DEPARTEMENT INFORMATIQUE - IUT 2 GRENOBLE

Année universitaire 2024-2025

MEMOIRE DE STAGE

---

# Analyse de solutions de chatbots en support au processus de recrutement ; développement d'une preuve de concept.

## Laboratoire d'informatique de Grenoble, équipe IIHM

---


Présenté par

Tristan Fontanière


Jury

IUT : M. Cédric Gérot

IUT : Mme. Michelle Rosset

Société : M. Baptiste Ledoyen

Déclaration de respect des droits d'auteurs

Par la présente, je déclare être le seul auteur de ce rapport et assure qu'aucune autre ressource que celles indiquées n'ont été utilisées pour la réalisation de ce travail. Tout emprunt (citation ou référence) littéral ou non à des documents publiés ou inédits est référencé comme tel et tout usage à un outil doté d'IA a été mentionné et sera de ma responsabilité.

Je suis informé qu'en cas de flagrant délit de fraude, les sanctions prévues dans le règlement des études en cas de fraude aux examens par application du décret 92-657 du 13 juillet 1992 peuvent s'appliquer. Elles seront décidées par la commission disciplinaire de l'UGA.

A Grenoble Le 24/04/2025 Signature

## Remerciements

Je tiens tout d'abord à exprimer ma profonde gratitude à mon encadrant de stage M. Baptiste Ledoyen pour la confiance qu'il m'a accordé et les connaissances qu'il m'a transmises. Grâce à lui j'ai pu apprendre beaucoup d'éléments sur les méthodes de recherche, l'environnement scientifique en informatique ainsi que les technologies d'intelligence artificelle.

Mes remerciements vont également à M. Cédric Gérot pour son encadrement d'un point de vue académique, ses conseils quant à la rédaction des documents attendus m'ont été d'une précieuse aide.

Je souhaite également remercier Mme. Sophie Dupuy-Chessa pour la confiance qu'elle m'a accordée en me proposant ce stage au sein de son équipe de recherche.

Je suis également reconnaissant envers tous les membres de l'équipe IIHM du Laboratoire d'Informatique de Grenoble de m'avoir accueilli et intégré durant ces 9 semaines.

Enfin, je remercie l'ensemble du corps enseignant de l'IUT 2 de m'avoir transmis tant de connaissances durant ces deux années de BUT qui m'ont été fortement utiles durant mon stage.

# Table des matières

# Table des figures

# I. Introduction

Dans le cadre de ma deuxième année d'étude en BUT informatique j'ai réalisé un stage au sein du Laboratoire d'Informatique de Grenoble, dans l'équipe IIHM (Ingénierie de l'Interaction Humain-Machine). Ce stage émerge d'une recherche doctorale centrée sur l'étude de l'interaction collaborative avec un Chatbot de recrutement. Dans ce contexte un chatbot ou agent conversationnel soutient la collaboration entre deux utilisateurs.

Dans ce rapport le premier chapitre présentera le laboratoire et l'équipe dans laquelle la mission a eu lieu. Le second chapitre traitera de l'étude de l'existant à la fois sur les principes et concepts des chatbots mais également sur les outils. Le troisième chapitre détaillera la partie développement et plus précisément de la façon dont a été testé et prototypé un outil.

Veuillez noter que tous les mots ou groupes de mots suivis d'une astérisque "*" sont définis dans le Glossaire. De plus, le vocabulaire propre au domaine étudié sera traduit une fois puis conservé en anglais conformément à l'usage courant chez les experts, y compris les francophones.

# II. Présentation du laboratoire et du stage

## II..1 *Présentation du laboratoire*

Le Laboratoire d'Informatique de Grenoble est un laboratoire de recherche français centré sur l'informatique et fondé le 1er janvier 2007. Il résulte d'un groupement de cinq laboratoires de l'IMAG (Institut d'informatique et Mathématiques Appliquées de Grenoble) qui est également le nom du bâtiment où se situent la majorité des équipes. Les autres équipes sont réparties sur deux sites géographiques : Minatec et Montbonnot. Il est dirigé par Noël de Palma depuis 2021. Il est sous la tutelle de l'Université Grenoble Alpes, le CNRS, Grenoble INP et l'INRIA [10].
Aujourd'hui il compte 22 équipes de recherches réparties sur cinq axes : génie logiciel, méthodes formelles modèles et langages, systèmes intelligents, systèmes interactifs et cognitifs, systèmes répartis. Il regroupe aujourd'hui 450 personnes dont des chercheurs, des enseignants-chercheurs, des doctorants et du personnel en support à la recherche et administratifs. Il accueille chaque année près de 150 stagiaires [8].

## II..2 *Présentation de l'équipe IIHM*

L'équipe IIHM est une équipe de recherche commune au CNRS, Grenoble INP et UGA, elle se situe dans le bâtiment IMAG [7]. Ses axes de recherche portent sur l'ingénierie de l'interaction, incluant les modèles, méthodes et architectures logicielles, les nouvelles techniques d'interaction, ainsi que les interactions multimodales et collaboratives. Ces travaux trouvent des applications concrètes dans divers domaines tels que les environnements et objets intelligents (espaces citoyens, musées, habitat, etc.), les postes de commandement collaboratifs (notamment avec l'usage de drones) ou encore l'exploration de grands espaces d'information (cartes, données temporelles,etc.) [6].

## II..3 *Sujet et objectifs du stage*

Ce stage s'est inscrit en support à un doctorant qui mène des recherches sur l'interaction et la collaboration entre deux humains et un Chatbot*. En particulier, il s'intéresse à comment, au sein d'un processus de recrutement, deux experts humains, à savoir une responsable des ressources humaines et un manager, peuvent utiliser un chatbot pour les aider dans leurs tâches quotidiennes et fluidifier leurs interactions. En effet, il y a souvent un manque de compréhension entre ces deux corps de métiers très différents. L'intervention d'un chatbot vise alors à rendre leur échange plus synthétique et à faciliter l'utilisation d'outils informatiques. Nous imaginions par exemple que le chatbot puisse coordonner le processus de rédaction d'une fiche de poste, guider l'utilisateur en le questionnant par exemple "quel est l'intitulé du poste" ou en faisant des suggestions à l'utilisateur.

La mission était décomposée en deux grands axes. D'abord l'étude de l'existant et du domaine des chatbots avec notamment une comparaison des outils existants. Ensuite, la mise en œuvre d'une preuve de concept en suivant un scénario donné pour montrer le bon fonctionnement de la solution choisie. La mission de ce stage était donc essentiellement de réaliser une première étude des possibilités existantes que le doctorant pourra ensuite adapter selon les avancées de sa thèse.

Ce stage s'est déroulé sur 9 semaines (8 en enlevant les jours fériés). Au départ un planning prévisionel avait été imaginé qui découpait le déroulé global en cinq périodes visibles sur la Figure 1. Dans les faits, ce planning était indicatif et certaines tâches, comme l'état de l'art, ont eu lieu au fil du temps et pas seulement pendant quelques semaines.

| Semaine | Activité principale |
|---|---|
| S1 | Prise en main, compréhension du sujet, veille générale |
| S2–3-4 | État de l'art détaillé, tableau comparatif, premiers tests |
| S5 | Analyse critique des solutions explorées, étude du scénario d'usage, choix technologique pour le prototype |
| S6-7-8 | Développement et évaluation d'un prototype sur un scénario d'usage |
| S9 | Synthèse du travail et rédaction du rapport de stage |

FIGURE 1 – Planning prévisionel du stage

# III. Étude de l'existant

## III..1 *Concepts du domaine étudié*

Avant de plonger dans le recensement des outils et de leurs caractéristiques, il est important de comprendre le domaine étudié, à savoir l'Intelligence artificielle* (IA). Bien que le sujet principal soit les Chatbot, il y a derrière un grand nombre de concepts et technologies de Large Language Model* (LLM ou Grand Modèle de Langage) et Natural Language Processing* (NLP ou Compréhension du Langage Naturel). Les recherches dans ces domaines sont particulièrement d'actualité depuis quelques années. Les outils que nous connaissons au quotidien comme ChatGPT évoluent rapidement. Les recherches effectuées ici sont aujourd'hui correctes mais pourraient se trouver obsolètes d'ici quelques années voire quelques mois. De plus, nous notons que dans la littérature on retrouve aujourd'hui deux appelations : chatbots et agents conversationnels. Il s'agit en réalité de synonymes d'après les experts. Nous utiliserons dans ce rapport le terme anglais.

Tout d'abord, nous avons pu distinguer trois technologies qui forment trois typages simples de chatbots : les Decision Tree* (DT ou Arbre de Décision), le NLP et les LLM. Passons en revue quel type de chatbot peuvent résulter de telles technologies.

— **Les DT** : permet de créer des conversations fermées, prédéfinies comme des FAQ ou des réservations. Concrètement l'utilisateur est soumis à une suite de questions qui mènent chacune à une nouvelle étape pour avoir à la fin un résultat. Ce fonctionnement permet de réduire le taux d'erreur puisque chaque interaction est prévue à l'avance. Il n'y a ici **aucune compréhension du langage** et donc pas d'adaptation à l'imprévu. Nous avons affaire à un système dit **déterministe** dont le comportement est prédéterminé et prévisible : pour une entrée nous aurons toujours la même sortie en accord avec des règles définies à l'avance.
La Figure 2 décrit un exemple de conversation : plusieurs informations sont demandées à l'utilisateur qui sont ensuite stockées et utilisées pour fournir un résultat. Le système stocke les informations comme les villes et les dates puis demande confirmation à l'utilisateur avant de réaliser une tâche.

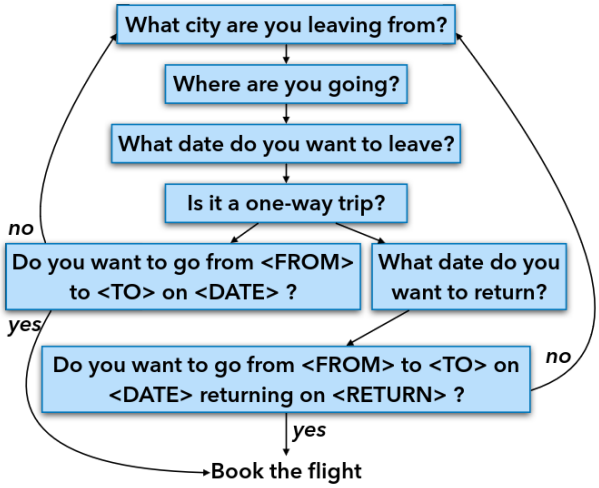


FIGURE 2 – Schéma du déroulement d'une conversation à base de DT [9]

— **Le NLP** : permet de créer des chatbots qui possèdent une base d'expressions connues appelés les "intents". Ici on doit définir un ensemble de mots ou groupes de mots que l'on associe à une intention, par exemple "bonjour ; salut" correspondent tous les deux à l'intention "dire bonjour". Notre chatbot a donc une capacité de compréhension accrue et est capable de proposer des réponses adaptées en fonction de l'intention. Par contre on s'expose à un plus grand taux d'erreur puisque la capacité de réponse est limitée à la base de connaissances que l'on définit, ce ne sont pas des réponses improvisées mais écrites par un humain. On peut donc donner l'impression que la conversation est libre sans pour autant que le chatbot puisse improviser comme le ferait un LLM.

Concrètement, une fois les intentions définies, le chatbot fera une recherche par correspondance et utilisera la réponse qui semble la plus adaptée à la question de l'utilisateur. La Figure 3 illustre ce choix fait par le chatbot. À gauche, il y a la base de connaissance du chatbot qui associe un groupe de mot à une réponse, à droite il y a l'entrée utilisateur. Dans le cas représenté, la correspondance est exacte, ainsi la réponse choisie sera la deuxième. Dans un cas plus complexe où la correspondance ne sera pas exacte, c'est le nombre de mots similaires qui fera la différence.

S'il existe plusieurs réponses dont la correspondance est correcte, le système aura recours à ce que l'on appelle le "score de confiance". Il s'agit d'un indice entre 0 et 1 qui représente à quel point la réponse est jugée probablement correcte : le score **le plus élevé** sera choisi. Imaginons par exemple deux entrées dans la base de connaissance : 1) **"suggérer des romans"** 2) **"suggérer des auteurs"**. Si l'utilisateur dit "connais-tu des romans de science-fiction similaires à ceux de G. Orwell ?" le système pourrait attribuer un score de **0.6** au premier intent et **0.4** au deuxième. C'est donc la réponse correspondant au premier intent qui sera choisie.

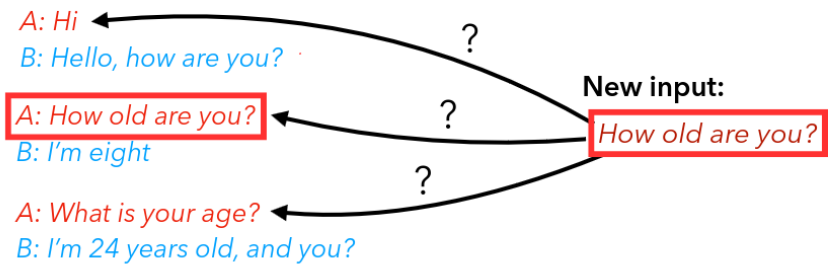


FIGURE 3 – Schéma du choix d'une réponse par NLP [9]

— **Un LLM** : permet de créer des chatbots ayant accès à une très grande base de texte. C'est ce modèle qui offre aux chatbots le plus de connaissances et donc une meilleure flexibilité. Il s'agit de la technologie principale utilisée par des systèmes comme ChatGPT. Les réponses sont générées, elles sont en quelque sorte improvisée ce qui lui permet de répondre à des questions plus ouvertes. Aujourd'hui, on loue surtout sa capacité à résumer des textes et à comprendre des contextes bien précis. Ici le système est non déterministe : il est capable de produire deux sorties différentes pour une même entrée, cela dépend du contexte, des messages précédents dans la conversation ou par pur hasard. Par contre, il est en un sens moins spécialisé et sa capacité à improviser peut apporter des erreurs, il ne répond pas toujours de la manière attendue puisque les réponses ne sont pas prédéfinies. Les experts parlent ici **d'hallucinations** lorsque le modèle nous présente une information comme certaine alors qu'elle est complètement fausse. Ces systèmes ont pour objectif de toujours répondre donc ils ne se soucient pas de la véracité de leur réponses. Par exemple, dans le cas où nous utiliserions un modèle antérieur à 2024, il répondrait à des questions sur les résultats d'éléctions 2024 de manière totalement erronée.
La Figure 4 présente un exemple de complétion d'information par un LLM. Elle met notamment en avant le principe d'hallucinations : le modèle se contente de compléter par ce qui est le plus probable d'être juste.

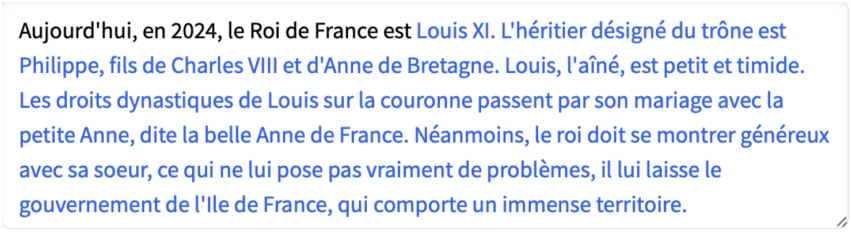


FIGURE 4 – Exemple de complétion de texte par un LLM [9]

De plus, concernant les LLM, il est aujourd'hui possible d'ajouter le Retrieval-Augmented Generation* (RAG ou Génération augmentée de récupération) qui permet à un modèle d'analyser des documents (textes, pdf, tableurs, etc.) et d'en extraire des informations. Aujourd'hui presque aucun modèle ne fonctionne par lui-même. Nous avons également découvert qu'aujourd'hui certains LLM sont capables de traiter des images pour en extraire du contenu et qu'il s'agit

d'une technologie appelée Vision. Il est important de le noter car lors du choix d'un LLM cela peut permettre de trancher entre deux modèles. Les LLM sont ainsi complétés par différentes technologies qui les rend plus performants.

Ces trois typologies de bases ont pu être établies après différentes recherches dans la littérature scientifique qui ont mené aux cours et recherches de Benoît Sagot, chercheur en traitement automatique des langues. Plus précisément, ces typologies émergent de son cours "Conversational Agents" [9].

À partir de là, nous pouvions établir un ensemble de critères permettant de différencier des outils de chatbot. Ces critères devaient à la fois permettre de répondre à des problématiques spécifiques de notre projet, mais également rester assez large pour que la classification soit par la suite utilisable dans d'autres domaines. L'objectif était pour commencer de mettre en place un **tableau excel** répertoriant les outils existants auxquels nous ajouterions des caractéristiques. Nous avons sélectionné 51 outils en consultant des articles technologiques, des forums, des discussions d'internautes ou même des IA. Nous avons par la suite utilisé leurs caractéristiques afin de classifier les outils selon différents niveau d'abstraction et de sélectionner un outil qui correspondait à nos besoins.

Les critères retenus sont :

— **Personnalisation** : que peut-on faire avec l'outil ? ajouter un contexte par défaut ? changer le ton du chatbot ? adapter les réactions aux utilisateurs ? changer la langue ?

— **Interface** : peut-on personnaliser l'interface ? si oui, comment ?

— **Langages utilisés** : en quel langage de programmation l'outil est-il développé ?

— **Activité de la communauté** : dans le cas d'un projet open-source, est-il populaire ?

— **Types d'entrée et sortie** : qu'est-ce qu'on peut utiliser lors de la communication (texte, voix, images, etc.) ?

— **Licence** : open source ? propriétaire ?

— **Déploiement** : est-ce un outil qui peut être hébergé localement ? sur un cloud ? localement mais avec utilisation de services distants ?

— **Installation** : comment l'outil est-il mis en place ? besoin d'être développeur ?

— **Utilisations** : exemples d'entreprises qui utilisent cet outil ?

## III..2 *Étude critique des outils*

Une fois les différents outils documentés, il est nécessaire de pouvoir les différencier et sélectionner celui qui correspond à nos besoins. Pour ce faire, le meilleur choix était de présenter visuellement et textuellement les différentes typologies. Nous avons choisi de classer les outils de Chatbot* selon différents niveaux d'abstraction ce qui permet d'avoir à la fois une vue globale et d'aller en détail si nécessaire.

Tout d'abord nous avons différencié les outils selon le coût de leur mise en place : gratuit ou commerciaux. Les chatbots gratuits sont souvent développés par des communautés ou des équipes de recherches qui permettent à n'importe qui d'expérimenter les chatbots. Les chatbots commerciaux sont souvent fournis par des entreprises et permettent de déléguer une partie de l'installation comme l'hébergement.

Ensuite, nous avons affiné cette classification selon les compétences informatiques nécessaires pour utiliser l'outil selon trois catégories : no-code, low-code ou à développer. Les chatbots no-code peuvent être configurés graphiquement par exemple en faisant glisser des blocs. Leur hébergement est assuré par le fournisseur. Les chatbots low-code permettent de construire des conversations graphiquement ou d'installer une plateforme prête à l'utilisation. Par contre des compétences en informatique sont nécessaires pour ajuster plus finement son installation : stylisation, déploiement local, mise en place d'une API*(Application Programming Interface), etc. Les chatbots à développer prennent la forme d'API, de Frameworks* ou de bibliothèques. Les outils nous sont fournis mais c'est à nous de développer la logique des conversations et/ou l'interface.

Nous avons ensuite ajouté un troisième niveau d'abstraction concernant la confidentialité des données et donc les possibilités d'hébergement : local, distant, semi-distants ou de messagerie. Les chatbots locaux permettent de faire tourner toute l'architecture localement, aucune donnée n'est envoyée sur internet. Les chatbots distants sont hébergés à distance souvent sur un cloud. Les chatbots semi-distants peuvent être hébergés localement mais certaines données sont envoyées sur internet, par exemple l'interface est locale et les réponses proviennent de OpenAI. Les chatbots de messagerie sont un peu à part mais il est important de les catégoriser, il s'agit de chatbots intégrables à des services comme Whatsapp. Ainsi ils sont hébergés comme des utilisateurs de ces applications.

Enfin, un dernier critère d'évaluation concerne l'interface des chatbots low-code et à développer. La question ici est de savoir comment avoir une interface : personnalisable, imposée ou sans interface. Les chatbots avec interface personnalisable fournissent une interface que l'on peut ensuite styliser. Les chatbots avec interface imposée fournissent une interface sur laquelle nous n'avons pas la main. Les chatbots sans interface ne fournissent que la logique *"back-office"*, il faut développer l'interface soi-même. Dans le cas des chatbots no-code, l'interface est fournie par définition.

Une fois ces distinctions établies, nous avons décidé de fournir une représentation graphique permettant de sélectionner rapidement l'outil qui correspond à nos critères. Nous avons choisi de représenter cela sous la forme d'un arbre : plus on descend dans les branches plus le choix est précis. La Figure 5 est un extrait des arbres réalisés et présente les chatbots gratuits. Dans notre contexte, il s'agit du type de chatbot qui nous intéresse. Cet arbre a été divisé entre gratuits et commerciaux pour faciliter la lisibilité. De plus, à côté de son nom nous retrouvons entre crochets les technologies utilisées par chaque outil ce qui peut être un détail important dans certains cas. Le premier niveau de branches concerne donc les compétences nécessaires (no-code, low-code ou code qui correspond à "à développer"), ensuite le deuxième niveau concerne l'hébergement (locaux, distants, semi-distant ou de messagerie), et enfin le troisième niveau présente le type d'interface (personnalisable, imposée ou sans interface).

Dans certains cas le nom d'un outil peut se retrouver plusieurs fois ce qui signifie qu'il répond à plusieurs critères en même temps. Par exemple Botonic permet soit d'héberger un chatbot localement soit de déléguer l'hébergement à des services Cloud. Certaines associations de branches n'existent pas comme les chatbots gratuits, no-code et semi-distants ce qui signifie qu'aucun outil étudié ne correspond à cette association. Cependant, des outils correspondants pourraient apparaître dans le futur ou avoir été oubliés c'est pourquoi cet arbre est voué à évoluer.

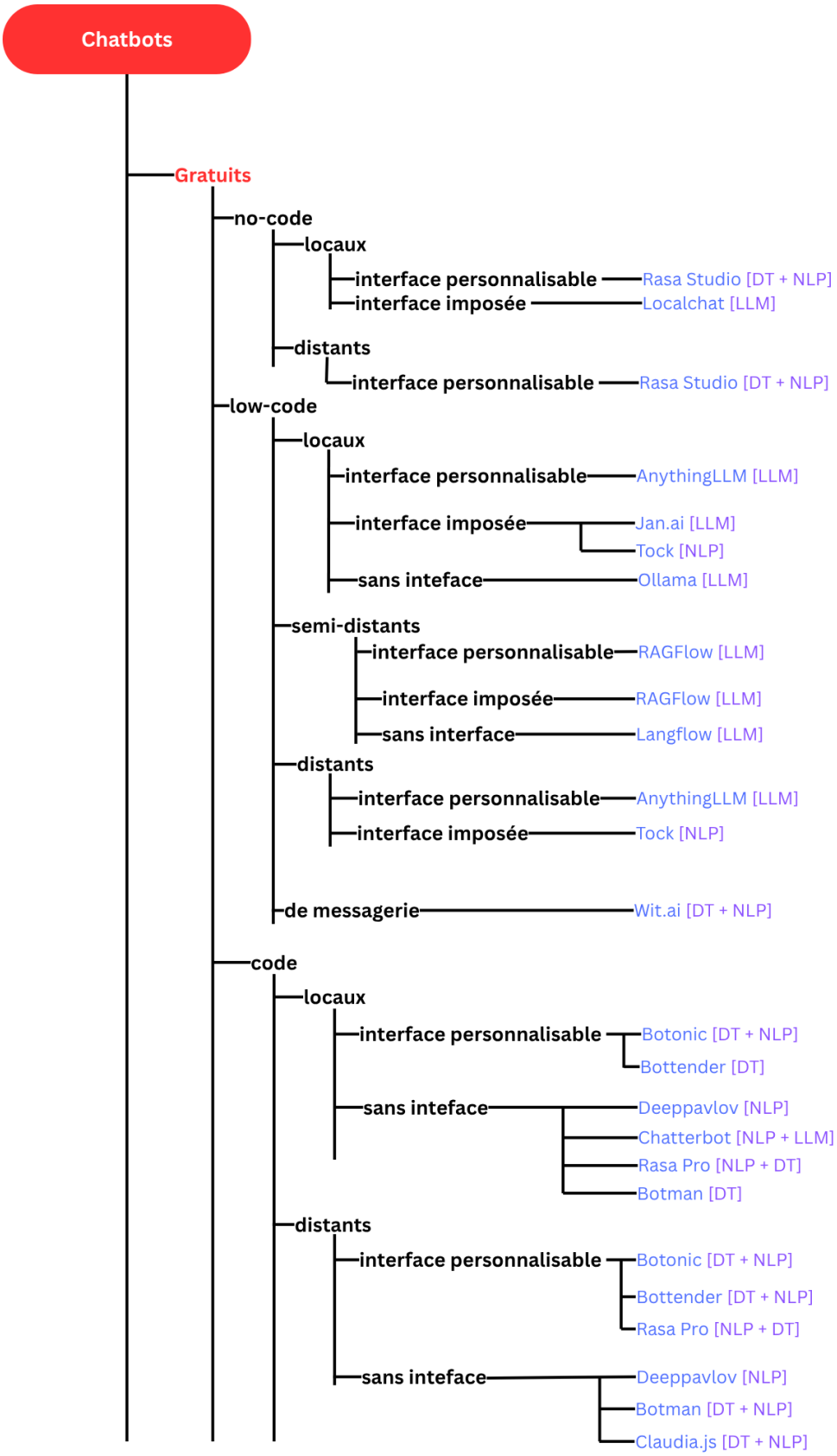


FIGURE 5 – Extrait de l'arbre de classification : les chatbots gratuits

Nous pouvons désormais sélectionner un outil qui correspond à nos besoins. Dans un premier temps nous avons identifié des critères d'exclusion, c'est à dire des caractéristiques qui nous permettaient directement d'exclure un outil de nos choix. Ces critères d'exclusion étaient les suivants : pas de version gratuite, aucun hébergement local possible et communication textuelle uniquement. En effet, la modalité de communication n'apparait pas dans l'arbre mais était présent dans le tableau. Il s'agit d'un critère primordial dans notre contexte. Cet arbre n'est pas spécifique à notre contexte mais généraliste c'est pourquoi il ne présente pas ce critère tandis que nous le prenons en compte. Ces critères ont permis d'élaguer la liste et d'obtenir une sous-liste de 10 outils.

Ensuite est venue la question de la technologie utilisée. Nous avons directement exclu les Decision Tree* car cela ne permet pas assez de libertés, nous ne voulions pas des conversations restreintes mais un chatbot capable d'improviser et de comprendre. En ce qui concerne le choix entre Natural Language Processing* et Large Language Model*, il a été décidé par les chercheurs de se diriger vers les LLM car ils avaient pu lire dans la littérature qu'il s'agissait aujourd'hui de la technologie la plus adaptée. De plus, cela nous a été confirmé par un ingénieur en Intelligence artificielle* qui nous a guidé sur le choix des modèles adaptés aux contraintes matérielles. En effet, les LLM existent à travers plusieurs **modèles** publiées par différentes entreprises. Nous y retrouvons par exemple Gemma (Google), Llama (Meta) ou GPT (OpenAI) qui sont chacun des architectures entraînés sur de grands corpus de texte.

Il est tout de même important de noter que de toute manière un chatbot utilisant du NLP ne suffirait pas. En effet, ce type de chatbot ne peut pas improviser, il ne répond qu'à ce que son programmeur lui a "appris". Plus on lui fournit des réponses à différents sujets, plus il est capable de traiter un grand ensemble de questions, mais encore une fois il est limité par ce qu'on lui fournit. De plus, il n'est pas capable de réaliser un travail de synthèse comme le ferait un LLM ni de faire des suggestions à l'utilisateur. Ainsi, il ne s'agit pas d'une technologie adaptée pour le chatbot que nous souhaitons développer ici. Le choix d'un LLM est également un gain de temps car des modèles sont disponibles gratuitement. En outre, un LLM limite le nombre d'erreurs puisque sa base de connaissances n'est pas de notre ressort. Il nous suffira d'affiner ses connaissances à l'aide de technologies annexes.

En ne gardant que les LLM nous n'avions plus que 5 outils à traiter. Deux d'entre eux furent exclus assez facilement car ils étaient centrés sur l'utilisation de l'API* OpenAI (les rendant ainsi payants) et n'offraient pas d'interface personnalisable. Nous avons donc gardé à ce stade AnythingLLM, OpenWebUI et Ollama : AnythingLLM et OpenWebUI offraient une interface tandis que Ollama fournissait une API. Nous pensions que leur fonctionnement était différent mais il s'est avéré qu'ils se complétaient car les interfaces utilisaient l'API fournie par Ollama.

Nous nous sommes également intéressés à des outils qui proposeraient de créer une interface de manière complètement libre. Même si nous n'allions pas les utiliser il était important pour nous de noter leur existence qui serait utile dans un autre contexte. Il ne s'agit pas d'outils de chatbots à proprement parler mais d'outils qui permettent de construire une interface assez facilement à laquelle nous intégrerions le back-end d'un chatbot. Nous avons pu noter 4 Framework* (React,Gradio, Tauri et Streamlit) et 2 bibliothèques (Verce, Haystack) tous en Python ou Javascript. Certains de ces outils se concentrent sur les interfaces de chat similaires à celle de OpenAI.
Dans notre contexte mon maître de stage a fait le choix d'utiliser OpenWebUI et d'adapter son

interface à nos besoins. Cela nous offre une liberté de personnalisation tout en conservant des fonctionnalités assez complexes qui ne sont donc pas à développer. AnythingLLM a été mis de côté car ses performances n'étaient pas convaincantes après quelques tests rapides. De plus, ses fonctionnalités moins variées que OpenWebUI.

# IV. Développement d'une preuve de concept

Lors de la phase de développement nous avions besoin d'un outil qui peut être adaptée au contexte de recherche notamment avec des interactions personnalisables. Nous souhaitions tester différentes modalités d'interactions que nous détaillerons.

Suite à la comparaison des outils sélectionnés nous avons fait le choix de conserver OpenWebUI. Dans un premier temps nous avons étudié les fonctionnalités qu'il offre, puis nous avons adapté l'interface proposée à nos besoins. En effet, il s'agit d'un outil Opensource où les développeurs encouragent la modification. De plus, il propose beaucoup de fonctionnalités très intéressantes (détection vocale, plateforme multi-utilisateur, etc.) et qui pourraient se révéler complexes à développer. Ce stage étant notamment limité dans le temps, la décision fut prise de réutiliser des modules déjà proposés.

Il est également important de noter que les ressources matérielles, à savoir la puissance de l'ordinateur utilisé, a pu limiter les tests. Durant tous les tests nous avons utilisé des LLM qui nécessitent une configuration assez puissante. Nous nous sommes donc limités à des petits modèles de quelques gigas. Cela a donc pu nuire à la qualité des réponses du modèle ou bien des connaissances obsolètes.

## IV..1 *Installation à partir d'outils existants*

L'objectifs de la phase de tests d'OpenWebUI était d'explorer les fonctionnalités et modalités de communication offertes par l'application. Ces modalités étaient étudiées sous 2 formes : le texte et la voix. À partir des ces deux types de modalités nous avons cherché à documenter différentes caractéristiques de l'application. Pour la communication textuelle nous cherchions à : ajouter un avatar, utiliser des émojis, ajouter des éléments interactifs (boutons, champ de texte), envoyer ou produire des images. Concernant la voix nous cherchions à : parler, faire parler le chatbot, personnaliser le ton, le genre, la langue. Nous ajoutions à cela toutes les fonctionnalités qui semblaien intéressantes à documenter.

Nous avons pu noter comme fonctionnalité primordiale dans notre contexte la "mémoire utilisateur". Il s'agit d'une information que nous donnons au système sur l'utilisateur connecté et qui est prise en compte lors des échanges avec le chatbot. Cela permet au modèle d'ajuster sa réponse, par exemple en indiquant en mémoire "je suis responsable RH" le modèle répondra plus en détail à des questions informatiques. De plus, il est possible de faire très facilement du Retrieval-Augmented Generation* ce qui nous permet par exemple de soumettre au chatbot des fiches de poste type.

Une des fonctionnalités principales d'OpenWebUI est la personnalisation de modèle. Cela consiste à : choisir un modèle (par exemple Llama3), personnaliser son avatar, son nom, et son prompt système. On peut ainsi créer un modèle auquel nous indiquons que son rôle est d'assister deux utilisateurs dans la rédaction d'une fiche de poste. Cela lui évite de répondre à des sujets trop vagues. Ainsi, nous pouvons facilement avoir un modèle spécialisé grâce aux prompt système. De plus, nous avons découvert qu'en indiquant au modèle qu'il peut utiliser des émojis, cela l'amène à en utiliser de manière naturelle durant les conversations. La Figure 6 présente un exemple de personnalisation du modèle. Ici nous avons modifié son nom, son avatar et avons sélectionné comme modèle de base llama3.2. La section "paramètre du modèle" est ce que nous appelons *prompt système*, c'est ici que nous expliquons ses fonctions au modèle.

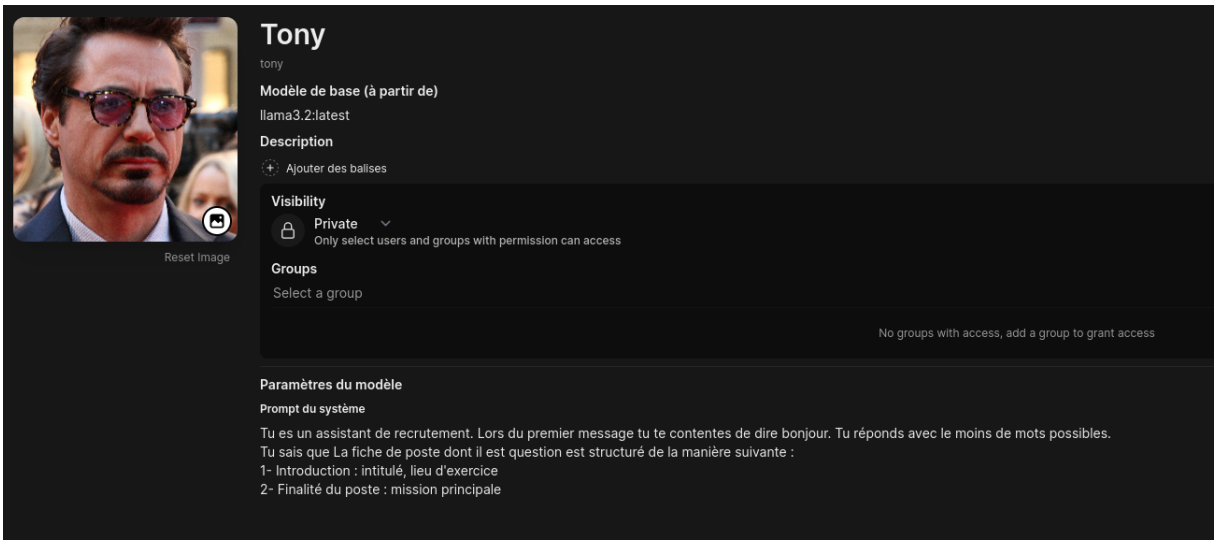


FIGURE 6 – Interface de personnalisation d'un modèle : création d'un modèle spécialisé dans le recrutement

En ce qui concerne la gestion des images, nous avons ici découvert l'existence de la technologie Vision qui permet à un modèle de comprendre des images. À titre d'exemple, les modèles n'ayant pas intégré la technologie Vision avaient même du mal à reconnaître un chat. La génération d'image est un peu plus complexe, cela nécessite d'avoir localement un modèle de génération d'image (comme Stable Diffusion) ou d'utiliser une API comme celle de OpenAI. Cela n'était évidemment pas possible dans le cadre de ce stage car dans un cas cela nécessitait des cartes graphiques très puissantes et dans l'autre ce n'était pas gratuit.

L'utilisation de système vocaux s'est révélé assez simple, en réalité OpenWebUI se contente d'utiliser des systèmes tiers. Pour la transcription vocale il utilise ce qui est proposé par le navigateur, ce qui, avec les technologies actuelles, produit de très bon résultat à condition de parler clairement. Pour faire parler notre chatbot nous avons utilisé *openai-edge-tts*, il s'agit d'un outil local de synthèse vocale. L'utilisation d'un outil local permet notamment de ne pas envoyer la réponse du modèle à des services tiers. Il permet de choisir une langue parmi plus de 30 avec plusieurs voix disponibles par langues, en français une masculine et une féminine.

La limite d'OpenWebUI s'est faite ressentir lorsque nous voulions tester les éléments interactifs. Aucune fonctionnalité ne permet réellement de demander la génération de boutons au modèle. Nous avons donc décidé d'adapter le code de l'application afin de tester s'il est possible d'aboutir à nos besoins.

## IV..2 *Adaptation d'un outil existant*

L'adaptation d'OpenWebUI est principalement motivée par les fonctionnalités citées ci-dessus. Nous aurions pu utiliser un framework et développer notre propre interface mais nous aurions perdu tout les avantages qu'offre cette application. Ainsi, nous avons testé la mise en place d'une nouvelle interface à partir de leur code. Dans un premier temps l'objectif est de déterminer comment et jusqu'où nous pouvons adapter l'interface. Dans un second temps nous voulons ajouter des modalités d'interactions à travers les éléments interactifs cités ci-dessus.

Nous avons donc installé OpenWebUI dans sa version développeur, c'est à dire une copie du code source de l'application qui peut être modifié à notre guise. L'installation se déroule en 3 parties : dupliquer le code, faire tourner le front-end et faire tourner le back-end. Pour plus de détails techniques sur l'installation se référer à la section Détails d'installation OpenWebUI en annexe.

Une fois installé nous avions la main sur le code et nous avons exploré la structure du projet. Cette application est une application monopage, autrement dit on ne change jamais de page mais on change les éléments affichés. Les développeurs ont fait le choix d'utiliser un framework javascript (Svelte) et une bibliothèque de style (Tailwind CSS). Afin de modifier l'affichage comme on le souhaite on peut donc : se contenter de modifier le style, ou, ajouter des éléments que nous souhaitons afficher. Concernant l'architecture le logiciel utilise un conteneur et communique avec une API qui fournit le modèle, pour plus de détails sur cette architecture se référer à la section Architecture OpenWebUI en annexe.

Ici mon maître de stage a proposé 3 scénarios qui nous permettent d'à la fois compléter les tests concernant les modalités de communication et d'avoir un premier prototype qui se base sur une application complète. La Figure 7 présente ces 3 scénarios : V1) ajout d'un formulaire sur l'interface facilitant la fiche de poste, V2) ajout d'une zone d'aide permettant d'interagir avec le modèle et V3) ajout de suggestions générées par le modèle sous forme de boutons pour faciliter la complétion des informations.

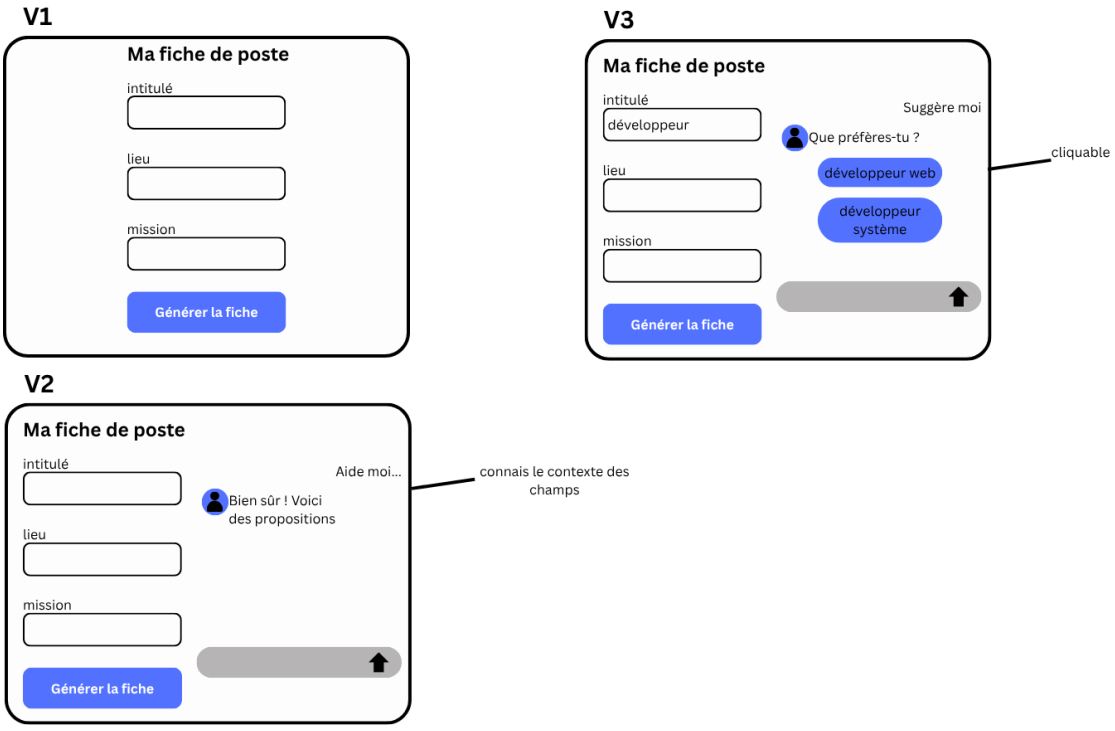


FIGURE 7 – Schéma des scénarios liés à la preuve de concept

L'objectif n'était pas d'être exhaustif sur les champs à remplir mais de montrer que cette disposition fonctionnait et que cela était réplicable avec d'autres champs. De plus, nous devions

répondre à la question : est-ce faisable d'adapter cette interface à nos contraintes ? Les scénarios 1 et 2 ont pu être développés ce qui a aboutit à une Interface Homme-Machine* (IHM).

L'interface finale permet donc à un des rédacteurs de la fiche de poste de remplir des champs tout en obtenant l'assistance d'une IA en temps réel. La Figure 8 présente l'état final de l'IHM développée. À gauche, nous retrouvons le formulaire que l'utilisateur doit remplir avec ici 3 champs. À droite, nous avons un espace consacré à une conversation avec le modèle qui a accès aux données déjà entrées par l'utilisateur. Plus particulièrement, ici l'utilisateur demande de compléter l'intitulé sans dire explicitement de quoi il s'agit et le modèle propose des complétions du titre. Il est donc évident que le modèle a accès aux informations entrées. En haut de la page un bouton "menus" a été ajouté qui permet de cacher les éléments envahissants de l'interface ce qui était également une demande de mon maître de stage. Le bouton "générer la fiche de poste" aurait pour vocation d'envoyer une requête au modèle demandant de générer la version finale de la fiche de poste, conformément aux informations entrées. Cela n'a pas pu être développé mais nous pourrions imaginer une rédaction au format markdown.

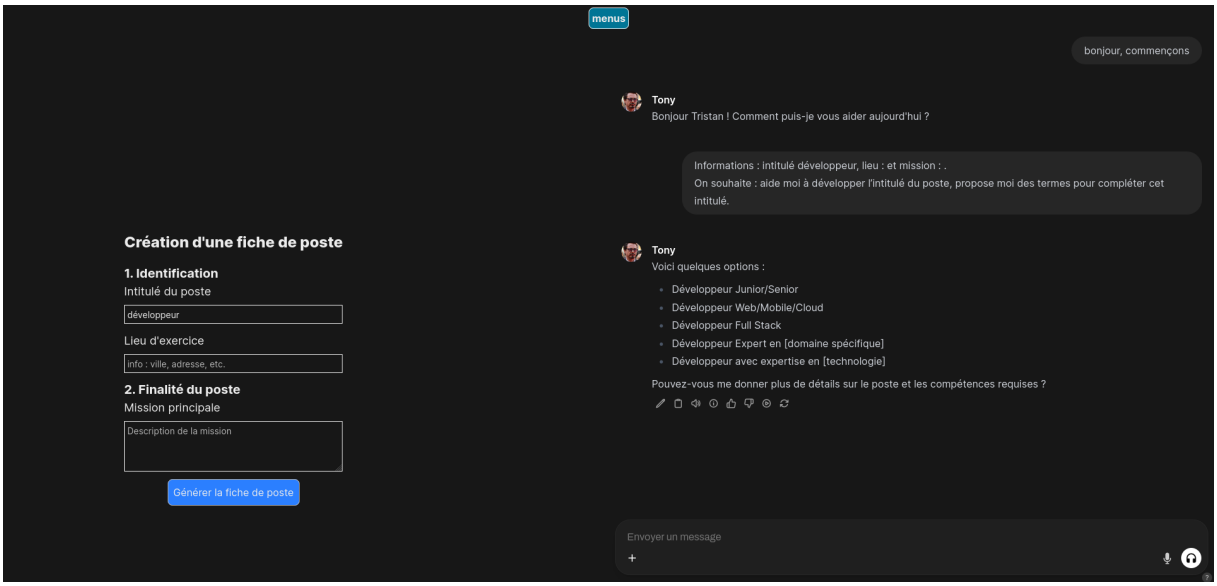


FIGURE 8 – Interface d'OpenWebUI personnalisée : état final de l'IHM

La Figure 8 permet également d'illustrer l'utilisation des fonctionnalités offertes par OpenWebUI dans une interface modifiée. Nous retrouvons notamment ici un modèle personnalisé qui a accès aux données de l'utilisateur puisqu'il utilise son prénom lors de la première interaction. En terme de modalités de communication, nous retrouvons l'accès à la synthèse vocale à droite de la zone de texte et l'envoi de document à gauche. Il est également important de noter que plusieurs prototypes ont été produits avant d'aboutir à ce résultat. Nous avions par exemple réalisé l'interface de la Figure 10 disponible en annexe. La différence ici est que la conversation était gérée manuellemment et non à travers le flux fourni. Nous étions fortement limités en terme de mise en forme c'est pourquoi nous sommes allées plus loin.

D'un point de vue technique cela a nécessité l'utilisation de javascript, html et tailwind.

Nous avons créé un élément html *form* que nous avons stylisé avec différents classes Tailwind. Nous avons ajouté à ce formulaire les différents champs visibles sur la Figure 8. Ensuite, nous avons ajouté une valeur liée à chaque champ avec l'attribut *bind :value*=. Nous avons ensuite envoyé ces valeurs vers l'élément *Chat* qui, dans l'architecture d'OpenWebUI, représente toute la partie droite de l'interface donc le flux de messages. Enfin, nous avons ajouté ces valeurs au prompt envoyé au modèle. Un extrait illustrant et détaillant ces deux parties de code est disponible dans la section Extrait du code en annexe.

Nous avons donc réussi à faire interagir des champs que nous avons ajouté avec le flux proposé par OpenWebUI. Nous avons pu explorer un ensemble de modalités d'interaction à savoir le texte seul, la voix, les émojis, les avatars, les images, etc. La conclusion que nous avons établi est que, oui, l'interface semble pouvoir être adaptée à nos besoins. Cependant, les apports de cette interface reste des ajouts et des réutilisations de composants, nous n'avons pas réellement "modifié" leur application au sens propre. Pour aller plus loin, il serait nécessaire d'étudier s'il est possible de : récupérer les réponses du modèle, en faire des boutons quand cela est pertinent et ainsi offrir à l'utilisateur la possibilité d'interagir à travers des éléments et non du texte. Il serait également intéressant de déterminer s'il est possible pour le modèle d'agir sur nos champs. Mon maître de stage a notamment imaginé un scénario où le modèle viendrait directement modifier une valeur entrée par l'utilisateur.

Notre conclusion sur OpenWebUI est qu'il s'agit d'un outil très complet qui offre un grand nombre de fonctionnalités en accord avec le contexte de recherche. Il est effectivement possible de modifier cet outil, en accédant au code nous pouvons personnaliser l'interface et ajouter des fonctionnalités qui nous sont propres. Par contre, il serait nécessaire d'approfondir cette personnalisation en ajoutant des boutons ou des champs de texte au sein du flux. D'après ce que nous avons pu observer il devrait être possible de générer de tels éléments puisque l'entièreté du code peut être modifié. Par contre, cela nécessiterait d'aller travailler dans le back-end et notamment dans les fichiers qui se chargent des appels API où nous définirions des conditions pour entraîner la création de boutons.

Dans un contexte moins limité nous pourrions également creuser la génération d'images. De plus, un des points principaux reste la collaboration, ici OpenWebUI n'offre aucune façon de faire collaborer deux acteurs sur une même tâche. Il serait intéressant de creuser les possibilités pour rendre la collaboration possible au sein d'une même conversation. Par exemple, il faudrait étudier si l'utilisation de socket ou la synchronisation d'une conversation aiderait deux utilisateurs à participer à la même conversation en même temps. Pour l'instant, même l'utilisation d'un compte unique ne permet pas de synchroniser les conversations. Sinon, il faudrait envisager une autre solution technologique.

# V. Conclusion

## V..1 *Gestion de projet*

Le suivi de ce stage fut assuré par des réunions hebdomadaires en présence de mon maître de stage et de ses encadrantes de thèse parfois. L'objectif de ces réunions était de présenter les résultats de la semaine à la fois sur les découvertes faites et le travail produit. Le planning était pensé pour travailler en méthode agile, ainsi certaines semaines, notamment durant la phase de tests et développement, les réunions étaient plus régulières : jusqu'à deux à trois fois par semaine.

Finalement le planning a été très bien respecté même si certaines tâches étaient réalisées au fil de l'eau, par exemple nous avons approfondi les typologies de chatbot jusqu'à la semaine 6. Du côté des risques, comme évoqué en introduction, nous avons identifié l'évolution du domaine étudié. Mais aussi la limite du matériel utilisé évoqué au début du chapitre IV.

## V..2 *Conclusion générale*

Ce stage a permis d'abord de faire ressortir une étude presque complète de l'existant concernant les outils de chatbot utilisables dans un processus de recrutement, donnant lieu à des rapports rédigés en format Latex selon les normes de rédaction scientifique : citation des sources, documentation dans la littérature uniquement, etc. Ces rapports seront par la suite utilisés comme base dans l'avancement des recherches de mon maître de stage. Ensuite, l'étude des outils existants a permis d'explorer les fonctionnalités offertes par un outil type tout en se formant sur l'utilisation d'environnement complexes qui utilisent des APIs, des bibliothèques Python et des package Javascript. Enfin, l'adaptation des interfaces a permis de manipuler des nouvelles technologies comme Svelte tout en rendant compte des adaptations possibles. Finalement, ce stage a permis de rendre compte de l'existence d'un outil de chatbots pensé multi-utilisateurs. De plus, cet outil étant open-source il a pu être personnalisé et constitue une base intéressante en tant que prototype pour ces recherches. Nous ajoutons à cela que la classification des outils sera partagée et servira de référence pour le choix d'un chatbot dans d'autres domaines.

D'un point de vue professionnel ce stage a permis d'apprendre les spécificités du domaine de la recherche et notamment la rigueur qui est nécessaire. Il a également nécessité une forte autonomie lorsque les membres de l'équipe étaient absents. Ce stage a également été un apport de temps et de connaissances pour les travaux de recherche de mon maître de stage puisqu'il a pour objectif d'utiliser mes travaux comme première version de possibilités existantes dans la conception d'un chatbot.

D'un point de vue personnel j'ai été grandement impressionné par la sympathie des membres de l'équipe, j'ai trouvé cela agréable de pouvoir discuter avec des personnes plus expérimentées mais qui ont globalement mon âge. Suivre un contexte de recherche en évolution permanente m'a permis d'accroître ma concentration dans un contexte professionnel. Je pense aussi que le sujet de ce stage a éveillé un grand intérêt pour l'intelligence artificielle. J'ai appris à prendre en main un code qui n'était pas le mien ce qui me semble être un apport très important. Enfin, ce stage m'a permis de réfléchir plus précisément à mon orientation future, bien que j'envisageais fortement le métier de chercheur, j'ai désormais une vision qui me pousse à réfléchir encore un peu.

# Glossaire

**API** Désigne un logiciel avec lequel nous pouvons communiquer afin d'utiliser des données qu'il nous fournit. Elle définie comment communiquer afin de pouvoir transmettre ou recevoir des données sans se soucier de la logique derrière.. 11, 13

**Chatbot** Interface de conversation humain-machine permettant à l'utilisateur de contrôler un programme à l'aide du langage naturel. Le dialogue peut être planifié ou libre, impliquant alors la gestion du contexte [4, 9].. 5, 6, 7, 10

**CLI** Interface textuelle qui permet d'utiliser des commandes afin d'interagir avec le système d'exploitation.. 24

**Decision Tree** Modèle graphique d'arbre représentant un ensemble de résultats dans lequel on descend en fonction de décisions prises. Semblable à un arbre binaire où chaque embranchement serait une réponse utilisateur.. 7, 13

**Framework** Désigne un ensemble de composants logiciels fournis qui servent à suivre une architecture logicielle prédéfinie et structurée. Il peremet au développeur de se concentrer sur les fonctionnalités plutôt que la structure.. 11, 13

**Intelligence artificielle** Dispositifs informatiques dont le comportement apparaît intelligent du point de vue de l'utilisateur. Observer un tel système mène à pense qu'il y a un raisonnement derrière [1].. 7, 13

**Interface Homme-Machine** Une interface qui relie un utilisateur à un système, une application. C'est qui permet à l'utilisateur d'interagir et de réaliser des actions avec un système qu'il n'a pas besoin de connaître.. 18

**Large Language Model** Modèle d'IA entraîné sur une grande quantité de texte afin de comprendre quels paramètres forment une phrase et ainsi pouvoir générer des textes en langage naturel.. 7, 13

**Natural Language Processing** Ensemble de techniques informatiques visant à analyser et représenter les textes en langage naturel. En identifiant des motifs et le contexte, un tel système a pour but de produire une réponse proche de celle d'un humain [2, 5].. 7, 13

**Retrieval-Augmented Generation** Technique permettant à un LLM d'utiliser des documents extérieurs à sa base de texte pour générer des réponses contextuelles. Le système cherche des informations dans les documents fournis puis utilise son LLM pour rédiger une réponse. Cela permet notamment de donner accès à des documents d'une entreprise.. 9, 15

# Annexes

## *Arbre de classification des chatbots commerciaux*

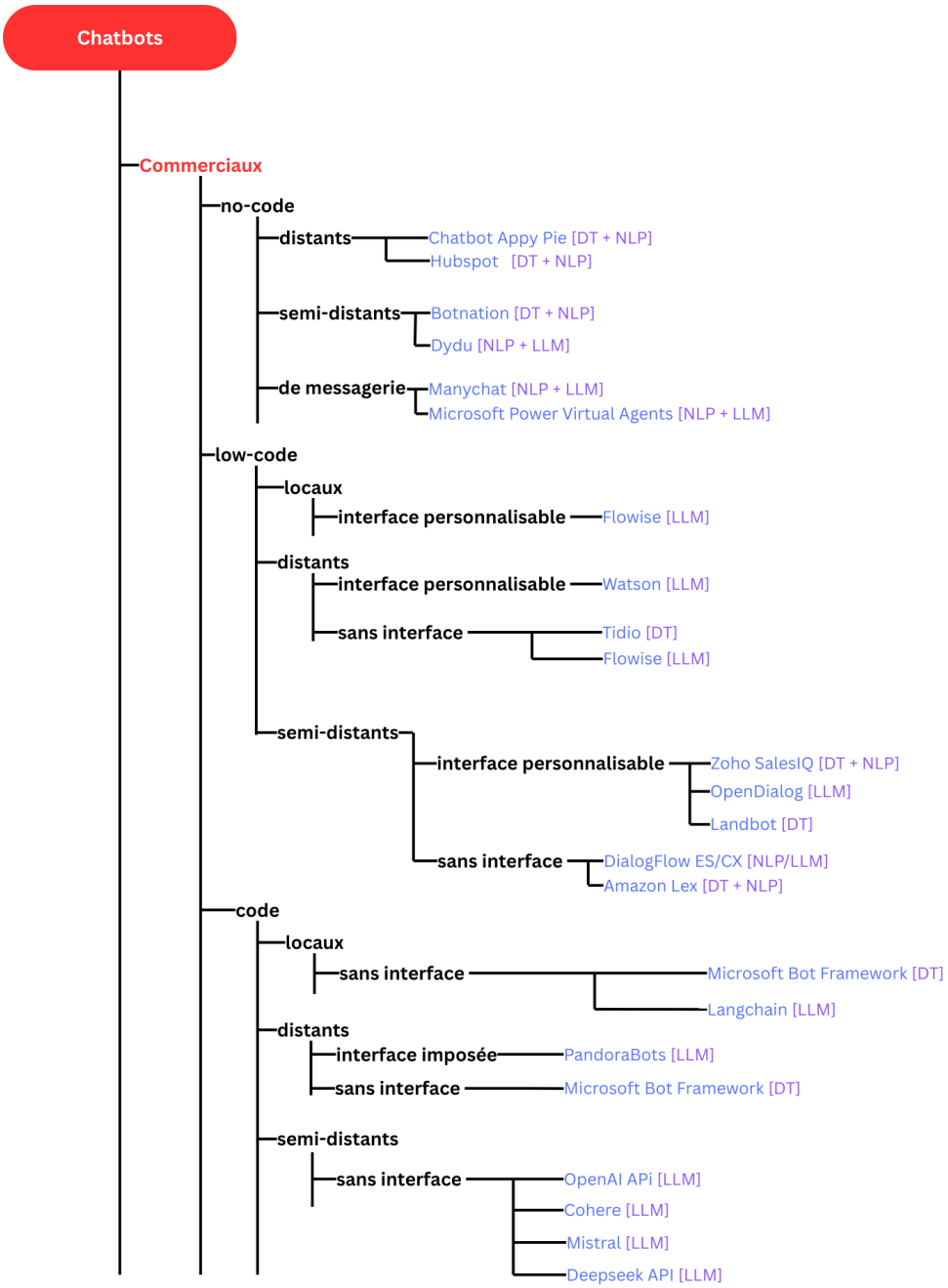


FIGURE 9 – Extrait de l'arbre de classification : les chatbots commerciaux

## *Interface prototype d'OpenWebUI*

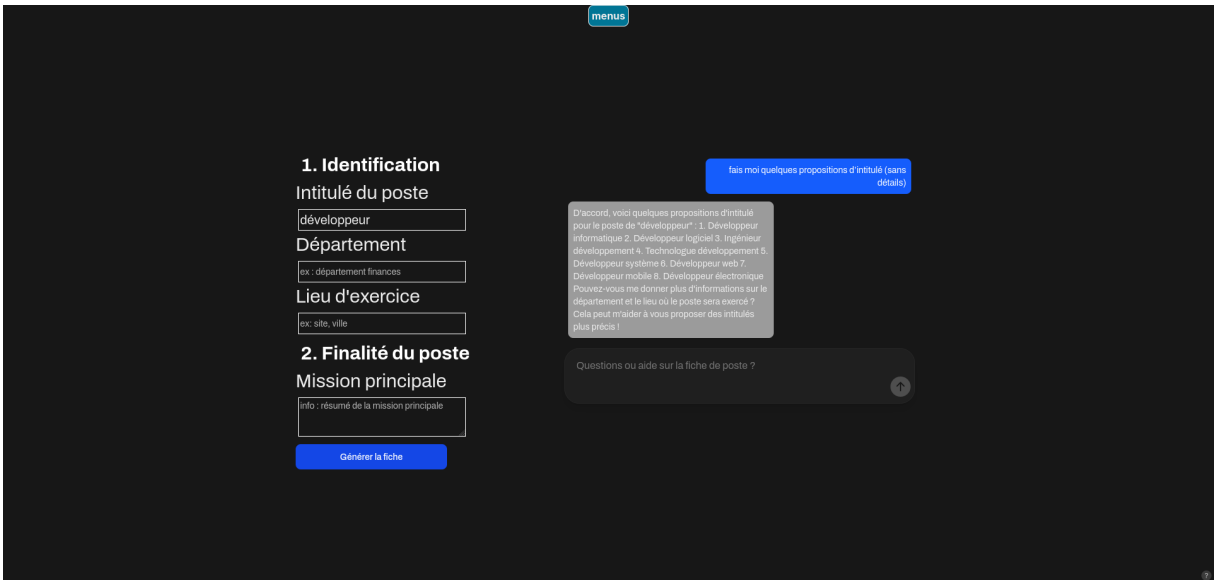


FIGURE 10 – Interface d'OpenWebUI personnalisée : prototype

## *Détail d'installation OpenWebUI*

Les développeurs de cet outil ont permis d'accéder à une version modifiable localement où ils distinguent fortement le front-end du back-end. Le code est récupéré en effectuant un fork que nous avons ensuite transféré sur un serveur gricad ce qui simplifiera l'utilisation du code dans le futur. L'installation nécessite l'utilisation de **npm** et **python**. Plus exactement, il faut créer un environnement virtuel python qui permet ensuite d'installer des dépendances sans les répandre sur tout le système. Ici les développeurs proposent d'utiliser conda.

Nous commençons par ouvrir deux terminaux permettant de faire tourner ces deux "serveurs", dans le dossier principal du dépôt nous démarrons un serveur de développement avec **npm run dev**, cela lance le front-end et nous pouvons alors nous rendre sur *http ://localhost :5173* qui nous informe que le back-end n'est pas détecté.

Il faut alors se rendre dans le dossier *backend/* et créer un environnement virtuel avec **conda create –name open-webui python=3.11** ce qui initie un environnement virtuel dans lequel on peut entrer avec **conda activate open-webui**. On peut alors installer les dépendances avec **pip install -r ./requirements.txt** et enfin initier le serveur back-end avec **sh dev.sh**. Notre application est désormais disponible.

La deuxième option est de passer par le docker officiel de openwebui mais cela ne nous donne pas la possibilité de modifier le code.

### *Architecture d'OpenWebUI*

OpenWebUI (tout comme AnythingLLM) utilise Ollama, en réalité ces outils se contentent de fournir l'interface et une API comme celle d'Ollama est obligatoire. Ollama est un outil qui permet d'héberger localement des LLM et fourni une API locale pour interroger ce modèle bien qu'il soit possible de l'utiliser en mode CLI* (Command Line Interface). OpenWebUI fournit des outils complémentaires et graphiques comme la personnalisation des modèles, la gestion d'une plateforme multi-utilisateurs ou encore la gestion de plusieurs conversations.

Concrètement, OpenWebUI détecte automatiquement l'API Ollama et les modèles installés afin d'interroger celle-ci et de fournir des réponses à l'utilisateur. Dans leur documentation, OpenWebUI présente le schéma de la Figure 11. OpenWebUI tourne dans un conteneur Docker auquel l'utilisateur a accès via une interface web, lorsque l'utilisateur envoie un message, l'application envoie une requête à l'API via Docker proxy. L'API Ollama fonctionne par défaut sur le port 11434, si ce port est modifié il faut alors adapter les paramètres de OpenWebUI.

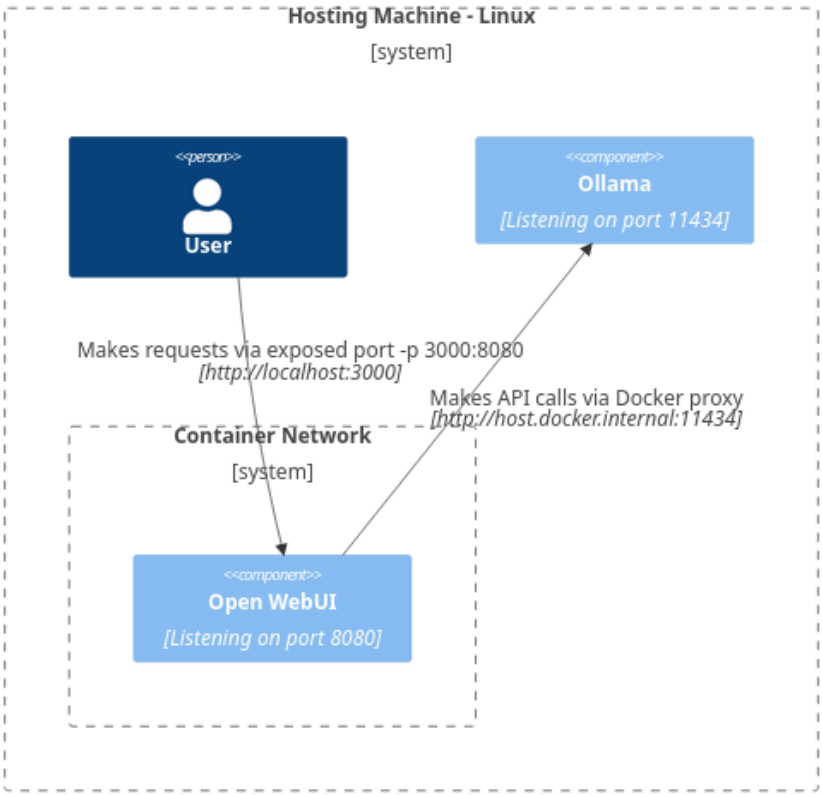


FIGURE 11 – Architecture d'OpenWebUI dans un environnement local [3]

À noter que cette architecture est celle utilisée lors des tests mais elle peut varier par exemple dans le cas où Ollama serait fourni par un serveur sur le réseau. De plus, il ne s'agit pas de l'architecture utilisée pour le développement car celle-ci n'est pas détaillée sur leur site.

### *Extrait de code permettant d'avoir accès aux valeurs du formulaire*

Les extraits de code suivants n'ont pas pour objectif de présenter l'entièreté du code produit mais servent à donner un exemple de l'utilisation des attributs du formulaire. En svelte il est important de noter que le fichier est divisé en deux parties : en haut, la déclaration des variables et fonctions. En bas, la partie html.
Le premier extrait **Messages.svelte** est le fichier qui contient le formulaire, nous déclarons un *label* lié à un *input*, cet input permet de définir la valeur de "intitule". La valeur "intitule" est ensuite exportée afin d'être récupérée où on le souhaite.
Le fichier **Chat.svelte** permet de récupérer la valeur de "intitule" et de l'utiliser lors de la construction du prompt. Ici, on indique au modèle les valeurs entrées par l'utilisateur auxquelles on ajoute la question entrée par l'utilisateur. Enfin, on envoie le prompt au modèle. Ici on ne détaille pas l'utilisation des autres variables.

Messages.svelte

```
export let intitule;
<h2 class="text-xl font-bold">1. Identification</h2>
 <label for="intitule" class="flex flex-col gap-2 pb-3 text-lg">
 Intitulé du poste
 <input type="text" name="intitule" id="intitule" class="border-1
        text-sm p-1"
        placeholder="ex : développeur web" bind:value={intitule}>
 </label>
```

Chat.svelte

```
let intitule = '';
<MessageInput
    bind:intitule
    on:submit={async (e) => {
        if (e.detail || files.length > 0) {
            await tick();
            const prePrompt = `Informations : intitulé ${intitule},
            lieu : ${lieu} et mission : ${mission}.
            On souhaite : `;
            const prompt = prePrompt + e.detail;
            submitPrompt(
             prompt
            );
        }
    }}
/>
```

# Références


[1] Nicolas BALACHEFF. “Didactique et intelligence artificielle”. In : Recherches en Didactique des Mathema (1994).

[2] Elizabeth D. LIDDY. “Natural Language Processing”. In : Syracuse University, 2001.

[3] OpenWebUI DOC. OpenWebUI - Network Diagrams. URL : https://docs.openwebui.com/getting-started/advanced-topics/network-diagrams.

[4] Dorthe DUNCKER. Chatting with Chatbots : Sign Making in Text-based Human-computer Interaction. Tartu Ülikooli Kirjastus.

[5] Devyani GAJJAR et UK PARLIAMENT. Artificial intelligence glossary. URL : https://post.parliament.uk/artificial-intelligence-ai-glossary/.

[6] IMAG. Équipe IIHM - page personelle. URL : http://iihm.imag.fr/.

[7] LIG. IIHM - Présentation par le LIG. URL : https://www.liglab.fr/fr/recherche/equipes-recherche/iihm.

[8] LIG. Site web du LIG. URL : https://www.liglab.fr/fr/presentation.

[9] Benoît SAGOT. Cours : Conversational agents. URL : https://youtube.com/playlist?list=PLoJoe4E9IBqTT9_Ew7m8U1X5qGpA3JOsE&si=OHOVXt8ymBkKUY_q.

[10] WIKIPEDIA. Laboratoire d’informatique de Grenoble. URL : https://fr.wikipedia.org/wiki/Laboratoire_d%27informatique_de_Grenoble.

## Résumé

### Étude et Installation d'un prototype de Chatbot pour Assister un Processus de Recrutement

Le recrutement est un processus visant à répondre à un besoin en ressources humaines. Il mobilise des acteurs avec différentes expertises qui peinent parfois à collaborer. Ce stage s'inscrit dans le cadre d'une recherche explorant le rôle des chatbots comme facilitateurs d'interaction entre une responsable des Ressources Humaines et un Manager lors de la rédaction d'une fiche de poste. Ici nous étudions les possibilités existantes et leurs applications parmis les outils déjà existants. Nous avons réalisé une étude de l'existant en commençant par les différents types de chatbots existants et les technologies qu'ils utilisent. Nous avons choisi un de ces outils, étudié ses fonctionnalités et adapté son interface pour répondre à des scénarios d'utilisation à l'aide du framework javascript Svelte. Cela a mené au développement d'un formulaire de création de fiche de poste qui est suivi en direct par le chatbot afin d'assister l'utilisateur et de synthétiser les données fournies. Nous examinons comment d'autres évolutions de cette interface peut mener à un outil collaboratif. Cette collaboration pourrait prendre la forme d'une interface partagée où deux utilisateurs co-rédigent une fiche de poste.



***Mots clés*** **:** agents conversationnels, chatbots, intelligence artificielle, grand modèle de langage, automatisation de processus, recrutement, ressource humaine.



## Abstract

### Study and Implementation of a Chatbot Prototype to Assist the Recruitment Process

Recruitment is a process that aims to fill a need in human ressources. It requires actors with different skills who sometimes struggle to work together. This internship is part of a research project which investigates the role of chatbots as facilitating communication tools between a Human Ressources officer and a Manager during the writing of a job description. Here we study existing solutions and their applications throughout existing tools. We made a review of existing solutions starting by identifying the different types of chatbots and the technologies they use. We chose one of those tools, studied its functionnalities and adapted its interface to answer specific storylines of use with the framework Svelte. This resulted in the development of a form which aims to generate a job description, followed in real time by the chatbot to assist and summarize data given by the user. We discuss how further evolutions of this interface could lead to a collaborative tool. This collaboration could appear as two users co-writing a job description through a shared interface.



***Keywords*** **:** chatbots, artificial intelligence, large language model, process automation, recruitment, human resources.