Institut national des langues
et civilisations orientales

Orange Bank

Agent conversationnel pour la relation client - Technologies, architectures et cas d'usage au domaine bancaire

Pierre Rochet

Encadrants Université
Damien Nouvel
Elvis Mboning

Encadrant Entreprise
Djamel Mostefa

2020

Remerciements

Je souhaite avant tout remercier mes deux professeurs et encadrants de mémoire Damien Nouvel et Elvis Mboning, auprès desquels j'ai énormément appris durant cette dernière année. Je tiens également à témoigner ma gratitude à Djamel Mostefa, *Head of IA* et encadrant d'entreprise qui m'a fait confiance et permis de réaliser ce travail passionnant chez Orange Bank. Je remercie aussi tous les membres de l'équipe IA qui m'ont merveilleusement intégré, particulièrement Nicolas, Antoine et Nassim pour tout ce qu'ils m'ont appris. Enfin j'exprime ma reconnaissance à mes parents qui m'ont toujours soutenu et qui ont largement contribué à la réussite de mes études.

Résumé

Nous proposons un agent conversationnel destiné à des conseillers clientèles dans le domaine bancaire. Nous montrons qu'il est possible de créer une architecture complètement basée sur des outils open source fiables. Nous nous appuyons sur le *framework* RASA, et basons nos choix des modèles d'apprentissage sur l'équilibre coûts-performances. L'automatisation entre modélisation et implémentation ainsi que la structure en thématiques facilitent grandement la flexibilité du système. Enfin nous présentons une perspective d'évolution pour intégrer la gestion d'une FAQ dans l'architecture proposée. Ce composant, qui allie recherche d'information et système de questions-réponses, permettrait d'élargir le périmètre de l'agent pour peu d'efforts tout en restant très modulable.

Mots-clés— Agent conversationnel, *chatbot*, relation client, banque, *open source*

Table des matières

Liste des tableaux

Table des figures

Introduction

Dans le monde contemporain, les outils numériques occupent une place de plus en plus centrale dans nos vies. Les modes d'interactions ont beaucoup évolué ces dernières années, depuis l'utilisation du clavier puis de la souris, les interfaces web avec liens et formulaires, et plus récemment la mise en œuvre d'agents conversationnels. Ces derniers mettent l'accent sur l'interaction en temps réel pour informer, guider, et parfois exécuter des actions à la place de l'utilisateur.

Les avantages apportés par les agents conversationnels représentent un grand intérêt. Leurs nombreuses qualités couplées à l'amélioration croissante de leurs performances ont provoqué un véritable engouement de la part des entreprises.

C'est le cas d'Orange Bank qui, depuis 2016, développe et maintient son assistant virtuel Djingo destiné à ses clients. A la demande du client ou lorsque la question est trop complexe pour Djingo, une redirection est automatiquement effectuée vers le centre relation client (CRC). Un conseiller réalise alors un diagnostic et répond à sa demande à l'aide d'une large documentation. C'est dans le but d'assister les conseillers dans la réalisation de ces tâches que l'entreprise a souhaitée développer un nouvel agent. L'objectif est de standardiser les diagnostics et les réponses tout en facilitant et dynamisant l'accès à l'information.

Cette demande a été l'occasion d'analyser le besoin par rapport aux technologies disponibles sur le marché. De prendre conscience que les dernières technologies état de l'art ne sont pas forcément adaptées aux besoins des entreprises, que ce soit par manque de modularité ou pour les trop grandes ressources en calculs qu'elles nécessitent.

Les réflexions menées et les choix présentés ont toujours été formulés avec un objectif opérationnel. Ce travail retrace donc les étapes de la mise en place d'un projet *chatbot* dans une entreprise, de la problématique initiale jusqu'à l'amélioration continue. Les réflexions menées et les démarches réalisées dressent les contours d'une méthodologie pour qui doit amorcer un projet de *chatbot* similaire. Elle se veut également différente car la plupart des travaux de *chatbot* pour la relation client se destinent à des interactions client alors que le nôtre est destiné à une équipe interne à l'entreprise.

Il répond donc à la problématique : Comment mettre en place une solution *chatbot* dans une entreprise ? L'objectif final étant de proposer une architecture pratique, légère, flexible et pour un coût minimal.

Nous considérons que les termes agent conversationnel et *chatbot* sont équivalents. Nous utiliserons davantage le second par souci de fluidité de lecture.

Chapitre 1

Les chatbots : historique, représentation et modélisation

Le concept des agents conversationnels, aussi appelé chatbot n'est pas récent. En 1950, Alain Turing [Turing, 1950] présente le célèbre test de Turing afin d'évaluer la capacité d'une intelligence artificielle à converser comme un humain.

En 2015, l'aspect conversationnel à pris une place prépondérante sur le web, notamment avec les réseaux sociaux qui s'orientent de plus en plus vers la communication instantanée. Les entreprises l'ayant bien compris ont souhaité exploiter ces canaux pour atteindre un maximum d'utilisateurs. Les *chatbots* ont alors rapidement été destinés à investir ce domaine.

En avril 2016, Facebook contribue largement à la démocratisation des agents conversationnels en proposant un service permettant aux développeurs de déployer facilement leurs *chatbots* directement dans l'application Messenger. Le succès fut immédiat avec plusieurs dizaines de milliers de *chatbots* déployées les premiers mois. Beaucoup d'entreprises en ont fait leur nouveau média commercial.

Les avantages majeurs d'un agent conversationnel sont les suivants :

— sa disponibilité, il peut fonctionner 24h/24 et 7j/7,

— une amélioration générale de la relation client,

— l'ajout d'un canal e-commerce souvent à moindre coût (comparé aux autres canaux),

— un accès à l'information facilité et dynamisé,

— un gain de productivité pour l'entreprise

1.1 Chatbot et intelligence artificielle

Le succès des agents conversationnels s'accompagne d'un fantasme historique : pouvoir interagir avec les machines par le dialogue en langage naturel, le mode de communication le plus commode de l'homme. Cependant si une machine est aujourd'hui capable de dialoguer plus ou moins naturellement avec l'homme, nous sommes encore très loin d'avoir atteint l'intelligence artificielle forte imaginée par les films de science-fiction.

Si nous affirmons que les *chatbots* entrent dans le domaine de l'intelligence artificielle, nous en sommes en réalité réduit à créer des systèmes extrêmement limitées. La capacité des systèmes « intelligents » que nous concevons est parfaitement illustrée par l'expérience de pensée « la chambre chinoise » de [Searle, 1980]. En 1980 Searle réalise une expérience de

pensée à propos du raisonnement des machines. Searle imagine une personne enfermée dans une pièce avec un manuel de linguistique chinoise. Cette personne n'a aucune connaissance de cette langue. L'opérateur reçoit des phrases écrites en chinois et, en appliquant les règles qu'il a à sa disposition, il produit d'autres phrases en chinois. L'interlocuteur enfermé semble alors communiquer comme un véritable chinois. En réalité il ne fait qu'appliquer les règles et n'a donc aucune compréhension des phrases qu'il produit. Imaginons maintenant que l'interlocuteur enfermé devient si habile avec les règles et sinogrammes qu'aucun moyen ne permet de le distinguer d'un véritable chinois. En revanche il n'est toujours pas capable d'expliquer le signifié des mots utilisés dans les phrases. Cette expérience suppose qu'il ne suffit pas d'ordonner des formes dans un ordre précis pour maîtriser le langage. La volonté de signifier et la conscience des concepts rendent impossible la création d'une intelligence artificielle forte.

La réalisation d'un *chatbot* reste donc qu'un ensemble de règles et de connaissances qui régissent un dialogue. Il n'y a pas de nécessité à réellement comprendre ce que dit l'utilisateur pour dialoguer efficacement avec lui. Nous souhaitons plutôt tendre vers une imitation convaincante de l'intelligence humaine. C'est cet objectif que nous tentons d'atteindre dans la réalisation des agents conversationnels. En les spécialisant à un domaine ou un cas d'usage particulier, nous obtenons des imitateurs très utiles.

1.2 Pourquoi un chatbot pour l'entreprise ?

D'après [Vidal F., 2016], les entreprises américaines fortement digitalisées ont une croissance deux fois supérieure et la rentabilité pour les actionnaires est trois fois supérieure à la moyenne. Cela fait maintenant plusieurs années que les *chatbots* sont intégrés dans les entreprises et notamment dans les stratégies de relation-client. Cette technologie fait donc intégralement partie de cet enjeu de digitalisation. Ces derniers répondent efficacement aux besoins métiers en étant au service des clients tout en diminuant la charge de travail des conseillers clientèles.

En 2018, [DEBOS, 2018] présente les résultats d'un sondage effectué sur 187 cadres et dirigeants d'entreprises dans les services de l'innovation, de l'informatique, et du marketing.

— 38% des entreprises interrogées développaient un *chatbot* alors que 52% l'envisageaient sérieusement. Parmi elles, 39% pensaient que les *chatbots* révolutionneront l'entreprise. 25% se disaient non prêtent au changement. Ces chiffres indiquent un très grand intérêt envers ces technologies.

— Concernant les secteurs d'activités, c'est 50% des entreprises qui ciblent les centre d'assistance pour la première utilisation de leur *chatbot*. 35% se tournent vers le service client et 12% seulement vers le marketing et le e-commerce.

— Pour 78% des entreprises, l'acquisition d'un *chatbot* doit créer une relation personnalisée avec le client pour le fidéliser, 74% veulent réduire les coûts, 68% faciliter un processus d'achat et 66% souhaitent proposer un accompagnement client simplifié et disponible 24h/24 et 7j/7.

— Dans une moindre mesure les entreprises évoquent également les objectifs suivant : apporter un service novateur (62%), avoir une meilleure traçabilité (60%), améliorer l'expérience utilisateur (45%), diminuer les sollicitations envers le service client (43%), automatiser les tâches simples avec peu de valeur ajoutée et résoudre les problématiques dites de premier niveau (32%).

— 28% pensent que la compréhension du langage est l'élément technologique le plus déterminant, 19% mentionnent l'analyse des questions et l'apprentissage automatique. 16% s'intéresse également à la mise en récit des conversations (Story Telling) adaptés aux utilisateurs.

— Après la démonstration d'un *chatbot* 80% des entreprises sont convaincues.

Au-delà des intérêts évoqués ci-dessus, de nombreux industriels sont convaincus que les applications mobiles seront peu à peu remplacées par des agents conversationnels au mode d'interaction plus intuitif. L'analyse rapporte également que le marché des *chatbots* est évalué à plus d'un milliard d'euro en 2024 en engendrant des économies estimées à près de 8 milliards d'euros

1.3 Quelques chatbots emblématiques

— L'une des premières tentatives, ELIZA apparaît en 1966 [Weizenbaum, 1966]. Sa mission : imiter le questionnement d'un psychanalyse s'adressant à son patient. Malgré un système symbolique relativement simple, certains développèrent une empathie spontanée suite à leur discussion avec ELIZA. Un phénomène appelé « effet Eliza ». [Weizenbaum, 1976, Hofstadter, 1996]

— ALICE créé en 1995. Ce programme fortement inspirée d'ELIZA a remporté trois fois le prix Loebner attribué au meilleur *chatbot* en 2000, 2001 et 2004.

— En 2011, Watson d'IBM remporte le jeu télévisé américain Jeopardy où les candidats doivent répondre à des questions posées. Il prouve alors sa capacité à comprendre le langage naturel. Rapidement les briques technologiques constituant Watson ont été commercialisées. Elles sont depuis proposées aux entreprises qui souhaitent intégrer cette technologie dans leurs services.

— La première version de Siri voit également le jour en 2011 sur l'Iphone 4S. Elle est et reste depuis l'assistant intelligent pour les utilisateurs des produits Apple.

— Viens ensuite le tour d'Amazon qui annonce la sortie de son assistant Alexa en 2014 destiné à devenir le principal canal de communication de son enceinte connectée Echo.

— La même année Cortana de Microsoft apparait pour équiper les Windows phones et les ordinateurs sous Windows 10.

— Peu de temps après, en 2016, c'est au tour de Google de sortir son assistant virtuel « l'assistant Google ». Il équipera les *smartphones* Android à partir de l'année 2017

C'est globalement entre les années 2014 et 2016 que les géants du numériques se dotent d'un assistant virtuel. Les tendances des recherches Google nous confirment que dans cette même période l'engouement pour ce service s'est considérablement accéléré.

FIGURE 1.1 – Tendances de recherche Google du terme *chatbot*

1.4 Les différents types de chatbot

Niveau	Description
1	L'état le plus simple d'un *chatbot*. Il envoie des notifications sous forme de message dans des applications de messagerie. Il prévient par exemple lorsqu'une promotion vient d'apparaître chez un commerçant ou lorsque vous devez renouveler un abonnement. Il s'agit en réalité d'un simple message prédéfini déclenché par évènement et habillé d'une fenêtre de messagerie. Il n'y a donc pas de système de compréhension du langage
2	Le type d'agent le plus courant. Il comprend les questions simples et peut y répondre. Il renvoie souvent des réponses assez générales et manque de précision. Il ne peut cependant pas prendre en compte le contexte de la conversation. Le niveau de profondeur des dialogues est le plus souvent de 1 ou 2 maximums. Il est souvent utilisé pour rendre une FAQ plus dynamique.
3	A ce niveau l'agent intériorise le contexte. L'historique des messages précédents et les informations déduites sont considérés comme acquis pour les échanges suivants. Pour répondre à une demande, le bot est capable de poser différentes questions successives pour recueillir des informations complémentaires. Le niveau de profondeur des dialogues peut donc être plus ou moins important suivant la difficulté de la demande initiale.
4	Le bot est capable d'apprendre à connaître l'utilisateur avec le temps. Il mémorise les goûts et les habitudes à long terme. Il intervient de façon autonome en fonction du contexte. Il connaît la personnalité d'un utilisateur fréquent, ce qui lui permet d'apporter une aide personnalisée.
5	Une intelligence artificielle indépendante qui connaît personnellement son utilisateur. Elle peut effectuer une grande partie des actions à sa place et en toute autonomie. L'IA est capable de gérer un processus complexe de A à Z sans aucune assistance humaine en se basant sur ses connaissances précises de son utilisateur. Elle est également capable de proposer des offres personnalisées et trouver instantanément les services les plus adaptés d'un client. Elle possède un pouvoir de décision élaborée.

TABLE 1.1 – Niveau d'évolution des *chatbots* par Alan Nichol

1.4.1 Les différents niveaux d'évolution

L'évolution rapide des technologies et des formalismes couplés à la grande diversité des fonctions des différents systèmes de dialogue crée un écosystème extrêmement dense et dynamique. Cet environnement foisonnant rend difficile une classification d'évolution. Alan Nichol, co-fondateur du *framework* Rasa, propose dans son article [Nichol, 2020] une hiérarchisation macroscopique en 5 niveaux (voir le tableau 1.1). La majeure partie des systèmes de *chatbot* actuels appartiennent à la deuxième et troisième catégories.

1.4.2 Les deux grandes classes de chatbot : conversationnels VS orientés but

Les systèmes de dialogue sont habituellement classés en deux catégories :

— Les agents de dialogue orientés sur la conversation : Ils peuvent par exemple servir d'interlocuteur alternatif dans le domaine de la santé ou simplement pour tenir compagnie aux personnes seules. L'accent est d'avantage mis sur la capacité à pouvoir tenir une conversation cohérente et fluide. Ses domaines de discussions sont souvent assez larges. L'avancée de ces systèmes toujours plus humains s'accompagne de questions éthiques cruciales.

— Les systèmes de dialogue orientés tâches : Ils sont utilisés pour aider l'utilisateur à accomplir diverses tâches dans un ou plusieurs domaines spécifiques. L'accent est davantage mis sur l'action, mais sa capacité à dialoguer reste un élément essentiel de sa qualité. Ce sont ces derniers qui intéressent majoritairement les industriels. Ils peuvent être déployés dans divers contextes commerciaux, tels que le marketing, les médias sociaux (achats personnalisés, procédures d'achat simplifiées, etc.) ou la relation client.

1.5 Comprendre le degré d'autonomie

Comme nous l'avons vu, les agents conversationnels sont très appréciés en entreprise. Ils peuvent être destinés aussi bien aux clients (aide aux achats en ligne, informations clients, etc) qu'aux collaborateurs (aide aux ressources humaines, recrutement, communication interne, aide à la relation client, etc). Il y a donc de nombreuses situations où un *chatbot* peut venir optimiser un processus. Les promesses autour de cet engouement amènent de nombreuses attentes de la part des services concernés, des attentes qui surestiment parfois les capacités des systèmes. En réalité un *chatbot* est rarement autonome sur son activité. La tendance qui se dégage est un support et non un remplacement des ressources humaines. Les chabots rentabilisent davantage par leurs capacités à augmenter la productivité des salariés et non pas à les remplacer. Traiter plus de requêtes plus rapidement répond à l'exigence d'instantanéité des clients et des collaborateurs. Les équipes physiques ont alors la possibilité de se concentrer sur l'aspect émotionnel des échanges et de répondre aux problématiques les plus complexes. D'une manière générale c'est cette complémentarité *chatbot*-humain qui enrichit et améliore les processus.

Concrètement il y aura toujours une situation que le bot ne pourra pas gérer, soit parce que sa capacité à comprendre le langage est limitée soit parce que la demande de l'interlocuteur est hors de son périmètre. Trois réactions sont alors envisageables : il détecte et communique son incapacité à répondre et la conversation s'arrête, il tente une désambiguïsation ou il effectue une redirection vers un humain qui prendra en charge le problème.

Les cas d'usage et le périmètre du bot doivent donc être consciemment délimités. Connaître précisément les limites du bot permettra de mieux gérer les cas non pris en charges. Il est courant de voir certaines conversations tourner en boucle devant un problème mal géré, provoquant généralement une grande frustration chez l'utilisateur. Une délimitation claire permettra également une meilleure vision des perspectives d'évolutions.

1.6 Comment modéliser les conversations ?

Lorsqu'une entreprise souhaite développer un *chatbot*, elle sait plus ou moins quelles sont les informations que le bot doit restituer. Elle veut amener l'utilisateur dans un processus dialogique provoqué par une demande initiale. Pour les cas les plus simples, une demande peut déclencher la réponse instantanément sans qu'il y ait d'échanges intermédiaires, par exemple des salutations attendent d'autres salutations, un remerciement attend une locution de politesse, etc. Mais de nombreux cas n'ont pas de réponse directement associée, il sera alors nécessaire d'effectuer des échanges intermédiaires avec l'utilisateur afin de désambiguïser la demande, apporter des détails ou bien effectuer un diagnostic.

Il est donc crucial de déterminer les cas d'usages, le périmètre, les différentes réponses que le bot est capable d'apporter et surtout les cheminements logiques menant à chaque réponse. Il s'agit finalement de créer les capacités dialectiques [1] artificielles du bot.

L'expérience chez Orange Bank a permis d'identifier des critères déterminants pour réaliser cette modélisation. En premier lieu, il s'agit de savoir qui sera en charge de réaliser ce travail. Il y a en général deux possibilités : les experts métier (les personnes expertes du service concerné par le bot), ou le développeur. Ce choix aura un impact sur la suite de la démarche.

Le cas d'usage réalisé chez Orange Bank a montré que la réalisation par les métiers fonctionnait relativement bien. Ils ont l'expertise et le vocabulaire adapté à la rédaction des réponses, la clairvoyance sur les processus et les traitements des cas d'usages ainsi que leurs complexités. Ils pourront alors dans un premier temps se consacrer exclusivement à la résolution des problèmes traités par le futur *chatbot* pendant que le développeur adaptera les développements pour accueillir au mieux la modélisation des métiers. Il pourra également, en cas de besoin, rappeler les limitations techniques influençant la modélisation. Voir la figure 1.2. Cette répartition des tâches ne doit donc pas exclure l'étroite collaboration qui doit persister entre les équipes.

1. Dialectique du Moyen-Âge reposant sur la logique formelle d'Aristote

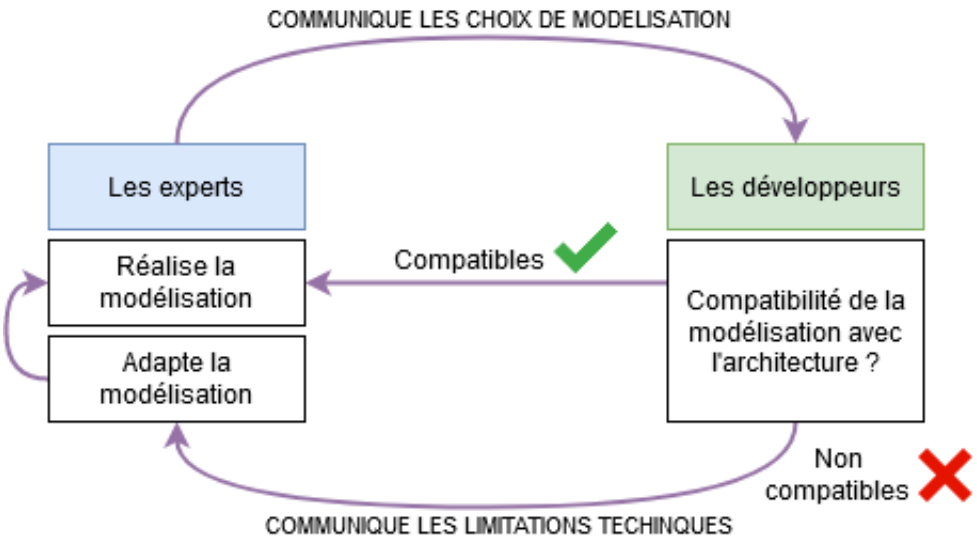

FIGURE 1.2 – Collaboration entre les équipes métiers et développeurs

Le travail effectué chez Orange Bank a également permis d'identifier plusieurs critères. Les entreprises évoluent constamment, ce qui impacte nécessairement les outils qu'ils utilisent. C'est typiquement le cas lorsque l'offre de l'entreprise change, que l'organisation interne est modifiée ou qu'un nouveau problème émerge. Il y a donc de fortes probabilités que les réponses ou les diagnostics doivent être modifiées au cours de la vie du *chatbot*. Même si les équipes n'en formulent pas la demande initialement, il faudra toujours considérer cette possibilité. Il sera donc nécessaire de mettre en place une stratégie de modélisation évolutive. La modélisation est une conceptualisation destinée à transiter entre les collaborateurs, il doit être structuré et compréhensible par tous. La sémiotique utilisée doit donc être déterminée en amont. Le choix de l'outil utilisé ne doit pas être négligé. Rappelons cependant qu'il ne doit pas orienter les choix logiques de la modélisation. On souhaite également réduire les efforts entre la phase de modélisation et la phase d'implémentation. On anticipera donc le format pour une éventuelle automatisation.

On retiendra les points suivants :

— modélisation évolutive,
— compréhensible par tous,
— facilité de transfert,
— potentiellement automatisable.

1.6.1 Exemple de modélisation par graphe

Le format est relativement libre sous conditions de rassembler les critères énumérés ci-dessus. L'aspect arbre de décision que dessine une conversation avantage nettement le format par graphe très utilisé pour la modélisation des conversations.

La représentation par graphe est depuis longtemps utilisée pour modéliser les dialogues en arbre de décision. La figure 1.3 est une modélisation par graphe issue de Djingo le *chatbot* d'Orange Bank destiné aux clients. Il modélise le cas où l'utilisateur souhaite connaître les frais d'opposition. La demande initiale est en rouge, les réponses sont en vert et les conditions en orange.

L'utilisateur formule d'abord sa demande, s'il est non identifié le bot apporte la réponse, s'il est identifié le bot demande le moyen de paiement signifié avec des boutons (aussi appelé « clictext »). S'il s'agit d'une carte bancaire, le bot effectue une requête API afin de connaître le type : « classique » ou « premium » puis apporte sa réponse.

Cette modélisation (figure 1.3) réalisée avec le logiciel Drawio a l'avantage d'être extrêmement claire. Elle est intuitive et reprend l'ensemble des éléments requis pour l'implémentation, notamment les identifiants pour les demandes (aussi appelé « intention ») et les réponses. Ces identifiants sont visibles dans les losanges.

Malheureusement, cette modélisation n'a aucune interopérabilité avec la phase d'implémentation. En observant les développements continus de Djingo, on constate que l'ajout d'un cas d'usage requiert une phase de modélisation et une phase d'intégration. Ces deux phases très dépendantes l'une de l'autre sont extrêmement chronovores.

En réalité les technologies utilisées pour la création de *chatbot* négligent la phase de modélisation et ne fournissent pas d'outil intégré. La modélisation se fait donc presque systématiquement sur un outil externe, ce qui rend difficile l'intégration automatique depuis la modélisation. Les formats utilisés (souvent pdf, png, jpef, etc.) renforcent cette difficulté.

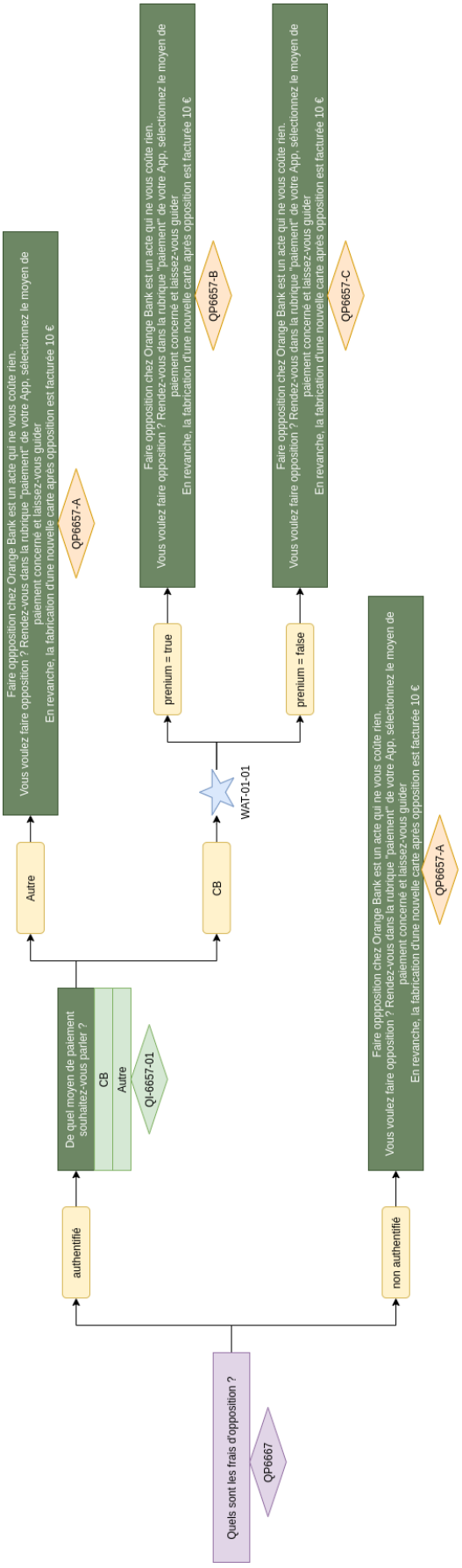

FIGURE 1.3 – Exemple d'une modélisation par graphe

En fonction des besoins et du contexte il est tout à fait possible de réaliser son propre format. C'est ce que nous avons choisi de faire chez Orange Bank afin de mieux répondre aux besoins. Nous disposions des données initiales sous la forme de documents Powerpoints. Afin d'accélérer le processus de modélisation nous avons converti et adapté ces documents en format tabulaire sous Excel. Nous détaillons ce traitement dans la partie Cas d'usage : Un chatbot pour la relation client destiné aux conseillers.

Chapitre 2

Les différentes architectures et technologies

2.1 La méthode traditionnelle déterministe

2.1.1 Détection par mots-clés

Une des premières approches pour réaliser un *chatbot* est d'établir des règles pour détecter des informations significatives dans le message d'entrée. Les éléments extraits permettent d'identifier le type de demande et la réponse à apporter. ELIZA utilise cette méthode sous sa forme la plus triviale. Elle utilise la correspondance de mots-clés ainsi qu'un contexte très limité. Lorsqu'un mot-clé est détecté dans le message de l'utilisateur, des règles associées à ce mot sont appliquées pour générer une nouvelle réponse. Ces règles intègrent quelques connaissances conceptuelles basiques du monde. Par exemple un message comportant le mot-clé « mère » ou « fils » ou tout autre mot lié à un lien de parenté provoque une réponse du type « Parlez-moi de votre famille ».

2.1.2 La langage AIML

AIML (*Artificial Intelligence Markup Language*) est un langage de balisage basé sur XML créé par [Wallace, 2003] et initialement développé pour le *chatbot* ALICE. Il s'agit d'une norme pour créer des flux de conversation. Il gagne fortement en popularité lorsqu'ALICE remporte trois fois le prix Loebner. Par exemple, La balise <pattern> permet de détecter la demande à l'aide d'expressions plus ou moins générales alors que la balise <template> associe la réponse à envoyer. Il est également possible d'effectuer d'autres actions comme par exemple la détection de certaines entités pour aider la compréhension ou les réutiliser dans les réponses.

L'avantage est qu'il permet un contrôle total sur la compréhension des messages, la conséquence est un travail d'analyse poussé et un nombre de règles qui croît rapidement avec la complexité du périmètre du bot. Il peut être très pratique pour des *chatbots* simples mais ne sera généralement pas adapté pour des *chatbots* plus évolués. Il a été très utilisé dans les années 2000 mais ce raréfie avec la popularité du *machine-learning*. Néanmoins certains projets de ces dernières années utilisent encore AIML.

[Yamaguchi et al., 2018] proposent une méthode d'extraction automatique de règles AIML qui réduiraient considérablement l'effort humain. Le processus de conversion des données en un ensemble de règles AIML comprend les étapes suivantes :

— Prétraitement : Les messages non signifiants sont filtrés. Ce sont des messages qui contiennent généralement seulement des hyperliens, des hashtags, ou des noms.

— Normalisation : Les dialogues sont mappés dans des balises AIML <context>, <pattern> et <template>. Le contexte est identifié grâce à la méthode TF-IDF après une segmentations en morphèmes, l'annotation en partie du discours et la suppression des « mots-vides » comme les prépositions, prénoms, adjectifs, verbes auxiliaires, conjonctions, etc.

— Une règle est créée pour chaque triplet <context>, <pattern> et <template>.

Les utilisateurs ont montré qu'une architecture très simple basée sur AIML pouvait donner des résultats acceptables avec un très grand nombre de règles : plus de 600 000 ont été nécessaires. Les difficultés rencontrées lors de l'évaluation sont également très intéressantes. Globalement, le contexte des tours de paroles précédents à un impact non négligeable et la sélection de la réponse par similarité TF-IDF est clairement insuffisante. Autre fait relevé et applicable à toute évaluation de *chatbot* : les utilisateurs ont tendance à jouer avec le *chatbot* plutôt que d'effectuer une conversation réelle. Lors des tests, les règles de conversations classiques n'ont pas été respectées à de nombreuses reprises. Le système étant entraîné sur des conversations entre utilisateurs réels ne peut donc pas trouver de réponse adéquate. Les auteurs suggèrent deux possibilités : former les utilisateurs à dialoguer correctement avec un *chatbot* ou ne pas préciser qu'il s'agit d'un robot.

2.1.3 L'outil Chatscript

Chatscript est un outil open source populaire pour la création de *chatbot* crée en 2011 par Bruce Wilcox. Suzette, un *chatbot* écrit avec ChatScript, a remporté le prix Loebner 2010. Il utilise un moteur basé sur des règles qui doivent être écrites dans un script. Il peut également utiliser des outils de *machine-learning* pour améliorer les flux des dialogues. ChatScript tente de réduire les efforts liés à la création de règles en proposant une syntaxe extrêmement concise, l'objectif étant d'écrire des scripts le plus rapidement possible. Les modèles utilisés pour les règles peuvent avoir une profondeur de complexité extrêmement variée. L'utilisation courante est généralement un rassemblement de multiples concepts, qui sont des listes de mots ayant une signification commune.

```
#
# file: food.top
#
topic: ~food []

#! I like spinach
s: ( I like spinach ) Are you a fan of the Popeye cartoons?

        a: ( ~yes )  I used to watch him as a child. Did you lust after Olive Oyl?
            b: ( ~no ) Me neither. She was too skinny.
            b: ( yes ) You probably like skinny models.

        a: ( ~no ) What cartoons do you watch?
                b: ( none ) You lead a deprived life.
                b: ( Mickey Mouse ) The Disney icon.

#! I often eat chicken
u: ( ![ not never rarely ] I * ~ingest * ~meat ) You eat meat.

#! I really love chicken
u: ( !~negativeWords I * ~like * ~meat ) You like meat.

#! do you eat bacon?
?: ( do you eat _ [ ham eggs bacon] ) I eat '_0

#! do you like eggs or sushi?
?: ( do you like _* or _* ) I don't like '_0 so I guess that means I prefer '_1.

#! I adore kiwi.
s: ( ~like ~fruit ![~animal _bear] )  Vegan, you too...

#! do you eat steak?
?: ( do you eat _~meat ) No, I hate _0.

#! I eat fish.
s: ( I eat _*1 > )
  $food = '_0
  I eat oysters.|
```

FIGURE 2.1 – Exemple d'un script pour ChatScript

2.1.4 L'utilisation des ontologies

Des ontologies de domaines peuvent également être utilisées pour représenter la connaissance dans un *chatbot*. La mise en place d'un tel système permet au *chatbot* d'explorer des concepts et leurs relations qui pourraient être évoqués dans une conversation. Théoriquement, cela permettrait au bot d'identifier des relations jamais observées auparavant. Récemment [Shi et al., 2020] ont utilisé une ontologie pour rendre explicable la sortie générée par un modèle d'apprentissage automatique. Ils proposent un *chatbot* d'apprentissage de l'anglais basé sur la génération de réponse par un modèle GPT-2. L'ontologie permet de visualiser les connexions du réseau neuronal et d'expliquer les phrases de sortie du modèle. En plus de représenter un véritable intérêt académique ceci permet de mieux comprendre le fonctionnement du modèle de langue.

2.2 La méthode par modèles d'apprentissage

2.2.1 Les modèles d'apprentissage de type BERT

L'apprentissage automatique (*machine-learning* en anglais) est aujourd'hui majoritairement utilisé dans les système de traitement automatique du langage dont les tâches liées aux systèmes *chatbot*. La machine ne comprenant que les nombres, il est nécessaire de représenter chaque élément linguistique (souvent des mots ou des n-grammes de mots) de notre texte par une liste de valeurs numériques. Grâce à l'apprentissage profond nous n'avons pas besoin de déterminer explicitement ces listes de valeurs. Dans le domaine du TAL, cela se traduit par

des représentations vectorielles des éléments linguistiques contenus dans les textes analysés. Ces représentations sont appelées plongements de mots (ou *word embeddings*) : ce sont des vecteurs représentant les mots au moyen de caractéristiques apprises en regardant de grandes quantités de textes. La tendance est d'utiliser ces modèles pré-entraînés, capturant les spécificités du langage concerné, et d'affiner les modèles pour traiter des tâches spécifiques de TAL. C'est ce que nous appelons le *fine-tuning*. Les modèles les plus populaires de ces dernières années reposent sur l'architecture transformer qui permet d'apprendre des encodages contextuels très efficaces. Il s'agit très souvent de variantes de l'architecture BERT (*Bidirectional Encoder Representations from Transformers*) proposé par [Devlin et al., 2018]. Ces modèles étant nettement supérieurs aux méthodes d'apprentissage antérieurs, ils sont aujourd'hui massivement utilisés.

Voici les modèles BERT-*like* pour le français :

— CamemBERT : Le premier modèle de langue de type BERT (plus précisément RoBERTa) pour le français par [Martin et al., 2020]. Il a été entraîné sur le corpus OSCAR, une section française du jeu de données CommonCrawl (un jeu de données contenant les textes d'une grande quantité de pages Web). Les auteurs comparent les performances de leur modèle de langage français de type BERT à des modèles de référence, dont des modèles multilingues basés sur BERT (mBERT et UDify) et à un modèle qui n'utilise pas de plongements de mots contextuels (UDPipe Future). Ils démontrent alors la plus-value d'un modèle français en termes de performance pour différentes tâches de TAL. Leurs résultats montrent que, pour la reconnaissance d'entités nommées (NER), la performance, mesurée par le F-score, s'améliore considérablement (de 82,75 avec mBERT à 87,93 avec CamemBERT). Ce résultat prouve donc que de tels modèles sont très bénéfiques pour nos cas d'usages. En plus de prouver l'intérêt du modèle français, ils montrent que l'utilisation de modèles contextuels améliore significativement les résultats par rapport à un modèle non contextuel (UDPipe Future).

— FlauBERT : FlauBERT [Le et al., 2020] sort quelques semaines après CamemBERT. Pour l'entraîner, les auteurs utilisent une configuration similaire à celle de CamemBERT. Leurs résultats montrent eux-aussi qu'un modèle en langue française améliore les résultats par rapport à des modèles BERT similaires (multilingues) ainsi qu'à d'autres modèles basés sur le français. Les performances de FlauBERT et de CamemBERT sont très proches. Cependant, pour des tâches assez similaires de classification séquentielle, de *parsing* en constituants et POS-tagging, la performance de FlauBERT est légèrement meilleure que celle de camemBERT. Plus intéressant encore, l'ensemble FlauBERT + CamemBERT donne les meilleurs résultats.

Certaines des méthodologies présentées ci-dessous utilisent des modèles BERT classiques puisqu'elles sont à l'origine développées pour l'anglais. Nous les présentons car il est théoriquement envisageable de les reproduire en utilisant un modèle de langue pour le français avec au choix FlauBert, CamemBert ou une combinaison des deux.

Il existe deux approches courantes pour réaliser un *chatbot* par modèle d'apprentissage : La première est une approche de bout en bout. L'entrée de l'utilisateur sera directement utilisée pour générer la réponse sans modèle intermédiaire, le système est constitué d'un unique module. La seconde utilise des composants indépendants pour la compréhension du langage (NLU), la génération de la réponse (NLG) et un moteur du dialogue intermédiaire

(DP) qui reçoit la sortie du NLU afin de déterminer l'action à envoyer au NLG. L'avantage considérable à pouvoir contrôler chaque élément du système a rendu cette dernière approche extrêmement populaire auprès des entreprises.

Elle repose donc en général sur trois composants : le NLU (*Natural Language Understanding*), le gestionnaire du dialogue (Dialogue Policy), lui-même découpable en deux modules (*State Tracking et Dialogue Policy*) et le NLG (*Natural Language Generation*) responsable de la réponse à renvoyer. Il s'agit d'un schéma général auquel s'ajoutent de très nombreuses variantes. Certaines simplifient l'architecture alors que d'autres ajoutent des composants supplémentaires.

2.2.2 L'architecture par composants

2.2.2.1 La compréhension du langage (NLU)

Le NLU est généralement représenté par deux tâches : la détection d'intention et l'extraction des entités nommées.

La tâche de détection d'intention peut être de différentes natures :

— Classification d'une phrase : Étant donné une phrase, le modèle l'identifie comme appartenant à l'une des classes prédéfinies.

— Similarité de textes : Il s'agit d'une tâche de régression. Le modèle doit prédire un score indiquant la similitude sémantique de deux phrases.

— Classification de paire de phrases : Étant donné une paire de phrases, le modèle détermine la relation des deux phrases en fonction d'un ensemble d'étiquettes prédéfinies. Par exemple, elle peut consister à déterminer si deux phrases d'une paire sont sémantiquement équivalentes.

La détection d'intention est le plus souvent représentée par une tâche de classification : pour un message donné, le modèle doit déterminer l'intention parmi une liste prédéfinie.

Concernant l'extraction des entités nommées, il s'agit souvent d'identifier des éléments clés de la phrase afin de les utiliser plus tard pour effectuer des actions. Par exemple si un utilisateur énonce la phrase : « Je souhaite faire un virement de 100 euros » nous pouvons identifier une intention « Je veux effectuer un virement », et une entité « 100 », la valeur de la transaction, très utile pour des réponses sous conditions et dans le cas où le système doit réaliser l'action. En réalité les systèmes de NLU peuvent être beaucoup plus complexes en intégrant des analyseurs syntaxique, sémantique, pragmatique et bien d'autres. Nous ne nous intéresserons pas à ces derniers.

Les recherches de ces dernières années font ressortir deux tendances : Une utilisation massive des modèles de langue pré-entrainés tels que BERT, GPT2, ou encore ELMO et la volonté de rassembler détection de l'intention et extraction des entités.

[Chen et al., 2019] effectuent le *fine-tuning* d'un modèle BERT pour prédire simultanément l'intention et les positions des entités. L'intention est détectée en utilisant le premier état caché du *token* spécial [CLS], introduit préalablement comme premier *token* de chaque séquence. L'intention est prédite avec $Y^i = (W^i h_1 + b^i)$ avec h1 le premier état caché de [CLS]. Les entités sont extraites grâce aux états finaux des autres *tokens* $h_2, ..., h_T$ toujours par une fonction *softmax*. Cette méthode a l'avantage de mettre à profit la performance et la

généralisation des modèles de langues préentrainés dans une architecture très simple à mettre en place pour les deux tâches indispensables du NLU.

Mais ce type de système n'est pas sans conséquences, les coûts des calculs des entraînements (*fine-tuning* compris) sont considérables. Il s'agit d'une limite non négligeable qui freine la création d'un *chatbot* dans les entreprises qui ne disposent pas des ressources adaptées. Rappelons également qu'un *chatbot* est généralement sujet à de multiples évolutions, ce qui demande un réentraînement régulier des modèles.

Plus récemment et dans le même esprit [Bunk et al., 2020] ont introduit une architecture plus spécialisée et particulièrement adaptée à la création d'agent conversationnel : DIET (*Dual Intent and Entity Transformer*). Il donne la possibilité d'ajouter facilement un modèle de langue préentrainés qui sera combiné à des n-grammes de mots et de caractères. Ils montrent également que l'utilisation des n-grammes sans modèle préentrainé suffit à concurrencer l'état de l'art des meilleurs systèmes. Ceci est d'autant plus intéressant que les résultats sont obtenus six fois plus rapidement. Pour ce faire un *token* ___CLS___ est ajouté à la phrase préalablement segmentée. Les séquences de *tokens* et de n-grammes (n 5) sont combinées avec les plongements de mots issus du modèle de langue. La sortie est envoyée dans un transformer à 2 couches. Un CRF (*Conditional Random Field*) sera en charge de détecter les positions des entités. Après être tout deux passés dans une dernière couche d'encodage, un produit scalaire est utilisé comme calcul de similarité entre le plongement du *token* ___CLS___ et l'intention à prédire.

Une tâche de prédiction de *MASK* est également ajoutée dans le pipeline. Pour une séquence d'entrée, 15% des *tokens* sont sélectionnés, dans 70% des cas ils sont substitués par le *token* spécial ___*MASK*___, dans 10% des cas il est substitué par un *token* aléatoire et dans les 20% le *token* d'origine est conservé. L'hypothèse avancée est que cette modification aléatoire des séquences d'entrée et la prédiction du *MASK* devraient normaliser et aider le modèle à apprendre des caractéristiques plus générales.

2.2.2.2 Le gestionnaire du dialogue (DM)

Il se charge de représenter l'état courant de la conversation à partir des états précédents. Il garde en mémoire l'historique des tours de parole afin de se représenter un contexte conversationnel (*State Tracking*). Cet état l'aide à déterminer la prochaine action à effectuer : le type de réponse à envoyer, une requête API, etc. (*Dialogue Policy*). C'est généralement le composant s'appuyant le plus sur la modélisation du dialogue.

Les systèmes les plus triviaux fonctionnent par une collection de réponses indexées. Chaque réponse est associée à un contexte, souvent une intention ou une suite d'intentions et d'entités. L'avantage est un contrôle précis des actions du bot. En revanche les systèmes les plus évolués demanderont de multiples règles longues et pénibles à établir. On retrouve donc la même problématique que les architectures par règles symboliques tels que les systèmes AIML.

Beaucoup des recherches récentes se concentrent sur l'utilisation des réseaux de neurone pour l'appariement du contexte et des actions. Ceci permet notamment de produire des tours de paroles sur la base de corpus pré-annotés. Dans la pratique, cette méthode qui ne demande pas la création de règles ne fait que déplacer les efforts dans l'annotation des données qui sont souvent demandées en grandes quantités.

[Wu et al., 2018] proposent une méthode intéressante nécessitant des données d'entraînement non annotées. Ils construisent un ensemble de données non étiquetées à partir de

conversations humaines. Une annotation est effectuée pour fournir des signaux de correspondance entre les tours de parole. Ces signaux sont ensuite exploités dans un apprentissage supervisé. C'est une méthode qui pourrait être envisageable dans le domaine de la relation client. En effet, les nombreux échanges par chat entre les conseillers et les clients peuvent constituer un corpus de conversations quantitatif et qualitatif.

2.2.2.3 Le générateur de réponse (NLG)

Le NLG est responsable de construire puis envoyer une réponse textuelle à l'utilisateur. Les systèmes orientés tâches sont presque exclusivement basés sur des *templates* préalablement définis. Ceci permet de garder un contrôle permanent des réponses, ce qui garantit une syntaxe cohérente et une orthographe sans erreur. Les systèmes conversationnels libres peuvent reposer sur des modèles génératifs qui apportent des réponses à la fois variées et personnalisées mais très difficilement contrôlables. Le contenu renvoyé peut donc être syntaxiquement ou sémantiquement incompréhensible.

[Ritter et al., 2011] utilisent une méthode s'inspirant de la traduction automatique pour générer des réponses automatiquement. Partageant des ressemblances, la tâche de question-réponse est bien différente de celle de la traduction et les performances s'en font ressentir. [Shang et al., 2015, Vinyals and Le, 2015, Sordoni et al., 2015] vont donc l'adapter à la génération de réponse. A ce stade le système ne considère pas le contexte (les tours de paroles précédents), pourtant indispensable dans une conversation. [Lowe et al., 2017] vont donc tenter de résoudre ce problème en résumant les informations des tours précédents et les intégrer au modèle initial. Malgré des résultats encourageant, on constate encore trop d'incohérences entre les réponses, créant des conversations avec des tours de parole déconnectés les uns des autres. [Li et al., 2016, Li et al., 2017] vont produire des conversations plus naturelles grâce à de l'apprentissage par renforcement ou encore des réseaux adverses.

Finalement, cette technique de traduction, bien que très intéressante reste trop difficilement contrôlable, que ce soit sur le contenu des réponses ou la cohérence globale de la conversation. Ces inconvénients déterminants en font, à l'heure actuelle, un mauvais choix pour la réalisation d'un *chatbot* orienté-but en entreprise.

2.2.3 L'approche de bout en bout (end2end)

Le système end-to-end (E2E) [Wen et al., 2016, Qiu et al., 2017, Young et al., 2017] est représenté par un unique composant qui assurera à lui seul les fonctions nécessaires. Malheureusement aucune architecture basée sur le E2E n'a réellement fait ses preuves malgré les nombreuses tentatives. [Wen et al., 2016] proposent une méthode se basant sur la logique en composants présentée précédemment.

Centraliser l'ensemble du traitement a l'avantage ne pas avoir à disposer des données d'entraînement spécifiques à chaque composant. Généralement ces systèmes demandent un large corpus de dialogues pour effectuer un unique entraînement. Des corpus souvent difficiles à obtenir. De plus, la création de la réponse qui s'apparente à un modèle génératif apporte son lot de contraintes : le manque de contrôle des réponses et la difficulté d'ajouter des actions supplémentaires par des règles symboliques. Ils sont donc plus souvent utilisés pour des *chatbots* à domaines ouverts comme [Adiwardana et al., 2020]

Les résultats très intéressants obtenus avec ce type d'architecture posent la question de l'existence des liaisons intrinsèques des composants. A moins que ces derniers soient la sur-externalisation d'un unique composant incomplet en charge d'une logique dialogique trop

évoluée.

2.2.4 Comment combiner composants modulables et modèle end2end ?

Les modèles de dialogue neuronaux de bout en bout affichent de bonnes performances pour les systèmes de dialogue orientés conversation, mais le manque de flexibilité reste un frein déterminant pour les utiliser dans les systèmes de dialogue orientés but. Ajouté à cela qu'ils sont souvent très gourmands en données d'entraînement et lent à entraîner en fait souvent un mauvais choix pour les entreprises. À l'inverse, les systèmes de dialogue en composants ont des modèles solides avec des structures explicites et très modulables.

[Mehri et al., 2019] présentent plusieurs approches pour introduire de la structure dans les systèmes à base de modèle de bout en bout. Un module à base de réseaux de neurones est construit pour les trois composants de l'architecture traditionnelle (NLU, DM, NLG). Le NLU utilise un encoder LSTM pour représenter l'entrée, il passe cette représentation dans une couche linéaire, puis une fonction sigmoïde pour obtenir les scores de prédiction de chaque intention. Le DM passe la sortie du NLU et de la base de données dans une couche linéaire avec activation ReLU puis une autre couche linaire avec activation sigmoïde qui prédit l'action. Le NLG utilise la sortie des deux modules précédents ainsi le base de données qu'il passe dans une couche linéaire, un décodeur, puis une autre couche linaire avec activation *softmax*.

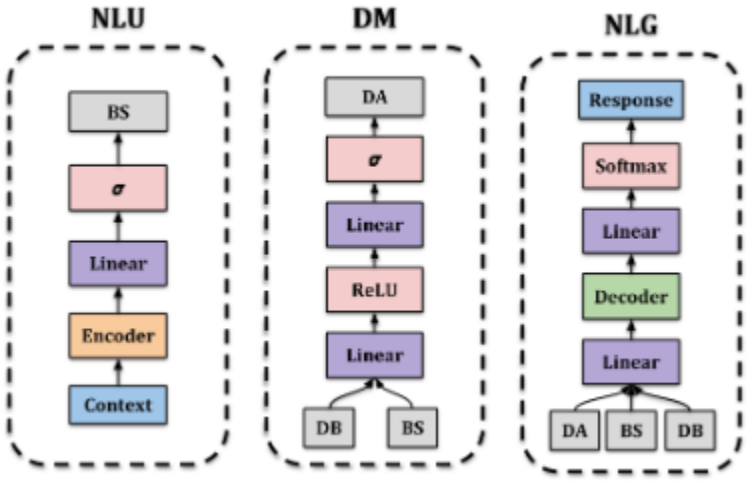

FIGURE 2.2 – Architecture de base proposée par [Mehri et al., 2019]

Les auteurs proposent ensuite différentes méthodes de fusion de ces modules dont la SFN (*Structured Fusion Networks*) semble se démarquer.

Elle consiste à entraîner des modèles indépendants pour chaque module. Un modèle de plus haut niveau est ajouté comme surcouche des trois sous-modules pour effectuer la génération de réponse de bout en bout. Les meilleures performances sont obtenues en entraînant seulement le modèle partagé, les modèles de bas niveaux sont donc figés.

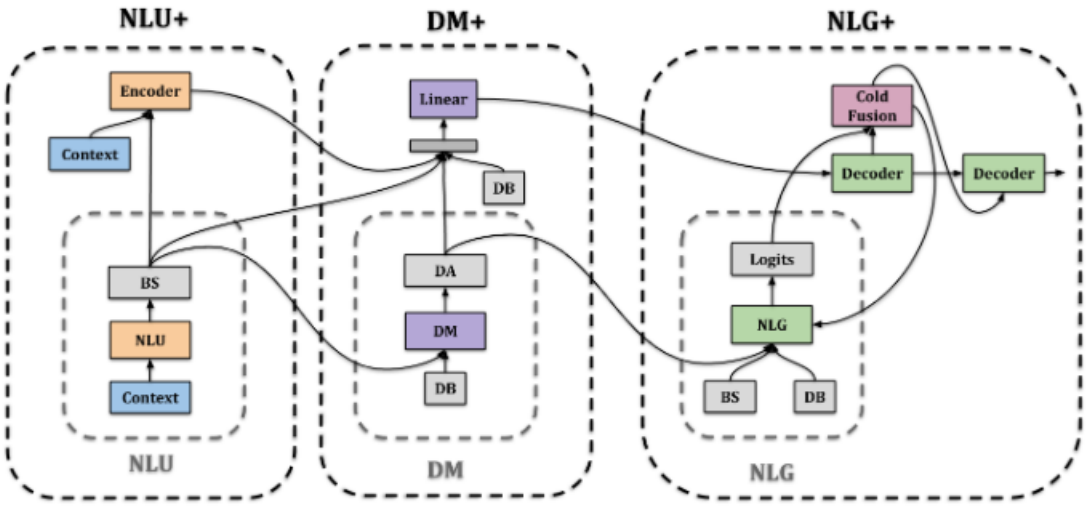

FIGURE 2.3 – Architecture SFN proposée par [Mehri et al., 2019]

Finalement on constate que les dernières avancées recherchent la modularité aussi bien à l'échelle des composants qu'à l'échelle des modèles. NLU et DM peuvent être relativement liés mais les usages obligent à garder une certaine flexibilité : le périmètre du NLU est souvent évolutif, le DM peut être lié à une base de données et utiliser des actions par règles symboliques, etc. Concernant le NLG, les systèmes de génération de réponses ainsi que les E2E ne sont pas suffisamment fiables pour les industriels. Ils favorisent le contrôle total des réponses, ce qui passe souvent par des systèmes de *template*. Même si l'effort pour la rédaction des réponses est parfois considérable, la fiabilité et la qualité des réponses sont favorisées. Dans les entreprises les plus importantes, les réponses sont validées par différents services tels que la communication et la conformité, preuve de toute l'attention qui y est apportée.

2.2.5 Pourquoi utiliser le deep learning avec parcimonie ?

2.2.5.1 Des coûts non négligeables

Comme nous l'avons vu précédemment, de nombreuses recherches reposent sur des réseaux de neurones formés sur des grands ensembles de données. Cependant, ces améliorations dépendent de ressources en calculs extrêmement importants, ce qui provoque une consommation d'énergie tout aussi importante. La conséquence sont des méthodes coûteuses aussi bien pour sur le coût des calculs, le temps d'entraînement nécessaire et l'impact environnemental. De plus, même si les progrès récents reposent sur ces méthodes, rien de garantit qu'elles fonctionnent systématiquement.

[Strubell et al., 2019] ont fait une analyse de ces coûts pour les modèles les plus populaires utilisés en NLP ainsi que pour le pipeline LISA [Strubell et al., 2018] présenté à l'EMNLP (*Empirical Methods in Natural Language Processing*) de 2018. Le temps total de formation est obtenu en entraînant les modèles jusqu'à l'achèvement de l'apprentissage et avec les recommandations matérielles des articles originaux.

La consommation électrique en kilowattheures est obtenue par la puissance électrique moyenne (en watts) de tous les CPU, la consommation moyenne de tous les DRAM (mémoire principale), la consommation moyenne d'un GPU et le nombre de GPU utilisés. La consommation électrique totale est estimée par la combinaison des consommations des GPU,

CPU, et DRAM multipliée par la PUE (*mPower Usage Effectiveness*) qui représente l'énergie supplémentaire requise pour soutenir l'infrastructure de calcul (principalement le refroidissement). Le coefficient PUE est de 1,58, la moyenne mondiale 2018 des data-centers.

La puissance totale requise à une instance donnée pendant la formation est obtenue par :

$$P_t = \frac{1.58t(p_c + p_r + gp_g)}{1000} \tag{2.1}$$

avec p_c la moyenne totale des CPU, p_r la moyenne totale pour tous les DRAM P_g la puissance moyenne d'un GPU et g le nombre de GPU.

La production moyenne de CO2 (en livres par kilowattheure) fournit par l'EPA (*Environmental Protection Agency*) pour l'énergie consommée aux États-Unis (EPA, 2018) est utilisée pour convertir l'énergie en émissions de CO2 avec

$$CO_2e = 0.954p_t \tag{2.2}$$

Consumption	**CO_2e (lbs)**
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468
Transformer (big)	192
w/ neural architecture search	626,155

FIGURE 2.4 – Estimation des émissions de CO_2 pour la formation de modèles de NLP par rapport à la consommation habituelle par [Strubell et al., 2019]

Les expériences ont été réalisées sur les 4 architectures les plus populaires : Transformer, ELMO, BERT, GP2

Model	Hardware	Power (W)	Hours	kWh·PUE	CO_2e	Cloud compute cost
$\text{Transformer}_{base}$	P100x8	1415.78	12	27	26	$41–$140
Transformer_{big}	P100x8	1515.43	84	201	192	$289–$981
ELMo	P100x3	517.66	336	275	262	$433–$1472
BERT_{base}	V100x64	12,041.51	79	1507	1438	$3751–$12,571
BERT_{base}	TPUv2x16	—	96	—	—	$2074–$6912
NAS	P100x8	1515.43	274,120	656,347	626,155	$942,973–$3,201,722
NAS	TPUv2x1	—	32,623	—	—	$44,055–$146,848
GPT-2	TPUv3x32	—	168	—	—	$12,902–$43,008

FIGURE 2.5 – Coût estimé de la formation d'un modèle en termes d'émissions de CO2 (lbs) et de coût de calcul cloud (USD) par [Strubell et al., 2019]

On observe que l'entraînement d'un modèle BERT$_b ase$ à une consommation de $C0_2$ de même ordre de grandeur que la consommation d'un vol d'avion entre New York - San Francisco pour un passager. Le coût en dollar pour ce même modèle est estimé entre 3751 et 12 571 pour un entraînement sur 64 cartes V100.

Ces inconvénients peuvent être déterminants pour les entreprises. De plus, les projets intègrent généralement une phase d'amélioration continue des modèles alourdissant encore les coûts demandés.

Pour mieux répondre à la réalité du terrain, les évaluations des nouveaux systèmes devraient systématiquement signaler le temps nécessaire à la convergence ainsi que la configuration requise, mais surtout les modèles devraient davantage être comparés par le rapport coûts-bénéfices. Les consommateurs pourraient alors mieux connaître la comptabilité des modèles avec leur besoin et le matériel dont ils disposent. Plus difficile à obtenir, il serait également souhaitable d'avoir la variation des performances sur diverses données et les procédures pour la recherche des hyperparamètres.

2.2.5.2 Quelle solution pour les chatbots ?

[Henderson et al., 2020] ont entraîné un encodeur appelé ConveRT aux niveaux des mots et des phrases sur un grand corpus de conversations issues de la plate-forme Reddit. Ils ont pu montrer que cette méthode était plus performante que la classification d'intention à l'aide des modèles préentrainés comme BERT et ELMO.

L'objectif de ConveRT est d'abord de déterminer une réponse pour un unique message donné. Il existe également une version multi-contexte combinant le contexte immédiat avec l'historique du dialogue en cours pour sélectionner la réponse. Des améliorations significatives sont observées pour les deux modèles. Les encodages formés lors de l'entraînement sont ensuite utilisés pour améliorer la classification d'intention.

10 millions de phrases de reddit sont d'abord segmentées en mots puis en sous-mots par extraction des préfixes. Les séquences sont ensuite passées dans une architecture transformer à 6 couches avec tous les poids partagés entre les messages d'entrées (input) et les réponses (label). Les deux sorties subissent une réduction de dimension. Les poids sont mis à jour pour maximiser le score de similarité cosinus entre l'input et la réponse pour les exemples positifs et le minimiser pour les scores négatifs.

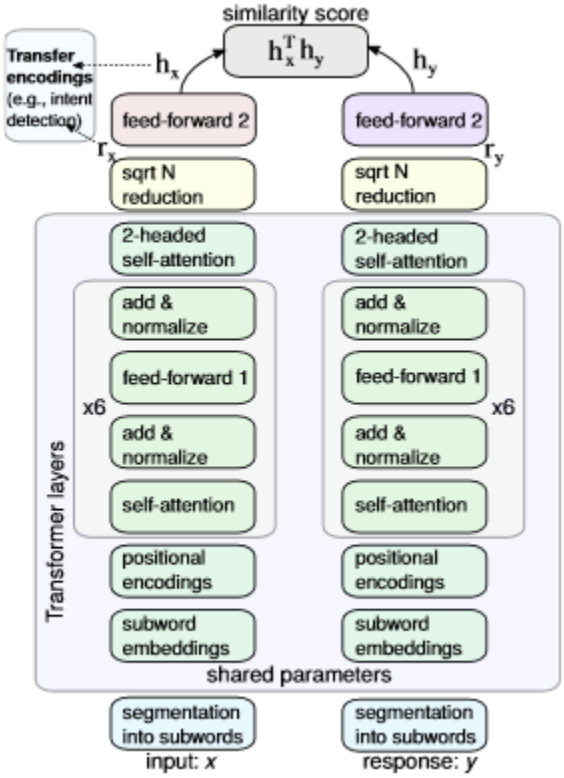

FIGURE 2.6 – Architecture du modèle ConveRT de [Henderson et al., 2020] pour un contexte simple.

Les encodages récupérés après la réduction de dimensions sont utilisés comme encodages de phrase pour la détection d'intention. Le modèle est un simple réseau de neurones à deux couches avec descente de gradient. Les résultats sont comparés avec certains des modèles les plus populaires.

	Banking	Shopping	Company FAQ
USE-LARGE	92.2	94.0	62.4
BERT-LARGE	93.2	94.3	61.2
ConveRT	92.7	94.5	64.3

FIGURE 2.7 – Comparaison des résultats de la détection d'intention avec l'encodeur ConveRT par [Henderson et al., 2020]

Mise à part les résultats qui concurrencent l'état de l'art, les auteurs observent un gain de rapidité 40 fois supérieur au modèle BERT-LARGE. Ce qui représente un avantage considérable. Cette méthode a en grande partie été reprise pour le NLU de [Bunk et al., 2020] présenté précédemment.

Finalement, il est clair que les recherches d'aujourd'hui sont très concentrées sur la formation de modèles d'apprentissage automatique. Nous sommes clairement entrés dans une utilisation massive des réseaux de neurones, des architectures transformers et de leurs variantes, apportant une amélioration significative des performances. Cependant, le peu de gains apportés aujourd'hui ne devrait plus être considéré comme suffisant vis à vis des nombreux

inconvénients qu'amènent les modèles : coûts économiques, environnementaux et manque de contrôle. Le développement d'agents conversationnels est d'autant plus concerné que leurs modèles et composants sont presque systématiquement voués à évoluer. Certaines recherches récentes semblent vouloir répondre à ces défis en proposant des solutions plus adaptées. Les réseaux de neurones restent toujours très appréciés pour les performances qu'ils apportent mais les grands modèles de langues pré-entraînés auraient tendance à être évités au profit d'encodeurs plus spécifiques aux tâches et moins lourds à entraîner à l'image de ConveRT et DIET classifier.

Chapitre 3

Cas d'usage : Un chatbot pour la relation client destiné aux conseillers

3.1 Contexte et situation initiale

3.1.1 Le contexte chez Orange Bank

Orange Bank est une banque en ligne française 100% mobile. Créée initialement par Groupama en 2003 alors sous le nom de Groupama Banque. Elle est renommée Orange Bank en 2017 suite au rachat de 65% des parts par le groupe Orange l'année précédente. Ses services sont exclusivement proposés via les applications web et mobile. Elle ne possède donc pas d'agence physique contrairement aux banques traditionnelles.

Pour répondre aux besoins de ses clients, l'entreprise a déployé dès ses débuts un agent virtuel disponible 24h/24 et 7j/7 basé sur la technologie IBM Watson. Cet assistant nommé Djingo permet de répondre à des demandes d'information simples ou effectuer des actions relativement complexes comme des manipulations de gestion de compte : virement, activation de service, etc.

Ses 3 principaux cas d'usage sont :

— répondre aux questions sur les offres Orange Bank,

— effectuer des actions pour la gestion du compte,

— résoudre les potentiels problèmes des clients.

Djingo en quelques chiffres :

— 3000 conversations quotidiennes

— 390 intentions

— 1000 réponses différentes

FIGURE 3.1 – Fenêtre d'accueil de Djingo sur le site d'Orange Bank

Si Djingo constate qu'il n'est pas en mesure d'aider son interlocuteur, il propose une redirection vers le CRC où un conseiller humain pourra prendre le relais. Les analyses ont montré que 43% des conversations étaient redirigées vers le CRC. Avec une moyenne de 4 000 conversations Djingo par jour, le CRC prend en charge 33% d'entre elles. À cela s'ajoute les sollicitations hors Djingo : les formulaires, les emails, et les questions déposées sur le forum.

3.1.2 Les données disponibles : Les standards

Le centre relation client dispose de procédures précises pour la résolution des problèmes les plus fréquents. Pour chaque situation, un questionnement est posé. La réponse donnée (à sélectionner parmi une liste prédéfinie), renvoie le traitement adapté. Ces procédures appelées « standards » sont également destinées à être présentées lors des phases de formation des nouveaux conseillers. Ils sont à l'origine sous format PowerPoint.

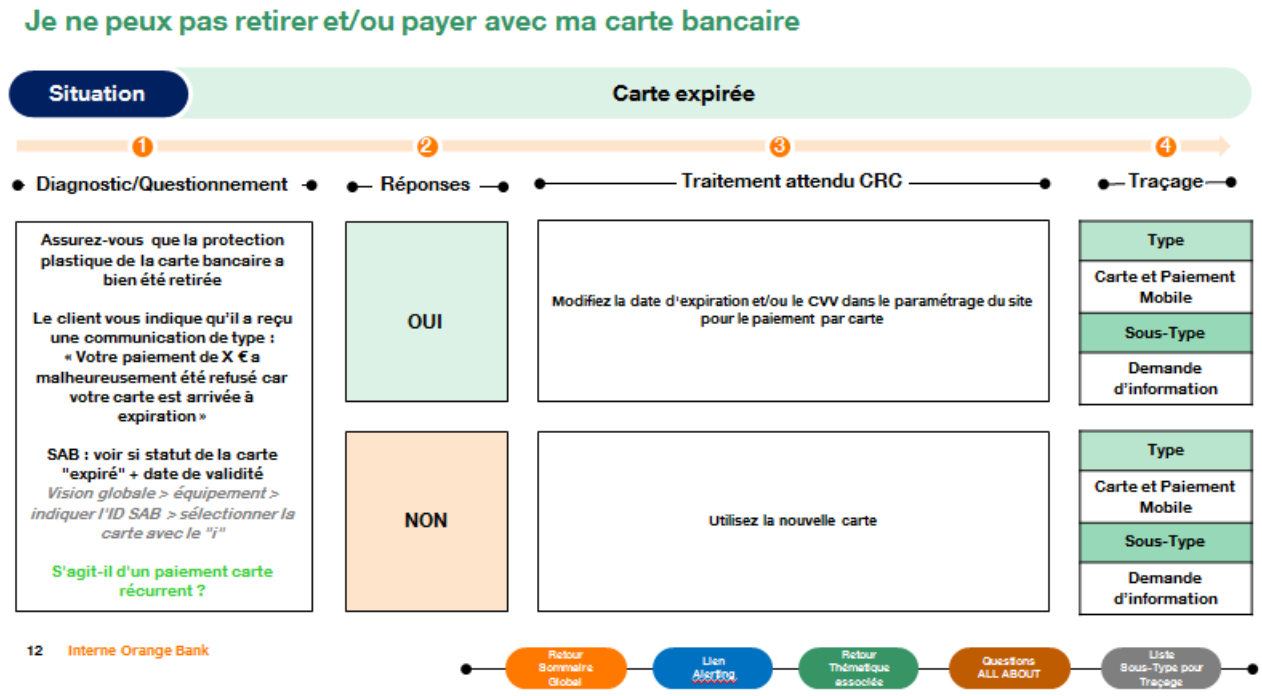

FIGURE 3.2 – Exemple d'une page d'un standard au format PowerPoint. Problème : Je ne peux pas payer avec ma carte bancaire. Situation : La carte a expirée

Nous allons exclusivement nous appuyer sur ces standards pour générer et maintenir à jour l'ensemble des données du *chatbot*.

3.1.3 Un format tabulaire sur mesure pour la modélisation

Le format Powerpoint des standards ne pouvant pas être automatisé, est abandonné au profit d'un format Excel plus facilement manipulable. Une fois converti, nous obtenons des données parfaitement structurées.

Je ne peux pas retirer et/ou payer avec ma carte bancaire et j'ai reçu une communication/alerting

Situation	Carte expirée			
Diagnostic/Questionnement	Réponses	Traitement attendu CRC	Traçage - Type	Traçage - Sous-Type
Assurez-vous que la protection plastique de la carte bancaire a bien été retirée Le client vous indique qu'il a reçu une communication de type : « Votre paiement de X € a malheureusement été refusé car votre carte est arrivée à expiration » SAB : voir si statut de la carte "expiré" + date de validité Vision globale > équipement > indiquer l'ID SAB > sélectionner la carte avec le "i" S'agit-il d'un paiement carte récurrent ?	OUI	Modifiez la date d'expiration et/ou le CVV dans le paramétrage du site pour le paiement par carte	Carte et Paiement Mobile	Demande d'informations
Assurez-vous que la protection plastique de la carte bancaire a bien été retirée Le client vous indique qu'il a reçu une communication de type : « Votre paiement de X € a malheureusement été refusé car votre carte est arrivée à expiration » SAB : voir si statut de la carte "expiré" + date de validité Vision globale > équipement > indiquer l'ID SAB > sélectionner la carte avec le "i" S'agit-il d'un paiement carte récurrent ?	NON	Utilisez la nouvelle carte	Carte et Paiement Mobile	Demande d'informations

FIGURE 3.3 – Exemple d'une page d'un standard au format Excel après transformation. Problème : Je ne peux pas payer avec ma carte bancaire. Situation : La carte à expirée

Ces fichiers représentent l'interface de modification et d'évolution du périmètre du *chatbot*. Ainsi, lorsque les experts du centre relation client développent une nouvelle thématique, ils créent un nouveau standard qui s'intégrera automatiquement aux capacités actuelles du bot.

Les standards existaient avant le démarrage du projet. Cette solution a permis de tirer avantage des outils déjà présents dans l'entreprise. Les collaborateurs, déjà familiers du format, ont pu exclusivement se consacrer à l'évolution du contenu. L'intérêt majeur de cette solution est un processus de modélisation considérablement accéléré. Ce choix a également été motivé par le fait que les standards dessinent indirectement des arbres de décisions, ce qui les rends particulièrement adaptés au développement d'un *chatbot*.

Voici les thématiques actuellement modélisées

— Accès application
— Carte bancaire
— Chèque
— Domiciliation bancaire
— Informations personnelles
— Paiement mobile
— Virement

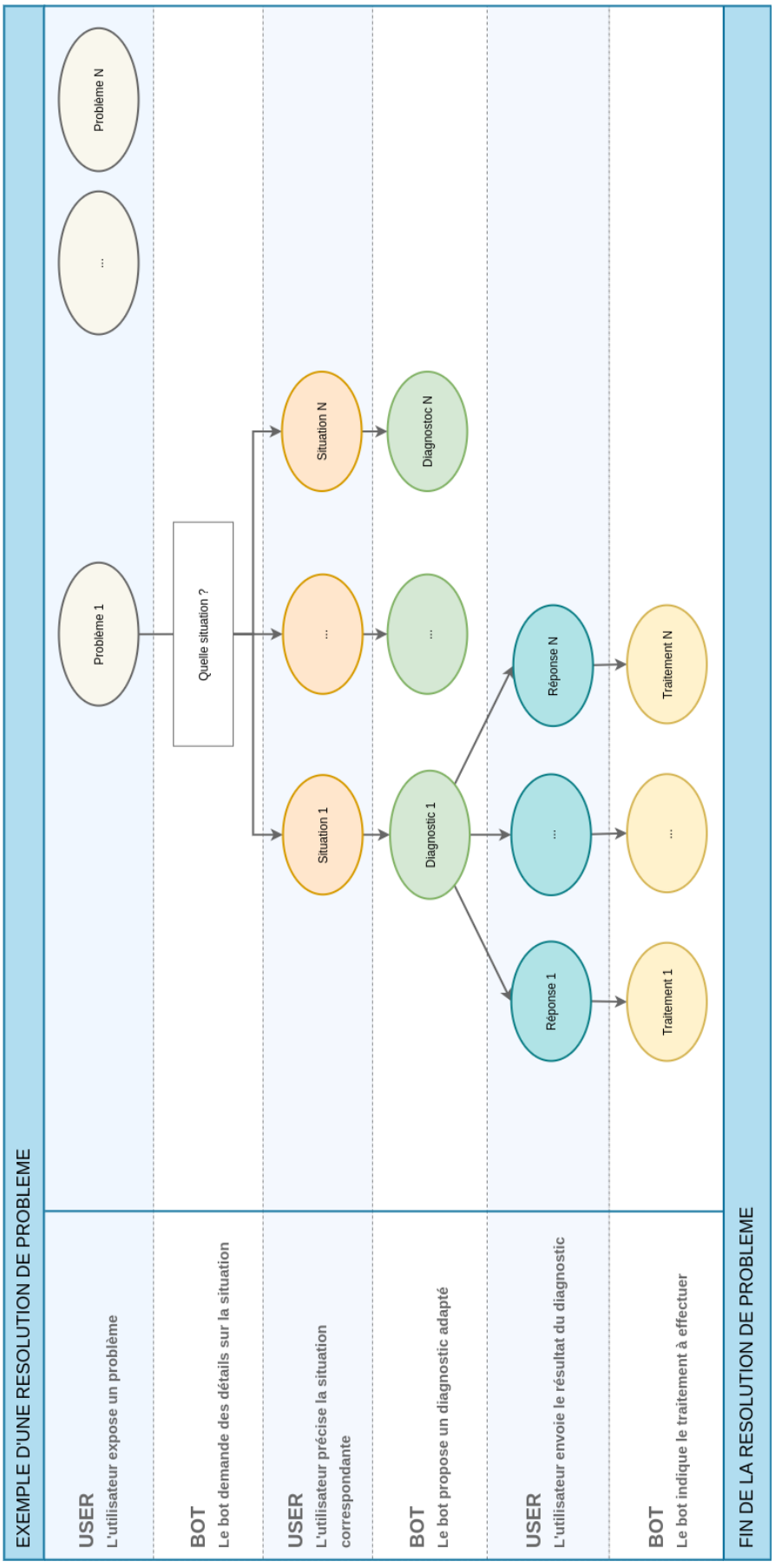

FIGURE 3.4 – Modélisation générale des standards

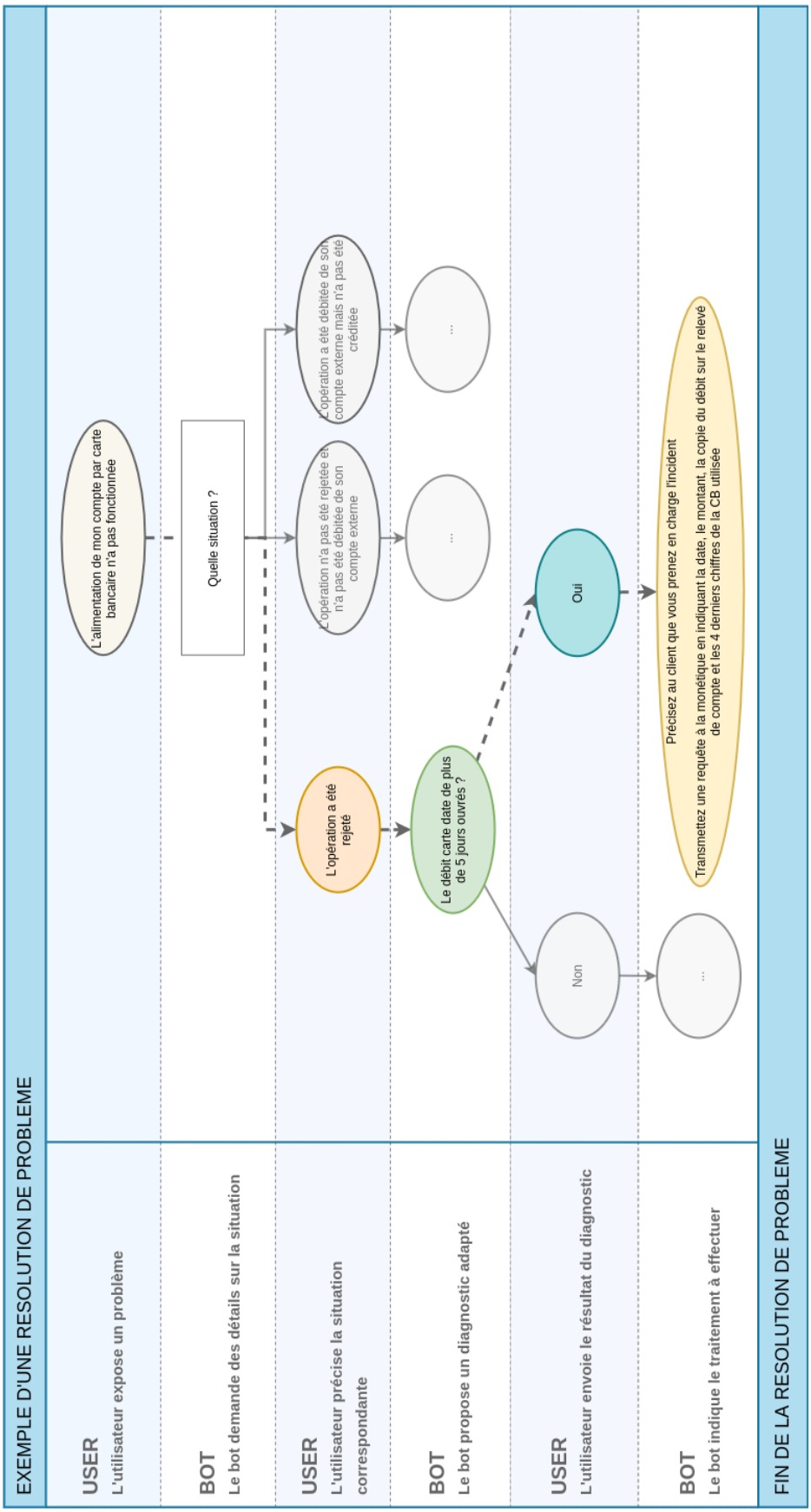

FIGURE 3.5 – Exemple de modélisation d'une demande : L'alimentation de mon compte par carte n'a pas fonctionnée.

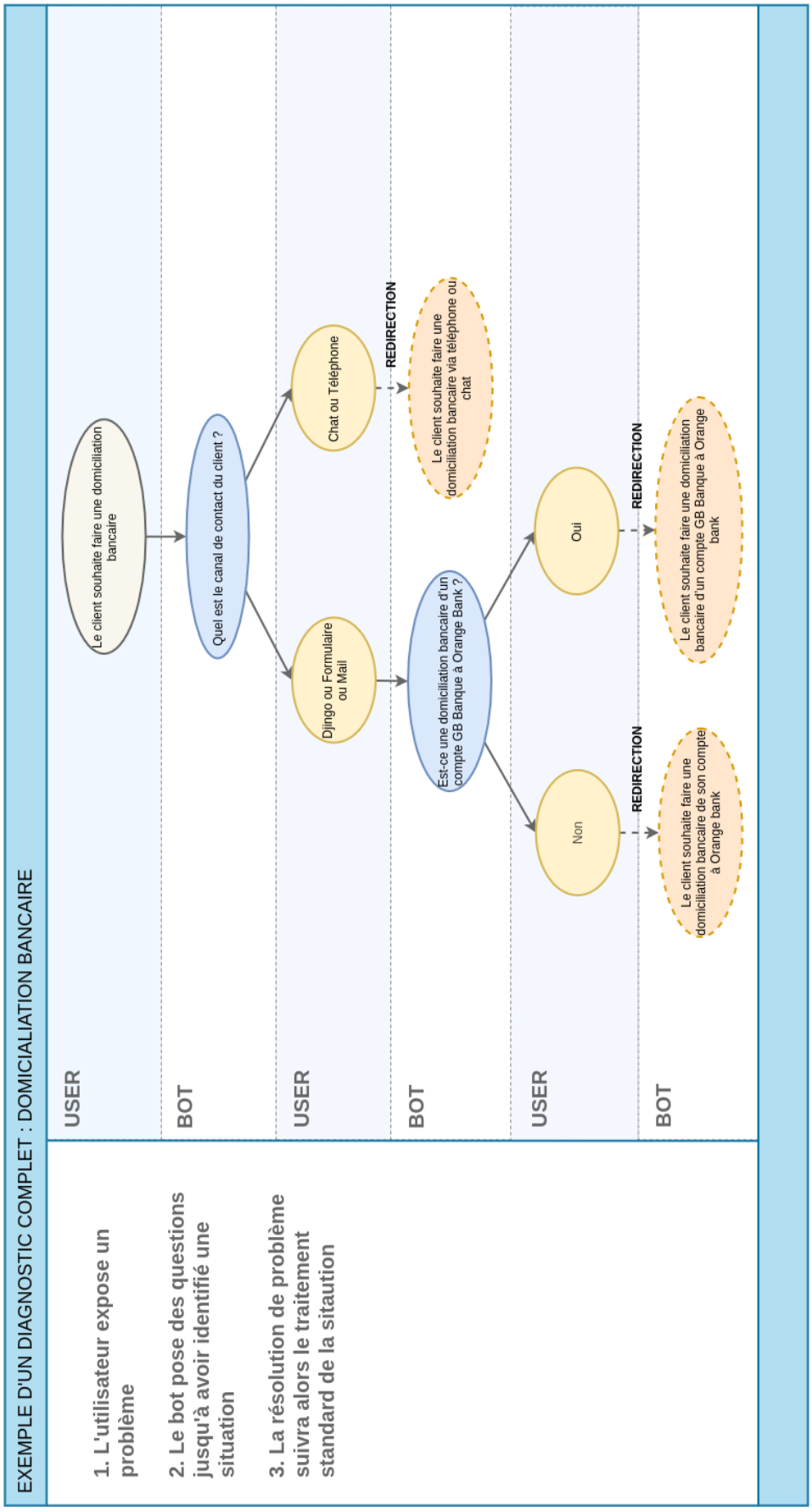

FIGURE 3.6 – Exemple de modélisation en arbre de décisions : Domiciliation Bancaire

3.1.4 Choix du logiciel de développement

Pour structurer le projet et faciliter les développements il est souvent judicieux de choisir un cadre de développement (*framework*). Ceci nous fournira la base du projet et les éléments essentiels sans avoir à recoder l'essentiel.

Il existe aujourd'hui de nombreuses options, des plateformes commerciales telles que Dialogflow de Google (anciennement Api.ai), LUIS de Microsoft, Watson d'IBM, Wit.ai de Facebook, AmazonLex, Recast.ai, Botfuel.io mais également des solutions open source, telles que Snips.ai et Rasa. [Liu et al., 2019a] ont comparé la performance des NLU des plus populaires d'entre eux : Rasa, Watson d'IBM, LUIS de Microsoft et Dialogflow de Google sur un ensemble de données de 21 domaines différents et rassemblant 25 000 énoncés d'utilisateurs.

Voici la liste des domaines : alarme, audio, livre, audio, calendrier, cuisine, heure, email, jeu, général, IoT, listes, musique, actualités, podcasts, question/réponse générales, radio, recommandations, social, plats à emporter, transports et météo.

	Intent			Entities			Combined		
	pre	rec	F1	pre	rec	F1	prec	rec	f1
Rasa	0.863	0.863	0.863	0.859	0.694	0.768	0.862	0.787	0.822
Dialogflow	0.870	0.859	0.864	0.782	0.709	0.743	0.832	0.791	0.811
LUIS	0.855	0.855	0.855	0.837	0.725	0.777	0.848	0.796	0.821
Watson	0.884	0.881	0.882	0.354	0.787	0.488	0.540	0.838	0.657

FIGURE 3.7 – Comparaison des performances des solutions *chatbots* les plus populaires par [Liu et al., 2019a] sur la langue anglaise.

Les auteurs ont choisi les configurations de base pour chaque outil, sans chercher à spécialiser les algorithmes sur les données. Cette comparaison est donc très discutable. Nous pouvons quand même remarquer que Watson semble particulièrement bon sur la détection d'intention, il l'est en revanche nettement moins sur l'extraction d'entités nommées.

Notons que cette approche compare seulement la partie NLU de chaque plate-forme. De plus cette évaluation a été effectuée sur un ensemble de donnée en anglais. Puisque nous travaillons sur la langue française nous avons besoin d'un outil modulable pouvant se spécialiser sur nos données. Idéalement nous souhaitons pouvoir intervenir sur le choix des algorithmes utilisés et leurs configurations.

RASA[Bocklisch et al., 2017] sera notre choix pour les raisons suivantes : Rasa est un outil open source permettant de développer facilement et rapidement des agents conversationnels. Il met à disposition une base d'algorithmes et d'architectures et est basé sur le modèle NLU-DP-NLG. L'avantage est qu'il est tout à fait possible de personnaliser différents éléments fonctionnels du *chatbot*. Nous aurons donc la possibilité de spécialiser l'architecture selon notre besoin et spécialiser les modèles pour la langue française. Le fait qu'il soit une des rare alternative open-source le rend extrêmement personnalisable comparé aux autres offres du marché. Ajoutez à cela le contrôle total de la sécurité de nos données en fait l'outil le plus adapté.

Un *framework* attend un format de données bien particulier. Voici les fichiers demandés par RASA :

— nlu.md : Rassemble toutes les intentions et leurs variations associées pour entraîner le modèle du NLU. Format markdown.

— stories.md : Rassemble les tours de paroles grâce aux identifiants des intentions et des réponses. Format markdown.

— domain.yml : Un fichier les *templates* utilisés par les réponses. Il contient également la liste des identifiants des intentions et des réponses. Format yaml.

Les standards au format Excel préalablement préparés par le CRC sont envoyés dans un module créé sur mesure pour générer les fichiers requis.

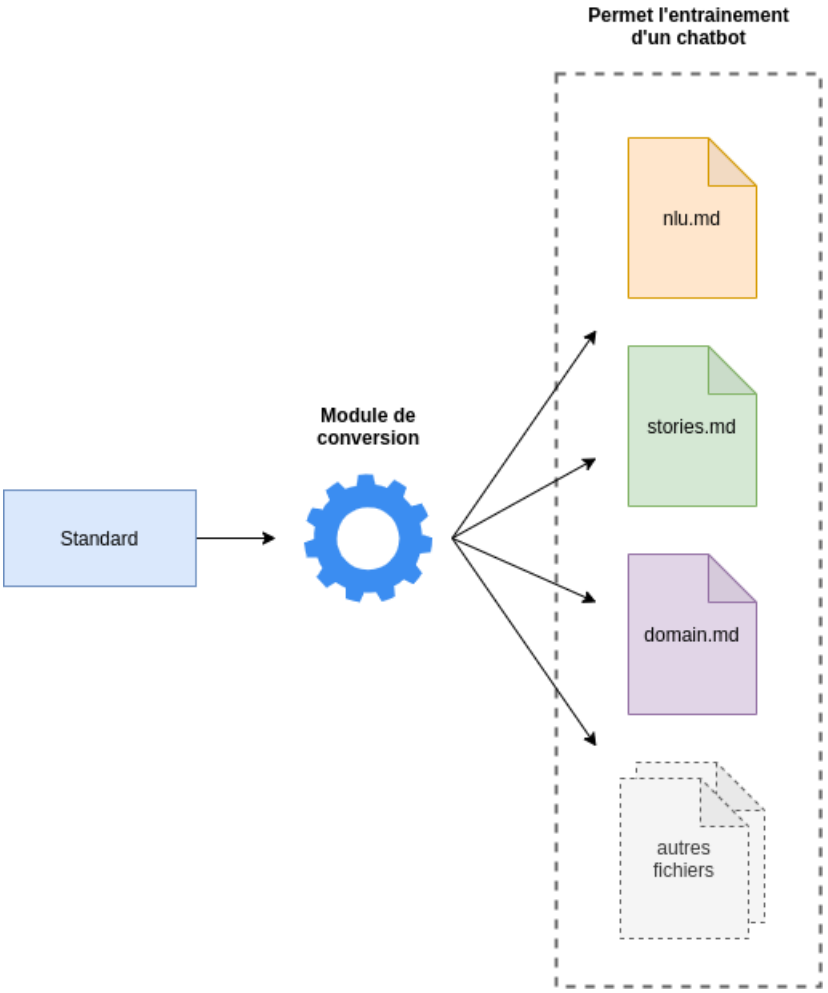

FIGURE 3.8 – Conversion d'un standard en fichiers demandés par RASA

Nous créons en réalité des fichiers indépendants pour chaque standard, pour cette raison certains fichiers de configurations sont également créés. Les fichiers d'entraînement de chaque *chatbot* sont utilisés ensemble pour entraîner des modèles globaux et ainsi constituer un unique *chatbot*. Finalement c'est donc bien un unique entraînement qui est effectué. Cet entraînement va simplement fusionner les données de chaque sous agent. Chaque agent peut également être entraîné de la même manière. Cette méthode favorise l'évolutivité. Elle nous donne la possibilité d'activer ou désactiver des thématiques sur demande (par des fichiers de configurations) en plus de nous faciliter les futures mises à jour.

Finalement un unique entraînement est requis, lors de cet entraînement le système va fusionner toutes les données de tous les agents. Si besoin, un ou plusieurs agents peuvent être désactivés dans la configuration pour que le *chatbot* global ignore les données associées. Il est également possible d'entraîner chaque sous système de façon indépendant, ce qui est notamment utile pour le débogage.

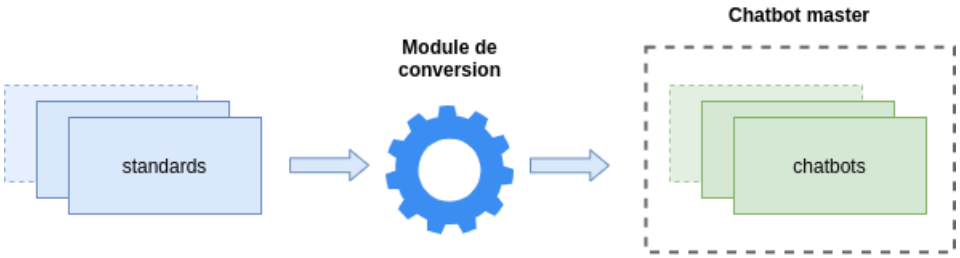

FIGURE 3.9 – Conversion des standards en plusieurs *chatbots* constituant ensemble un *chatbot* global

3.2 Mise en place de l'architecture

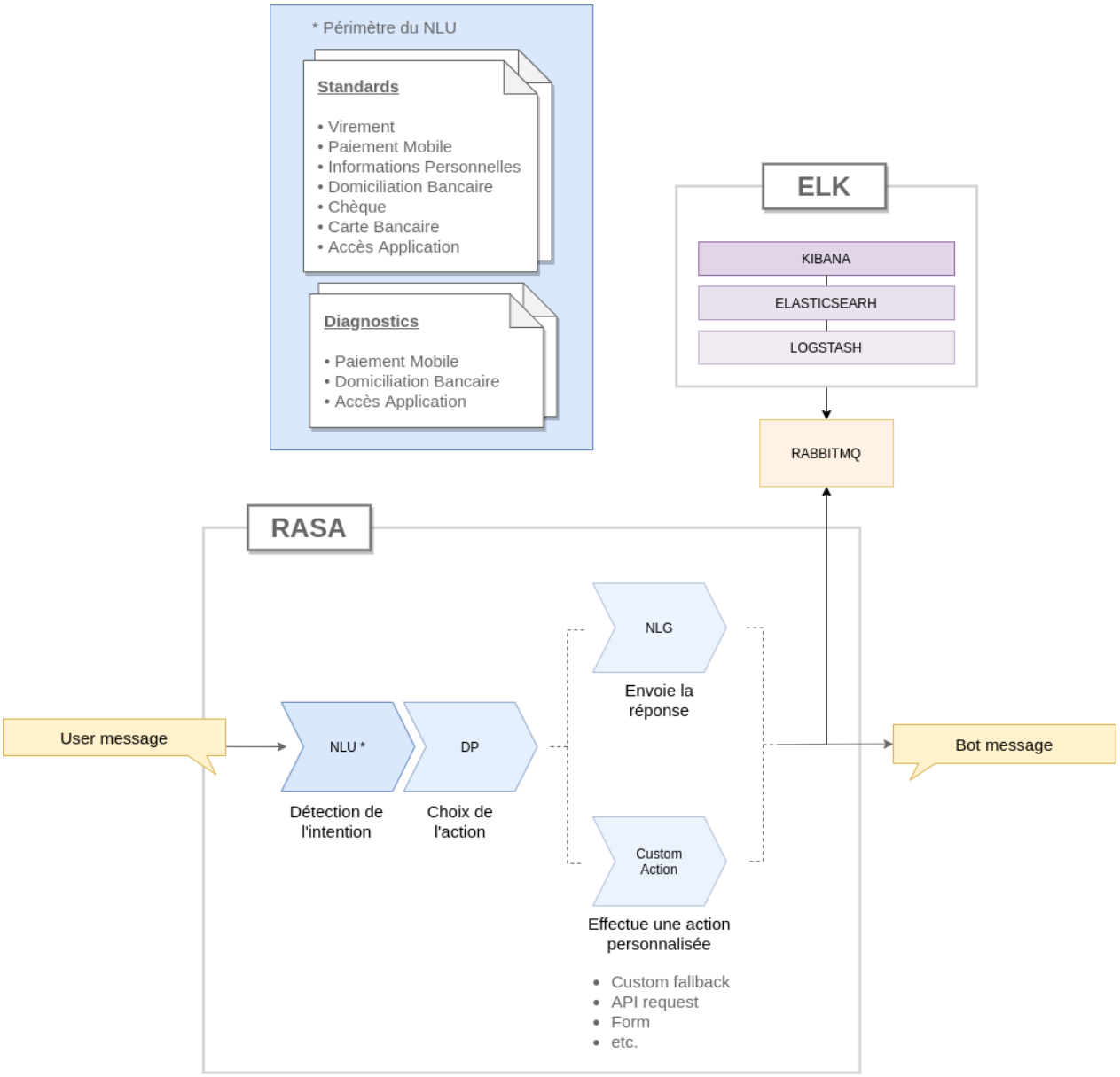

FIGURE 3.10 – Architecture du bot

3.2.1 Les principaux composants du chatbot

3.2.2 Détecteur de l'intention (NLU)

Le *framework* propose différents algorithmes pour la classification d'intention et la détection d'entités nommées. Suite à l'étude effectuée plus haut dans ce devoir, nous choisissons d'utiliser DIET classifier qui affiche de très bonnes performances et est relativement rapide à entraîner. Sa compatibilité avec les modèles de langues nous permettra de spécialiser les encodages sur le français. A noter que l'état actuel du projet n'a pas nécessité de détection d'entités, nous désactivons donc le module en charge de cette tâche. En pratique nous serons bien en présence d'entités telles que des montants, des noms, des références clients, etc.

mais les standards qui constituent le périmètre du système n'a pas fourni l'annotation de ces entités. Elles seront peut-être considérées dans une future mise à jour.

3.2.2.1 Analyse comparative de différentes configurations

Les données actuelles du projet sont créées artificiellement, par leur qualité et leur faible quantité, elles manquent de fiabilité et ne permettent pas de déterminer la meilleure configuration pour l'entraînement d'un modèle. Pour une meilleure visualisation on réalise un test sur un autre ensemble de données plus conséquent. Nous utilisons les intentions du *chatbot* Djingo destiné aux clients. Il s'agit d'un ensemble de données volumineux et créée à partir de verbatims réels. Chaque intention rassemble des formulations très différentes, ce qui nous permettra de bien observer le comportement du modèle devant des verbatims très éloignés de l'ensemble d'entraînement, ce qui est très souvent le cas en situation réelle.

- En cas de fraude vous faites quoi v
- Que faites-vous en cas de tentative et/ou fraude
- vous m'alertez si je me fais piraté ?
- vous envisagez quoi par rapport a la sécurité orange bank ?
- vous prévenez le client s'il se fait piraté son compte ?
- comment etre certaine ke ta bank est sure ?
- comment être certain que orange est fiable
- Quels sont les moyens mis en place par Orange Bank en cas de fraude
- Peut on pirater un compte sur livret ?
- Quelle est la protection proposée par Orange bank
- vous prévenez le client si vous suspectez une fraude sur le compte bancaire ?
- Quelles sont les sécurités mises en place par Orange Bank
- est ce que orange bank détecte les fraudes ?
- comment est-ce protégé
- Quel est la politique de sécurité
- vous détectez les opérations frauduleuses ? est ce que je suis prévenu en cas d'opération frauduleuse sur mon compte ?
- Au faite les conte son t'il bien protégé ?

FIGURE 3.11 – Exemple de variations. intention : Quelles sont les sécurités mises en place par Orange Bank

Le *dataset* original étant trop volumineux pour réaliser nos expériences dans un délai convenable on extrait un échantillon de 5353 variations pour 287 intentions. Chaque intention possède en moyenne 18.65 variations dont la plus faible possède 9 variations. Nous prenons 20% de chaque variation arrondi au supérieur pour notre ensemble de test. Nous obtenons les proportions suivantes :

	nombre de variations
train	4222
test	1131

TABLE 3.1 – Répartition du jeu de données

Nous nous intéressons plus particulièrement à DIET classifier qui nous semble intéressant par sa rapidité d'entraînement et sa capacité à pouvoir utiliser des encodages de type Bert. Nous souhaitons savoir à quel point utiliser des encodages issus de modèle Bert améliorent les résultats par rapport à la durée d'entraînement nécessaire, durée d'entraînement qui, rappelons-le, est loin d'être négligeable lorsque le modèle est censé être ré-entraîné régulièrement.

On décrit ci-dessous les différentes configurations.

spacy + svm (1) Première configuration *baseline*. Spacy est utilisé avec le modèle « fr core news lg ». Il fournit les *tokens* et les vecteurs de mots. Ils sont envoyés à un SVM linéaire avec gamma à 0.1 et C déterminé par une technique grid-search.

camembert + svm (2) Deuxième configuration *baseline*. Spacy est cette fois remplacé par Camembert pour la segmentation des mots et leurs encodages. L'idée étant de voir à quel point les encodages contextuels de type Bert sont efficaces sans toucher à l'algorithme d'apprentissage. Un SVM identique à (2) est utilisé pour l'apprentissage.

spacy$_{tokens-only}$ **+ countvect + diet**$_{mask}$ **(3)** Spacy est utilisé seulement pour la segmentation des mots. L'encodage est effectué par un CountVectorizer (que nous notons « countvect ») qui extrait les n-grammes de mots ($n \leq 2$) et les n-grammes de caractères ($n \leq 5$). L'apprentissage est réalisé par DIET classifier avec les configurations décrites ci-dessus. L'entrée des encodages Bert est donc désactivée. Nous voulons ici observer le potentiel de l'algorithme sans les encodages Bert.

Nous déterminons quelques paramètres que nous appliquerons par défaut à chaque entraînement. Les valeurs sont obtenues en réalisant quelques entraînements de test que nous ne détaillerons pas. Nous activons la tâche de prédiction de MASK avec l'hypothèse qu'elle permettra une meilleure généralisation. Les exemples de nos intentions étant très différents, nous espérons un impact notable sur les performances de notre modèle.

name	value
batch size	[64, 256]
epochs	200
learning rate	0.0008
embedding dimension	128
drop rate	0.2
drop rate attention	0.0
use masked language model	True

TABLE 3.2 – Configurations de DIET classifier

Les d'autres paramètres sont laissés par défauts. Tous les paramètres sont décrient dans la documentation.

camembert + countvect + diet$_{mask}$ (4) On ajoute maintenant Camembert à la configuration (4). Spacy n'est plus utilisé car la segmentation est directement gérée par le modèle Camembert.

camembert$_{split-intent}$ + countvect + diet$_{mask}$ (5) Similaire à (4) avec pour seule modification l'activation de la segmentation des intentions. Nos intentions sont de la forme : QP6425..Quelles..sont..les..sécurités..mises..en..place..par..Orange..Bank, En segmentant sur le symbole «..» nous espérons améliorer la similarité entre les encodages des intentions et des variations.

camembert$_{split-intent}$ + countvect + diet (6) L'unique modification par rapport à (5) est la désactivation du *MASK*. Nous voulons connaître son degré d'importance dans les performances du modèle.

Le matériel utilisé est un ordinateur classique avec un Processeur Intel core i7-9750 GHz à 6 cœurs et 32Go de RAM. Chaque configuration est entraînée sur l'ensemble d'entraînement (4422 variations) et est évaluée sur l'ensemble de test (1131 variations). Voici les résultats obtenus :

	pre	rec	f1	acc	training
spacy + svm (baseline) (1)	0.325	0.298	0.286	0.301	00h 04m 12s
camembert + svm (baseline) (2)	0.653	0.569	0.577	0.569	00h 09m 48s
spacy$_{tokens-only}$ + countvect + diet$_{mask}$ (3)	0.590	0.569	0.548	0.566	1h 51m 48s
camembert + countvect + diet$_{mask}$ (4)	0.620	0.603	0.582	0.600	2h 37m 28s
camembert$_{split-intent}$ + countvect + diet$_{mask}$ (5)	0.618	0.604	0.582	0.602	2h 39m 12s
camembert$_{split-intent}$ + countvect + diet (6)	0.602	0.558	0.545	0.554	2h 36m 51s

TABLE 3.3 – Comparaison des performances pour différentes configurations du NLU. Nous avons dans l'ordre : macro précision, macro rappel, macro fscore, exactitude et la durée d'entraînement.

A noter que toutes les configurations, à part (1), ont eu des résultats supérieurs ou égaux à 0.99 (pour toutes les métriques) sur les données d'entraînement.

Comme attendu notre première *baseline* (1), qui n'utilise ni encodage de type Bert ni algorithme à base de transformers, affiche les performances les plus basses. En utilisant un modèle de type Bert avec le même SVM les résultats concurrences les modèles suivant à base de DIET classifier. Notons également qu'il affiche la meilleure précision. Le temps d'entraînement de (2) est doublé par rapport à (1) mais reste inférieur à 10 minutes alors que toutes les configurations à base de DIET classifier dépassent 1 heure d'entraînement. Sans les encodages par Bert, (3) est moins performant que la *baseline* (2) et pourtant considérablement plus long à entraîner. (6), également inférieur à la *baseline* (2), affiche les plus mauvaises performances parmi les configurations à base de DIET classifier, et prouve l'importance de la tâche de prédiction de MASK avec une différente de plus de 0.03 f1 avec (5).

(5) et (6) réalisent les meilleures performances. Leurs résultats sensiblement identiques montrent que la segmentation des intentions n'est pas utile pour notre ensemble de données.

Pour la suite, nous choisissons de conserver la configuration (4).

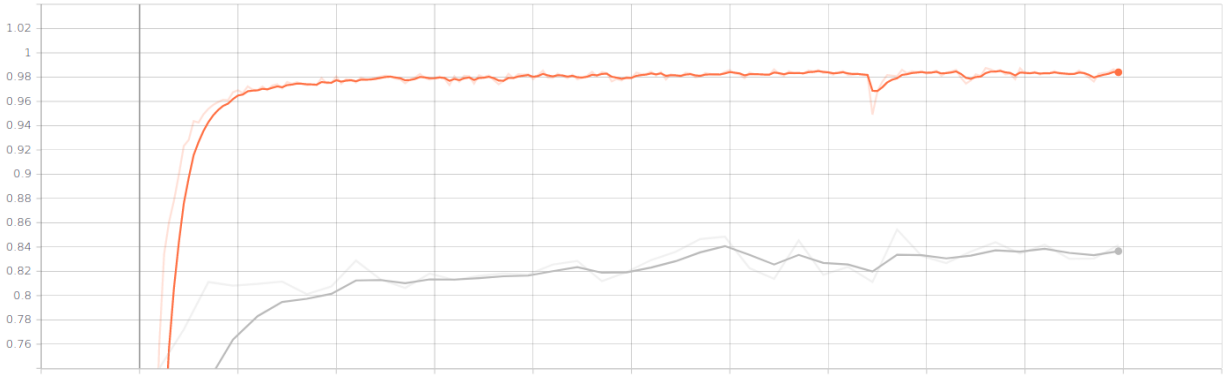

FIGURE 3.12 – Évolution de l'exactitude de la configuration (4) par époque sur l'échantillon des données djingo. En rouge l'ensemble d'apprentissage et en gris l'ensemble de validation. L'ensemble de validation rassemble 3 variations par intention pour un total de 861 variations. Un lissage est effectué avec α égal à 0.6

3.2.2.2 Comment détecter les intentions à désambiguïser ?

Nous allons maintenant étudier plus spécifiquement les résultats. La méthodologie présentée ici est identique à celle qui sera appliquée sur les données du *chatbot* en cours de réalisation. Les résultats sont issus de la configuration (4).

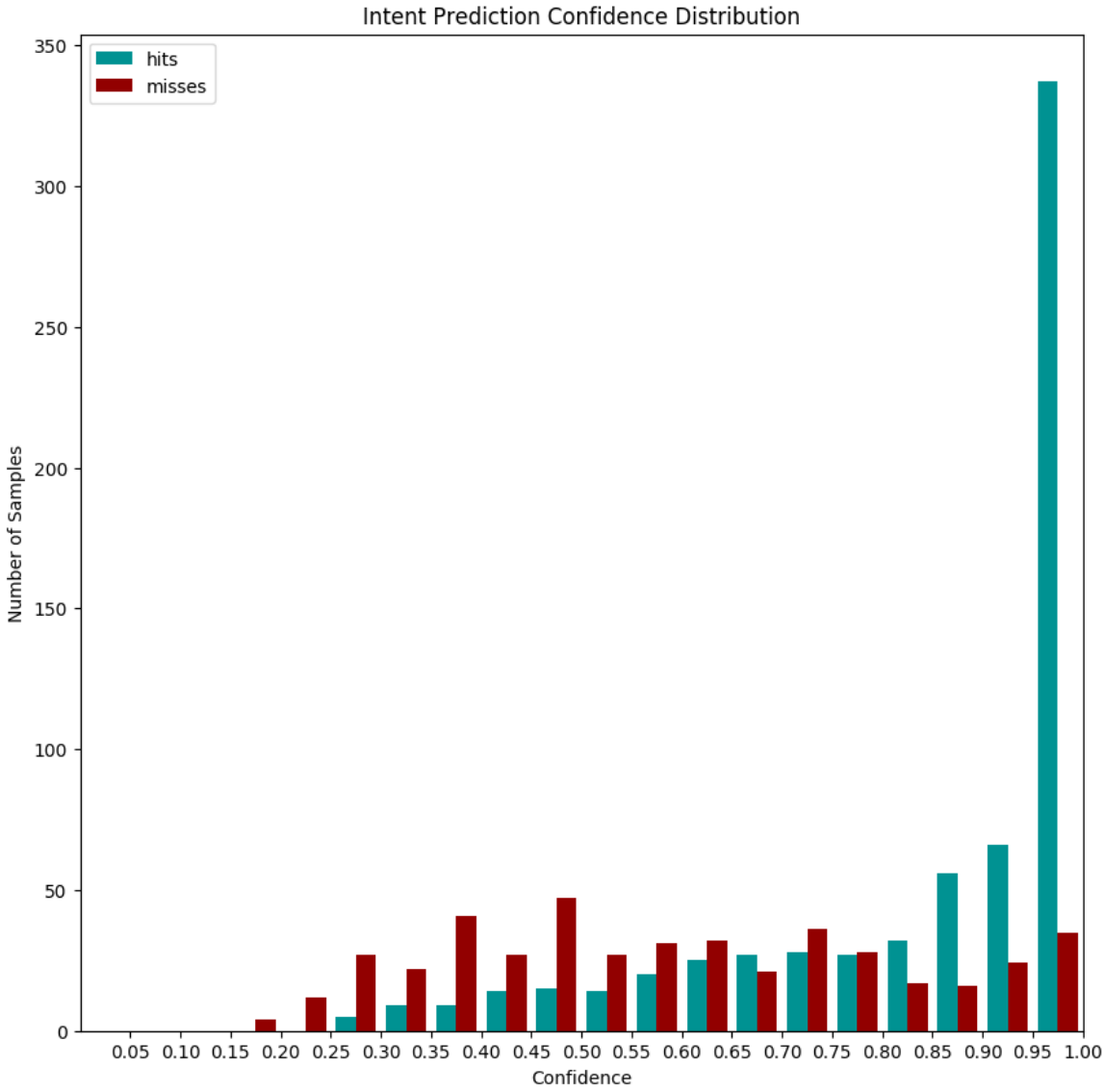

FIGURE 3.13 – Distribution des scores de confiance sur les données de test

La distribution des scores de confiance nous permet de savoir à partir de quelle valeur nous pouvons estimer la prédiction correcte. Ce diagramme est donc particulièrement utile pour déterminer des seuils optimaux. On observe sans surprise que l'intervalle de confiance le plus élevé se situe entre 0.95 et 1. Malgré tout 35 des prédictions avec un score supérieur à 0.95 sont incorrectes. Si nous fixons un seuil à 0.95 cela signifie que ce sont les intentions liées à ces 35 erreurs qu'il faut corriger en priorité. On peut visualiser les identifiants des intentions concernées grâce à la matrice de confusion suivante :

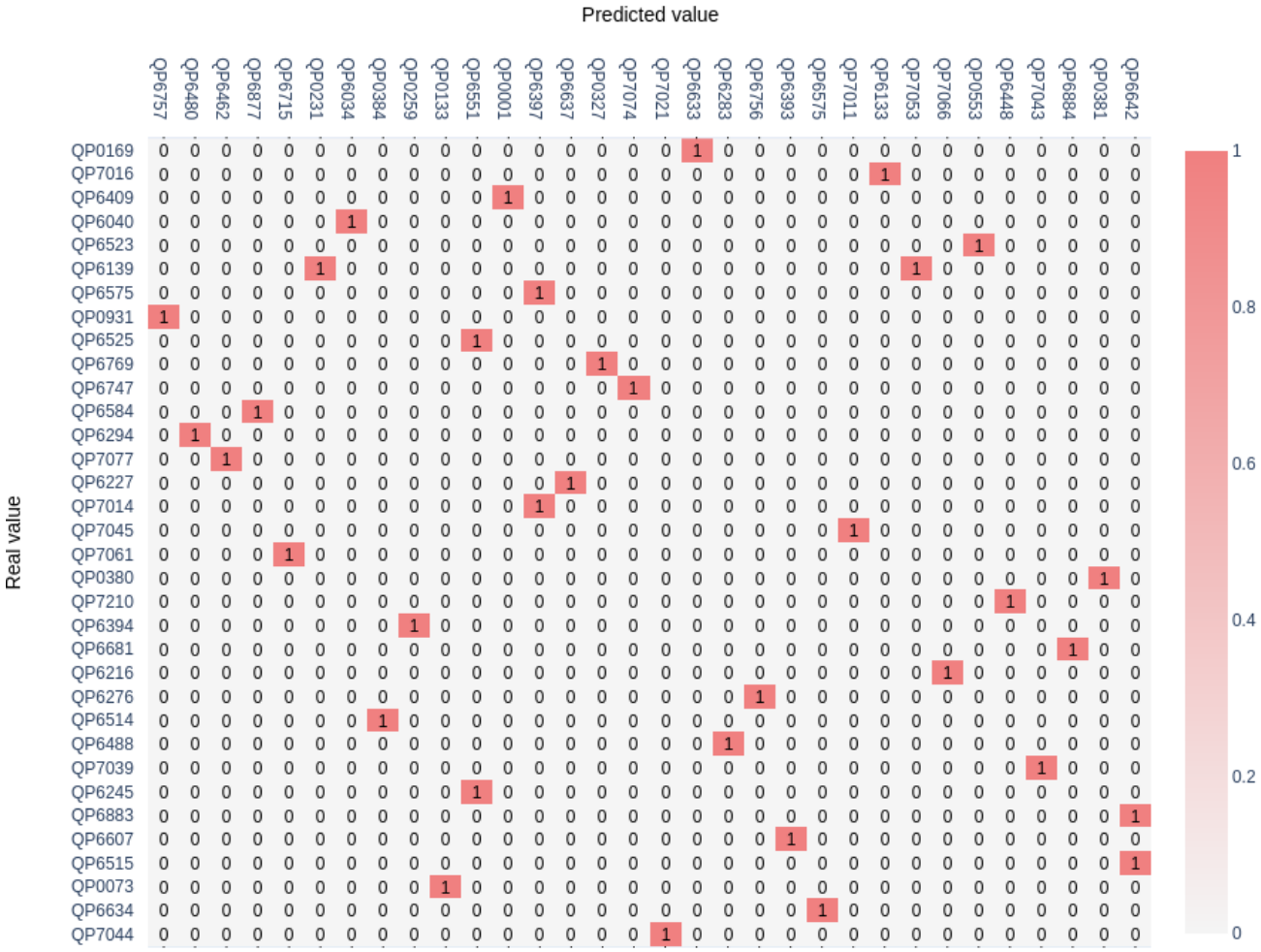

FIGURE 3.14 – Matrice de confusion sur le jeu de test pour les 35 intentions ayant un score de confiance supérieur à 0.95 et inférieur à 1.

Voyons maintenant les intentions les moins bien classées. On relève les intentions ayant un fscore moyen égal à 0. On obtient la matrice de confusion suivante :

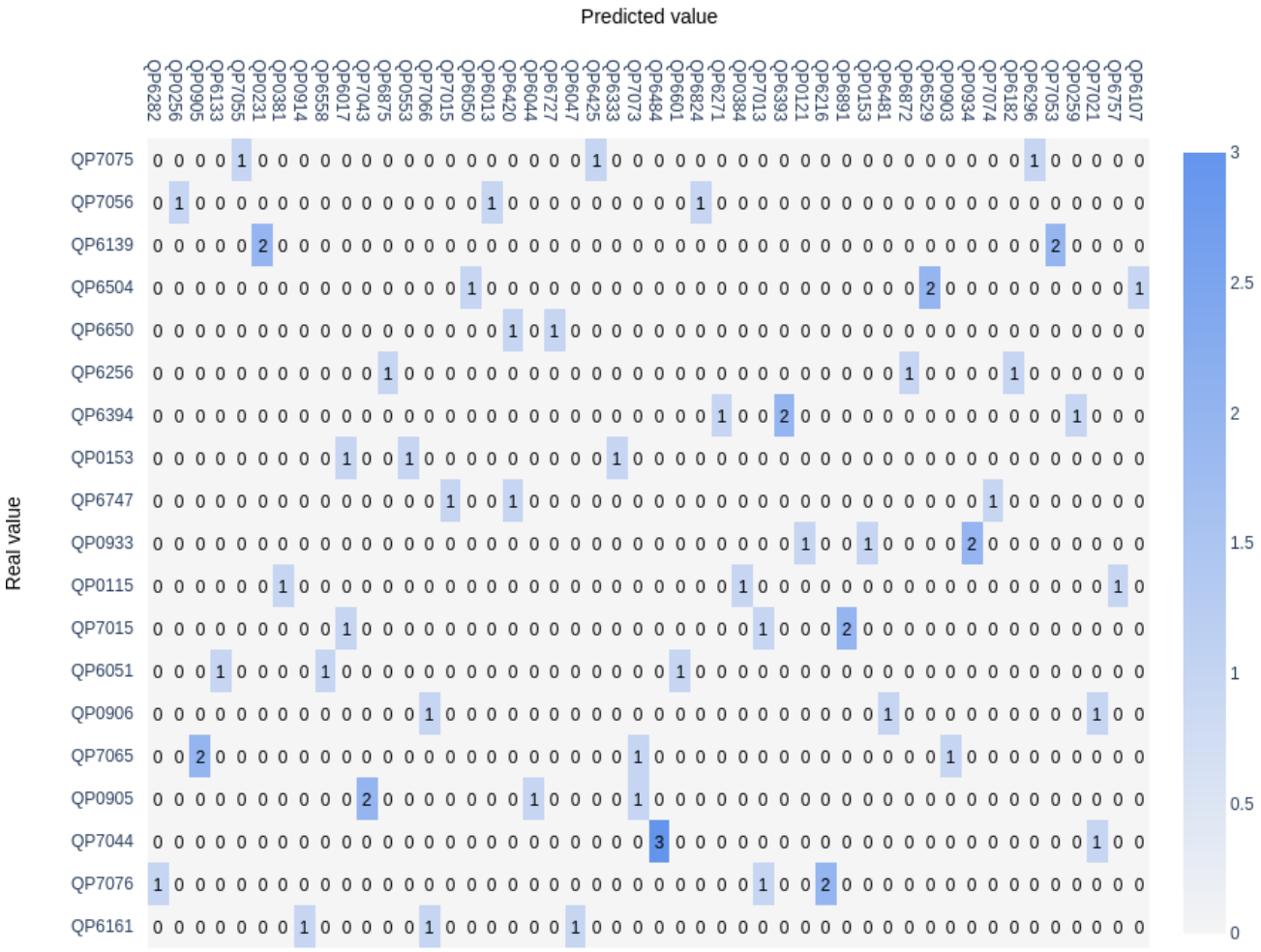

FIGURE 3.15 – Matrice de confusion sur le jeu de test pour les 19 intentions ayant un fscore moyen égal à 0.

Les erreurs sont assez homogènes. Nous retenons cependant les intentions QP7044 et QP6484 à désambiguïser en priorité avec 3 confusions. Elles correspondent respectivement à « Je ne suis pas satisfait » et « Pourquoi as-tu été créé ».

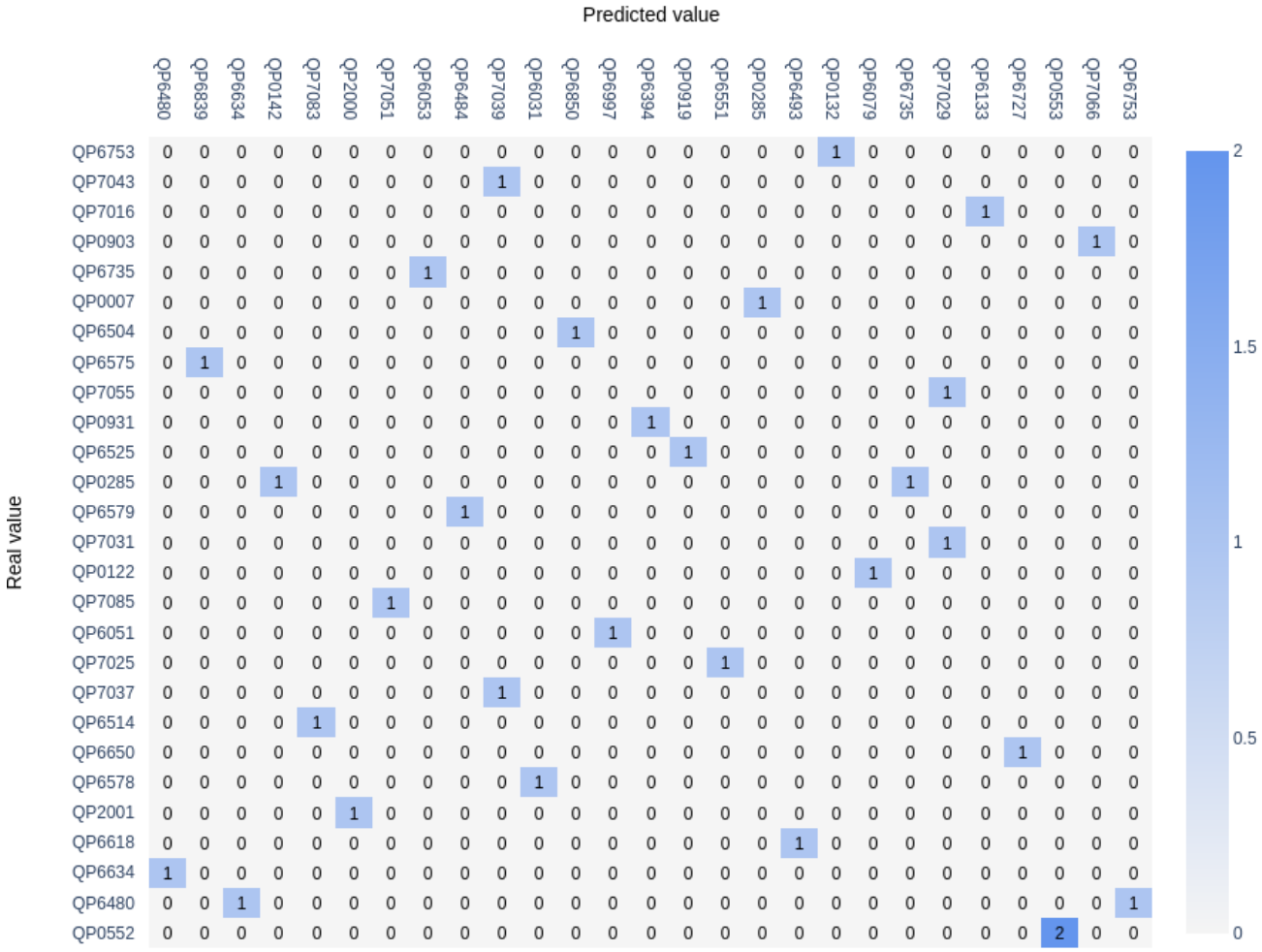

FIGURE 3.16 – Matrice de confusion des 30 erreurs de prédiction sur le jeu d'entraînement

En observant les erreurs de l'ensemble d'entraînement on s'aperçoit que certaines intentions sont déjà confondues, avec une faiblesse accentuée entre les intentions QP0552 et QP0553 qui correspondent respectivement à « Le destinataire du SMS n'arrive pas à renseigner ses coordonnées » et « Je suis destinataire d'une transaction par SMS et je ne sais pas comment faire ». La distribution des mots pour les deux intentions semble indiquer une fréquence trop similaire des mots « sms » et « virement ».

FIGURE 3.17 – Distribution des mots pour les intentions QP0552 et QP0553. Quelques mots vides de base ont été exclus.

Cette méthode peut être appliquée dès que nécessaire pour identifier les intentions qui doivent être davantage désambiguïsées dans les données d'entraînement.

3.2.2.3 Entraînement sur les données du projet

Pour entraîner une première version du NLU, les variations ont été créées manuellement. L'ensemble des données contient 225 intentions pour 851 variations. On extrait un ensemble de validation de 225 variations afin d'obtenir au moins une variation par intention. Une

évaluation est effectuée toutes les 5 époques. L'entraînement est réalisé avec la configuration (4).

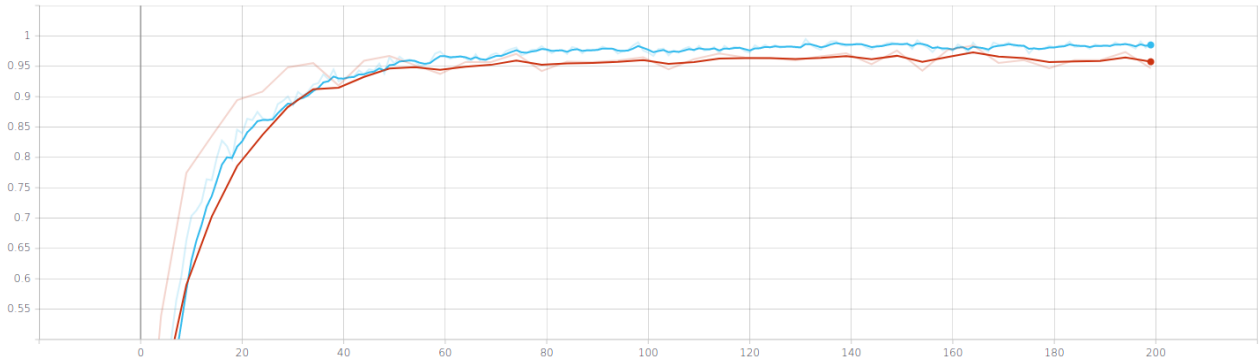

FIGURE 3.18 – Évolution de l'exactitude du NLU par époque. En bleu l'ensemble d'apprentissage et en rouge l'ensemble de validation. Un lissage est effectué avec α égal à 0.6

3.2.3 Des templates pour le NLG

Le NLG est un simple système de *template* reprenant les réponses initialement présentes dans les standards. Il rassemble donc les diagnostics, les traitements, les traçages, et les sources d'information.

Voici un exemple pour illustrer chaque type de *template* :

— diagnostic : Le client a t'-il supprimé la carte Orange Bank dans son Wallet (ou l'appli GPAY) ?

— traitement : Informez le client que l'option « Bloquer mon paiement mobile » depuis son application permet un blocage temporaire de son paiement et qu'il n'est pas nécessaire de supprimer la carte du Wallet (ou l'appli GPAY)

— traçage : type : Carte et Mobile, sous-type : Activer/désactiver le paiement mobile

— source : Standard Paiement mobile

3.2.4 Le gestionnaire du dialogue (DP)

Le gestionnaire de dialogue va déterminer pour une intention donnée quelle est l'action à exécuter. Généralement, il s'agit de renvoyer une réponse textuelle mais nous pouvons également créer des actions plus complexes. Dans notre cas cela nous sera particulièrement utile pour personnaliser les incompréhensions du *chatbot*.

Le gestionnaire TED (*Transformer Embedding Dialogue*) [Vlasov et al., 2020] est basé sur une architecture transformers. Il reprend et simplifie l'architecture REDP (*Recurrent Embedding Dialogue Policy*) proposé par [Vlasov et al., 2018]. Le mécanisme d'auto-attention permet de mettre en évidence les tours de parole les plus utiles à la prédiction de l'action. Les auteurs partent du constat qu'une conversation ne peut pas se résumer à une unique séquence, mais est souvent constituée de plusieurs segments de discours qui se chevauchent. Il s'agit parfois de seulement considérer les tours de paroles précédents sans tenir compte du dernier message, au contraire il peut s'agir dans d'autres cas d'ignorer l'historique et de se concentrer exclusivement sur la dernière intervention. Les RNN et LSTM ont été étudiés

pour résoudre ce problème mais les performances requièrent un nombre de données beaucoup trop important et souvent non disponibles dans le développement de *chatbot*. Le transformer étant naturellement adapté pour ignorer ou exposer des segments il représente une solution intéressante notamment avec son mécanisme d'auto-attention.

Pour ce faire, une fonction de similarité est maximisée entre les plongements de chaque état du système et les actions correspondantes. Les actions sont les étiquettes associées à chaque état des données d'entraînement. Lors de l'inférence, l'action ayant la plus grande similarité avec l'état courant est renvoyée. Un état est représenté par l'intention et les entités fourni par le NLU.

name	value
max history	5
epochs	15
batch size	[8, 32]
hidden layers sizes	dialogue : [256, 128], label : []
number of transformer layers	1
transformer size	128
embedding dimension	128
number of negative examples	20
maximum positive similarity	0.8
maximum negative similarity	-0.2

TABLE 3.4 – Configurations de TED policy

Les données sont directement extraites des standards. Toutes les séquences sont donc générées automatiquement. Voici un exemple de séquence :

```
## story_17
* Le..client..n'arrive..pas..à..se..connecter..via..empreinte..
digitale..ou..reconnaissance..faciale
  - utter_AADIA11
* Le..client..a..déjà..ouvert..son..application..OB..via..
l'une..des..deux..méthodes
  - utter_AAPRO17
  - utter_AATRA6
  - utter_SRCAA
```

Les codes commençant par utter_ sont les identifiants des actions à effectuer. Dans notre cas il s'agit toujours de réponses textes à renvoyer. Nous obtenons au total 293 séquences avec 404 actions distinctes. Les séquences sont relativement courtes et avec des schémas très similaires. L'historique est donc limité à 5, ce qui semble suffisamment large pour discriminer les séquences. Nous obtenons 1351 exemples pour 293 classes. Après prétraitement par le TED policy, 3477 exemples sont obtenus pour 405 classes. Nous choisissons alors un ensemble de validation de 405 exemples.

On intègre également un autre gestionnaire appelé Memorization Policy fournit par RASA. Il mémorise simplement les séquences des données d'entraînement. Si les données d'entraînement contiennent une séquence identique à l'état donnée alors il prédit l'action

associée avec un score de 1, sinon il renvoie un score de 0. Une action prédite avec un score de 1 est prioritaire devant le TED Policy. Les conversations de nos données d'entraînement sont spécialement conçues pour anticiper un maximum de situations, il y a donc de fortes chances pour que nos données contiennent le contexte d'entrée. On souhaite que la prédiction des actions reposent au maximum sur la qualité de nos séquences et donc de nos standards et c'est exactement ce que nous permet de faire ce composant. En d'autres mots, nous cherchons à profiter au maximum du Memorization Policy et réduire les prédictions du TED policy autant que possible.

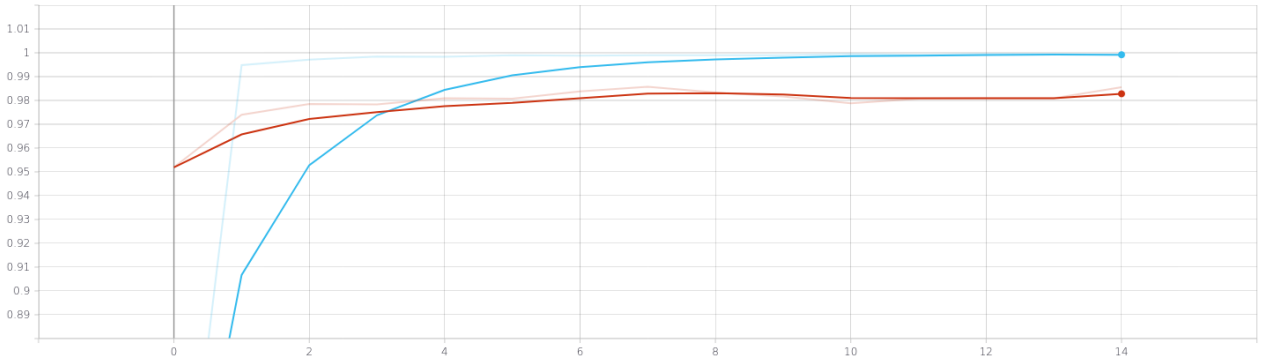

FIGURE 3.19 – Évolution de l'exactitude du Gestionnaire de dialogue par époque. En bleu l'ensemble d'apprentissage et en rouge l'ensemble de validation. Un lissage est effectué avec α égal à 0.6. L'ensemble de validation obtient un score d'exactitude supérieur 0.98 à la fin de l'entraînement

Rappelons que les résultats obtenus à la figure 3.19 sont issus d'un ensemble de données créé artificiellement. L'objectif étant d'obtenir un premier bot avec un ensemble minimal de fonctionnalités. Le prototype doit être soumis à un ensemble d'utilisateurs de test afin de récupérer des verbatims et des séquences de discours d'utilisateurs réels. On ne s'attardera donc pas davantage sur les résultats présentés ci-dessus. Ils confirment seulement le bon fonctionnement du système et ne font preuve d'aucune performance réelle.

3.2.5 Comment gérer les incompréhensions du chatbot ?

Le système tel qu'il est actuellement souffre d'un défaut majeur. Si le NLU prédit la mauvaise intention, l'erreur sera automatiquement propagée au gestionnaire de dialogue qui a de très fortes chances de répondre avec une action non désirée. De plus on souhaitera toujours maximiser la précision du système plutôt que le rappel. En d'autres mots, nous voulons garantir la pertinence d'une réponse renvoyée. Si à l'inverse l'incertitude de la réponse est trop élevée on préférera ne pas l'envoyer en acceptant le risque de bloquer des réponses potentiellement correctes. Pour ce faire, on fixe le seuil du NLU à 0.85. Seules les intentions avec un score supérieur à cette valeur seront envoyées aux gestionnaires du dialogue. On ajoute à cela un second seuil à 0.10, qui correspond à la différence entre les scores de confiance des deux intentions les mieux classées. Le score de l'intention prédite doit être au moins 0.10 supérieur au score de la seconde.

On ajoute un dernier seuil sur la prédiction du gestionnaire de dialogue. L'action est bloquée si son score est inférieur à 0.25.

Si l'un des seuils bloque le processus une action particulière appelée « action de » est utilisée pour prévenir l'utilisateur de l'incompréhension. Éventuellement des boutons à cliquer lui sont proposés pour l'aider à préciser sa demande.

Pour améliorer notre action de *fallback* nous introduisons un détecteur de thématique. Lors de l'entraînement du NLU, un modèle supplémentaire est donc entraîné. Les thématiques sont équivalentes aux standards Excel préparés par le CRC. (voir la section 3.1.3) :

— accès application
— carte bancaire
— chèque
— domiciliation bancaire
— informations personnelles
— paiement mobile
— virement

A cela s'ajoute une classe « non » qui réunit les intentions génériques de type salutations, formules de politesse, etc. Ce qui fait un total de 8 classes.

L'organisation de notre système nous met déjà à dispositions tout ce dont nous avons besoin. Les variations d'une intention d'un standard sont utilisées comme données d'entraînement pour la thématique associée. Pour ne pas complexifier l'architecture et augmenter le temps d'entraînement tout en garantissant des performances acceptables, on choisit d'utiliser un SVM (*support vector machine*) avec noyau linaire sur un double encodeur n-grammes de mots ($n < 5$) et n-grammes de caractères ($n < 5$) [1]. La valeur C est fixée à 0.01 avec une stratégie de classification « crammer_singer ».

On extrait 20% de l'ensemble sont extraites comme données de test. Nous avons 700 exemples pour l'entraînement et 176 pour le test. Nous obtenons des performances très satisfaisantes :

theme	precision	recall	f1-score	support
Accès Application	0.85	0.91	0.88	32
Carte Bancaire	1.00	1.00	1.00	47
Chèque	1.00	1.00	1.00	28
Domiciliation Bancaire	1.00	0.86	0.92	7
Informations Informationnels	0.88	0.79	0.83	19
Other	0.86	1.00	0.92	6
Paiement Mobile	0.96	1.00	0.98	24
Virement	1.00	0.92	0.96	13
accuracy			0.95	176
macro avg	0.94	0.93	0.94	176
weighted avg	0.95	0.95	0.95	176

TABLE 3.5 – Performances du modèle en charge de la détection de la thématique sur l'ensemble de test

1. Les n-grammes sont limités aux mots

Lors des échanges avec le *chatbot*, une entité thématique va marquer chaque message de l'utilisateur. Si un des seuils n'est pas franchi et que l'action de *fallback* est déclenchée alors on applique le traitement suivant :

1. Si le score des deux intentions les plus élevées est supérieur à 0.75 alors on propose à l'utilisateur de choisir entre une de ces deux intentions. (Cas de *fallback* n° 1)
2. Sinon si la thématique détectée est autre que « nan » avec un score supérieur à 0.20, on propose à l'utilisateur les différents cas traités pour cette thématique. (Cas de *fallback* n° 2)
3. Si le score des intentions est inférieur à 0.75, si aucune thématique a été détectée ou si la score de la thématique est inférieur à 0.20 alors on envoie un message d'incompréhension ainsi que la liste des sujets dans le périmètre du bot. En cliquant sur l'un des sujets, le bot proposera les cas dont il est capable de répondre. (Cas de *fallback* n° 3)

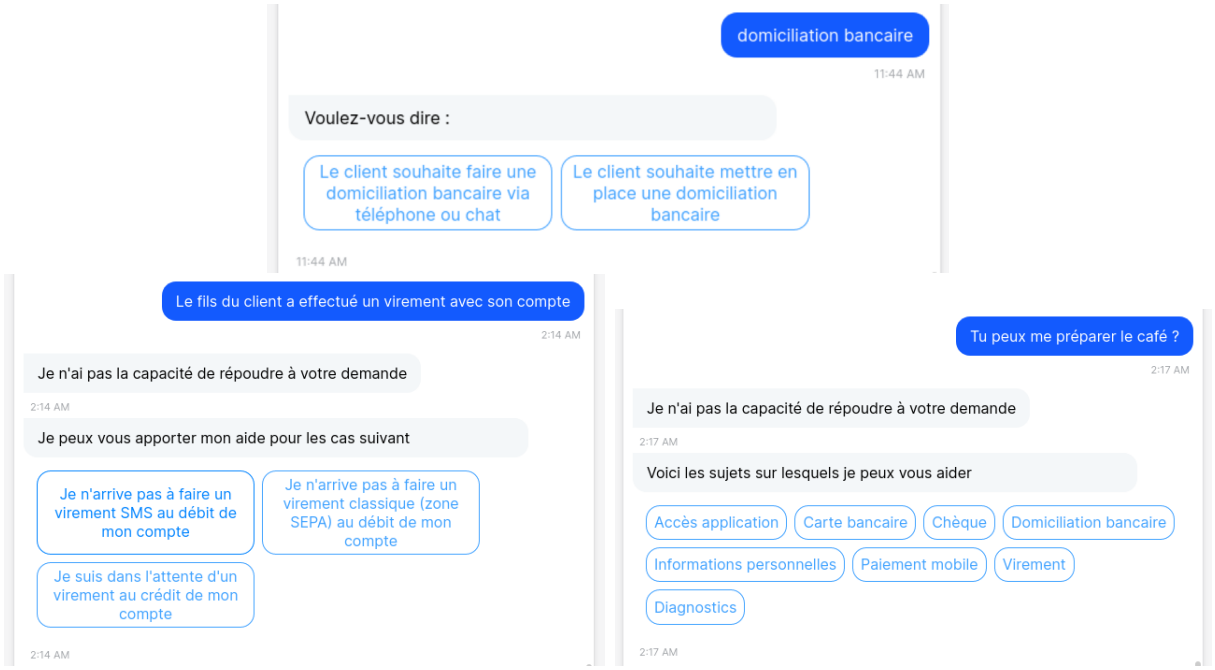

FIGURE 3.20 – Exemples de cas de fallback. En haut à gauche un cas n° 1, en bas à gauche un cas n° 2 et en bas à droite un cas n° 3.

3.3 L'amélioration continue

Développer un assistant virtuel robuste est très difficile car les utilisateurs trouveront toujours une phrase que le chabot ne pourra pas comprendre. La meilleure solution pour réduire les incompréhensions est d'écouter les conversations entre le bot et les utilisateurs afin de récupérer des verbatims réels au plus proche du cas d'usage destiné.

Cette démarche a également de fortes chances d'impacter les métiers annexes de l'entreprise. Plus spécifiquement, notre *chatbot* permettrait de détecter des problèmes récurrents non gérés initialement par le service relation client. Par exemple si une problématique non gérée par le bot revient régulièrement alors cela pourrait signifier qu'il faudrait modéliser une nouvelle résolution de problème.

3.3.1 Comment écouter les conversations ?

Les conversations entre l'agent et les utilisateurs contiennent des informations très importantes. Traquer les logs des échanges permet évidement d'améliorer les performances du système mais cela peut aussi permettre d'apporter des informations utiles à la stratégie de l'entreprise. Détecter un nombre de demande anormalement important d'une problématique permet par exemple d'identifier des anomalies à prioriser.

Pour mettre en place notre stratégie d'écoute et de visualisation nous choisissons d'utiliser l'outil ELK. ELK est un acronyme pour trois projets open source écrit en JAVA : Elasticsearch, Logstash et Kibana. Elasticsearch est un moteur de recherche et d'analyse basé sur Lucène. Il s'agit du coeur de la suite ELK. Logstash est destiné au traitement des données côté serveur. Il permet notamment d'intégrer simultanément des données provenant de multitude de sources, de les transformer puis de les envoyer vers un système de stockage, ici il s'agit d'Elasticsearch. Kibana est la dernière couche, il permet de visualiser facilement les données d'Elasticsearch avec des graphes et des tableaux hautement personnalisables. Tous les composants de la suite sont open source, ils s'interfacent donc facilement dans notre projet, flexible et peu coûteux.

Afin de sécuriser les échanges de données entre notre bot RASA et ELK nous introduisons un dernier logiciel appelé RabbitMQ. RabbitMQ est un « message broker », son rôle est de transporter et router les messages depuis des applications émettrices (notre bot) vers des applications consommatrices (notre ELK et plus spécifiquement Logstash). Il est écrit en langage Erlang et se base sur le protocole AMQP (*Advanced Message Queuing Protocol*). AMQP (pour *Advanced Message Queuing Protocol*) est un protocole d'échange pour les systèmes de messagerie orientés intergiciel (*middleware*). Tout à fait adapté à notre situation donc.

On obtient la structure suivante :

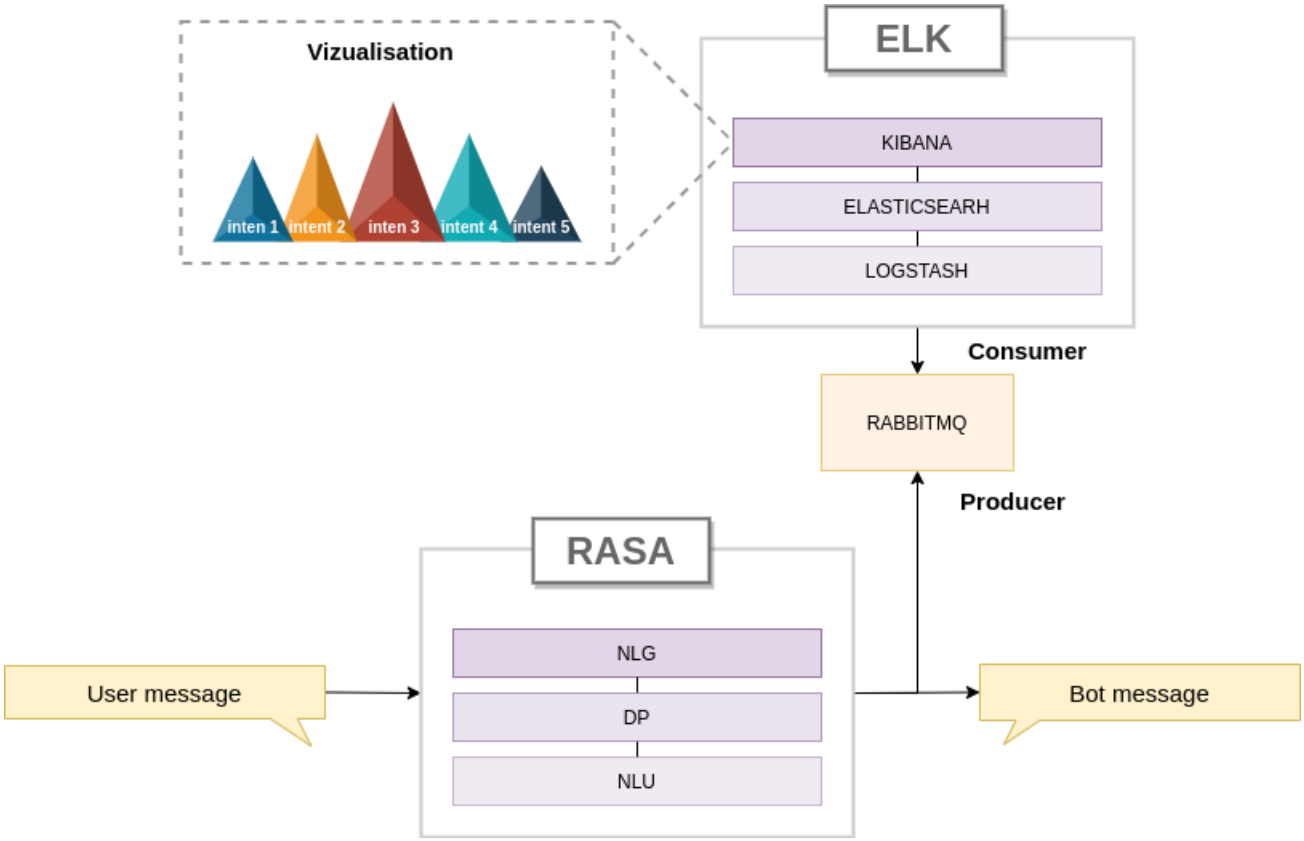

FIGURE 3.21 – Connexion du système entre ELK et RASA

3.3.2 Exemple pour la détection d'une anomalie

On simule l'utilisation du *chatbot* pendant une heure pour montrer l'intérêt d'un tel suivi.

En observant la distribution des thématiques, on aperçoit que 60% d'entre elles concernent l'accès à l'application, ce qui en fait la thématique la plus populaire de la période. La seconde étant seulement à 20

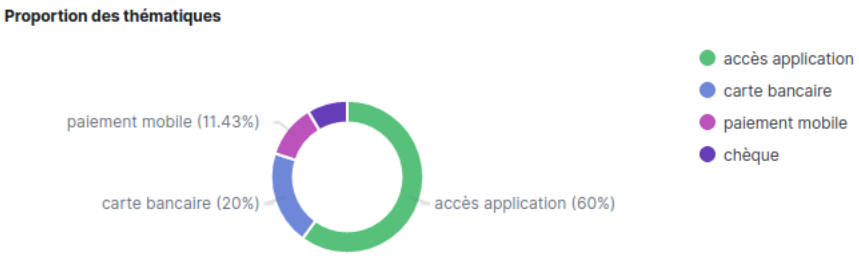

FIGURE 3.22 – Proportion des thématiques

En s'intéressant donc à la thématique « accès application » sur l'ensemble de la période. On ne détecte pas de pic particulier, elle semble récurrente sur l'ensemble de la période, même si elle est effectivement plus fréquente que les autres.

FIGURE 3.23 – Proportion des thématiques sur la période, en haut l'ensemble des thématiques, en bas seulement la thématique « accès application »

On observe maintenant la proportion des intentions. Les plus populaires concernent effectivement des problèmes liés à l'application. Les intentions nous donnent davantage de précisions sur la nature du problème. Les quatre premières « Le client a reçu un message d'erreur identifié » (21.69%), «Le client a un message d'erreur 'Service momentanément interrompu'» (20.48%), « Le client a reçu un message d'erreur » (4.82%) et « Diagnostic complet accès application » (3.61%) indiquent clairement que l'application semble mal fonctionner sur la période donnée. On suppose que ces résultats proviennent de diagnostics clients ayant reçus un message d'erreur lorsqu'ils se connectaient ou naviguait sur l'application. Ces statistiques obtenues en temps réels sont donc très utiles pour diagnostiquer un problème récurrent ou une anomalie immédiate. Elles permettent également de quantifier les prises en charge du problème en vue de futures améliorations.

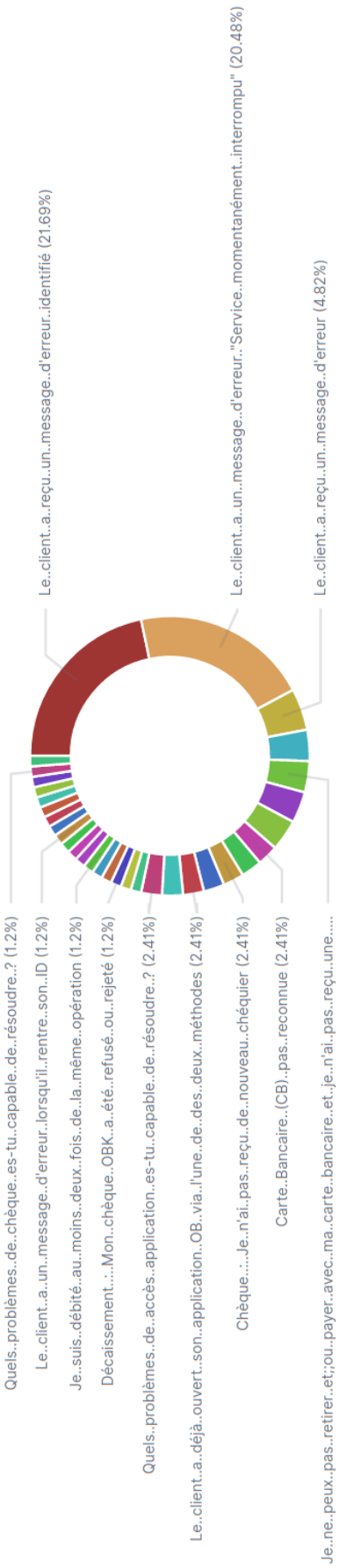

FIGURE 3.24 – Proportion des intentions par période

Une des autres utilités de l'outil est qu'il permet de faire des recherches sur tout ce dont nous avons besoin. Il est par exemple fréquent de vouloir récupérer les verbatims utilisés pour une intention pour augmenter ses variations et améliorer sa détection. Ici on effectue une recherche sur les verbatims avec ayant un score de confiance inférieur à 0.5, ceci afin d'identifier les verbatims ayant une forte probabilité d'avoir été mal classées. C'est cette même méthodologie que nous utiliserons pour enrichir les données d'entraînements du *chatbot*.

text	confidence
inpossible de payer avec le mobile	0.398
message d'erreur sur l'application	0.384
le client a recu un message d'errur sur l'pplication	0.381
l'application affiche service momentannément intérrompu	0.358
le client ne peut pas utiliser l'application	0.343
Comment réparer le paiement par mobile ?	0.337
impossible de payer avec le mobile	0.332
le client ne peut pas se connecter à l'application	0.326
Le client ne peut pas se conencter sur l'application	0.321
le client a recu un message identifi&é	0.248

TABLE 3.6 – Échantillon de verbatims filtrés par score de confiance inférieur à 0.5. Les verbatims ont été restitués telles qu'elles ont été envoyées au système. Les fautes sont donc volontaires.

Chapitre 4

Perspective d'évolution : intégrer un composant de gestion de FAQ

En plus des standards, le CRC utilise une base de connaissances très complète qui rassemble les réponses aux questions les plus fréquentes. Cette base est appelée « FAQ All About » (voir la figure 4.1). Pour réunir les différentes sources d'informations utilisées par le CRC, nous voulons intégrer dans le *chatbot* un nouveau module en charge de cette FAQ. Or, les textes non structurés et de tailles conséquentes de la FAQ ne sont pas directement compatibles avec l'architecture précédemment présentée. Un développement indépendant et personnalisé est donc nécessaire. Pour le réaliser nous allons procéder en deux étapes : Identifier le document pertinent (le doublet question-réponse), puis extraire l'élément précis de la réponse.

A noter que cette partie est purement théorique, elle propose certaines approches intéressantes qui semblent répondre au besoin du futur composant.

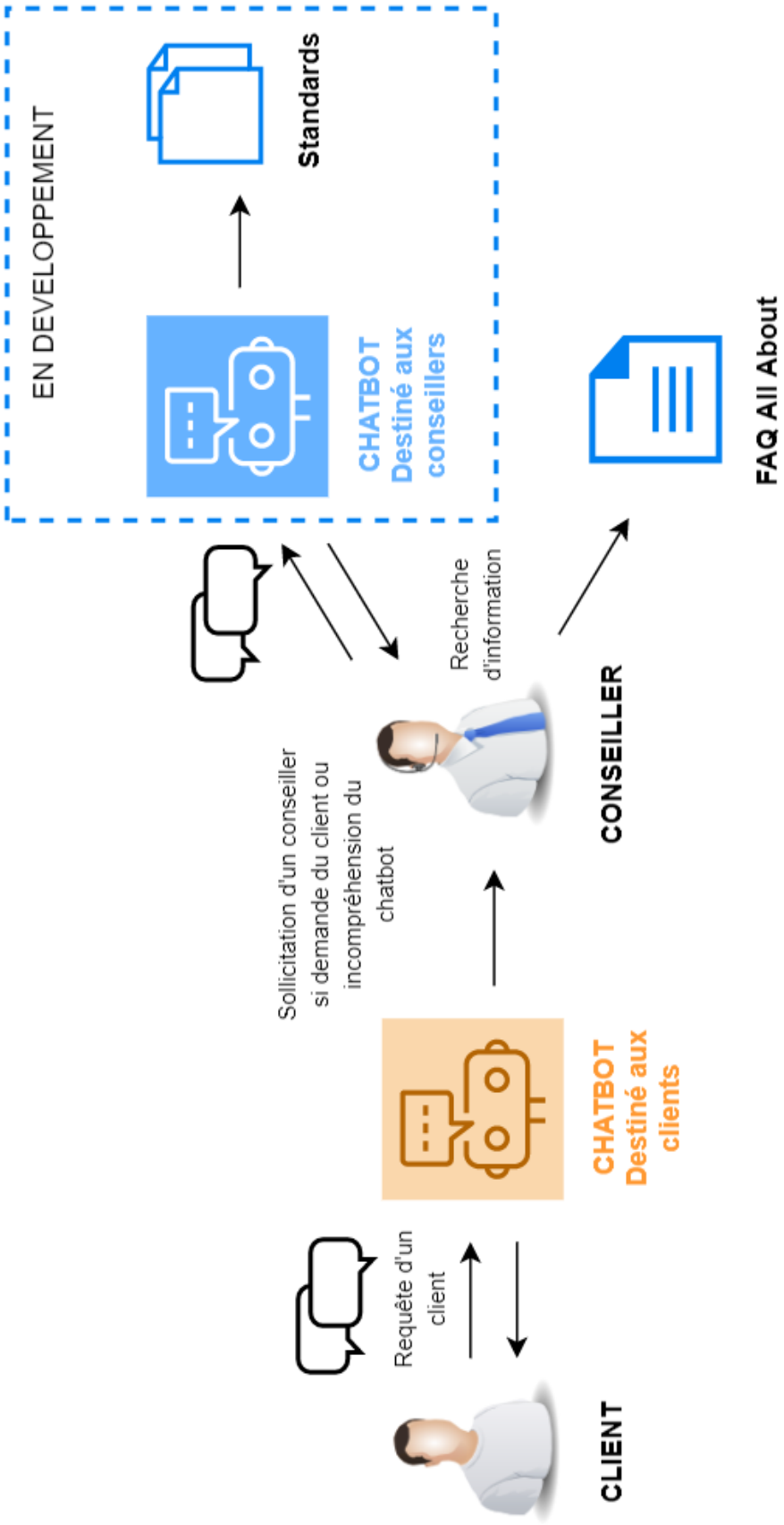

FIGURE 4.1 – Procédure d'accès à l'information : du client jusqu'à la source d'information

4.1 Les données : La FAQ AllAbout

Cette FAQ contient près de 300 paires de questions-réponses. Il s'agit d'un corpus très complet avec une multitude d'informations. Les réponses généralement très développées permettent aux conseillers d'obtenir un maximum d'informations pour une question donnée. Un avantage qui devient problématique lorsque le conseiller requiert une information simple et courte. Par exemple pour chercher le numéro de téléphone d'un service. Il doit connaître la question traitant du service demandé puis ensuite chercher le numéro dans la masse d'information qu'il reçoit. La conséquence de cette démarche est une dépense non négligeable de temps. Cette observation fait apparaître le besoin suivant : un système de recherche d'information dynamique et optimisé.

Ci-dessous un exemple de question-réponse issu de la FAQ :

1. — Que faire en cas de perte ou de vol du mobile ?

Contexte : Avec Orange Bank, le téléphone du client devient également un moyen de paiement. Voici la marche à suivre en cas de perte ou vol du téléphone.

Réponse : Lorsque le client perd ou se fait voler son téléphone mobile, il doit avoir un réflexe : Pour bloquer votre paiement mobile, vous devez déclarer le vol de votre mobile en appelant le +33(0)9.69.32.82.88 appel non surtaxé puis tapez 2. Un message vous demandera de saisir votre numéro de mobile volé/perdu et déclenchera le blocage de votre mobile. De plus, pour sécuriser l'application il est également nécessaire d'effectuer une manipulation sur le logiciel Inwebo. Le CRC peut solliciter un CT pour effectuer l'action La suppression du téléphone volé ou perdu des devices est nécessaire afin d'ajouter un nouveau terminal de confiance. Voir la question « Comment supprimer le téléphone de confiance ? » A noter : Cette demande de blocage n'a aucune incidence sur le fonctionnement de la carte bancaire.

Niveau de sécurisation : L'authentification via le widget est obligatoire pour toute demande client. Voir la question : « Comment authentifier mon client par le biais du widget ?

Pas à pas du conseiller : La création de deux requêtes est obligatoire :

- Type d'enregistrement de requête → Demande
- Canal d'origine → Appel entrant ou Chat à définir - selon le cas
- Type → Accessibilité
- Sous-type → Accès mobile
- Statut → A faire
- Priorité → Normal

et

- Type d'enregistrement de requête → Demande
- Canal d'origine → Appel entrant ou Chat à définir selon le cas
- Type → Paiement mobile
- Sous-type → Opposition
- Statut → A faire
- Priorité → Normal

Bonne pratique : N'oubliez pas qu'il est également nécessaire de tracer les éléments suivants et récapituler l'échange : - Pour la partie chat > traçage automatique dans la « transcription live chat » et renseigner le numéro de la transcription live chat dans la requête. - Pour la partie téléphonie > « Voice-Inbound » à compléter.

FIGURE 4.2 – Exemple de question-réponse. Document pour la question « Que faire en cas de perte ou de vol de mon mobile ? »

4.2 Comment identifier le bon document ?

La situation nous oriente clairement vers une tâche de recherche d'information. Le principe des *chatbots* basés sur la recherche d'informations est de répondre à l'intervention d'un utilisateur en allant chercher la réponse dans un corpus de documents. Les différences entre ces systèmes résident dans la façon dont ils utilisent le corpus (indexation, encodage, etc.) et comment ils décident de la réponse appropriée (fonction de similarité, modèle d'apprentissage, etc.). Généralement les systèmes basés sur la recherche d'information possèdent un

large choix d'algorithme pour chercher la réponse appropriée.

Pour une requête q et un corpus C, calculer les similarités entre chaque document et la requête q puis renvoyer le document le plus similaire à q. N'importe quelle fonction de calcul de similarité peut être utilisée. Le choix le plus commun est une similarité cosinus calculée sur les documents vectorisés en TF-IDF. [Yan et al., 2016] représente bien cette méthodologie dont le moteur utilisent trois composants :

— Récupération des meilleurs candidats C dans D basé sur q : $C = Retrieve(Q, D)$ avec chaque S $\in$ D est une réponse candidate

— Classement de toutes les réponses candidates dans C et sélectionne la réponse la plus pertinente : $T = \arg\max_{S \in C} Rank(S, Q)$

— Déclenchement de la réponse, il est décidé ici si le score de confiance est suffisamment élevé pour renvoyer la réponse : $I = Trigger(T, Q)$ avec I valeur binaire

4.2.1 La méthode traditionnelle : Par calcul de similarité

Les systèmes de questions-réponses (QA) qui utilisent des méthodes similaires sont devenus des services web très populaires. Le fonctionnement traditionnel de ces systèmes est de comparer la requête de l'utilisateur avec chaque paire de question-réponse. La paire avec le plus haut score de similarité est alors renvoyé. Les performances du système sont donc très dépendantes de la pertinence du calcul de similarité.

Les calculs de similarité peuvent être statistiques ou sémantiques, généralement obtenue avec des outils tels que WordNet ([Burke et al., 1997, Song et al., 2007]). Il est souvent judicieux de maximiser cette similarité en effectuant quelques prétraitements sur le texte. C'est le cas de [Song et al., 2007] qui proposent trois traitements :

— Un étiquetage en partie du discours (POS tagging) qui sera utilisé à la fois pour chercher les mots sémantiquement proches dans WordNet mais également pour apporter davantage d'informations pour le calcul de similarité avec la requête.

— Une racinisation pour réduire les discriminations des mots par leurs fléxions

— La suppression des « mots-vides » depuis une liste prédéfinie

Le calcul de similarité est une combinaison entre une similarité statistique et une similarité sémantique. La similarité statistique est basée sur les occurrences de mots, la requête est mappée avec chaque question de la base. Pour chaque paire un set de vocabulaire est obtenu par union des vocabulaires des deux séquences. Ensuite pour chaque séquence les vecteurs sont construits en récupérant les fréquences associées à chaque mot.

La similarité sémantique est obtenue par score de similarité WordNet. Un premier mapping de la requête sur la question est effectué puis un second où c'est la question qui est mappée sur la requête. La similarité est calculée par la moyenne des valeurs obtenues après les deux itérations. Rapide et légère en calculs, elle s'intègre facilement dans un projet. Elle a également l'avantage d'être très modulable, les prétraitements peuvent être adaptés pour mieux correspondre au corpus de travail.

4.2.2 Les systèmes à base de Bert

Avec la réapparition des réseaux de neurones de ces dernières années, des systèmes se sont d'abord vu doter de réseaux de neurones à convolution (CNN) [Karan and Snajder, 2016],

puis de réseaux de neurones récurrents à mémoire court et long terme (LSTM) [Gupta and Carvalho, 2019]. Ces tentatives, bien que prometteuses, ont l'inconvénient de demander de grandes quantités de données étiquetées (des requêtes utilisateurs associées aux couples de question-réponse) pour l'entraînement des modèles.

Depuis le grand succès des architectures transformers et des modèles pré-entraînés, des nouveaux systèmes combinant des méthodes traditionnelles non supervisées à des modèles Bert sont apparues.

Exemple 1 [Sakata et al., 2019] combinent la score de similarité TSUBAKI entre la requête et la question, et un score de pertinence basé sur un modèle Bert entre la requête et la réponse. [Sakata et al., 2019] proposent donc un système basé sur deux critères :

— La similarité entre la requête d'un utilisateur et les questions de la FAQ

— La pertinence entre la requête et la réponse à l'aide d'un modèle Bert

Les auteurs motivent leur choix en précisant que cette méthode non-supervisée a l'avantage de ne pas nécessiter de corpus annoté.

La similarité q-Q (requête-question) est calculée grâce à la similarité TSUBAKI [Shinzato et al., 2012]. TSUBAKI considère l'analyse en dépendances d'une phrase afin de fournir une représentation précise. Elle utilise également des synonymes extraits automatiquement de dictionnaires et de corpus.

BERT [Devlin et al., 2018] est utilisé pour calculer la pertinence q-A (requête-réponse). Il est basé sur l'architecture transformer pour l'encodage du texte. BERT est appliqué sur un classificateur de paires de phrases (requête et réponses). Le score de pertinence est obtenu par inférence du modèle sur la requête et la réponse.

Les deux systèmes sont ensuite combinés. Les dix scores les plus élevés envoyés par BERT sont conservés. Parmi elles, seules sont conservées celles ayant un score de TSUBAKI supérieur à α, le seuil de pertinence. Elles sont ensuite triées par ordre croissant du score de TSUBAKI. Les autres sont classées par ordre de similitude : $Score = (q, Q)t + pertinence(q, A)$ avec t comme hyperparamètre.

Le score de TSUBAKI est normalisé par le nombre de *tokens* et le nombre de dépendances de la requête puisqu'il a tendance à augmenter avec l'augmentation de ces derniers :

$$PertinenceScore = \frac{Score}{Count(ContentWords) \cdot k1 + Count(DependencyRelations) \cdot k2} \tag{4.1}$$

avec k1 et k2 comme hyperparamètres.

Exemple 2 [Mass Yosi, 2020] effectuent un classement par une similarité appelée BM25 [Robertson Stephen, 2009] puis utilisent deux modèles Bert indépendants, un pour la similarité requête-question et l'autre pour la similarité requête-réponse pour reclasser successivement les paires obtenues.

Trois composants sont donc utilisés :

— Similarité Q-(q+a) BM25 : Un score est calculé entre la requête de l'utilisateur et chaque couple de question-réponse. Un premier classement des couples est alors obtenu

— Similarité Q-a BERT : Un modèle est entraîné sur des couples positifs, c'est à dire une question et sa réponse associée et des couples négatifs, une question et une réponse aléatoire non associée. Il est ensuite appliqué pour obtenir la similarité entre la requête et les réponses du corpus. Seul le corpus FAQ est utilisé pour l'entraînement.

— Similarité Q-q BERT : Un autre modèle pour calculer la similarité entre la requête et les questions du corpus. Ce modèle est entraîné sur un *dataset* de paraphrases de questions générées automatiquement à l'aide d'une architecture GPT-2. Le bruit est minimisé en conservant uniquement les paraphrases ayant une sémantique proche de leur question. Le modèle est entraîné sur des couples positifs, une question et sa paraphrase associée et des couples négatifs, une question et une paraphrase aléatoire non associée.

Les scores des trois systèmes sont combinés pour obtenir un score de similarité final entre une requête et les couples de question-réponse. [Mass Yosi, 2020] proposent deux méthodes de combinaison. La première nommée combSUM [Kurland and Culpepper, 2018] calcule la somme de tous les scores obtenus pour chaque paire Q-A. La seconde nommée PoolRank, ajoute un modèle RMI [Lavrenko and Croft, 2001] sur la sortie de CombSUM pour reclasser l'ensemble des meilleurs candidats.

Une méthode similaire peut être envisagée sur notre FAQ. S'il n'existe pas de de modèle GPT2 pour la génération de paraphrase pour le français des alternatives tel que le corpus PAW-S (`https://github.com/google-research-datasets/paws`) peut être utilisé pour entraîner un modèle et prédire le score de similarité.

4.3 Comment extraire la réponse ?

Très récemment, sont sortis deux corpus de questions-réponses nativement français, FQUAD [Martin et al., 2020] et PIAF [Keraron et al., 2020]. FQUAD est issu d'articles Wikipédia, il est composé de plus de 25 000 échantillons de questions-réponses pour la version 1.0 et plus de 60 000 pour la version 1.1. PIAF est de taille plus modeste avec 3835 questions-réponses collectées pour sa version 1 et 7569 pour sa version 1.1. Les deux projets s'inspirant fortement du projet SQUAD utilisent un format similaire dont voici la structure :

```
{
  version: "Version du dataset"
  data:[
    {
      title: "Titre de l'article Wikipedia"
      paragraphs:[
        {
          context: "Paragraphe de l'article"
          qas:[
            {
              id: "Id de la la paire de question-reponse"
              question: "Question"
              answers:[
                {
                  "answer_start": "Position de la reponse"
                  "text": "Reponse"
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

FIGURE 4.3 – Format des *datasets* SQUAD-*like*

Les corpus sont utilisés pour fine-tuned des modèles de langues préentrainés sur des tâches de questions-réponses. Le module Transformers recense déjà plusieurs modèles Bert et CamemBert prêts à l'emploi, à utiliser comme tel ou à réentraîner sur des données spécifiques :

- `https://huggingface.co/fmikaelian/camembert-base-fquad`
- `https://huggingface.co/illuin/camembert-large-fquad`
- `https://huggingface.co/etalab-ia/camembert-base-squadFR-fquad-piaf`
- `https://huggingface.co/illuin/camembert-base-fquad`
- `https://huggingface.co/JDBN/t5-base-fr-qg-fquad`

Les modèles peuvent d'ores et déjà être utilisés en l'état comme dans l'ensemble suivant :

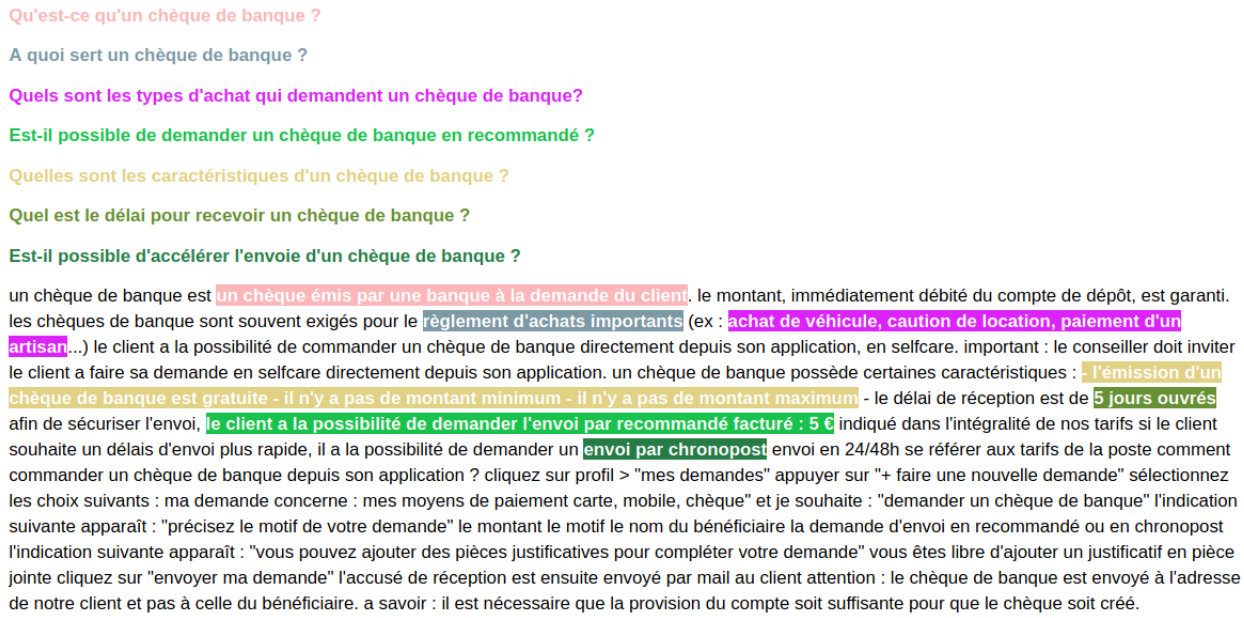

FIGURE 4.4 – Exemple sur le document « Comment commander un chèque de banque ? » avec le modèle camembert-large *fine-tuned* sur le corpus fquad

Cependant, les données présentent dans les entreprises sont souvent propre à leur domaine d'activité ou au service concerné. Les modèles étant entraînés sur des données à domaines générals, risquent de manquer de performances sur des données spécifiques. Notre FAQ appartient au domaine bancaire et contient du vocabulaire spécifique à l'entreprise, nous sommes donc en présence de caractéristiques particulières. Ré-entrainer le modèle choisi sur l'ensemble de la FAQ semble donc recommandé. Pour obtenir un corpus annoté, une campagne d'annotation interne à l'entreprise doit être organisée. Pour ce faire [Keraron et al., 2020] ont développé une plateforme open-source spécifique à l'annotation de questions-réponses disponible à l'adresse `https://github.com/etalab/piaf`. Cette campagne d'annotation est en cours de réalisation à l'écriture de ce travail.

Une fois les modèles entraînés le composant peut être facilement connecté à l'architecture actuelle du *chatbot*. Si le NLU identifie une demande d'information, le message est marqué par une entité. Le manager du dialogue détecte cette entité et déclenche une action personnalisée qui appelle le composant QA. Un premier module recherche les documents pouvant potentiellement contenir la réponse. Ensuite un second module sera en charge d'extraire la réponse de ces documents. La réponse est finalement envoyée à l'utilisateur (voir la figure 4.5).

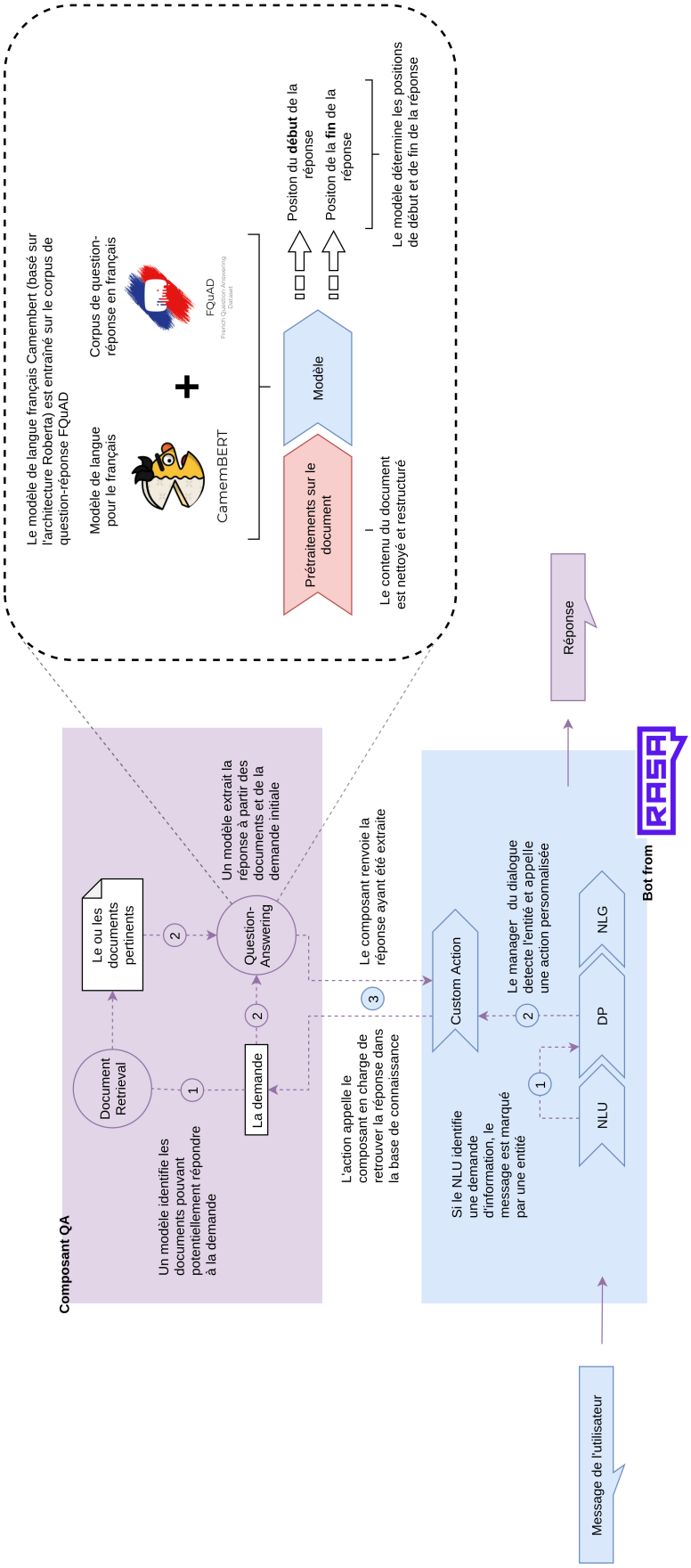

FIGURE 4.5 – Intégration du composant en charge de la FAQ sur la structure actuelle du système, la partie Rose est en cours de développement

Conclusion

L'évolution des technologies en traitement automatiquement des langues et en apprentissage automatique ont permis de grandes avancées des solutions *chatbots*. Nous avons proposé dans ce travail une démarche opérationnelle axée sur la modularité et le développement open source. La solution a la capacité d'évoluer facilement avec peu d'efforts grâce à l'automatisation entre modélisation et intégration. Tous les composants utilisés sont open source et gratuits. Les *frameworks* tels que Rasa facilitent l'intégration des algorithmes état de l'art existant tout en donnant la possibilité de créer nos propres modèles. Nous avons également vu que la visualisation n'est pas négligeable aussi bien pour l'amélioration continue que pour la détection d'anomalie.

En revanche, la création des variations pour le corpus d'apprentissage du NLU reste un problème. Nous avons créé quelques verbatims pour vérifier le bon fonctionnement de la solution proposée mais le test du modèle formé pour le NLU a dû être effectué sur un ensemble de données différent afin de ne pas biaiser les résultats. Former un corpus d'entraînement adapté aux cas d'usages s'inscrit dans un processus relativement long. Une phase de test du bot est actuellement en cours pour récupérer des verbatims réels et créer les données nécessaires.

Enfin nous avons proposé une solution théorique pour le développement d'une extension permettant la prise en charge d'une FAQ. Le composant est basé sur la recherche de document et l'extraction fine de réponse. L'avantage est une augmentation facile du périmètre du *chatbot* sans modélisation et corpus d'apprentissage. Un corpus de questions-réponses annoté est cependant nécessaire pour spécialiser le modèle d'extraction.

Bibliographie

[Adiwardana et al., 2020] Adiwardana, D., Luong, M.-T., So, D. R., Hall, J., Fiedel, N., Thoppilan, R., Yang, Z., Kulshreshtha, A., Nemade, G., Lu, Y., and Le, Q. V. (2020). Towards a human-like open-domain chatbot.

[Bocklisch et al., 2017] Bocklisch, T., Faulkner, J., Pawlowski, N., and Nichol, A. (2017). Rasa : Open source language understanding and dialogue management.

[Bunk et al., 2020] Bunk, T., Varshneya, D., Vlasov, V., and Nichol, A. (2020). Diet : Lightweight language understanding for dialogue systems.

[Burke et al., 1997] Burke, R., Hammond, K., Kulyukin, V., Lytinen, S., Tomuro, N., and Schoenberg, S. (1997). Question answering from frequently asked question files : Experiences with the faq finder system. *AI Magazine*, 18:57–66.

[Chen et al., 2019] Chen, Q., Zhuo, Z., and Wang, W. (2019). Bert for joint intent classification and slot filling.

[DEBOS, 2018] DEBOS, F. (2018). Les chatbots, vecteur humanisant ou financier ?

[Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert : Pre-training of deep bidirectional transformers for language understanding.

[Gupta and Carvalho, 2019] Gupta, S. and Carvalho, V. R. (2019). Faq retrieval using attentive matching. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 929–932, New York, NY, USA. Association for Computing Machinery.

[Henderson et al., 2020] Henderson, M., Casanueva, I., Mrkšić, N., Su, P.-H., Wen, T.-H., and Vulić, I. (2020). Convert : Efficient and accurate conversational representations from transformers.

[Hofstadter, 1996] Hofstadter, D. R. (1996). Fluid concepts and creative analogies : Computer models of the fundamental mechanisms of thought, new york, basic bookse.

[Hussain et al., 2019] Hussain, S., Sianaki, O., and Ababneh, N. (2019). *A Survey on Conversational Agents/Chatbots Classification and Design Techniques*, pages 946–956.

[Karan and Snajder, 2016] Karan, M. and Snajder, J. (2016). Faqir - a frequently asked questions retrieval test collection. In *TSD*.

[Keraron et al., 2020] Keraron, R., Lancrenon, G., Bras, M., Allary, F., Moyse, G., Scialom, T., Soriano-Morales, E.-P., and Staiano, J. (2020). Project piaf : Building a native french question-answering dataset.

[Kurland and Culpepper, 2018] Kurland, O. and Culpepper, J. S. (2018). Fusion in information retrieval : Sigir 2018 half-day tutorial. In *The 41st International ACM SIGIR Conference on Research amp ; Development in Information Retrieval*, SIGIR '18, page 1383–1386, New York, NY, USA. Association for Computing Machinery.

[Lavrenko and Croft, 2001] Lavrenko, V. and Croft, W. B. (2001). Relevance based language models. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, page 120–127, New York, NY, USA. Association for Computing Machinery.

[Le et al., 2020] Le, H., Vial, L., Frej, J., Segonne, V., Coavoux, M., Lecouteux, B., Allauzen, A., Crabbé, B., Besacier, L., and

Schwab, D. (2020). Flaubert : Unsupervised language model pre-training for french.

[Li et al., 2016] Li, J., Monroe, W., Ritter, A., Galley, M., Gao, J., and Jurafsky, D. (2016). Deep reinforcement learning for dialogue generation.

[Li et al., 2017] Li, J., Monroe, W., Shi, T., Jean, S., Ritter, A., and Jurafsky, D. (2017). Adversarial learning for neural dialogue generation.

[Liu et al., 2019a] Liu, X., Eshghi, A., Swietojanski, P., and Rieser, V. (2019a). Benchmarking natural language understanding services for building conversational agents.

[Liu et al., 2019b] Liu, X., He, P., Chen, W., and Gao, J. (2019b). Multi-task deep neural networks for natural language understanding.

[Lowe et al., 2017] Lowe, R., Pow, N., Serban, I., Charlin, L., Liu, C., and Pineau, J. (2017). Training end-to-end dialogue systems with the ubuntu dialogue corpus. *Dialogue Discourse*, 8:31–65.

[Martin et al., 2020] Martin, d., Maxime, V., Wacim, B., and Tom, B. (2020). FQuAD : French Question Answering Dataset. *arXiv e-prints*, page arXiv:2002.06071.

[Martin et al., 2020] Martin, L., Muller, B., Ortiz Suárez, P. J., Dupont, Y., Romary, L., de la Clergerie, É., Seddah, D., and Sagot, B. (2020). CamemBERT : a tasty French language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7203–7219, Online. Association for Computational Linguistics.

[Mass Yosi, 2020] Mass Yosi, Carmeli Boaz, R. H. K. D. (2020). Unsupervised FAQ retrieval with question generation and BERT. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 807–812, Online. Association for Computational Linguistics.

[Mehri et al., 2019] Mehri, S., Srinivasan, T., and Eskenazi, M. (2019). Structured fusion networks for dialog.

[Nichol, 2020] Nichol, A. (2020). 5 Levels of Conversational AI : 2020 Update.

[Qiu et al., 2017] Qiu, M., Li, F.-L., Wang, S., Gao, X., Chen, Y., Zhao, W., Chen, H., Huang, J., and Chu, W. (2017). AliMe chat : A sequence to sequence and rerank based chatbot engine. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2 : Short Papers)*, pages 498–503, Vancouver, Canada. Association for Computational Linguistics.

[Ren et al., 2018] Ren, L., Xie, K., Chen, L., and Yu, K. (2018). Towards universal dialogue state tracking.

[Ritter et al., 2011] Ritter, A., Cherry, C., and Dolan, W. B. (2011). Data-driven response generation in social media. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 583–593, Edinburgh, Scotland, UK. Association for Computational Linguistics.

[Robertson Stephen, 2009] Robertson Stephen, Z. H. (2009). The probabilistic relevance framework : Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

[Sakata et al., 2019] Sakata, W., Shibata, T., Tanaka, R., and Kurohashi, S. (2019). Faq retrieval using query-question similarity and bert-based query-answer relevance. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 1113–1116, New York, NY, USA. Association for Computing Machinery.

[Searle, 1980] Searle, J. R. (1980). Minds, brains, and programs. *Behavioral and Brain Sciences*, 3(3):417–57.

[Shang et al., 2015] Shang, L., Lu, Z., and Li, H. (2015). Neural responding machine for short-text conversation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural*

Language Processing (Volume 1 : Long Papers), pages 1577–1586, Beijing, China. Association for Computational Linguistics.

[Shi et al., 2020] Shi, N., Zeng, Q., and Lee, R. (2020). The design and implementation of language learning chatbot with xai using ontology and transfer learning.

[Shinzato et al., 2012] Shinzato, K., Shibata, T., Kawahara, D., and Kurohashi, S. (2012). Tsubaki : An open search engine infrastructure for developing information access methodology. *Journal of Information Processing*, 20(1):216–227.

[Song et al., 2007] Song, W., Feng, M., Gu, N., and Wenyin, L. (2007). Question similarity calculation for faq answering. In *Third International Conference on Semantics, Knowledge and Grid (SKG 2007)*, pages 298–301.

[Sordoni et al., 2015] Sordoni, A., Galley, M., Auli, M., Brockett, C., Ji, Y., Mitchell, M., Nie, J.-Y., Gao, J., and Dolan, B. (2015). A neural network approach to context-sensitive generation of conversational responses.

[Strubell et al., 2019] Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in nlp.

[Strubell et al., 2018] Strubell, E., Verga, P., Andor, D., Weiss, D., and McCallum, A. (2018). Linguistically-informed self-attention for semantic role labeling.

[Turing, 1950] Turing, A. (1950). Computing machinery and intelligence.

[Vidal F., 2016] Vidal F., . V. J.-M. (2016). Analyse entretien : Eric labaye, président du mckinsey global institute. *Trends Tendances*, 33:22–24.

[Vinyals and Le, 2015] Vinyals, O. and Le, Q. (2015). A neural conversational model.

[Vlasov et al., 2018] Vlasov, V., Drissner-Schmid, A., and Nichol, A. (2018). Few-shot generalization across dialogue tasks.

[Vlasov et al., 2020] Vlasov, V., Mosig, J. E. M., and Nichol, A. (2020). Dialogue transformers.

[Wallace, 2003] Wallace, R. (2003). The elements of aiml style.

[Weizenbaum, 1966] Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45.

[Weizenbaum, 1976] Weizenbaum, J. (1976). Computer power and human reason : from judgment to calculation.

[Wen et al., 2016] Wen, T.-H., Vandyke, D., Mrksic, N., Gasic, M., Rojas-Barahona, L. M., Su, P.-H., Ultes, S., and Young, S. (2016). A network-based end-to-end trainable task-oriented dialogue system.

[Wu et al., 2018] Wu, Y., Wu, W., Li, Z., and Zhou, M. (2018). Learning matching models with weak supervision for response selection in retrieval-based chatbots.

[Yamaguchi et al., 2018] Yamaguchi, H., Mozgovoy, M., and Danielewicz-Betz, A. (2018). A chatbot based on aiml rules extracted from twitter dialogues. pages 37–42.

[Yan et al., 2016] Yan, Z., Duan, N., Bao, J., Chen, P., Zhou, M., Li, Z., and Zhou, J. (2016). DocChat : An information retrieval approach for chatbot engines using unstructured documents. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, pages 516–525, Berlin, Germany. Association for Computational Linguistics.

[Young et al., 2017] Young, T., Cambria, E., Chaturvedi, I., Huang, M., Zhou, H., and Biswas, S. (2017). Augmenting end-to-end dialog systems with commonsense knowledge.