prometteuses. ILEAD doit encore subir certaines améliorations notamment pour garantir la stabilité de l'apprentissage et un niveau de performance quasi-parfait avant d'être effectivement employé en conditions réelles. Enfin, nous avons proposé une architecture ainsi qu'un scénario d'usage prospectif permettant d'intégrer au sein d'un même assistant apprenant les différents agents développés dans ces travaux.

Conclusion et Perspectives

We can only see a short distance ahead, but we can see plenty there that needs to be done.

— Alan Turing, *Mind,* 1950

Conclusion générale

Dans ces travaux, nous avons exploré différentes facettes des capacités de généralisation que l'on peut attendre d'assistants numériques apprenants. À la croisée de l'apprentissage par interaction et de l'apprentissage développemental, nous avons souhaité développer des systèmes qui font émerger leurs propriétés de généralisation à partir de la supervision langagière d'un utilisateur ou plus généralement d'un partenaire social.

Nous avons d'abord proposé AidMe afin de répondre au défaut de généralisation linguistique que présentent ces systèmes. AidMe est un module de NLU qui permet à un assistant d'intégrer de nouvelles classes d'intentions au fur et à mesure qu'elles sont introduites par un utilisateur. Il s'appuie sur un modèle de similarité sémantique pour comparer les requêtes de l'utilisateur entre elles et ne nécessite pas d'apprentissage préalable.

Nous avons ensuite exploré la question de la généralisation procédurale, c'est-à-dire généraliser les procédures apprises à de nouveaux objets. C'est une problématique qui est formulée sous l'angle de la généralisation systématique par la communauté de Deep RL (Hill et al., 2019 ; Ruis et al., 2020). Le langage permet de formuler les tâches réalisables de manière compacte et abstraite. En apprenant à représenter ces tâches directement à partir de son expérience dans l'environnement, l'agent peut généraliser immédiatement sur de nouvelles tâches formulées à partir du même vocabulaire. L'agent peut aussi mettre à profit ces représentations et certaines propriétés de composition du langage pour exploiter les affordances des objets de l'environnement et apprendre à réaliser de nouvelles tâches en totale autonomie. On a montré que la structure de l'environnement ainsi que celle de

l'architecture jouaient un rôle central dans le développement des capacités de généralisation systématique de ces agents.

Enfin, nous avons appliqué cette approche à un environnement domotique en lien avec les applications industrielles envisagées par Cloud Temple. Ces environnements posent des difficultés spécifiques, notamment pour modéliser des environnements numériques manipulables par des algorithmes de Deep RL et en raison d'une asymétrie perceptuelle entre l'assistant et l'utilisateur. L'architecture proposée dans le cadre du simulateur d'OpenHAB démontre des résultats préliminaires encourageants qui nous permettent de proposer l'usage intégré d'AidMe, Jarvis et Ilead en conditions réelles.

On peut aussi voir en filigrane de ces travaux l'influence des Sciences Cognitives. Grandement inspiré par Vygotsky notamment, nous avons aussi montré comment les idées développées depuis plusieurs décennies en psycholinguistique développementale trouvent un écho des plus naturels en Intelligence Artificielle, même lorsque l'on cherche à développer des assistants numériques apprenants. Au-delà des théories, la psycholinguistique reste un vivier d'idées encore trop peu exploitées et pourtant utilisables pour développer des mécanismes d'apprentissage artificiel.

Les défis restent cependant nombreux pour disposer d'un assistant qui concilie efficacement apprentissage par interaction rapide et capacité de généralisation suffisamment large. Après avoir listé les articles publiés dans le cadre de ces travaux, on revient dans ces dernières pages sur certains de ces défis et on évoque certaines pistes de réflexion à travers trois grands thèmes. On souhaiterait d'abord conduire nos systèmes vers des contextes plus réalistes permettant d'intégrer un humain directement dans la boucle d'apprentissage. Après avoir été grandement inspiré par les travaux de Vygotsky et sa vision du langage comme un outil du développement cognitif, on se tourne vers Tomasello et la notion d'attention partagée (Carpenter et al., 1998) qui offre une nouvelle manière d'envisager l'apprentissage en interaction. Enfin, on s'interroge avec pragmatisme sur les opportunités qu'offrent les approches hybrides symboliques et connexionnistes dans le cadre des assistants apprenants qui doivent concilier efficacité dans l'apprentissage et généralisation.

Liste des publications

— Lair, N., Colas, C., Portelas, R., Dussoux, J.-M., Dominey, P. & Oudeyer, P.-Y. (2019). Language grounding through social interactions and curiosity-driven multi-goal learning. *ViGIL workshop (NeurIPS 2019) : Visually Grounded Interaction and Language*

— Lair, N., Delgrange, C., Mugisha, D., Dussoux, J.-M., Oudeyer, P.-Y. & Dominey, P. F. (2020). User-in-the-loop adaptive intent detection for instructable digital

assistant. *International Conference on Intelligent User Interfaces, Proceedings IUI*, 116-127. DOI : 10.1145/3377325.3377490

— Colas, C., Karch, T., Lair, N., Dussoux, J.-M., Dominey, P. F. & Oudeyer, P.-Y. (2020). Language as a Cognitive Tool to Imagine Goals in Curiosity-Driven Exploration. *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*

— Uchida, T., Lair, N., Ishiguro, H. & Dominey, P. F. (2021). A Model of Online Temporal-Spatial Integration for Immediacy and Overrule in Discourse Comprehension. *Neurobiology of Language*, *2*(1), 83-105. DOI : 10.1162/nol_a_00026

Vers des systèmes plus réalistes

Interaction humaine vs interaction simulée

Comme évoqué en Introduction, le domaine de l'apprentissage par interaction rassemble différentes communautés. Pour certains, la présence d'un véritable humain dans la boucle d'apprentissage est indispensable. C'est notamment la position défendue par Laird et al. (2017) qui propose la définition du domaine d'Interactive Task Learning comme le fait de donner la possibilité à un système d'apprendre de nouvelles tâches, pour lesquelles il n'a pas été programmé, au moyen d'interactions aussi naturelles que possible. Notre démarche, modulo une acceptation souple de la notion « d'interaction naturelle », s'inscrit dans cette démarche. Cependant, les auteurs restreignent leur définition exclusivement aux agents qui apprennent en *one-shot*, c'est-à-dire pour lesquels une seule interaction suffit pour apprendre une nouvelle tâche. Ils le justifient en s'inspirant de la manière dont les humains apprennent entre eux, arguant qu'un exemple suffit généralement pour un humain. Ils excluent ainsi de facto les systèmes d'apprentissage statistique au motif qu'ils apprennent trop lentement.

Nous souhaitons au contraire étendre ce cadre aux systèmes exploitant le ML. En effet, si les adultes sont capables d'apprendre en *one-shot*, ce n'est qu'après plusieurs années de développement biologique et cognitif. Dans le cadre de l'approche développementale, on envisage l'agent dont les compétences émergent de son expérience et qui ne possède pas nécessairement les capacités d'apprendre en *one-shot* dès le début. Plus qu'un pré-requis, nous considérons que cette capacité doit être un objectif. De plus, s'il reste pour le moment difficile d'apprendre en *one-shot* pour un système de ML, certains sont capables d'apprendre en *zero-shot* c'est-à-dire de généraliser sur des tâches pour lesquelles ils n'ont jamais vu d'exemple. Évaluer la généralisation systématique permet de mesurer cette capacité.

Partant du constat qu'il n'était pas réaliste de superviser l'apprentissage des agents de Deep RL par de véritables utilisateurs, nous avons proposé en Partie 4.8 une méthode pour exploiter l'apprentissage effectué en simulation dans le cadre de véritables interactions utilisateurs. Pour autant, il serait intéressant de déterminer dans quelles conditions cet apprentissage pourrait s'effectuer directement au contact des utilisateurs. On détaille par la suite des pistes qui permettraient d'aller dans cette direction.

Un langage naturel plus naturel

Pour le moment, les systèmes comme Imagine et Ilead ont été testés avec un langage simple dans lequel les phrases sont essentiellement constituées d'un verbe indiquant le type de transformation de l'environnement attendu, d'une expression permettant d'identifier l'objet qui doit subir cette transformation et éventuellement d'un paramètre permettant de préciser l'action. Dans le cadre de l'intégration présentée en Partie 4.8, le rôle d'AidMe est de convertir le langage de l'utilisateur vers ces formes simplifiées pour permettre à Ilead d'exécuter correctement ces actions élémentaires.

Pour être supervisé par de vrais utilisateurs, on doit d'abord étudier la manière dont ces systèmes gèrent un langage naturel plus réaliste (Sumers et al., 2020). Imagine et Ilead apprennent leur propre modèle de langage concomitamment à la politique d'action. Cela peut permettre à ces systèmes d'acquérir une notion de similarité sémantique et de disposer d'une certaine robustesse quant à la formulation du but en langage naturel. L'une des pistes à explorer pourrait consister à laisser l'oracle formuler de plusieurs manières différentes un même objectif. Ainsi, l'agent serait naturellement en mesure de généraliser sur la variabilité des formes verbales employées par l'utilisateur. On peut penser que l'utilisation de modèles de langage pré-entraînés peut conduire à des gains de performance substantiels, l'agent ayant ainsi connaissance dès le début de son apprentissage d'une forme de sémantique du langage.

Pour Jarvis, chaque intention de l'utilisateur était associée à une procédure et possédait des arguments propres nécessaires à sa réalisation. Cette représentation discrète de l'espace d'intention de l'utilisateur est un frein aux possibilités de généralisation. Ainsi, les requêtes « Allume toutes les lampes de la cuisine » et « Allume toutes les lampes de la cuisine et du salon » sont catégorisées comme des intentions différentes par AidMe, car leurs arguments sont différents. Avec AidMe, nous avons introduit une notion de distance entre les classes d'intentions, calculée par un modèle de similarité sémantique appris en cours d'interaction. Ainsi, AidMe est en mesure d'apprécier la proximité sémantique entre deux intentions et de généraliser sur les formes verbales de l'utilisateur. Pour autant, du point de vue de la politique d'action, chaque intention est traitée indépendamment. L'utilisation de formes verbales variées pour décrire chacune des tâches de la politique d'action d'un agent

comme ILEAD, permet de s'affranchir de la notion de classe d'intention et de se placer dans un espace d'intention continue où la proximité sémantique de deux formes verbales induit naturellement une proximité procédurale, et peut permettre une généralisation à la fois linguistique et procédurale.

Améliorer la *sample efficiency*

L'une des limites des algorithmes de Deep RL aujourd'hui est leur manque d'efficacité en termes d'usage de données (*sample efficiency*). C'est la raison principale qui empêche l'apprentissage d'un système comme ILEAD en conditions réelles. Ils ont en effet besoin à la fois de s'exercer dans l'environnement pour observer les effets de leurs actions, et de recueillir une forme de récompense quant à leurs réussites. Concernant cette dernière, nous avons montré que l'apprentissage d'une fonction de récompense interne permet de diminuer le retour du partenaire social en permettant à un agent de s'entraîner en autonomie à partir du signal de la fonction de récompense (Partie 3.7.6). Cependant, le système a toujours besoin d'agir dans un environnement pour comprendre les effets de ses actions. En conditions réelles, cela peut s'avérer très ennuyeux pour un utilisateur qui verrait ses objets fonctionner de manière intempestive. C'est notamment l'une des raisons qui nous a poussé à utiliser un simulateur.

L'amélioration de la *sample efficiency* fait partie des grands défis en Deep RL (Y. Yu, 2018). On évoque ici certaines idées que l'on souhaiterait investiguer dans le cadre d'un agent comme ILEAD :

Apprendre un modèle de l'environnement

L'une des problématiques des systèmes de Deep RL est leur besoin d'observer les effets de leurs actions. On peut s'inspirer de l'approche de Ha et Schmidhuber (2018) qui apprend un modèle de la dynamique de jeux Atari pour apprendre ensuite une politique d'action qui apprend à jouer à ces jeux non pas en jouant effectivement aux jeux mais en exploitant le modèle appris. Dans le contexte d'assistants domotiques, on peut exploiter la même idée. L'avantage d'un environnement structuré comme OpenHAB est que l'on peut envisager d'apprendre un modèle général en simulation et de le *fine-tuner* en conditions réelles chez un utilisateur. Une fois la dynamique de l'environnement et une fonction de récompense interne apprise, la politique d'action pourrait être en partie apprise en simulation. Cela réduirait naturellement les interactions nécessaires entre l'agent et l'environnement physique réel de l'utilisateur.

Supervision distribuée

L'intérêt des assistants apprenants est la possibilité pour l'utilisateur de personnaliser très finement son assistant à ces usages. Pour autant, on peut légitimement penser qu'une partie de l'apprentissage est mutualisable entre plusieurs utilisateurs. Dans l'intégration proposée en Partie 4.8, on distingue les actions élémentaires, apprises en simulation, des actions de haut niveau, apprises par explication auprès de l'utilisateur. Les actions élémentaires sont donc par définition les mêmes pour tous les utilisateurs. Si on souhaite les apprendre non plus en simulation mais directement en conditions réelles, il semble donc raisonnable d'essayer de mutualiser cet apprentissage en partageant les données collectées par une instance particulière de l'assistant aux autres assistants.

Nous avons déjà évoqué cette idée pour le partage d'un modèle de similarité sémantique en Partie 2.6.2. Pour la politique d'action, on peut envisager deux formes de partage entre utilisateurs :

1. *Le partage d'une partie des réseaux neuronaux* : Les premières couches des réseaux de neurones de la politique d'action calculent une représentation de l'environnement et de l'espace d'actions. On peut envisager de partager tout ou partie de ces couches entre plusieurs instances.
2. *Le partage de certaines données d'interaction avec l'utilisateur et l'environnement domotique* : Les données collectées par certaines instances de la politique d'action pourraient être utilisées par d'autres en utilisant un mécanisme de Replay similaire à ceux employés pour IMAGINE ou ILEAD. L'enjeu est de déterminer quelles instances peuvent mutualiser leurs données. En effet, deux instances qui manipulent des configurations d'objets très différentes n'ont peut-être rien à s'apporter. On peut envisager une approche similaire à celle développée par S. M. Nguyen et Oudeyer (2018) qui emploient des techniques d'*active learning* pour déterminer le partenaire social le plus approprié à suivre en fonction d'une mesure du progrès en apprentissage.

Notons cependant qu'en pratique, ces idées soulèvent des questions quant aux partages des données des utilisateurs et à la protection de la vie privée qui ne sont pas facilement résolues.

Enrichir les interactions

On a montré notamment dans les Chapitres 1 et 3 que les problématiques en Intelligence Artificielle trouvaient un écho assez naturel en Sciences Cognitives et que l'on pouvait s'inspirer du développement humain pour concevoir des systèmes apprenants (Colas et al., 2020). La multiplication récente des travaux qui mêlent les deux domaines est

très prometteuse (Lake et al., 2017 ; Dupoux, 2018 ; Matthew Botvinick et al., 2019 ; James L. McClelland et al., 2020). On souhaite donc approfondir les liens avec les Sciences Cognitives.

On est en effet forcé d'admettre que le type d'interaction envisagée dans les travaux présentés est peu naturel. L'interaction est univoque, d'un oracle / partenaire social vers l'agent. De plus, il s'agit exclusivement d'une supervision langagière. Cette supervision faible, requérant peu d'implications du partenaire, permet à l'agent d'associer une phrase avec d'autres éléments (d'autres phrases pour AidMe, l'état de l'environnement pour Imagine et Ilead) et d'y détecter des régularités pour apprendre le sens des mots employés et adapter son comportement aux instructions d'un utilisateur. Elle répond à certaines exigences des assistants autonomes de ne pas importuner trop fréquemment l'utilisateur.

On peut cependant envisager d'autres formes d'interaction plus naturelles et qui soient complémentaires de celles utilisées jusqu'à présent. Les modes d'interaction entre un enfant et les adultes sont en effet diversifiés et la richesse de l'interaction entre un enfant et ses parents est un facteur essentiel pour le développement de ses capacités cognitives (Bruner, 1983). On peut distinguer deux formes d'interaction. Celles dites *passives* du point de vue de l'adulte lorsque l'enfant imite ou écoute l'adulte ou *actives* lorsque l'adule s'implique dans la relation par le jeu ou le dialogue.

Partager l'attention de l'utilisateur

Nous avons montré le rôle crucial de l'attention partagée dans le développement de l'enfant (Partie 1.5.2). Elle lui permet en effet de concentrer naturellement son attention sur un objet d'intérêt et d'inférer les raisons qui motivent les actions de l'adulte. On peut envisager de nouvelles opportunités d'interaction à travers le prisme de l'attention partagée. En effet, la possibilité de focaliser l'attention d'un agent apprenant est très intéressante pour l'apprentissage. Carpenter et al. (1998) distinguait trois phrases dans le développement de cette forme d'attention et notamment le fait de suivre l'attention d'autrui ou de la diriger. Chacune de ces phases peut donner lieu à de nouvelles formes d'interaction entre un assistant apprenant et un utilisateur.

Suivre l'attention de l'utilisateur

Donner la possibilité à l'agent de suivre l'attention de l'utilisateur est une opportunité pour l'agent d'apprendre en observant. On peut envisager un cadre d'interaction dans lequel l'assistant est en mesure d'observer les effets des actions de l'utilisateur dans son environnement numérique. Il peut alors se servir de ces trajectoires comme des

modèles à reproduire, et notamment identifier les trajectoires les plus fréquentes. Bien qu'il n'en connaisse pas le sens, il sait qu'elles représentent quelque chose de pertinent pour l'utilisateur et peuvent faciliter l'exploration de l'environnement. Ce nouveau mode d'interaction présente l'avantage de ne pas représenter une charge supplémentaire pour l'utilisateur.

Diriger l'attention de l'utilisateur

On peut envisager une seconde forme d'interaction, à l'initiative de l'agent, lui en permettant de diriger l'attention de l'utilisateur sur un objet ou une tâche bien choisie. En estimant sa compétence, l'agent peut identifier les objets ou les tâches qui lui posent des difficultés particulières et demander explicitement des démonstrations ou des explications. On s'inspire ici des approches exploitant la motivation intrinsèque pour orienter l'apprentissage d'un agent (Colas et al., 2019a ; Lair et al., 2019). Ces systèmes sont capables de maintenir un modèle interne de leur compétence ou de leur progrès dans le but de donner plus d'autonomie à l'agent. En exploitant ces mêmes mesures, on peut aussi orienter les modes d'interaction de l'agent. Afin d'éviter d'importuner trop fréquemment, on peut doter l'agent d'un budget d'interaction qu'il doit apprendre à utiliser efficacement (Misra et al., 2018).

Apprentissage interactif et curriculum

On a souligné en Partie 1.5.5 le rôle que jouait l'enseignant ou le parent dans le développement des apprentissages de l'enfant en le maintenant sur une trajectoire d'apprentissage optimale. Souvent transposé sous la forme de l'apprentissage par curriculum en Intelligence Artificielle (Bengio et al., 2009), peu de travaux ont cependant étudié une forme d'apprentissage par curriculum interactif (Narvekar et al., 2020). Pourtant, cela pourrait permettre de mieux identifier la manière dont les humains conçoivent un curriculum en interaction avec un système apprenant. Cela constitue aussi une opportunité intéressante pour notre assistant apprenant qui pourrait bénéficier d'un guidage actif de l'utilisateur.

Cependant, dans le cadre d'assistant apprenant par interaction, l'implication de l'utilisateur doit rester raisonnable. Celle-ci doit être motivante et rester peu intrusive. Cela ouvre aussi des questions intéressantes sur la manière dont un agent peut conserver l'intérêt de l'utilisateur et le motiver à l'aider.

On a ainsi envisagé 4 cadres d'interactions différents dans lesquels l'implication de l'utilisateur est de plus en plus nécessaire :

1. *Utilisateur observé par l'agent.* Cela permet à l'agent d'acquérir des démonstrations mais pas d'informations langagières relatives aux tâches.
2. *Agent observé par l'utilisateur.* C'est le cadre présenté dans ces travaux.
3. *Utilisateur sollicité par l'agent.* Ce cadre permet à l'agent de s'engager dans des activités d'attention partagée similaires avec l'utilisateur (Tomasello, 1999).
4. *Agent guidé par l'utilisateur.* L'agent bénéficie alors d'une supervision active, dans l'esprit de la théorie de l'échafaudage de Bruner (Wood et al., 1976).

Chacune des formes d'interaction évoquée ci-dessus fait déjà l'objet de nombreux travaux en Intelligence Artificielle R. Kaplan et al. (2017), K. Nguyen et al. (2018), S. M. Nguyen et Oudeyer (2018), Salimans et R. Chen (2018), Najar et al. (2020) et K. Nguyen et al. (2021). Cependant, à notre connaissance, aucune étude ne s'y est intéressée sous l'angle des Sciences Cognitives. Que représentent ces modes d'interaction pour un enfant / agent ? Quelle typologie d'informations en retire-t-il ? Comment ces modes d'interactions peuvent-ils s'articuler dans le développement d'un agent artificiel ?

Les bénéfices de l'Intelligence Artificielle hybride pour un assistant autonome

Il y a un débat dans la communauté en Intelligence Artificielle sur la pertinence des approches hybrides qui mêlent symbolisme et connexionnisme (M. Botvinick et al., 2017 ; Lake et al., 2017 ; Goldstein et al., 2020 ; G. Marcus, 2020 ; Santoro et al., 2021). Sans vouloir opposer les deux approches ni prétendre que seule une combinaison des deux peut mener vers le développement de systèmes plus intelligents, on constate que d'un point de vue purement pragmatique, les faiblesses des deux approches peuvent se compenser lorsque l'on combine ces systèmes entre eux. C'est notamment le cas pour les assistants numériques apprenants et l'intégration de Jarvis, AidMe et Ilead (Partie 4.8).

Renforcer l'intégration

En effet, l'une des problématiques à laquelle ces systèmes sont confrontés est de concilier une interaction fluide et efficace avec l'utilisateur, et la capacité de généraliser la plus large possible. Un système comme Jarvis est très efficace dans son apprentissage mais faible dans sa capacité de généralisation. À l'inverse Ilead est lent à apprendre mais peut généraliser efficacement. AidMe constitue une forme d'entre deux.

L'intégration proposée est une intégration fonctionnelle : chaque module met au service des autres ses fonctionnalités. En ce sens, l'intégration reste partielle. On peut envisager

un deuxième niveau d'intégration dans lequel chaque module exploite les compétences acquises par les autres pour poursuivre son propre développement. Les modules deviennent alors interdépendants.

Jarvis peut ainsi jouer le rôle d'un oracle démonstrateur pour Ilead. En effet, Jarvis apprend à composer les actions élémentaires apprises par Ilead et collecte ainsi auprès de l'utilisateur des séquences d'actions constituant de nouvelles procédures. Cela fournit à l'utilisateur un moyen simple et efficace pour réaliser des actions de haut niveau. Cependant, Jarvis reste incapable de généraliser une procédure à de nouveaux contextes (nouveaux objets par exemple). En revanche, si Ilead apprenait à exécuter ces nouvelles procédures, il pourrait être en mesure de généraliser à de nouveaux objets. Jarvis pourrait ainsi jouer le rôle d'un démonstrateur expert permettant de superviser de nouveaux apprentissages de Ilead.

De même, on a évoqué en conclusion du Chapitre 2 que l'une des limites d'AidMe étaient l'absence d'*embodiment*. AidMe pourrait être augmenté d'une capacité de perception à partir des représentations de l'environnement apprises par Ilead. Cela permettrait de corréler les requêtes de l'utilisateur avec l'état courant de l'environnement et de détecter des régularités, par exemple le fait qu'il est peu probable qu'une nouvelle requête de l'utilisateur concerne le volume de la télévision si celle-ci n'est pas allumée.

Le cadre des systèmes hybrides multi-agents peut être pertinent pour étudier les dépendances enter les différents modules et permettre des apprentissages croisés (Srinivasan et Choy, 2010).

Le potentiel d'explicabilité

L'intégration de ces systèmes entre eux permet aussi d'envisager une forme d'explicabilité croisée. On peut envisager de conceptualiser explicitement l'apprentissage d'une politique d'action d'un système de Deep RL, par exemple au sein d'un graphe de connaissance. Un agent numérique comme Ilead est immergé dans un espace qui possède une structure forte mais dont on ne connaît pas à l'avance la sémantique associée. Le rôle d'Ilead est d'apprendre la sémantique des objets qu'il manipule : découvrir la notion d'objets, de Channels, d'actions et de paramètres, et comprendre les similarités et différences qui lient ces objets. Nous avons montré qu'une approche basée sur l'apprentissage statistique permettait dans ce cas d'apprendre cette sémantique et d'être en mesure de généraliser à des structures connexes.

Une fois la sémantique de la structure de l'espace d'action appréhendée, on peut essayer d'exploiter cette structure pour conceptualiser la manière dont l'agent interagit avec l'espace d'action. De la même manière que l'on a déterminé les règles de grammaire à partir des usages du langage, on peut essayer de déterminer les règles qui expliquent les

actions de l'agent. Par exemple, pour allumer un objet, si l'objet dispose d'un CHANNEL de type Switch dont la sémantique de la description se rapproche de *power*, l'action TurnOn de ce CHANNEL permet certainement de l'allumer.

L'objectif n'est pas de remplacer l'agent par ces règles inférées mais de pouvoir proposer une explication aux actions de l'agent. En effet, l'un des reproches souvent formulé à l'encontre des systèmes d'apprentissage statistique et qui nuit à leur déploiement en conditions réelles est leur manque d'explicabilité (Teso et Kersting, 2019). L'approche proposée ici est envisageable dès que le système considéré évolue dans un espace très structuré comme ILEAD. L'apprentissage statistique permet d'acquérir une notion du sens de cette structure, d'apprendre à y naviguer. Le domaine XAI (*Intelligence Artificielle Explicable*) est en pleine émergence, notamment en Deep RL et peut proposer des opportunités intéressantes pour dispose de systèmes plus interprétables (Arrieta et al., 2020 ; Heuillet et al., 2021).

Annexes

Annexe A

Grammaire utilisateur : *UserGrammar*

On présente ci-dessous *UserGrammar* qui nous a permis de conduire les simulations pour AIDME (Partie 2.4.1). `arguments_dict` contient les différentes valeurs que peuvent prendre les arguments en fonction de leur type. `intention_list` contient la liste des intentions. Chaque intention est représentée par un dictionnaire contenant la liste des patterns/constructions correspondantes ainsi que le domaine auquel il se réfère. Les paramètres de chacune des constructions sont typées afin que les phrases générées aient un sens mais on rappelle cependant qu'AIDME n'a aucune idée du typage des arguments.

```
arguments_dict = {
    "__loc": ["paris", "london", "tokyo", "Honolulu", "Berlin",
              "Miami", "Stockholm", "Beirut", "Rome", "France",
              "Italy", "Germany"],
    "__num": [str(i) for i in range(10)],
    "__pers": ["Bob", "Brian", "Alice", "Jack", "Joe",
               "Bill", "Robert"],
    "__date": ["Friday", "Monday", "Sunday", "Tuesday",
               "Wednesday", "Thursday", "Saturday"],
    "__var": ["table", "chair", "apple", "bird", "bakery",
              "plan", "train", "dog", "wheelchair", "car",
              "orange", "meal", "friendship", "suitcase", "bike",
              "tree", "vegetable", 'spider'],
    "__search": ["wikipedia", "dictionary", "web"],
    "__device": ["light", "alarm", "television", "speaker", "heater"]
}

intention_list= [
    {"patterns": ["search for __var1",
```

```
20                  "search the definition of __var1",
21                  "show me the definition of __var1",
22                  "what is a __var1 ?",
23                  "do you know what is a __var1",
24                  "search for __var1",
25                  "What does a __var1 mean",
26                  "what is the definition of __var1"],
27      "domains": ["search"]},
28     {"patterns": ["open __search1",
29                  "go to __search1",
30                  "Take me to __search1 page",
31                  "Open __search1 website"], "domains": ["search"]},
32     {"patterns": ["send a new mail to __pers1",
33                  "send an email to __pers1",
34                  "Please, send an email to __pers1",
35                  "I need to send an email to __pers1",
36                  "Can you send an email to __pers1"],
37      "domains": ["mail"]},
38     {"patterns": ["set the subject of the email to __var1",
39                  "set the subject to __var1",
40                  "The subject of the email is __var1",
41                  "Write __var1 as the subject of the email"],
42      "domains": ["mail"]},
43     {"patterns": ["send the draft",
44                  "ok, now send the draft",
45                  "Ok, now you can send the draft",
46                  "Press the send button"],
47      "domains": ["mail"]},
48     {"patterns": ["close the inbox tab",
49                  "Thanks now you can close the inbox tab"],
50      "domains": ["mail"]},
51     {"patterns": ["Take the definition of __var1 on __search1 and send it to
       ↪  __pers1",
52                  "Prepare an email for __pers1 with the definition from
                    ↪  __search1 of __var1",
53                  "Get the definition of __var1 on __search1 and write about it
                    ↪  to __pers1",
54                  "send the definition from __search1 of __var1 to __pers1",
55                  "tell the definition you find on __search1 of __var1 to
                    ↪  __pers1"],
```

```
56      "domains": ["mail", "search"]},
57     {"patterns": ["set the content with __var1",
58                   "write __var1 in the content",
59                   "__var1 is the content",
60                   "set __var1 as the content",
61                   "Put __var1 in the content"],
62      "domains": ["mail"]},
63     {"patterns": ["send the email",
64                   "send the email through Inbox"],
65      "domains": ["mail"]},
66     {"patterns": ["close __search1",
67                   "now close __search1 tab",
68                   "please close the __search1 tab",
69                   "you can close the __search1 tab"],
70      "domains": ["search"]},
71     {"patterns": ["create a mail draft",
72                   "can you create a draft please",
73                   "create a new draft",
74                   "go to inbox and create a draft"],
75      "domains": ["mail"]},
76
77     {"patterns": ["set the recipient with __pers1",
78                   "The recipient of the message is __pers1",
79                   "fill the recipient of the message __pers1",
80                   "this mail is for __pers1"],
81      "domains": ["mail"]},
82     {"patterns": ["put it in my message and send the message to __pers1",
83                   "Write it in my message and send the mail to __pers1",
84                   "You can put that in the message and send it to __pers1",
85                   "Send that in a message to __pers1",
86                   "Put that in a message for __pers1 and send it",
87                   "Can you add it to my mail and send it to __pers1"],
88      "domains": ["mail"]},
89
90     {"patterns": ["can you buy __num1 tickets",
91                   "buy __num1 concert tickets",
92                   "Can you book __num1 tickets",
93                   "Please buy __num1 tickets",
94                   "Please book __num1 tickets",
95                   "I need __num1 tickets"],
```

```
96      "domains": ["ticket"]},
97     {"patterns": ["buy __num1 tickets for the __pers1 concert",
98                   "there is __pers1 concert soon, can you buy __num1 tickets for
                     ↪  me",
99                   "Buy me __num1 tickets for the next concert of __pers1",
100                  "I need __num1 tickets for __pers1 concert",
101                  "Get me __num1 tickets for __pers1 concert",
102                  "please book __num1 tickets for __pers1 concert"],
103     "domains": ["ticket"]},
104    {"patterns": ["exchange my __num1 tickets for a refund",
105                  "can you get me a refund for my __num1 tickets",
106                  "I need a refund for my __num1 tickets",
107                  "I need to cancel my booking for __num1 tickets",
108                  "Please cancel my __num1 tickets"],
109     "domains": ["ticket"]},
110    {"patterns": ["swap my tickets for __pers1 with the __pers2 concert",
111                  "Exchange the tickets for __pers1 concert against tickets for
                     ↪  __pers2 concert",
112                  "Switch the tickets for __pers1 concert for tickets for
                     ↪  __pers2 concert",
113                  "Can you switch the tickets for __pers1 concert for tickets
                     ↪  for __pers2 concert"],
114     "domains": ["ticket"]},
115    {"patterns": ["Can you ask __pers1 if he would be interested in the tickets
    ↪  for __pers2 concert",
116                  "Message __pers1 about the tickets for __pers2 concert",
117                  "Ask __pers1 about my tickets for __pers2 concert",
118                  "Offer __pers1 my tickets for __pers2 concert"],
119     "domains": ["ticket", "mail"]},
120    {"patterns": ["Ask __pers1 if he wants my tickets for __pers2 concert on
    ↪  __date1",
121                  "Offer __pers1 my tickets for __pers2 concert on __date1",
122                  "Give my tickets for __pers2 concert on __date1 to __pers1",
123                  "Message __pers1 about the tickets for __pers2 concert on
                     ↪  __date1"],
124     "domains": ["ticket", "mail"]},
125    {"patterns": ["search for __pers1 on __search1 then buy a ticket for his
    ↪  show",
126                  "check on __search1 for __pers1 and then try to buy a ticket
                     ↪  for his show",
```

```
127                 "Buy a ticket for __pers1 show by looking on __search1",
128                 "Book a ticket for __pers1 show by looking on __search1",
129                 "Look for __pers1 on __search1 and get me a ticket for his
                    ↪  show"],
130      "domains": ["ticket", "search"]},
131     {"patterns": ["Check my calendar and buy tickets for __pers1 concert",
132                 "If I am available, can you get me tickets for __pers1
                    ↪  concert",
133                 "Book tickets for __pers1 concert if I am available",
134                 "Check my planning and get me tickets for __pers1 concert"],
135      "domains": ["ticket", "calendar"]},
136     {"patterns": [
137         "can you buy __num1 tickets for __pers1 concert, then inform __pers2
            ↪  about it",
138         "Buy __num1 tickets for __pers1 concert and send an email to __pers2",
139         "Get __num1 tickets for __pers1 concert and inform __pers2",
140         "Please book __num1 tickets for __pers1 concert and tell __pers2 about
            ↪  it"],
141         "domains": ["ticket", "mail"]},
142     {"patterns": ["buy __num1 tickets for the __date1 and add the event to my
        ↪  calendar",
143                 "please, buy __num1 tickets for __date1 then save it in my
                    ↪  calendar",
144                 "Can you book __num1 tickets for __date1 and put a reminder in
                    ↪  my calendar",
145                 "Add it to my calendar after buying __num1 tickets for
                    ↪  __date1",
146                 ],
147      "domains": ["ticket", "calendar"]},
148
149     {"patterns": ["when do I have a meeting with __pers1",
150                 "what day do I have a meeting with __pers1",
151                 "on what day do I meet with __pers1",
152                 "when do I meet __pers1",
153                 "when is my next meeting with __pers1",
154                 "When am I supposed to meet __pers1",
155                 "When is my next meeting with __pers1",
156                 "On what day is the meeting with __pers1",
157                 "Look for my next meeting with __pers1"],
158      "domains": ["calendar"]},
```

```
159
160     {"patterns": ["set a meeting with __pers1 on __date1",
161                   "I will meet with __pers1 on __date1",
162                   "set me a meeting on __date1 with __pers1",
163                   "I need to meet __pers1 on __date1"],
164      "domains": ["calendar"]},
165     {"patterns": ["when do I go to __loc1",
166                   "what day do I go to __loc1",
167                   "when am I supposed to go to __loc1",
168                   "When is my trip to __loc1 scheduled"],
169      "domains": ["calendar"]},
170     {"patterns": ["who is coming to the meeting on __date1",
171                   "who is attending __date1 meeting",
172                   "tell me who is attending __date1 meeting",
173                   "who will be at the meeting on __date1",
174                   "who is planning to attend the meeting on __date1",
175                   "who will attend the meeting on __date1"], "domains":
                      ↪  ["calendar"]},
176     {"patterns": ["look for flights from __loc1 to __loc2",
177                   "find me a flight from __loc1 to __loc2",
178                   "find a flight for me from __loc1 to __loc2",
179                   "find a flight from __loc1 to __loc2",
180                   "get me a flight from __loc1 to __loc2",
181                   'what are the flights from __loc1 to __loc2',
182                   ],
183      "domains": ["travel"]},
184     {"patterns": ["should I set it on __date1 or __date2",
185                   "According to my calendar, is __date1 or __date2 better",
186                   "What is better between __date1 and __date2",
187                   "Advise me the better between __date1 and __date2",
188                   "Would you recommend __date1 or __date2",
189                   ],
190      "domains": ["calendar"]},
191
192     {"patterns": ["I want to go on __date1",
193                   "The trip is on __date1",
194                   "The trip will be on __date1",
195                   "The trip is planned on __date1",
196                   "I will go on __date1",
197                   "I am going on __date1",
```

```
                "I need to go on __date1"],
 "domains": ["travel"]},

{"patterns": ["how far is it from __loc1 to __loc2",
                "what is the distance from __loc1 to __loc2",
                "how far is __loc1 from __loc2",
                "tell me how far __loc1 is from __loc2",
                "tell me the distance between __loc1 and __loc2"],
 "domains": ["travel"]},
{"patterns": ["Get me the price for flights from __loc1 to __loc2",
                "What is the price for flights from __loc1 to __loc2",
                "How much does the flights from __loc1 to __loc2 cost",
                "How much are the flights from __loc1 to __loc2",
                "how much does it cost to fly from __loc1 to __loc2",
                "how much is it to fly from __loc1 to __loc2"],
 "domains": ["travel"]},
{"patterns": ["Get me the price for ticket from __loc1 to __loc2 on
↪  __date1",
                "What is the price for a ticket from __loc1 to __loc2 on
                ↪  __date1",
                "On __date1 how much does it cost to travel from __loc1 to
                ↪  __loc2"],
 "domains": ["travel"]},
{"patterns": ["Remind __pers1 of our meeting on __date1",
                "Remind __pers1 about our __date1 meeting",
                "Remind __pers1 of our __date1 meeting",
                "Please remind __pers1 of our meeting on __date1",
                "Can you remind __pers1 of our meeting on __date1",
                "Please send a reminder to __pers1 about our meeting on
                ↪  __date1"],
 "domains": ["mail", "calendar"]},
{"patterns": ["Is there an airport in __loc1",
                "Is there any airport in __loc1",
                "Any airport in __loc1",
                "Check for an airport in __loc1",
                "Check for any airport in __loc1",
                "Check an airport in __loc1"],
 "domains": ["travel"]},
{"patterns": ["I need to organise a meeting with __pers1 to talk about
↪  __var1",
```

```
233                 "Plan a meeting with __pers1 about __var1",
234                 "Set up a meeting with __pers1 about __var1",
235                 "I need to meet with __pers1 to talk about __var1"],
236     "domains": ["calendar"]},
237    {"patterns": ["Set a meeting with __pers1 to talk about __var1 project on
       ↪  __date1",
238                 "I need to set a meeting with __pers1 to talk about __var1
                    ↪  project on __date1",
239                 "Plan a meeting with __pers1 about __var1 on __date1"],
240     "domains": ["calendar"]},
241    {"patterns": ["how long is the trip from __loc1 to __loc2",
242                 "how long is the trip between __loc1 to __loc2",
243                 "how long is it from __loc1 to __loc2",
244                 "how long do I need to go from __loc1 to __loc2",
245                 "how long do I need to go between __loc1 to __loc2",
246                 "how long will it take to go from __loc1 to __loc2"],
247     "domains": ["travel"]},
248    {"patterns": ["Get me the price for a trip to __loc1 between __date1 and
       ↪  __date2",
249                 "What is the price for a trip to __loc1 from __date1 to
                    ↪  __date2",
250                 "Between __date1 and __date2 how much does it cost to travel
                    ↪  to __loc1"],
251     "domains": ["search"]},
252    {"patterns": ["How is the weather in __loc1 on __date1",
253                 "How is the weather on __date1 in __loc1",
254                 "Will the weather be fine in __loc1 on __date1",
255                 "Will the weather be fine on __date1 in __loc1",
256                 "Check the weather in __loc1 on __date1",
257                 "Check the weather on __date1 in __loc1"],
258     "domains": ["search"]},
259    {"patterns": ["Turn on __device1 in my home in __loc1",
260                 "Turn on __device1 in __loc1",
261                 "Activate __device1 in __loc1",
262                 "Activate __device1 in my house __loc1",
263                 "Please make sure __device1 is turned on at my house in
                    ↪  __loc1",
264                 "Switch on the __device1 in my home in __loc1",
265                 "Switch on the __device1 in __loc1"],
266     "domains": ["IoT"]},
```

```
    {"patterns": ["When __device1 is off, please turn on  __device2",
                  "Control that when __device1 is turned off, __device2 is
                  ↪  turned on",
                  "When __device1 is switched off, automatically switch on
                  ↪  __device2",
                  "After you turn off __device1 , turn on __device2"],
     "domains": ["IoT"]},
    {"patterns": ["Add a reminder on my calendar on __date1 to repair
    ↪  __device1",
                  "Can you remind me to repair __device1 on __date1",
                  "Remind me to repair __device1 on __date1",
                  "on __date1 remind me to repair __device1",
                  "Set a reminder on __date1 to repair __device1",
                  "I need to remember to repair __device1 on __date1"],
     "domains": ["IoT", "calendar"]},
    {"patterns": ["Can you set up the alarm on __date1",
                  "Set up the alarm on __date1",
                  "Activate the alarm on __date1",
                  "Please turn on the alarm on __date1",
                  "Switch on the alarm on __date1",
                  "Please switch on the alarm on __date1"],
     "domains": ["IoT"]},
    {"patterns": ["Make sure the __device1 is always off on __date1",
                  "Verify that __device1 is always off on __date1",
                  "Check __device1 is off on __date1",
                  "on __date1 check that __device1 is off",
                  "I want __device1 off on __date1"],
     "domains": ["IoT"]},
    {"patterns": ["Always turn off __device1 before turning on __device2",
                  "__device1 needs to be off before __device2 is turned on",
                  "Check that __device1 is off before turning on __device2"],
     "domains": ["IoT"]},
    {"patterns": ["Are you connected to __device1",
                  "Can you take control of __device1",
                  "Take control of __device1",
                  "Connect to __device1",
                  "Please connect to __device1",
                  "Do you control __device1"],
     "domains": ["IoT"]},
    {"patterns": ["What is the status of __device1",
```

```
304                  "What is the power status of __device1",
305                  "Is power of __device1 on or off",
306                  "Tell me the status of __device1",
307                  "Is __device1 on or off"], "domains": ["IoT"]},
308     {"patterns": ["Get me a report of __device1 usage on __date1",
309                  "report on __device1 usage on __date1",
310                  "I want to know how __device1 was used on __date1",
311                  "How was __device1 used on __date1",
312                  "on __date1 how was __device1 used"],
313      "domains": ["IoT"]},
314 ]
```

Annexe B

Tâches de l'environnement *ArmToolsToys*

On présente dans cette partie l'ensemble des tâches en langage naturel définies dans l'environnement *ArmToolsToys* (Partie 3.5).

1. Shift the hand to the right
2. Shift the hand to the left
3. Shift the hand higher
4. Shift the hand lower
5. Move the hand close to the center
6. Move the hand to the top right area
7. Move the hand to the top left area
8. Move the hand to the bottom right area
9. Move the hand to the bottom left area
10. Grasp the magnetic stick
11. Grasp the scratch stick
12. Shift the magnetic stick to the right
13. Shift the magnetic stick to the left
14. Shift the magnetic stick higher
15. Shift the magnetic stick lower
16. Move the magnetic stick to the center
17. Move the magnetic stick to the top right area
18. Move the magnetic stick to the top left area
19. Move the magnetic stick to the bottom right area
20. Move the magnetic stick to the bottom left area
21. Shift the sticky stick to the right
22. Shift the sticky stick to the left
23. Shift the sticky stick higher
24. Shift the sticky stick lower
25. Move the sticky stick to the center
26. Move the sticky stick to the top right area
27. Move the sticky stick to the top left area
28. Move the sticky stick to the bottom right area
29. Move the sticky stick to the bottom left area
30. Bring the magnetic stick closer to the magnet
31. Bring the scratch stick closer to the scratch
32. Grasp the magnet
33. Grasp the scratch

34. Shift the magnet to the right
35. Shift the magnet to the left
36. Shift the magnet higher
37. Shift the magnet lower
38. Move the magnet to the center
39. Move the magnet to the top right area
40. Move the magnet to the top left area
41. Move the magnet to the bottom right area
42. Move the magnet to the bottom left area
43. Shift the scratch to the right
44. Shift the scratch to the left
45. Shift the scratch higher
46. Shift the scratch lower
47. Move the scratch to the center
48. Move the scratch to the top right area
49. Move the scratch to the top left area
50. Move the scratch to the bottom right area
51. Move the scratch to the bottom left area

Annexe C

Tâches de l'environnement *Playground*

C.1 La grammaire des tâches réalisables

On décrit ci-dessous la grammaire des tâches **réalisables** dans *Playground* (Partie 3.6.2). Les mots < > permettent de désigner les lieux de l'environnement, couleurs et noms d'objet et de catégorie selon le vocabulaire ci-dessous. Un objet peut être désigné par sa couleur uniquement (*blue thing*), son type d'objet ou de catégorie (*any cat* ou *any animal*), par sa couleur et son type d'objet/catégorie (*red cat* ou *red animal*).

1. Se déplacer : GO <***zone***> (ex : *go bottom left*)
2. Attraper :
   - GRASP ANY <***couleur***> THING (ex :*grasp any blue thing*)
   - GRASP <***couleur*** $\cup$ {*any*}> <***type*** $\cup$ ***catégorie***> (ex : *grasp red cat*)
3. Faire grandir :
   - GROW + ANY + <***couleur***> + THING (ex : *grow any red thing*)
   - GROW + <***couleur*** $\cup$ {*any*}> + <***être vivant*** $\cup$ {*living_thing, animal, plant*}> (ex : *grow green animal*)

— ***zone*** = {*center, top, bottom, right, left, top left, top right, bottom left, bottom right*}
— ***couleur*** = {*red, blue, green*}
— ***type*** = ***être vivant*** $\cup$ ***meuble*** $\cup$ ***provision***
— ***catégorie*** = {*living_thing, animal, plant, meuble, provision*}
— ***être vivant*** = ***animal*** $\cup$ ***plante***
— ***animal*** = {*dog, cat, chameleon, human, fly, parrot, mouse, lion, pig, cow*}
— ***plante*** = {*cactus, carnivorous, flower, tree, bush, grass, algae, tea, rose, bonsai*}
— ***meuble*** = {*door, chair, desk, lamp, table, cupboard, sink, window, sofa, carpet*}
— ***provision*** = {*water, food*}
— ***prédicat*** = {*go, grasp, grow*}

C.2 Ensemble $\mathcal{D}^{\text{test}}$ et $\mathcal{D}^{\text{imaginé}}$

On décrit dans les Tableaux C.1 et C.2 les ensembles des descriptions de $\mathcal{D}^{\text{test}}$ ainsi que les descriptions imaginables $\mathcal{D}^{\text{imaginé}}$ par l'heuristique CGH présenté en Partie 3.6.4. $\mathcal{D}^{\text{test}}$ a été construit de manière à pouvoir étudier 5 formes différentes de généralisation (voir Partie 3.6.2). $\mathcal{D}^{\text{train}}$ est simplement l'ensemble des tâches réalisables (générées par la grammaire ci-dessus) auquel on a soustrait $\mathcal{D}^{\text{test}}$.

Table C.1 – Ensemble des descriptions $\mathcal{D}^{\text{test}}$

Type	**Description des tâches**
Type 1	*Grasp blue door*, *Grasp green dog*, *Grasp red tree*, *Grow green dog*
Type 2	*Grasp any flower*, *Grasp blue flower*, *Grasp green flower*, *Grasp red flower*, *Grow any flower*, *Grow blue flower*, *Grow green flower*, *Grow red flower*
Type 3	*Grasp any animal*, *Grasp blue animal*, *Grasp green animal*, *Grasp red animal*
Type 4	*Grasp any fly*, *Grasp blue fly*, *Grasp green fly*, *Grasp red fly*
Type 5	*Grow any algae*, *Grow any bonsai*, *Grow any bush*, *Grow any cactus* *Grow any carnivorous*, *Grow any grass*, *Grow any living_thing*, *Grow any rose*, *Grow any tea*, *Grow any tree*, *Grow blue algae* *Grow blue bonsai*, *Grow blue bush*,*Grow blue cactus*, *Grow blue carnivorous* *Grow blue grass*, *Grow blue living_thing*, *Grow blue plant*, *Grow blue rose* *Grow blue tea*, *Grow blue tree*,*Grow green algae*, *Grow green bonsai* *Grow green bush*, *Grow green cactus*, *Grow green carnivorous*, *Grow green grass* *Grow green living_thing*, *Grow green plant*, *Grow green rose*, *Grow green tea* *Grow green tree*, *Grow red algae*, *Grow red bonsai*, *Grow red bush* *Grow red cactus*, *Grow red carnivorous*, *Grow red grass*, *Grow red living_thing* *Grow red plant*, *Grow red rose*, *Grow red tea*, *Grow red tree*, *Grow any plant*.

Table C.2 – Ensemble des descriptions imaginables par l'heuristique CGH

Type des tâches	**Description des tâches**
Descriptions de $\mathcal{G}^{\text{train}}$	$\mathcal{G}^{\text{train}}$
Descriptions de $\mathcal{G}^{\text{test}}$	Tâches de Type 1, 3, 4 and 5, voir Tableau C.1.
Descriptions syntaxiquement incorrectes	*Go bottom top*, *Go left right*, *Grasp red blue thing*, *Grow blue red thing*, *Go right left*, *Go top bottom*, *Grasp green blue thing*, *Grow green red thing*, *Grasp green red thing*, *Grasp blue green thing*, *Grasp blue red thing*, *Grasp red green thing*.
Descriptions syntaxiquement correctes mais correspondant à des tâches non réalisables	*Go center bottom*, *Go center top*, *Go right center*, *Go right bottom*, *Go right top*, *Go left center*, *Go left bottom*, *Go left top*, *Grow green cupboard*, *Grow green sink*, *Grow blue lamp*, *Go center right*, *Grow green window*, *Grow blue carpet*, *Grow red supply*, *Grow any sofa*, *Grow red sink*, *Grow any chair*, *Go top center*, *Grow blue table*, *Grow any door*, *Grow any lamp*, *Grow blue sink*, *Go bottom center*, *Grow blue door*, *Grow blue supply*, *Grow green carpet*, *Grow blue furniture*, *Grow green supply*, *Grow any window*, *Grow any carpet*, *Grow green furniture*, *Grow green chair*, *Grow green food*, *Grow any cupboard*, *Grow red food*, *Grow any table*, *Grow red lamp* , *Grow red door*, *Grow any food*, *Grow blue window*, *Grow green sofa*, *Grow blue sofa*, *Grow blue desk*, *Grow any sink*, *Grow red cupboard*, *Grow green door*, *Grow red furniture*, *Grow blue food*, *Grow red desk* , *Grow red table*, *Grow blue chair*, *Grow red sofa*, *Grow any furniture*, *Grow red window*, *Grow any desk*, *Grow blue cupboard*, *Grow red chair*, *Grow green desk*, *Grow green table*, *Grow red carpet*, *Go center left*, *Grow any supply*, *Grow green lamp*, *Grow blue water*, *Grow red water*, *Grow any water*, Grow green water, *Grow any water*, *Grow green water*.

Annexe D

Tâches du simulateur OpenHAB

Dans cette partie, on décrit les tâches qui sont réalisables pour chacun des objets utilisés dans les simulations présentées en Partie 4.6. On utilise une grammaire simple similaire à celle employée dans *Playground*.

1. **Store** :
   - Close <*name*>
   - Open <*name*>

2. **Plug** :
   - Turn on <*name*>
   - Turn off <*name*>

3. **BLight** :
   - Increase <*name*> brightness
   - Decrease <*name*> brightness
   - Set <*name*> brightness to low
   - Set <*name*> brightness to average
   - Set <*name*> brightness to high

4. **BTLight** :
   - Increase <*name*> brightness
   - Decrease <*name*> brightness
   - Set <*name*> brightness to low
   - Set <*name*> brightness to average
   - Set <*name*> brightness to high
   - Increase <*name*> temperature
   - Decrease <*name*> temperature
   - Set <*name*> temperature to low
   - Set <*name*> temperature to average
   - Set <*name*> temperature to high

5. **ColorLight** :

   — Increase <*name*> brightness
   — Decrease <*name*> brightness
   — Set <*name*> brightness to low
   — Set <*name*> brightness to average
   — Set <*name*> brightness to high
   — Increase <*name*> temperature
   — Decrease <*name*> temperature
   — Set <*name*> temperature to low
   — Set <*name*> temperature to average
   — Set <*name*> temperature to high
   — Set <*name*> color to red
   — Set <*name*> color to blue
   — Set <*name*> color to green

6. **Speaker** :

   — Increase <*name*> volume
   — Decrease <*name*> volume
   — Set <*name*> volume to low
   — Set <*name*> volume to average
   — Set <*name*> volume to high
   — Play <*name*>
   — Pause <*name*>

7. **Television** :

   — Increase <*name*> volume
   — Decrease <*name*> volume
   — Set <*name*> volume to low
   — Set <*name*> volume to average
   — Set <*name*> volume to high
   — Set <*name*> channel to 1
   — Set <*name*> channel to 2
   — Set <*name*> channel to 3
   — Play <*name*>
   — Pause <*name*>

Acronymes

API *Application Programming Interface* 10, 55, 65, 125, 127–131, 164

BoW *Bag of Words* 48

CGH *Construction Grammar Heuristic* xiii, 112, 113, 115, 194, 195

DDPG *Deep Deterministic Policy Gradients* 77, 81, 93, 105, 144

Deep RL *Deep Reinforcement Learning* 9, 71, 73, 75, 76, 81, 82, 84, 85, 88, 122, 128, 130, 134, 135, 137, 138, 144, 151, 157, 164, 165, 167, 168, 170, 171, 176, 177

DOM *Document Object Model* 129, 130

DQN *Deep Q-Network* 77, 80, 81, 144, 148, 158

IDF *Inverse Document Frequency* 48

iML *Interactive Machine Learning* 5, 7–9

MDP *Markov Decision Process* 77, 78

ML *Machine Learning* 4–7, 9, 18, 37, 169

NLP *Natural Language Processing* 14, 18

NLU *Natural Language Understanding* 15–18, 37, 38, 55, 65, 67, 75, 167

REST *Representational state transfer* 130, 164

RL *Reinforcement Learning* 10, 77, 79, 151, 160

URI *Uniform Ressource Identifier* 138

URL *Uniform Ressource Locator* 138

VQA *Visual Question Answering* 21, 84, 139

ZPD *Zone de développement proximal* 31, 32, 74, 118, 120, 122

Bibliographie

[1] Allen, J. F., Byron, D. K., Dzikovska, M., Ferguson, G., Galescu, L. & Stent, A. (2001). Toward Conversational Human-Computer Interaction. *AI Magazine*, *22*(4), 27-27. DOI : 10.1609/AIMAG.V22I4.1590

[2] Allen, J. F., Chambers, N., Ferguson, G., Galescu, L., Jung, H., Swift, M. & Taysom, W. (2007). PLOW : a collaborative task learning agent. *Proceedings of the 22nd national conference on Artificial intelligence (AAAI'07)*, 1514-1519. https://dl.acm.org/doi/10.5555/1619797.1619888

[3] Amershi, S., Cakmak, M., Knox, W. B. & Kulesza, T. (2014). Power to the people : The role of humans in interactive machine learning. *AI Magazine*, *35*(4), 105-120. DOI : 10.1609/aimag.v35i4.2513

[4] Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Sünderhauf, N., Reid, I., Gould, S. & Van Den Hengel, A. (2018). Vision-and-Language Navigation : Interpreting visually-grounded navigation instructions in real environments. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3674-3683. https://bringmeaspoon.org

[5] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P. & Zaremba, W. (2017). Hindsight Experience Replay. *31st Conference on Neural Information Processing Systems (NIPS 2017)*. http://arxiv.org/abs/1707.01495

[6] Ankerst, M., Elsen, C., Ester, M. & Kriegel, H.-P. (1999). Visual classification : An Interactive Approach to Decision Tree Construction. *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '99*, 392-396. DOI : 10.1145/312129.312298

[7] Arrieta, A. B., Díaz-Rodríguez, N., Ser, J. d., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., Chatila, R. & Herrera, F. (2020). Explainable Artificial Intelligence (XAI) : Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI. *Information Fusion*, *58*, 82-115. DOI : 10.1016/J.INFFUS.2019.12.012

[8] Asada, M., Hosoda, K., Kuniyoshi, Y., Ishiguro, H., Inui, T., Yoshikawa, Y., Ogino, M. & Yoshida, C. (2009). Cognitive Developmental Robotics : A Survey. *IEEE Transactions on Autonomous Mental Development*, *1*(1), 12-34. DOI : 10.1109/TAMD.2009.2021702

[9] Azaria, A., Krishnamurthy, J. & Mitchell, T. M. (2016). Instructable Intelligent Personal Agent. *Proceedings of the 30th Conference on Artificial Intelligence (AAAI 2016)*, 2681-2689. https://dl.acm.org/doi/10.5555/3016100.3016277%20http://ai2-website.s3.amazonaws.com/publications/LearnByInst.pdf

[10] Bahdanau, D., Cho, K. H. & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *Proceedings of the International Conference on Learning Representations, ICLR 2015*. https://arxiv.org/abs/1409.0473v7

[11] Bahdanau, D., Hill, F., Leike, J., Hughes, E., Hosseini, A., Kohli, P. & Grefenstette, E. (2019). Learning to Understand Goal Specifications by Modelling Reward. *International Conference on Learning Representations (ICLR 2019)*. https://openreview.net/forum?id=H1xsSjC9Ym%20http://arxiv.org/abs/1806.01946

[12] Bahdanau, D., Murty, S., Noukhovitch, M., Nguyen, T. H., de Vries, H. & Courville, A. (2018). Systematic Generalization : What Is Required and Can It Be Learned ? *ICLR 2019*. http://arxiv.org/abs/1811.12889

[13] Baranes, A. & Oudeyer, P.-Y. (2013). Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, *61*(1), 49-73. DOI : 10.1016/j.robot.2012.05.008

[14] Bateson, M. C. (1975). Mother-infant Exchanges : The Epigenesis Of Conversational Interaction. *Annals of the New York Academy of Sciences*, *263*(1), 101-113. DOI : 10.1111/j.1749-6632.1975.tb41575.x

[15] Bengio, Y., Louradour, J., Collobert, R. & Weston, J. (2009). Curriculum learning. *ICML 09 : Proceedings of the 26th Annual International Conference on Machine Learning*, *382*, 1-8. DOI : 10.1145/1553374.1553380

[16] Bisk, Y., Zellers, R., Le Bras, R., Gao, J. & Choi, Y. (2020). PIQA : Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(05), 7432-7439. DOI : 10.1609/aaai.v34i05.6239

[17] Bodrova, E. & Leong, D. J. (2001). *Tools of the Mind : A Case Study of Implementing the Vygotskian Approach in American Early Childhood and Primary Classrooms. Innodata Monographs 7.* (rapp. tech.). UNESCO International Bureau of Education. International Bureau of Education, P.O. Box 199, 1211 Geneva 20, Switzerland. Web site : http ://www.ibe.unesco.org. http://www.ibe.unesco.org

[18] Bornstein, M. H., Tamis-LeMonda, C. S., Tal, J., Ludemann, P., Toda, S., Rahn, C. W., Pêcheux, M.-G. .-G., Azuma, H. & Vardi, D. (1992). Maternal Responsive-

ness to Infants in Three Societies : The United States, France, and Japan. *Child Development*, *63*(4), 808-821. DOI : 10.1111/j.1467-8624.1992.tb01663.x

[19] Botvinick, M. [M.], Barrett, D. G. T., Battaglia, P., de Freitas, N., Kumaran, D., Leibo, J. Z., Lillicrap, T., Modayil, J., Mohamed, S., Rabinowitz, N. C., Rezende, D. J., Santoro, A., Schaul, T., Summerfield, C., Wayne, G., Weber, T., Wierstra, D., Legg, S. & Hassabis, D. (2017). Building Machines that Learn and Think for Themselves : Commentary on Lake et al., Behavioral and Brain Sciences, 2017. *Behavioral and Brain Sciences*. http://arxiv.org/abs/1711.08378

[20] Botvinick, M. [Matthew], Ritter, S., Wang, J. X., Kurth-Nelson, Z., Blundell, C. & Hassabis, D. (2019). Reinforcement Learning, Fast and Slow. DOI : 10.1016/j.tics.2019.02.006

[21] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, *33*. https://commoncrawl.org/the-data/

[22] Bruinsma, Y., Koegel, R. L. & Koegel, L. K. (2004). Joint attention and children with autism : A review of the literature. DOI : 10.1002/mrdd.20036

[23] Bruner, J. (1983). *Child's talk : learning to use language*. Oxford : Oxford University Press.

[24] Cambria, E. & White, B. (2014). Jumping NLP curves : A review of natural language processing research. DOI : 10.1109/MCI.2014.2307227

[25] Cangelosi, A. (2006). The grounding and sharing of symbols. *Pragmatics & Cognition*, *14*(2), 275-285. DOI : 10.1075/pc.14.2.08can

[26] Cangelosi, A. & Schlesinger, M. (2018). From Babies to Robots : The Contribution of Developmental Robotics to Developmental Psychology. *Child Development Perspectives*, *12*(3), 183-188. DOI : 10.1111/cdep.12282

[27] Carpenter, M., Nagell, K., Tomasello, M., Butterworth, G. & Moore, C. (1998). Social Cognition, Joint Attention, and Communicative Competence from 9 to 15 Months of Age. *Monographs of the Society for Research in Child Development*, *63*(4), i. DOI : 10.2307/1166214

[28] Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I. & Specia, L. (2018). SemEval-2017 Task 1 : Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation, 1-14. DOI : 10.18653/v1/s17-2001

[29] Chan, H., Wu, Y., Kiros, J., Fidler, S. & Ba, J. (2019). ACTRCE : Augmenting Experience via Teacher's Advice For Multi-Goal Reinforcement Learning. *1st*

Workshop on Goal Specifications for Reinforcement Learning. http://arxiv.org/abs/1902.04546

[30] Chaplot, D. S., Sathyendra, K. M., Pasumarthi, R. K., Rajagopal, D. & Salakhutdinov, R. (2018). Gated-Attention Architectures for Task-Oriented Language Grounding. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 2819-2826. http://arxiv.org/abs/1706.07230

[31] Chaudhari, S., Mithal, V., Polatkan, G. & Ramanath, R. (2020). An Attentive Survey of Attention Models. DOI : 10.1145/1122445.1122456

[32] Chen, D. L. & Mooney, R. J. (2011). Learning to Interpret Natural Language Navigation Instructions from Observations. *AAAI Conference on Artificial Intelligence (AAAI), 2011*. http://www.cs.utexas.edu/users/ml/clamp/navigation/

[33] Chen, H., Suhr, A., Misra, D., Snavely, N. & Artzi, Y. (2019). TOUCHDOWN : Natural Language Navigation and Spatial Reasoning in Visual Street Environments. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 12538-12547. https://touchdown.ai.

[34] Chen, T. & Guestrin, C. (2016). XGBoost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 785-794. DOI : 10.1145/2939672.2939785

[35] Chevalier-Boisvert, M., Bahdanau, D., Lahlou, S., Willems, L., Saharia, C., Nguyen, T. H. & Bengio, Y. (2019). BabyAI : First Steps Towards Grounded Language Learning With a Human In the Loop. *International Conference on Learning Representations*. https://openreview.net/forum?id=rJeXCo0cYX

[36] Chomsky, N. [Noam.]. (1957). *Syntactic structures*. Mouton. http://www.linguist.univ-paris-diderot.fr/~edunbar/ling499b_spr12/readings/syntactic_structures.pdf

[37] Chomsky, N. [Noam]. (1965). *Aspects of the Theory of Syntax*. MIT Press. https://mitpress.mit.edu/books/aspects-theory-syntax

[38] Chu, J. & Schulz, L. (2020). Exploratory play, rational action, and efficient search. DOI : 10.31234/osf.io/9yra2

[39] Cideron, G., Seurin, M., Strub, F. & Pietquin, O. (2021). HIGhER : Improving instruction following with Hindsight Generation for Experience Replay, 225-232. DOI : 10.1109/ssci47803.2020.9308603

[40] Claessen, V., Schmidt, A. & Heck, T. (2017). A Study on the Usability and User Perception of Customer Service Systems for E-Commerce. *Proceedings of the 15th International Symposium of Information Science (ISI 2017)*, 116-130. https://www.chatbots.org/country/de/.

[41] Cobbe, K., Klimov, O., Hesse, C., Kim, T. & Schulman, J. (2019). Quantifying Generalization in Reinforcement Learning. *Proceedings of the 36th International Conference on Machine Learning ICML*, 1282-1289. http://proceedings.mlr.press/v97/cobbe19a.html

[42] Colas, C., Founder, P., Sigaud, O., Chetouani, M. & Oudeyer, P. Y. (2019a). CURIOUS : Intrinsically motivated modular multi-goal reinforcement learning. *36th International Conference on Machine Learning, ICML 2019*, *2019-June*, 2372-2387.

[43] Colas, C., Karch, T., Lair, N., Dussoux, J.-M., Dominey, P. F. & Oudeyer, P.-Y. (2020). Language as a Cognitive Tool to Imagine Goals in Curiosity-Driven Exploration. *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*.

[44] Colas, C., Sigaud, O. & Oudeyer, P.-Y. (2019b). *A Hitchhiker's Guide to Statistical Comparisons of Reinforcement Learning Algorithms*. http://arxiv.org/abs/1904.06979

[45] Conneau, A., Kiela, D., Schwenk, H., Barrault, L. & Bordes, A. (2017). Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. *EMNLP*. http://arxiv.org/abs/1705.02364

[46] Coradeschi, S., Loutfi, A. & Wrede, B. (2013). A Short Review of Symbol Grounding in Robotic and Intelligent Systems. *KI - Kunstliche Intelligenz*, *27*(2), 129-136. DOI : 10.1007/s13218-013-0247-2

[47] Co-Reyes, J. D., Gupta, A., Sanjeev, S., Altieri, N., Andreas, J., Denero, J., Abbeel, P. & Levine, S. (2019). Guiding Policies with Language via Meta-Learning. *International confernce on Learning Representations ICLR 2019*.

[48] Côté, M. A., Kádár, Á., Yuan, X., Kybartas, B., Barnes, T., Fine, E., Moore, J., Hausknecht, M., El Asri, L., Adada, M., Tay, W. & Trischler, A. (2019). TextWorld : A Learning Environment for Text-Based Games. *Communications in Computer and Information Science*, *1017*, 41-75. DOI : 10.1007/978-3-030-24337-1_3

[49] Cruz, F., Magg, S., Weber, C. & Wermter, S. (2014). Improving Reinforcement Learning with Interactive Feedback and Affordances. *Proceedings of the Fourth Joint IEEE ICDL-EpiRob '14*, 125-130. http://www.informatik.uni-hamburg.de/WTM/

[50] Cruz, F., Twiefel, J., Magg, S., Weber, C. & Wermter, S. (2015). Interactive reinforcement learning through speech guidance in a domestic scenario. *Proceedings of the International Joint Conference on Neural Networks*, *2015-September*. DOI : 10.1109/IJCNN.2015.7280477

[51] Daumé III, H. (2016). Language bias and black sheep. https://nlpers.blogspot.com/2016/06/language-bias-and-black-sheep.html

[52] Dautenhahn, K. & Billard, A. (1999). Studying robot social cognition within a developmental psychology framework. *1999 3rd European Workshop on Advanced Mobile Robots, Eurobot 1999 - Proceedings*, 187-194. DOI : 10.1109/EURBOT.1999.827639

[53] Dehghani, M., Boghrati, R., Man, K., Hoover, J., Gimbel, S. I., Vaswani, A., Zevin, J. D., Immordino-Yang, M. H., Gordon, A. S., Damasio, A. & Kaplan, J. T. (2017). Decoding the neural representation of story meanings across languages. *Human Brain Mapping*, *38*(12), 6096-6106. DOI : 10.1002/hbm.23814

[54] Delgrange, C. (2018). *Apprentissage basé sur l'usage en interaction humaine avec un assistant adaptatif* (thèse de doct.). Université de Lyon. Lyon. https://www.researchgate.net/publication/331973499_Apprentissage_base_sur_l%27usage_en_interaction_humaine_avec_un_assistant_adaptatif

[55] Delgrange, C., Dussoux, J. M. & Dominey, P. F. (2020). Usage-based learning in human interaction with an adaptive virtual assistant. *IEEE Transactions on Cognitive and Developmental Systems*, *12*(1), 109-123. DOI : 10.1109/TCDS.2019.2927399

[56] Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2019). BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171-4186. DOI : 10.18653/v1/N19-1423

[57] Dobrovsky, A., Borghoff, U. M. & Hofmann, M. (2017). An approach to interactive deep reinforcement learning for serious games. *7th IEEE International Conference on Cognitive Infocommunications, CogInfoCom 2016 - Proceedings*, 85-90. DOI : 10.1109/CogInfoCom.2016.7804530

[58] Dominey, P. F. & Boucher, J. D. (2005). Learning to talk about events from narrated video in a construction grammar framework. *Artificial Intelligence*, *167*(1-2), 31-61. DOI : 10.1016/j.artint.2005.06.007

[59] Dupoux, E. (2018). Cognitive Science in the era of Artificial Intelligence : A roadmap for reverse-engineering the infant language-learner. *Cognition*, *173*, 43-59. DOI : 10.1016/j.cognition.2017.11.008

[60] Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M. & Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, *542*(7639), 115-118. DOI : 10.1038/nature21056

[61] Ettinger, A. (2020). What BERT Is Not : Lessons from a New Suite of Psycholinguistic Diagnostics for Language Models. *Transactions of the Association for Computational Linguistics*, *8*, 34-48. DOI : 10.1162/tacl_a_00298

[62] Ettinger, A., Feldman, N. H., Resnik, P. & Phillips, C. (2016). Modeling N400 amplitude using vector space models of word representation. *Proceedings of the Annual Meeting of theCognitive Science Society*.

[63] Evans, N. & Levinson, S. C. (2009). The myth of language universals : Language diversity and its importance for cognitive science. *Behavioral and Brain Sciences*, *32*, 429-492. DOI : 10.1017/S0140525X0999094X

[64] Fails, J. A. & Olsen, D. R. (2004). Interactive machine learning. *dl.acm.org*, 39. DOI : 10.1145/604045.604056

[65] Fillmore, C. J. (1988). The Mechanisms of "Construction Grammar". *Annual Meeting of the Berkeley Linguistics Society*, *14*(0), 35. DOI : 10.3765/bls.v14i0.1794

[66] Fodor, J. A. & Pylyshyn, Z. W. (1988). Connectionnism and cognitive architecture : a critical analysis. *Cognition*, *28*, 3-71.

[67] Forestier, S. & Oudeyer, P.-Y. (2016). Modular active curiosity-driven discovery of tool use. DOI : 10.1109/IROS.2016.7759584

[68] Forestier, S., Portelas, R., Mollard, Y. & Oudeyer, P.-Y. (2017). Intrinsically Motivated Goal Exploration Processes with Automatic Curriculum Learning. *arXiv*. http://arxiv.org/abs/1708.02190

[69] Fu, J., Korattikara, A., Levine, S. & Guadarrama, S. (2019). From Language to Goals : Inverse Reinforcement Learning for Vision-Based Instruction Following. *International Conference on Learning Representations*. https://openreview.net/forum?id=r1lq1hRqYQ

[70] Gao, J., Galley, M. & Li, L. (2019). Neural Approaches to Conversational AI. *Foundations and Trends® in Information Retrieval*, *13*(2-3). DOI : 10.1561/1500000074

[71] Garg, N., Schiebinger, L., Jurafsky, D. & Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, *115*(16), E3635-E3644. DOI : 10.1073/PNAS.1720347115

[72] Gentner, D. (2016). Language as cognitive tool kit : How language supports relational thought. *American Psychologist*, *71*(8), 650-657. DOI : 10.1037/amp0000082

[73] Goksel-Canbek, N. & Mutlu, M. E. (2016). On the track of Artificial Intelligence : Learning with Intelligent Personal Assistants. *International Journal of Human Sciences*, *13*(1), 592. DOI : 10.14687/ijhs.v13i1.3549

[74] Goldberg, A. E. (2003). Constructions : A new theoretical approach to language. *Trends in Cognitive Sciences*, *7*(5), 219-224. DOI : 10.1016/S1364-6613(03)00080-9

[75] Goldstein, A., Zada, Z., Buchnik, E., Schain, M., Price, A., Aubrey, B., Nastase, S. A., Feder, A., Emanuel, D., Cohen, A., Jansen, A. & Hasson, U. (2020). Thinking

ahead : prediction in context as a keystone of language in humans and machines. *bioRxiv*, 2020.12.02.403477. DOI : 10.1101/2020.12.02.403477

[76] Gopnik, A., Meltzoff, A. N. & Kuhl, P. K. (1999). *The scientist in the crib : Minds, brains, and how children learn.* William Morrow & Co.

[77] Goyal, P., Niekum, S. & Mooney, R. J. (2019). Using Natural Language for Reward Shaping in Reinforcement Learning. *IJCAI 2019*. http://arxiv.org/abs/1903.02020

[78] Graves, A., Wayne, G. & Danihelka, I. (2014). Neural Turing Machines. http://arxiv.org/abs/1410.5401

[79] Green, E. J. & Quilty-Dunn, J. (2017). what is an object file ? *The British Journal for the Philosophy of Science*. DOI : 10.1093/bjps/axx055

[80] Gur, I., Rueckert, U., Faust, A. & Hakkani-Tur, D. (2019). Learning to Navigate the Web. *International Conference on Learning Representations (ICLR), 2019*. http://arxiv.org/abs/1812.09195

[81] Ha, D. & Schmidhuber, J. (2018). Recurrent World Models Facilitate Policy Evolution. *Advances in Neural Information Processing Systems*, *2018-December*, 2450-2462. http://arxiv.org/abs/1809.01999

[82] Harnad, S. (1999). The Symbol Grounding Problem. *Physica*, *42*, 335-346. DOI : 10.1016/0167-2789(90)90087-6

[83] Hauswald, J., Tang, L., Mars, J., A. Laurenzano, M., Zhang, Y., Li, C., Rovinski, A., Khurana, A., Dreslinski, R., Mudge, T. & Petrucci, V. (2015). Sirius : An Open End-to-End Voice and Vision Personal Assistant and Its Implications for Future Warehouse Scale Computers. *ACM SIGPLAN Notices*, *50*, 223-238. DOI : 10.1145/2775054.2694347

[84] He, P., Liu, X., Gao, J. & Chen, W. (2021). DeBERTa : Decoding-enhanced BERT with Disentangled Attention. *ICLR 2021*. https://arxiv.org/abs/2006.03654v3

[85] Henderson, J. C., Merkhofer, E. M., Strickhart, L. & Zarrella, G. (2017). MITRE at SemEval-2017 Task 1 : Simple Semantic Similarity. In S. Bethard, M. Carpuat, M. Apidianaki, S. M. Mohammad, D. M. Cer & D. Jurgens (Éd.), *Proceedings of the 11th International Workshop on Semantic Evaluation, SemEval@ACL 2017, Vancouver, Canada, August 3-4, 2017* (p. 185-190). Association for Computational Linguistics. DOI : 10.18653/v1/S17-2027

[86] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D. & Meger, D. (2017). Deep Reinforcement Learning that Matters. http://arxiv.org/abs/1709.06560

[87] Hermann, K. M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., Szepesvari, D., Czarnecki, W. M., Jaderberg, M., Teplyashin, D., Wainwright, M., Apps, C., Hassabis, D. & Blunsom, P. (2017). Grounded Language Learning in a Simulated 3D World. http://arxiv.org/abs/1706.06551

[88] Heuillet, A., Couthouis, F. & Díaz-Rodríguez, N. (2021). Explainability in deep reinforcement learning. *Knowledge-Based Systems*, *214*. DOI : 10.1016/j.knosys.2020.106685

[89] Hill, F. & Korhonen, A. (2014). Learning abstract concept embeddings from multi-modal data : Since you probably can't see what I mean. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, 255-265. DOI : 10.3115/v1/d14-1032

[90] Hill, F., Lampinen, A., Schneider, R., Clark, S., Botvinick, M., McClelland, J. L. & Santoro, A. (2019). Environmental drivers of systematicity and generalization in a situated agent. *ICLR 2020*. http://arxiv.org/abs/1910.00571

[91] Hinaut, X. & Dominey, P. F. (2013). Real-Time Parallel Processing of Grammatical Structure in the Fronto-Striatal System : A Recurrent Network Simulation Study Using Reservoir Computing (T. Boraud, Éd.). *PLoS ONE*, *8*(2), e52946. DOI : 10.1371/journal.pone.0052946

[92] Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735-1780. DOI : 10.1162/neco.1997.9.8.1735

[93] Holzinger, A. (2016). Interactive machine learning for health informatics : when do we need the human-in-the-loop ? *Brain Informatics*, *3*(2), 119-131. DOI : 10.1007/s40708-016-0042-6

[94] Hong, Y., Rodriguez-Opazo, C., Qi, Y., Wu, Q. & Gould, S. (2020). Language and Visual Entity Relationship Graph for Agent Navigation. *Advances in Neural Information Processing Systems*, *33*. https://evalai.cloudcv.org/web/challenges/challenge-page/97/overview

[95] Hosseini-Asl, E., Mccann, B., Wu, C.-S., Yavuz, S. & Socher, R. (2020). A Simple Language Model for Task-Oriented Dialogue. *Advances in Neural Information Processing Systems*, *33*, 20179-20191. https://github.com/

[96] Huang, H., Jain, V., Mehta, H., Ku, A., Magalhaes, G., Baldridge, J. & Ie, E. (2019). Transferable Representation Learning in Vision-and-Language Navigation. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 7404-7413.

[97] Intharah, T., Turmukhambetov, D. & Brostow, G. J. (2017). Help, It Looks Confusing : GUI Task Automation Through Demonstration and Follow-up Questions. *International Conference on Intelligent User Interfaces, Proceedings IUI*, 233-243. DOI : 10.1145/3025171.3025176

[98] Ji, H., Ke, P., Huang, S., Wei, F., Zhu, X. & Huang, M. (2020). Language Generation with Multi-Hop Reasoning on Commonsense Knowledge Graph. *Proceedings of the*

2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 725-736. https://github.

[99] Jia, S., Kiros, J. & Ba, J. (2019). DOM-Q-NET : Grounded RL on Structured Language. *International Conference on Learning Representations (ICLR), 2019*. http://arxiv.org/abs/1902.07257

[100] Jiang, L., Liu, S. & Chen, C. (2019). Recent research advances on interactive machine learning. *Journal of Visualization*, *22*(2), 401-417. DOI : 10.1007/s12650-018-0531-1

[101] Jiang, Y., Gu, S., Murphy, K. & Finn, C. (2019). Language as an Abstraction for Hierarchical Deep Reinforcement Learning. *33rd Conference on Neural Information Processing Systems (NeurIPS 2019),* http://arxiv.org/abs/1906.07343

[102] Johnson, J., Fei-fei, L., Hariharan, B., Zitnick, C. L., Maaten, L. V. D. & Girshick, R. (2019). Compositional Language and Elementary Visual Reasoning. *NAACL-Johnson, J., Fei-fei, L., Hariharan, B., Zitnick, C. L., Maaten, L. Van Der, & Girshick, R. (2019). Compositional Language and Elementary Visual Reasoning. NAACL.* https://arxiv.org/pdf/1612.06890.pdf

[103] Kaplan, F. & Hafner, V. V. (2004). The Challenges of Joint Attention. *Proceedings of the Fourth International Workshop on Epigenetic Robotics*. http://cogprints.org/4067/1/kaplan.pdf

[104] Kaplan, F. & Oudeyer, P.-Y. (2007). In Search of the Neural Circuits of Intrinsic Motivation. DOI : 10.3389/neuro.01.1.1.017.2007

[105] Kaplan, R., Sauer, C. & Sosa, A. (2017). Beating Atari with Natural Language Guided Reinforcement Learning. http://arxiv.org/abs/1704.05539

[106] Kemler Nelson, D. G., Russell, R., Duke, N. & Jones, K. (2000). Two-year-olds will name artifacts by their functions. *Child Development*, *71*(5), 1271-1288. DOI : 10.1111/1467-8624.00228

[107] Kiani, M., Andreu-Perez, J., Hagras, H., Filippetti, M. L. & Rigato, S. (2020). A type-2 fuzzy logic based explainable artificial intelligence system for developmental neuroscience. *IEEE International Conference on Fuzzy Systems*, *2020-July*. DOI : 10.1109/FUZZ48607.2020.9177711

[108] Kidd, C. & Hayden, B. Y. (2015). The Psychology and Neuroscience of Curiosity. DOI : 10.1016/j.neuron.2015.09.010

[109] Kiros, R., Zhu, Y., Salakhutdinov, R., Zemel, R. S., Torralba, A., Urtasun, R. & Fidler, S. (2015). Skip-Thought Vectors. *NIPS'15 Proceedings of the 28th International Conference on Neural Information Processing Systems*. http://arxiv.org/abs/1506.06726

[110] Klein, T. & Nabi, M. (2018). Attention Is (not) All You Need for Commonsense Reasoning. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 4831-4836.

[111] Kulkarni, T. D., Narasimhan, K. R., Saeedi, A., Tenenbaum, J. B. & Bcs, J. B. T. (2016). Hierarchical Deep Reinforcement Learning : Integrating Temporal Abstraction and Intrinsic Motivation. *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, *29*. http://arxiv.org/abs/1604.06057

[112] Kuo, Y.-L., Katz, B. & Barbu, A. (2020). Compositional Networks Enable Systematic Generalization for Grounded Language Understanding. *arXiv*. http://arxiv.org/abs/2008.02742

[113] Labutov, I., Srivastava, S. & Mitchell, T. (2018). LIA : A Natural Language Programmable Personal Assistant. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing : System Demonstrations*, 145-150. DOI : 10.18653/v1/D18-2025

[114] Lair, N., Colas, C., Portelas, R., Dussoux, J.-M., Dominey, P. & Oudeyer, P.-Y. (2019). Language grounding through social interactions and curiosity-driven multi-goal learning. *ViGIL workshop (NeurIPS 2019) : Visually Grounded Interaction and Language*.

[115] Lair, N., Delgrange, C., Mugisha, D., Dussoux, J.-M., Oudeyer, P.-Y. & Dominey, P. F. (2020). User-in-the-loop adaptive intent detection for instructable digital assistant. *International Conference on Intelligent User Interfaces, Proceedings IUI*, 116-127. DOI : 10.1145/3377325.3377490

[116] Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., Salvucci, D., Scheutz, M., Thomaz, A., Trafton, G., Wray, R. E., Mohan, S. & Kirk, J. R. (2017). Interactive Task Learning. *IEEE Intelligent Systems*, *32*(4), 6-21. DOI : 10.1109/MIS.2017.3121552

[117] Lake, B. M. & Baroni, M. (2017). Generalization without systematicity : On the compositional skills of sequence-to-sequence recurrent networks. *35th International Conference on Machine Learning, ICML 2018*, *7*, 4487-4499. http://arxiv.org/abs/1711.00350

[118] Lake, B. M., Ullman, T. D., Tenenbaum, J. B. & Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences*, *40*, e253. DOI : 10.1017/S0140525X16001837

[119] Landauer, T. K. & Dumais, S. T. (1997). A Solution to Plato's Problem : The Latent Semantic Analysis Theory of Acquisition, Induction, and Representation of Knowledge. *Psychological Review*, *104*(2), 211-240.

[120] Laversanne-Finot, A., Péré, A. & Oudeyer, P.-Y. (2018). Curiosity Driven Exploration of Learned Disentangled Goal Spaces. *Proceedings of The 2nd Conference on Robot Learning, CoRL*, 487-504. http://proceedings.mlr.press/v87/laversanne-finot18a.html

[121] Lazaridou, A., Pham, N. T. & Baroni, M. (2015). Combining language and vision with a multimodal skip-gram model. *NAACL HLT 2015 - 2015 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Proceedings of the Conference*, 153-163. DOI : 10.3115/v1/n15-1016

[122] Lee, B., Isenberg, P., Riche, N. H. & Carpendale, S. (2012). Beyond mouse and keyboard : Expanding design considerations for information visualization interactions. *IEEE Transactions on Visualization and Computer Graphics*, *18*(12), 2689-2698. DOI : 10.1109/TVCG.2012.204

[123] Li, F. & Bowling, M. (2019). Ease-of-Teaching and Language Structure from Emergent Communication. *Neural Information Processing Systems (NeurIPS) 2019*. http://arxiv.org/abs/1906.02403

[124] Li, T. J.-J., Azaria, A. & Myers, B. A. (2017). Sugilite. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems - CHI '17*, 6038-6049. DOI : 10.1145/3025453.3025483

[125] Li, T. J.-J., Mitchell, T. & Myers, B. (2020). Interactive Task Learning from GUI-Grounded Natural Language Instructions and Demonstrations. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics : System Demonstrations*, 215-223. DOI : 10.18653/v1/2020.acl-demos.25

[126] Li, T. J.-J., Popowski, L., Mitchell, T. M. & Myers, B. A. (2021). Screen2Vec : Semantic Embedding of GUI Screens and GUI Components. *15*(21), 2021. DOI : 10.1145/3411764.3445049

[127] Li, Y., He, J., Zhou, X., Zhang, Y. & Baldridge, J. (2020). Mapping Natural Language Instructions to Mobile UI Action Sequences. *Annual Conference of the Association for Computational Linguistics (ACL 2020)*. http://arxiv.org/abs/2005.03776

[128] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. & Wierstra, D. (2016). Continuous control with deep reinforcement learning. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*. https://goo.gl/J4PIAz

[129] Lindblom, J. & Ziemke, T. (2003). Social Situatedness of Natural and Artificial Intelligence : Vygotsky and Beyond. *Adaptive Behavior*, *11*(2), 79-96. DOI : 10.1177/10597123030112002

[130] Linzen, T., Dupoux, E. & Goldberg, Y. (2016). Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies. *Transactions of the Association for Computational Linguistics*, *4*, 521-535. DOI : 10.1162/tacl_a_00115

[131] Liu, E. Z., Guu, K., Pasupat, P., Shi, T. & Liang, P. (2018). Reinforcement Learning on Web Interfaces Using Workflow-Guided Exploration. *International Conference on Learning Representations (ICLR), 2018*. http://arxiv.org/abs/1802.08802

[132] Lowe, R., Gupta, A., Foerster, J., Kiela, D. & Pineau, J. (2020). On the interaction between supervision and self-play in emergent communication. *ICLR*. https://github.com/backpropper/s2p.

[133] Luketina, J., Nardelli, N., Farquhar, G., Foerster, J., Andreas, J., Grefenstette, E., Whiteson, S. & Rocktäschel, T. (2019). A Survey of Reinforcement Learning Informed by Natural Language. *IJCAI'19*. http://arxiv.org/abs/1906.03926

[134] Lungarella, M., Metta, G., Pfeifer, R. & Sandini, G. (2003). Developmental robotics : A survey. DOI : 10.1080/09540090310001655110

[135] Luong, M. T., Pham, H. & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *Conference Proceedings - EMNLP 2015 : Conference on Empirical Methods in Natural Language Processing*, 1412-1421. DOI : 10.18653/v1/d15-1166

[136] Maharjan, N., Banjade, R., Gautam, D., Tamang, L. J. & Rus, V. (2017). DT_Team at SemEval-2017 Task 1 : Semantic Similarity Using Alignments, Sentence-Level Embeddings and Gaussian Mixture Model Output. In S. Bethard, M. Carpuat, M. Apidianaki, S. M. Mohammad, D. M. Cer & D. Jurgens (Éd.), *Proceedings of the 11th International Workshop on Semantic Evaluation, SemEval@ACL 2017, Vancouver, Canada, August 3-4, 2017* (p. 120-124). Association for Computational Linguistics. DOI : 10.18653/v1/S17-2014

[137] Mandler, J. M. (2000). Perceptual and Conceptual Processes in Infancy. *Journal of Cognition and Development*, *1*(1), 3-36. DOI : 10.1207/S15327647JCD0101N_2

[138] Marcus, G. (2020). *The Next Decade in AI : Four Steps Towards Robust Artificial Intelligence* (rapp. tech.). http://arxiv.org/abs/2002.06177

[139] Mazumder, S. & Riva, O. (2020). FLIN : A Flexible Natural Language Interface for Web Navigation. http://arxiv.org/abs/2010.12844

[140] McClelland, J. L. [James L.], Hill, F., Rudolph, M., Baldridge, J. & Schütze, H. (2020). Placing language in an integrated understanding system : Next steps toward human-level performance in neural language models. *Proceedings of the National Academy of Sciences*, 201910416. DOI : 10.1073/pnas.1910416117

[141] Merkle, N. & Philipp, P. (2019). Cooperative web agents by combining semantic technologies with reinforcement learning. *K-CAP 2019 - Proceedings of the 10th*

International Conference on Knowledge Capture, 205-212. DOI : 10.1145/3360901.3364417

[142] Metz, L., Ibarz, J., Jaitly, N. & Davidson, J. (2017). Discrete Sequential Prediction of Continuous Actions for Deep RL. https://arxiv.org/abs/1705.05035v3

[143] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. *Advances in Neural Information Processing Systems (NIPS 2013)*. http://arxiv.org/abs/1310.4546

[144] Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D. & Miller, K. J. (1990). *Introduction to wordnet : An on-line lexical database* (rapp. tech. N° 4). DOI : 10.1093/ijl/3.4.235

[145] Mintz, T. H. (2003). Frequent frames as a cue for grammatical categories in child directed speech. *Cognition*, *90*(1), 91-117. DOI : 10.1016/S0010-0277(03)00140-9

[146] Mirolli, M. & Parisi, D. (2011). Towards a Vygotskyan cognitive robotics : The role of language as a cognitive tool. *New Ideas in Psychology*, *29*(3), 298-311. DOI : 10.1016/j.newideapsych.2009.07.001

[147] Misra, I., Girshick, R., Fergus, R., Hebert, M., Gupta, A. & Van Der Maaten, L. (2018). Learning by Asking Questions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 11-20. DOI : 10.1109/CVPR.2018.00009

[148] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529-533. DOI : 10.1038/nature14236

[149] Moon, S., Shah, P., Kumar, A. & Subba, R. (2018). OpenDialKG : Explainable Conversational Reasoning with Attention-based Walks over Knowledge Graphs. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 845-854.

[150] Nagai, Y., Hosoda, K., Morita, A. & Asada, M. (2003). A constructive model for the development of joint attention. *Connection Science*, *15*(4), 211-229. DOI : 10.1080/09540090310001655101

[151] Nair, A., Bahl, S., Khazatsky, A., Pong, V., Berseth, G. & Levine, S. (2019). Contextual Imagined Goals for Self-Supervised Robotic Learning. *Conference on Robot Learning (CoRL) 2019*. http://arxiv.org/abs/1910.11670

[152] Nair, A., Pong, V., Dalal, M., Bahl, S., Lin, S. & Levine, S. (2018). *Visual Reinforcement Learning with Imagined Goals* (rapp. tech.). https://sites.google.com/site/

[153] Najar, A., Sigaud, O. & Chetouani, M. (2020). Interactively shaping robot behaviour with unlabeled human instructions. *Autonomous Agents and Multi-Agent Systems*, *34*(2), 1-35. DOI : 10.1007/s10458-020-09459-6

[154] Narasimhan, K., Barzilay, R. & Jaakkola, T. (2018). Grounding language for transfer in deep reinforcement learning. *Journal of Artificial Intelligence Research*, *63*, 849-874. DOI : 10.1613/jair.1.11263

[155] Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E. & Stone, P. (2020). Curriculum Learning for Reinforcement Learning Domains : A Framework and Survey. *Journal of Machine Learning Research*, *21*, 1-50.

[156] Navigli, R. & Ponzetto, S. P. (2012). BabelNet : The Automatic Construction, Evaluation and Application of a Wide-Coverage Multilingual Semantic Network. *Artificial Intelligence*, *193*, 217-250.

[157] Nelson, K. (1977). First Steps in Language Acquisition. *Journal of the American Academy of Child Psychiatry*, *16*(4), 563-583. DOI : 10.1016/S0002-7138(09)61180-8

[158] Nelson, K. (1996). *Language in Cognitive Development*. Cambridge University Press. DOI : 10.1017/cbo9781139174619

[159] Nguyen, K., Dey, D., Brockett, C. & Dolan, B. (2018). Vision-based Navigation with Language-based Assistance via Imitation Learning with Indirect Intervention. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2019-June*, 12519-12529. http://arxiv.org/abs/1812.04155

[160] Nguyen, K., Misra, D., Schapire, R., Dudík, M. & Shafto, P. (2021). Interactive Learning from Activity Description. http://arxiv.org/abs/2102.07024

[161] Nguyen, S. M. & Oudeyer, P.-Y. (2018). Active choice of teachers, learning strategies and goals for a socially guided intrinsic motivation learner. *Paladyn, Journal of Behavioral Robotics*, *3*(3), 136-146. DOI : 10.2478/s13230-013-0110-z

[162] Oudeyer, P.-Y. (2019). Developmental Machine Learning. https://youtube.videoken.com/embed/xGAZ0GmRgjI?tocitem=2

[163] Oudeyer, P.-Y. & Kaplan, F. (2009). What is intrinsic motivation ? A typology of computational approaches. *Frontiers in Neurorobotics*, *3*(NOV), 6. DOI : 10.3389/neuro.12.006.2007

[164] Oudeyer, P.-Y., Kaplan, F. & Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, *11*(2), 265-286. DOI : 10.1109/TEVC.2006.890271

[165] Papineni, K., Roukos, S., Ward, T. & Zhu, W.-J. (2001). BLEU. *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, 311. DOI : 10.3115/1073083.1073135

[166] Pater, J. (2019). Generative linguistics and neural networks at 60 : Foundation, friction, and fusion. *Language*, *95*(1), e41-e74. DOI : 10.1353/lan.2019.0009

[167] Peng, B., Zhu, C., Li, C., Li, X., Li, J., Zeng, M. & Gao, J. (2020). Few-shot Natural Language Generation for Task-Oriented Dialog. *Findings of the Association for Computational Linguistics EMNLP*, 172-182.

[168] Pennington, J., Socher, R. & Manning, C. D. (2014). GloVe : Global vectors for word representation. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, 1532-1543. DOI : 10.3115/v1/d14-1162

[169] Pfeifer, R., Lungarella, M. & Iida, F. (2007). Self-organization, embodiment, and biologically inspired robotics. DOI : 10.1126/science.1145803

[170] Piaget, J. (2002). *Le langage et la pensée chez l'enfant* (Delachaux et Niestlé, Éd. ; 9e éd.). https://pure.mpg.de/rest/items/item_2375486_6/component/file_2375485/content

[171] Pierrot, T., Ligner, G., Reed, S., Sigaud, O., Perrin, N., Kas, D., Beguir, K. & De Freitas, N. (2019). Learning Compositional Neural Programs with Recursive Tree Search and Planning. *NIPS 2019*, 14673-14683. https://github.com/instadeepai/AlphaNPI

[172] Pong, V. H., Dalal, M., Lin, S., Nair, A., Bahl, S. & Levine, S. (2020). Skew-Fit : State-Covering Self-Supervised Reinforcement Learning. *ICML*. http://arxiv.org/abs/1903.03698

[173] Portelas, R., Colas, C., Weng, L., Hofmann, K. & Oudeyer, P. Y. (2020). Automatic curriculum learning for deep RL : A short survey. *IJCAI International Joint Conference on Artificial Intelligence*, *2021-January*, 4819-4825. DOI : 10.24963/ijcai.2020/671

[174] R. K. Branavan, S., S. Zettlemoyer, L. & Barzilay, R. (2010). Reading Between the Lines : Learning to Map High-level Instructions to Commands. *ACL 2010 - 48th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, 1268-1277.

[175] Radford, A., Narasimhan, K., Salimans, T. & Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training. https://gluebenchmark.com/leaderboard

[176] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. & Sutskever, I. (2019). *Language Models are Unsupervised Multitask Learners* (rapp. tech.). https://github.com/codelucas/newspaper

[177] Ram, A., Prasad, R., Khatri, C., Venkatesh, A., Gabriel, R., Liu, Q., Nunn, J., Hedayatnia, B., Cheng, M., Nagar, A., King, E., Bland, K., Wartick, A., Pan, Y., Song, H., Jayadevan, S., Hwang, G. & Pettigrue, A. (2017). Conversational AI : The Science Behind the Alexa Prize. *1st Proceeding of ALexa Prize 2017*. http://arxiv.org/abs/1801.03604

[178] Ramshaw, L. A. & Marcus, M. P. (1995). Text Chunking using Transformation-Based Learning. *Third Workshop on Very Large Corpora*. https://www.aclweb.org/anthology/W95-0107

[179] Roy, D. K. (2005). Semiotic schemas : A framework for grounding language in action and perception. *Artificial Intelligence*, *167*(1-2), 170-205. DOI : 10.1016/j.artint.2005.04.007

[180] Roy, D. K. & Pentland, A. P. (2002). Learning words from sights and sounds : a computational model. *Cognitive Science*, *26*(1), 113-146. DOI : 10.1207/s15516709cog2601_4

[181] Ruis, L., Andreas, J., Baroni, M., Bouchacourt, D. & Lake, B. M. (2020). A Benchmark for Systematic Generalization in Grounded Language Understanding. *Advances in Neural Information Processing Systems*, *33*. https://github.com/LauraRuis/groundedSCAN

[182] Rumelhart, D. E. & McClelland, J. L. [James L]. (1986). *On Learning the Past Tenses of English Verbs.* (rapp. tech.). Bradford Books/MIT Press.

[183] Sacha, D., Zhang, L., Sedlmair, M., Lee, J. A., Peltonen, J., Weiskopf, D., North, S. C. & Keim, D. A. (2017). Visual Interaction with Dimensionality Reduction : A Structured Literature Analysis. *IEEE Transactions on Visualization and Computer Graphics*, *23*(1), 241-250. DOI : 10.1109/TVCG.2016.2598495

[184] Salimans, T. & Chen, R. (2018). Learning Montezuma's Revenge from a Single Demonstration. *Deep RL Workshop, NeurIPS 2018*. http://arxiv.org/abs/1812.03381

[185] Santoro, A., Lampinen, A., Mathewson, K., Lillicrap, T. & Raposo, D. (2021). Symbolic Behaviour in Artificial Intelligence. http://arxiv.org/abs/2102.03406

[186] Schapire, R. E. & Singer, Y. (2000). BoosTexter : A Boosting-based System for Text Categorization. *Machine Learning*, *39*(3), 135-168.

[187] Schaul, T., Horgan, D., Gregor, K. & Silver, D. (2015a). Universal Value Function Approximators. *Proceedings of the 32nd International Conference on Machine Learning ICML*, 1312-1320. http://proceedings.mlr.press/v37/schaul15.html

[188] Schaul, T., Quan, J., Antonoglou, I. & Silver, D. (2015b). Prioritized Experience Replay. *ICLR*. http://arxiv.org/abs/1511.05952

[189] Schwartz, D., Toneva, M. & Wehbe, L. (2019). Inducing brain-relevant bias in natural language processing models. *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*. http://arxiv.org/abs/1911.03268

[190] Settles, B. (2009). *Active Learning Literature Survey* (rapp. tech.). University of Wisconsin. Madison, University of Wisconsin-Madison Department of Computer Sciences. https://minds.wisconsin.edu/handle/1793/60660

[191] Shah, P., Hakkani-Tur, D. & Heck, L. (2016). Interactive reinforcement learning for task-oriented dialogue management. *Workshop on Deep Learning for Action and Interaction, NIPS 2016 (2016)*. https://research.google/pubs/pub45734/

[192] Shao, Y. (2018). HCTI at SemEval-2017 Task 1 : Use convolutional neural network to evaluate Semantic Textual Similarity. *Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017)*, 130-133. DOI : 10.18653/v1/s17-2016

[193] Shi, T., Karpathy, A., Fan, L., Hernandez, J. & Liang, P. (2017). World of Bits : An Open-Domain Platform for Web-Based Agents. *Proceedings of the 34th International Conference on Machine Learning*, 3135-3144. http://universe.openai.com/

[194] Shridhar, M., Yuan, X., Côté, M.-A., Bisk, Y., Trischler, A. & Hausknecht, M. (2021). Alfworld : Aligning Text And Embodied Environments For Interactive Learning. *ICLR*.

[195] Silver, D. L., Yang, Q. & Li, L. (2013). Lifelong Machine Learning Systems : Beyond Learning Algorithms. *AAI Spring Symposium : Lifelong Machine Learning*, *SS-13-05* (AAAI Technical Report). www.aaai.org

[196] Siskind, J. M. (1996). A computational study of cross-situational techniques for learning word-to-meaning mappings. *Cognition*, *61* (1-2 SPEC. ISS.), 39-91. DOI : 10.1016/s0010-0277(96)00728-7

[197] Skinner, B. F. (1957). *Verbal behavior*. Appleton-Century-Crofts. DOI : 10.1037/11256-000

[198] Smith, L. B. & Slone, L. K. (2017). A Developmental Approach to Machine Learning ? *Frontiers in Psychology*, *8* (DEC), 2124. DOI : 10.3389/fpsyg.2017.02124

[199] Smolensky, P. (1987). Connectionist AI, symbolic AI, and the brain. *Artificial Intelligence Review*, *1* (2), 95-109. DOI : 10.1007/BF00130011

[200] Srinivasan, D. & Choy, M. C. (2010). Hybrid multi-agent systems. *Studies in Computational Intelligence*, *310*, 29-42. DOI : 10.1007/978-3-642-14435-6_2

[201] Stanley, K. O., Lehman, J. & Soros, L. (2019). Open-endedness : The last grand challenge you've never heard of. https://www.oreilly.com/radar/open-endedness-the-last-grand-challenge-youve-never-heard-of/

[202] Steedman, M. (1996). *Surface Structure and Interpretation | The MIT Press*. MIT press. https://mitpress.mit.edu/books/surface-structure-and-interpretation

[203] Steels, L. & Vogt, P. (1997). Grounding Adaptive Language Games in Robotic Agents. *Proceedings of the Fourth European Conference on Artificial Life*, 474-482. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.3876

[204] Sumers, T. R., Ho, M. K., Hawkins, R. D., Narasimhan, K. & Griffiths, T. L. (2020). Learning Rewards from Linguistic Feedback. *arXiv*. http://arxiv.org/abs/2009.14715

[205] Sutskever, I., Vinyals, O. & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, 3104-3112. https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks

[206] Sutton, R. S. & Barto, A. G. (2018). *Reinforcement learning : an introduction* (Second Edition). MIT Press. http://incompleteideas.net/book/the-book.html

[207] Tai, K. S., Socher, R. & Manning, C. D. (2015). Improved semantic representations from tree-structured long short-Term memory networks. *ACL-IJCNLP 2015 - 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, Proceedings of the Conference*, *1*, 1556-1566. DOI : 10.3115/v1/p15-1150

[208] Tan, H. & Bansal, M. (2020). Vokenization : Improving Language Understanding with Contextualized, Visual-Grounded Supervision. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2066-2080. DOI : 10.18653/v1/2020.emnlp-main.162

[209] Tellex, S., Kollar, T., Dickerson, S., Walter, M. R., Banerjee, A. G., Teller, S. & Roy, N. (2011). Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation. *Proceedings of the National Conference on Artificial Intelligence (AAAI 2011)*. http://spatial.csail.mit.edu/grounding.

[210] Teso, S. & Kersting, K. (2019). Explanatory Interactive Machine Learning. *AIES '19 : Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, 239-245. https://dl.acm.org/doi/abs/10.1145/3306618.3314293

[211] Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J. & Mannor, S. (2016). A Deep Hierarchical Approach to Lifelong Learning in Minecraft. *Proceedings of the 31th Conference on Artificial Intelligence (AAAI 2017)*. http://arxiv.org/abs/1604.07255

[212] Tian, J., Zhou, Z., Lan, M. & Wu, Y. (2017). ECNU at SemEval-2017 Task 1 : Leverage Kernel-based Traditional NLP features and Neural Networks to Build a Universal Model for Multilingual and Cross-lingual Semantic Textual Similarity. In S. Bethard, M. Carpuat, M. Apidianaki, S. M. Mohammad, D. M. Cer & D. Jurgens (Éd.), *Proceedings of the 11th International Workshop on Semantic Evaluation, SemEval@ACL 2017, Vancouver, Canada, August 3-4, 2017* (p. 191-197). Association for Computational Linguistics. DOI : 10.18653/v1/S17-2028

[213] Tomasello, M. (1999). *The cultural origins of human cognition*. Harvard University Press. http://ektr.uni-eger.hu/wp-content/uploads/2015/11/tomasello-the-cultural-origins-of-human-cognition.pdf

[214] Tomasello, M. (2000a). Do young children have adult syntactic competence ? *Cognition*, *74*(3), 209-253. DOI : 10.1016/S0010-0277(99)00069-4

[215] Tomasello, M. (2000b). The item-based nature of children's early syntactic development. *Trends in Cognitive Science*, *4*(4), 156-163.

[216] Tomasello, M. (2002). Things are what they do : Katherine Nelson's functional approach to language and cognition. *Journal of Cognition and Development*, *3*(1), 5-19. DOI : 10.1207/S15327647JCD0301_2

[217] Tomasello, M. (2003). *Constructing a Language : A Usage-Based Theory of Language Acquisition* (First edition). Harvard University Press.

[218] Tomasello, M. (2009). Universal grammar is dead. *Behavioral and Brain Sciences*, *32*(5), 470-471. DOI : 10.1017/S0140525X09990744

[219] Tomasello, M. & Olguin, R. (1993). Twenty-three-month-old children have a grammatical category of noun. *Cognitive Development*, *8*(4), 451-464. DOI : 10.1016/S0885-2014(05)80004-8

[220] Trevarthen, C. (1993). Predispositions to cultural learning in young infants. *Behavioral and Brain Sciences*, *16*(3), 534-535. DOI : 10.1017/s0140525x00031502

[221] Trevarthen, C. & Aitken, K. J. (2003). Intersubjectivité chez le nourrisson : Recherche, théorie et application clinique. *Devenir*, *15*(4), 309-428. DOI : 10.3917/dev.034.0309

[222] Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, *59*(236), 433-460. DOI : 10.1093/mind/LIX.236.433

[223] Uchida, T., Lair, N., Ishiguro, H. & Dominey, P. F. (2021). A Model of Online Temporal-Spatial Integration for Immediacy and Overrule in Discourse Comprehension. *Neurobiology of Language*, *2*(1), 83-105. DOI : 10.1162/nol_a_00026

[224] Underwood, J. (2017). Exploring AI language assistants with primary EFL students. *CALL in a climate of change : adapting to turbulent global conditions – short papers from EUROCALL 2017*, 317-321. DOI : 10.14705/rpnet.2017.eurocall2017.733

[225] Van Der Maaten, L. & Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, *9*(86), 2579-2605. http://jmlr.org/papers/v9/vandermaaten08a.html

[226] van Hasselt, H., Guez, A. & Silver, D. (2016). Deep reinforcement learning with double Q-Learning. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2094-2100. https://dl.acm.org/citation.cfm?id=3016191

[227] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. (2017). Transformer : Attention is all you need. *Advances in Neural Information Processing Systems*, *2017-Decem*, 5999-6009.

[228] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., ... Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, *575*(7782), 350-354. DOI : 10.1038/s41586-019-1724-z

[229] Vinyals, O. & Le, Q. (2015). A Neural Conversational Model. *Proceedings of the 31 st International Conference on Machine Learning ICML*. http://arxiv.org/abs/1506.05869

[230] Vogt, P. (2002). The physical symbol grounding problem. *Cognitive Systems Research*, *3*(3), 429-457. DOI : 10.1016/S1389-0417(02)00051-7

[231] Vygotsky, L. S. (1978). *Mind in Society* (M. Cole, V. John-Steiner, S. Scribner & E. Souberman, Éd.). Harvard University Press. http://home.fau.edu/musgrove/web/vygotsky1978.pdf

[232] Vygotsky, L. S. (1986). *Thought and language* (Alex Kozulin, Éd.). MIT press Cambridge. DOI : 10.3233/BEN-1992-5106

[233] Wang, S., Zhang, J. & Zong, C. (2018a). Associative multichannel autoencoder for multimodal word representation. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*, 115-124. DOI : 10.18653/v1/d18-1011

[234] Wang, S., Zhang, J. & Zong, C. (2018b). Learning Multimodal Word Representation via Dynamic Fusion Methods. *Thirty-Second AAAI Conference on Artificial Intelligence*. www.aaai.org

[235] Ware, M., Frank, E., Holmes, G., Hall, M. & Witten, I. H. (2001). Interactive machine learning : Letting users build classifiers. *International Journal of Human Computer Studies*, *55*(3), 281-292. DOI : 10.1006/ijhc.2001.0499

[236] Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards* (thèse de doct.). University of Cambridge.

[237] Weng, J., McClelland, J., Pentland, A., Sporns, O., Stockman, I., Sur, M. & Thelen, E. (2001). Autonomous Mental Development by Robots and Animals. *Science*, *291*(5504). DOI : 10.1126/science.291.5504.599

[238] Wieting, J., Bansal, M., Gimpel, K. & Livescu, K. (2015). Towards Universal Paraphrastic Sentence Embeddings. *International Conference on Learning Representations (ICLR 2016)*. http://arxiv.org/abs/1511.08198

[239] Wieting, J., Bansal, M., Gimpel, K., Livescu, K. & Roth, D. (2018). Erratum : "From Paraphrase Database to Compositional Paraphrase Model and Back". *Transactions of the Association for Computational Linguistics*, *3*, 600-600. DOI : 10.1162/tacl_a_00246

[240] Winograd, T. (1972). Understanding natural language. *Cognitive Psychology*, *3*(1), 1-191. DOI : 10.1016/0010-0285(72)90002-3

[241] Wood, D., Bruner, J. S. & Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry*, *17*(2), 89-100. DOI : 10.1111/j.1469-7610.1976.tb00381.x

[242] Wu, H., Huang, H., Jian, P., Guo, Y. & Su, C. (2017). BIT at SemEval-2017 Task 1 : Using Semantic Information Space to Evaluate Semantic Textual Similarity. In S. Bethard, M. Carpuat, M. Apidianaki, S. M. Mohammad, D. M. Cer & D. Jurgens (Éd.), *Proceedings of the 11th International Workshop on Semantic Evaluation, SemEval@ACL 2017, Vancouver, Canada, August 3-4, 2017* (p. 77-84). Association for Computational Linguistics. DOI : 10.18653/v1/S17-2007

[243] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C. & Yu, P. S. (2021). A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(1), 4-24. DOI : 10.1109/TNNLS.2020.2978386

[244] Yang, C. D. (2004). Universal Grammar, statistics or both ? *Trends in Cognitive Sciences*, *8*(10), 451-456. DOI : 10.1016/j.tics.2004.08.006

[245] Yao, K., Zweig, G., Hwang, M.-Y., Shi, Y. & Yu, D. (2013). Recurrent Neural Networks for Language Understanding. *Interspeech*. https://www.microsoft.com/en-us/research/publication/recurrent-neural-networks-for-language-understanding/

[246] Yu, C. & Ballard, D. H. (2004). On the Integration of Grounding Language and Learning Objects. *AAAI*, 488-494. http://www.aaai.org/Library/AAAI/2004/aaai04-078.php

[247] Yu, H., Zhang, H. & Xu, W. (2018). Interactive Grounded Language Acquisition and Generalization in a 2D World. *ICLR 2018*. http://arxiv.org/abs/1802.01433

[248] Yu, Y. (2018). Towards sample efficient reinforcement learning. *IJCAI International Joint Conference on Artificial Intelligence*, *2018-July*, 5739-5743. DOI : 10.24963/ijcai.2018/820

[249] Yu, Z., Yu, J., Cui, Y., Tao, D. & Tian, Q. (2019). Deep Modular Co-Attention Networks for Visual Question Answering. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 6281-6290.

[250] Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R. & Smola, A. J. (2017). Deep Sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan & R. Garnett (Éd.), *Advances in Neural Information Processing Systems* (p. 1-11). Curran Associates, Inc. http://arxiv.org/abs/1703.06114%20https://proceedings.neurips.cc/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf

[251] Zellers, R., Bisk, Y., Schwartz, R. & Choi, Y. (2018). SWAG : A Large-Scale Adversarial Dataset for Grounded Commonsense Inference. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*, 93-104. http://arxiv.org/abs/1808.05326

[252] Zhang, Y., Ou, Z. & Yu, Z. (2020). Task-Oriented Dialog Systems that Consider Multiple Appropriate Responses under the Same Context. *Association for the Advancement of Artificial Intelligence AAAI*. www.aaai.org

[253] Zhang, Z., Takanobu, R., Zhu, Q., Huang, M. L. & Zhu, X. Y. (2020). Recent advances and challenges in task-oriented dialog systems. *Science China Technological Sciences*, *63*(10), 2011-2027. DOI : 10.1007/s11431-020-1692-3

Titre : Langage et Apprentissage en Interaction pour des Assistants Numériques Autonomes

Mots clés : Langage – Apprentissage en interaction – Apprentissage automatique – Intelligence artificielle développementale – Assistant numérique – Cognition

Résumé : Les assistants numériques apprenant par interaction sont personnalisables par leurs utilisateurs, ces derniers pouvant leur enseigner de nouvelles procédures. Ces travaux s'appuient sur des méthodes d'apprentissage automatique pour enrichir leurs capacités de généralisation linguistique et procédurale. Pour concilier des facultés d'apprentissage rapide, nécessaire à une expérience utilisateur fluide, avec une capacité de généralisation suffisamment large, on s'inscrit dans le cadre de l'IA développementale. Ces travaux sont ainsi inspirés par certains processus cognitifs à l'œuvre chez l'enfant lors de l'apprentissage du langage. On propose un premier module de NLU qui, en interprétant les requêtes d'un utilisateur par des méthodes de comparaison sémantique, offre un outil de généralisation sur le langage qui s'adapte en continu. Un autre enjeu pour ces agents apprenants est leur capacité à transférer leur connaissance à de nouveaux objets et contextes. On propose alors une série d'agents de Deep RL capable d'exécuter des tâches exprimées en langage naturel dans des environnements variés. En exploitant le langage comme outil d'abstraction pour représenter les tâches, on montre, que dans un environnement structuré, ces agents peuvent généraliser des compétences à de nouveaux objets. On développe enfin un cas d'usage dans un environnement domotique. On propose un assistant apprenant qui intègre les systèmes mentionnés précédemment.

Title: Language and Interactive Learning for Autonomous Digital Assistants

Keywords: Language – Interactive learning – Machine learning – Developmental artificial intelligence – Digital assistant – Cognition

Abstract: Digital assistants learning by interaction allow users to personalise the way they respond to queries, by learning new procedures. This work leverages machine learning to enrich the linguistic and procedural generalisation capabilities of these systems. To reconcile rapid learning skills, necessary for a smooth user experience, with a sufficiently large generalisation capacity, we approach these issues from the perspective of developmental AI. This work is thus inspired by specific cognitive processes of the child during language learning. First, we propose a language processing module, which relies on semantic comparison methods to interpret the user's natural language requests. The variability in a user speech is indeed one of the central difficulties of these learning assistants. We provide them with a generalisation tool to continuously adapt to the user language. Another challenge for these learning agents is their ability to transfer their knowledge to new objects and contexts. We propose a series of architectures for Deep Reinforcement Learning agents that learn to perform tasks expressed in natural language in various environments. By exploiting language as an abstraction tool to represent tasks, we show that in structured environment, these agents can transfer their skills to new objects. Finally, we develop a use case in domotic environment. We propose a learning assistant that integrates the systems mentioned above.

Université Bourgogne Franche-Comté
32, avenue de l'Observatoire
25000 Besançon