Utilisation d'un LLM pour le couplage faible de services web

Séjourné Kevin[1] Lata Alexandru[1]
(1) Cloud Temple, Le Belvédère, 1-7 Cours Valmy SPACES, 92800 Puteaux, France
kevin.sejourne@cloud-temple.com, alexandru.lata@cloud-temple.com

RÉSUMÉ

Cet article décrit l'utilisation d'un grand modèle de langage (LLM) pour faciliter la maintenance des systèmes d'information (SI). Il présente l'approche adoptée pour interfacer un outil de gestion de procédures dynamiques avec un LLM, en vue de résoudre le problème d'appel d'API soumis aux contraintes de gestion des SI. Les expériences ont montré que l'utilisation du LLM peut réduire l'interdépendance entre les actions et les services web (WS).

ABSTRACT

This article describes the use of a large language model (LLM) to facilitate the maintenance of information systems (IS). It presents the approach used to interface a dynamic procedure management tool with an LLM to solve the problem of calling APIs with the constraints of IS management. The experiments have shown that using the LLM can reduce the coupling between actions and web services (WS).

MOTS-CLÉS :.

KEYWORDS: .

LLM, WebService, RAG, Couplage faible. LLM, WebService, RAG, Loosely coupling.

1 Introduction

Nous voulons un système capable de nous aider dans le maintien en condition opérationnelle des systèmes d'information (SI). La conception de ce type de système pose plusieurs défis, tels que l'évolution rapide des SI et la diversité en termes de technologies, de versions et de tailles. Il n'est pas possible de créer des procédures fixes dans ces conditions ; les possibilités d'automatisation sont limitées, et seule l'intervention humaine apporte la souplesse d'exécution requise.

Notre expérimentation repose sur un outil de gestion de procédures dynamiques.Cet outil propose aux humains de rédiger leurs procédures au fur et à mesure qu'ils les exécutent. Il présente une capacité d'appel de service web (WS) et d'enregistrement des données du contexte. Dans cet article, nous proposons de retrouver le WS et ses paramètres à partir d'un énoncé en langue naturelle.

Par exemple, à partir d'un énoncé de la forme :

```
Add an ERROR status notification on service 48658 with
message : storage is broken.
```

Le WS attendu serait :

```
{"action":"Post_monitoringServices_notifications",
"monitoringServiceId":"48658", "state":"ERROR", "content":"storage
is broken"}
```

En enregistrant non pas le WS, mais l'énoncé, le couplage reste faible et seulement associé au besoin de plus haut niveau. Les gains attendus vont de la souplesse de saisie des procédures à l'adaptation automatique aux changements de version des API de WS. Cet objectif désirable[5] se traduira aussi par une automatisation accrue de la gestion des SI

2 Présentation de l'approche

2.1 Le système de procédure et l'usage des WS

Nous disposons d'un outil de suivi de procédure qui permet l'édition et l'exécution simultanées des procédures. Le modèle de données est inspiré de IGARF [1][18][4](du monde de la robotique) et des modèles de dialogue homme machine de Allen basé sur une représentation explicite du temps[1].

Chaque processus forme un contexte d'interprétation. Les étapes du processus servent de cadre pour orienter le choix des étapes suivantes. Chaque étape comporte une action pouvant être un WS, une autre (sous-)procédure, ou un description du travail à réaliser. Les actions possèdent un titre et une description obligatoire. Pour l'utilisateur associé un WS revient à le sélectionner dans une liste d'API, puis a renseigner les paramètres à utiliser.

Les utilisateurs exécutant et modifiant à la volée la procédure (versionnée) capitalisent ainsi leurs connaissances. Ainsi ils spécifient, les cas particuliers qu'ils rencontrent, et le chemin de remédiation qu'ils emploient.

La ligne de commande l'outil permet le traitement d'énoncés en langue naturelle. C'est un moyen permettant de rechercher des WS, des paramètres de WS dans les étapes précédentes ou la description de l'étape en cours. Les résultats de l'appel du WS sont à leur tours intégré dans l'étape d'appel du WS et donc dans le contexte du processus.

Notre approche consiste à rendre le LLM responsable de la sélection du WS, tandis que l'appel du WS est effectué par le système de gestion de procédure, donnant lieu à une autre étape. L'objectif est de passer automatiquement de la description de l'étape de la procédure au choix d'un WS s'il existe.

Ultérieurement, la mise à jour des procédures lorsque les API changent sera facilité. De même de nombreux scripts utilise à la fois des appels vers des WS et un commentaire décrivant ce que fait l'appel. Dans un cas d'usage similaire, la mise à jour du script à partir du commentaire (et du WS précédent) sera un gain de temps. Ce second cas d'usage dépasse notre expérimentation actuelle.

Le traitement des énoncés regroupe les descriptions d'action et les commentaires de WS. Nous parlerons simplement d'*énoncé* en langue naturelle.

Les énoncés étaient initialement analysés par SNIPS[3] pour identifier les actions possibles. Cependant, cet outil manquait de précision, rendant son usage peu fiable. Il associait difficilement les bons paramètres aux actions, même lorsqu'il identifiait correctement l'action. La liste des WS utilisables n'était pas connue à l'avance, et cette implémentation des CRF[11] ne supportait pas l'apprentissage non supervisé. L'apprentissage des actions nécessitait alors un ensemble de phrases types.

Aujourd'hui, les améliorations des transformers, notamment les LLM, permettent de réaliser ce même travail plus précisément sur plus de paramètres et avec un apprentissage réduit ou proche de zéro (zero-shot learning). Les évolutions récentes des LLM (LLaMa[21], Mistral[8] & LoRa[7])

1. Taxonomy : Init Goal Action Result Final

permettent d'envisager leur fonctionnement sur un ordinateur portable, rendant l'usage à l'échelle possible. En effet, l'une des difficultés réside dans l'accès aux ressources de calcul, nécessitant une connexion à un centre de données sécurisé [2], capable de gérer les pics de charge. Cette difficulté serait résolue si le LLM pouvait fonctionner directement sur le poste de l'utilisateur final. Cette approche est également plus conforme à une politique de sécurité visant à réduire les surfaces d'attaque.

Les solutions existantes telles que Gorilla[17], utilisent des LLM avec un fine-tuning coûteux pour générer du code Python nécessitant une API Python, nous recherchons une solution plus légère. Dans les approches telles que ReAct[24] (Reason + Act), les LLM doivent être couplés à des fonctions programmées capables de réaliser les appels ainsi qu'une description(*Function Calling* pour Openai, *Function Tools* pour Llamaindex) ; cela ne nous convient pas car notre solution doit pouvoir s'accommoder d'API saisies à la volée par l'utilisateur(sans mise à jour de l'assistant/agent). Néanmoins, ReAct et le Chain-of-thought (Cot)[22] pourront être à terme un apport significatif sur le suivi de procédures et l'appel effectif du service une fois sélectionné. Dans toutes les solutions existantes que nous avons pu consulter, les services, actions ou outils qui peuvent être invoqués doivent être présents lors d'une étape de fine-tuning ou de programmation. Notre liste de WS n'étant pas connue *a priori*, ces solutions ne sont pas envisageables. Nos WS peuvent être mis à jour tous les jours par nos équipes ou par les plateformes tierces que nous utilisons.

3 Les données d'expérimentation

Le prompt du LLM en lui-même ne sert à rien s'il n'est pas possible d'extraire le résultat sous forme de la référence d'un service web à appeler. Chaque service web du contexte reçoit une clé de référence constituée de sa méthode HTTP, son chemin et ses différents paramètres possibles. Nous avons donc construit un parseur capable de transformer les complétions les plus courantes du LLM en clé de service web et d'identifier les paramètres, types et valeurs.

3.1 Choix de formats

Il y a plusieurs choses à choisir : le format des éléments du contexte, le format d'intégration du contexte, le prompt et le modèle. Tout ceci devant être testé devant un panel d'énoncés diversifié. Le format des éléments du contexte est déjà très automatisé (openAPI et WSDL). L'intégration du contexte se fait soit par échange de message avec le LLM (Discussion) soit au travers du RAG. La discussion a l'avantage de la simplicité. Plusieurs modèles peuvent être utilisés, ils doivent satisfaire les contraintes physiques de la machine mais aussi répondre assez vite. Les énoncés sont la partie qui demande de solliciter les équipes de l'entreprise pour en obtenir, cela relève du corpus d'apprentissage classique. L'expérimentation des prompts crée un nouveau cadre expérimental pour nous. Les prompts sont centraux dans l'interaction entre le format de WS et les énoncés. Leur mise au point implique préalablement de clarifier la méthode d'évaluation ultime du système.

3.2 Évaluation des résultats attendus

Les différents prompts sont évalués en comptant le nombre d'erreurs qu'ils engendrent sur une série d'énoncés de test. Les complétions doivent retourner la bonne méthode, la bonne clé de référence et les bonnes associations paramètres et valeurs. Les paramètres optionnels peuvent être tolérés s'ils

2. Conforme au référentiel d'exigences SecNumCloud

n'interfèrent pas avec le sens de l'énoncé ; par exemple, un paramètre modifiant le nombre de résultats par requête est souvent acceptable, sauf si l'énoncé indique explicitement le contraire. Le parseur est écrit suivant ces contraintes (voir 3.3). Cette évaluation nous donne un score où le meilleur résultat possible est zéro. $score = \sum_{allstatements} \sharp errors$

Nous ne faisons pas de différence entre deux énoncés incorrectement complétés. Connaissant pour chaque énoncé la forme attendue du résultat et tous les paramètres attendus, nous pouvons alors facilement calculer la précision.

Notre protocole est le suivant. Tout d'abord, choisir une liste d'énoncés utilisateurs. Ensuite, annoter les énoncés utilisateurs avec la forme de l'arbre de syntaxe logique attendu, en précisant les paramètres indispensables et les paramètres optionnels acceptables. Il faut alors choisir une liste de prompts à essayer. Ces prompts doivent être choisis pour leur brièveté, car plus le prompt sera long, moins l'énoncé utilisateur pourra être long en conséquence. Deux techniques de prompts peuvent être testées : avec ou sans apprentissage en contexte[25].

3.3 Ingénierie du prompt

Nous présentons un algorithme manuel. Un être humain doit l'exécuter. Le résultat est la création d'un ensemble de prompts. À ce stade précoce, une complétion est dite **correcte** si elle se termine par le WS attendu ou un WS halluciné de manière proche. L'hallucination est considérée comme proche si elle ne contient que des erreurs typographiques ou des changements de nom petits et non ambigus.

Algorithme de construction de prompt :

Etape 1 : Debut
Imaginer la demande au LLM de la manière la plus concise possible. Appelons là "prompt".
Etape 2 : Tester le prompt
Soit la longueur du *learning set* = 20 (énoncé + WS)
Pour un nombre incrémentale d'énoncés : [1 ;3 ;7 ;20] :
  Pour chaque énoncé :
    result := complétion(llm, prompt, énoncé)
    si result n'est pas correcte :
      choisir parmi :
        "améliorer le prompt"(étape 3)
        "oublier ce prompt"
    si le résultat est correct mais mal parsé :
      améliorer le parser
Etape 3 : Améliorer le prompt
  Utiliser les techniques connus d'amélioration de prompt :
  Recherche de synonyme, Reformulation, Détailler la tache, Ajout un exemple en multi-shot
Etape 4 : Sélection
  Sur l'ensemble des 20 exemples de l'ensemble d'apprentissage.
  Si au moins 10 aboutisse au bon WS :
    ajouter le prompt dans l'ensemble de ceux de l'expérimentation final.

Certains résultats sont difficiles à quantifier. Par exemple, si l'on ne demande pas explicitement au LLM de répondre par des "short answer no explanation", celui-ci va systématiquement compléter le texte avec une explication concernant l'usage du WS. Cette explication est parfois correcte, mais

systématiquement une hallucination car absente du contexte. L'explication peut être ignorée par le parser qui se contentera de la partie correctement identifiée, augmentant la précision. Afin de limiter ces hallucinations, des complétions courtes et sans explications sont préférables.

Au final notre corpus d'expérimentation est composé de *15* prompts créés à la main, incluant des prompts pour le *zero-shot learning* et des prompts pour le *one-shot learning*. Notre corpus d'énoncés, annoté en actions de service web et en paramètres, se compose de *150* énoncés répartis en *15* groupes de 10 énoncés présentant des niveaux de difficultés variés. Ces *150* énoncés d'API représentent *71* verbes de WS différents. Lorsque des verbes apparaissent en plusieurs exemplaires, ils le sont avec des conditions différentes avec des paramètres différents.

En moyenne, les énoncés utilisent *1,29* paramètres avec un écart type de *0.95*, le nombre minimum de paramètres étant *0* et le maximum *4*. Les contraintes de sécurité ne permettent pas de divulguer ici la liste complète des énoncés, car elles pourraient révéler le fonctionnement de systèmes critiques de l'entreprise.

Les deux premiers groupes d'énoncés (*20* énoncés au total) ont servi à la mise au point du parser et à l'ingénierie des *15* prompts. Les *13* groupes suivants n'ont été utilisés que dans le cadre d'une étude sur la généralisation des performances, sans correction ultérieure des prompts, du parser ou d'autres paramètres. Ces corrections seront évidemment réalisé lors du passage en production.

Les prompts courts présentent l'avantage de n'utiliser que peu de tokens (environ 30), mais laissent le LLM assez libre quant à la forme finale de la complétion. Pour remédier à ce problème, nous utilisons le multi-shot learning en fournissant un vecteur d'exemples au format attendu. Ces prompts font alors environ 100 tokens mais généralisent mieux la forme du résultat attendu.

4 Deux baselines

Avec SNIPS , un apprentissage était requis sur des phrases ou énoncé prédéfini auxquels les bons WS devaient être associés, de plus cet apprentissage devaient être refait pour chaque langue utilisé. Le passage au LLM est donc attendu comme étant des gains fonctionnelles. Ici quelque exemples des définitions attendus par cet outil.

Modèle d'intentions :
- send a new mail to [email :email] (user@domain.com)
- put the definition in my message and send the message to [email :email](user@domain.com)
- send an email to [email :email](user@domain.com)

Il faut aussi fournir des modèles pour les paramètres, dans le cas *email* c'est évident ; mais dans le cas d'identifiants techniques cela devient vite compliqué. Ces listes définitions permettent à SNIPS d'apprendre la structure des énoncés auxquels il sera exposé.

La précision sur des cas réels n'est jamais supérieur à 0.1 [3]. Ceci est trop faible pour aller en production, et le système dans cette configuration n'a pas pus être utilisé en production. Fonctionnellement, jamais un énoncé avec deux paramètres n'est correctement reconnus (même s'il en dispose dans sa base d'apprentissage). De même les formes d'énoncés employé par les utilisateurs sont plus divers que ceux employé pour l'apprentissage, alors même que les énoncés sont fonctionnement similaire. L'apprentissage n'est tout simplement pas assez orienté sur la sémantique. La description textuel

3. Sur deux groupes de 10 énoncés chacun.

est obligatoire dans l'outil de procédure, mais l'association du WS est finalement toujours en échec, résultant un non usage des cette fonctionnalité par nos utilisateurs.

Utilisation des WS "brutes" Dans cette baseline utilisant le LLM, nous utilisons les prompts zero-shot et les données de définition des WS au format openAPI-json (un document de plus de 10K lignes). Puis l'énoncé utilisateur est directement mis en entrée du LLM. Il s'agit d'une utilisation naïve mais rapide des WS et du contexte de RAG.

N° sentence group	0	1
RAG	0.15	0.00
Disc	0.10	0.00

TABLE 1 – Moyenne des précisions des *2* premiers groupes de *10* énoncés pour chaque méthode(zero-shot uniquement). RAG : le contexte des WS est chargé dans la base de donnée vectorielle du RAG. Disc : le contexte des WS est chargé comme une séquence de messages de Discussion avec le LLM. Les résultats de la baseline ne sont pas suffisamment bon pour aller en production.

Comme nous pouvons le constater dans le tableau 1 les résultats sont si faibles qu'il n'en sont pas meilleur que ceux avec l'outil précédent. Avoir obtenu un bon résultat résulte plus de la chance que d'une qualité propre de cette approche. A présent, améliorons ces résultats en retravaillant la présentation des données et le texte de prompts.

5 Premières expérimentation

Notre première approche consistait à inclure dans le prompt la totalité des services web (WS) visibles par la procédure au point considéré. Même sous une forme concise, ces descriptions sont bien trop longues pour une taille classique de prompt, de quelques milliers de tokens au mieux, alors que nous disposons de plusieurs centaines de WS. Notre seconde approche consiste à découper la liste WS en paquet de 10, puis les données au LLM en mode *chat*, comme une conversation. Une phase d'initialisation permet de donner au LLM tout les WS puis nous l'avons interrogé avec des énoncés utilisateurs.

Une autre approche est d'utilisé la Génération Augmentée par Récupération (RAG) [12] pour créer un contexte local pouvant être facilement modifié chaque fois qu'il est nécessaire d'évaluer un énoncé utilisateur à un point différent d'une procédure. Les RAG permettent de simuler des contextes beaucoup plus vastes que ceux possible dans une conversation. Les ordres de grandeurs ne sont pas les même. Plusieurs millions de token pour les RAG contre quelque milliers ou dizaine de millier pour les conversation utilisant les meilleurs modèles. Les RAG permettent aussi d'ajouter des données spécifiques sans nécessiter un fine-tuning.

5.1 Stratégies pour les WS et les prompts

Deux stratégies doivent être évalué simultanément dans cette expérience. L'une concernant la représentation des WebServices, l'autre concernant l'entrée du LLM (le prompt).

Les prompts : La technique du RAG consiste à réaliser une vectorisation des documents, puis à insérer ces vecteurs dans une base de données vectorielle, avant de récupérer les meilleurs vecteurs

compte tenu de l'énoncé utilisateur. Ces vecteurs ne représentent à chaque fois qu'une petite partie du contexte. En effet les documents du RAG sont découpé en blocs de 512 tokens. Lorsque le RAG est utilisé une requête est émise vers la base de donnée vectorielle afin de récupérer les vecteurs les plus proche de la requête. Il en résulte que si une partie d'un document à une logique qui prend plus de 512 tokens ce bloc logique est découpé et une partie ne sera pas remonté dans le contexte par la recherche vectorielle. In-fine, dans certains cas l'utilisation du RAG conduit à oublier localement une partie du contexte. Nous avons donc intérêt même dans l'utilisation du RAG a créer des descriptions de WS faisant moins de 512 tokens.

L'énoncé de l'utilisateur est concaténé à la suite d'un prompt qui est sélectionné par un mécanisme d'évaluation dédié à cette tâche lors de la phase d'ingénierie du projet. L'un des objectifs de l'expérimentation est de déterminer le prompt le plus adapté.

Les WS : À chaque énoncé de l'utilisateur, un bilan sur l'état en cours de la procédure est réalisé via sa logique de description afin de calculer le contexte. Ceci permet de sélectionner les API de WS à utiliser. Enfin, le format d'origine des WS est modifié avant leur insertion en DB du RAG ou conversation. Les WS sont écrits sous une forme courte où seul le chemin (PATH) du service, sa méthode et ses paramètres (typés) sont conservés.

Les web-services sont transformés en un format concis avant d'être insérés soit dans la conversation soit dans la DB du RAG (usage du LLM sans conversation). En effet, les représentations classiques sont beaucoup trop longues pour être utilisées avec cette technique. Par exemple, les formats WSDL et OpenAPI engendrent des fichiers de représentation en XML ou JSON de plusieurs centaines ou milliers de lignes pour chaque WS. Évidemment, ce format plus concis sont obtenues au prix d'une certaine perte d'informations ; nous jugeons cette perte acceptable car le développeur ne consulte lui-même que très rarement le détail des spécifications de WS, se fiant à son expérience pour deviner les bons paramètres.

Le format concis est construit en supprimant le paramètre de type s'il est 'chaîne de caractères', puis d'autres simplifications sont appliquées : '*entier*' (2 tokens) peut être facilement simplifié en '*int*' (1 token) ; de même, '*booléen*' devient '*bool*'. Les descriptions des commentaires sont parfois peu informatives en l'absence d'une description beaucoup plus longue et détaillée de l'emploi. Le nom simple, par contre, permet d'adopter des comportements par défaut assez cohérents, les développeurs des API essayant de coller aux conventions. Le format concis utilise jusqu'à 4 fois moins de tokens. Nous pouvons ajouter 3 fois plus de services *concis* dans le contexte que de services complets ; que ce soit dans la DB du RAG ou dans le contexte de la conversation. Ainsi, pour bien moins de tokens, nous pouvons encoder plus de WS. Nos expériences préliminaires montrent que des format concis conduisent à de meilleurs résultats.

5.2 Cadre expérimental

Pour l'expérimentation avec le LLM seul le lien entre énoncé et WS est testé. Dans la figure 1, la boucle de test est réalisée pour chacun des 15 prompts. Chaque prompt est testé avec les 150 énoncés, soit 2250 expériences pour le modèle avec RAG et autant pour le modèle avec messages de conversation. L'organisation de l'expérimentation permet d'isoler les expériences les unes des autres sans avoir à recharger le LLM entre deux, permettant un gain de temps expérimental considérable. Comme nous pouvons le voir sur la figure 1, c'est la forme après l'analyse du parser qui est testée car c'est bien cette forme qui peut ensuite être intégrée dans le modèle logique du suivi de procédure.

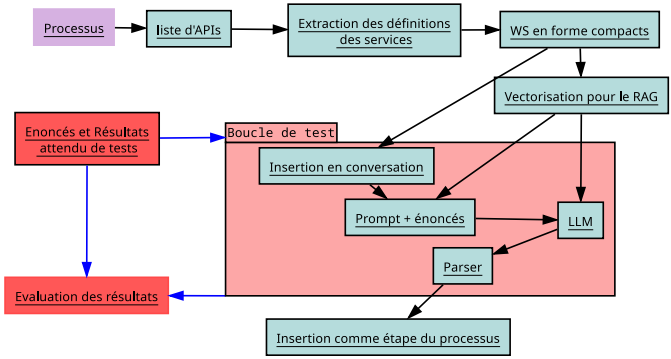

FIGURE 1 – Organisation de l'expérimentation.

Il y a deux chemins de tests, car il y a deux expériences, l'un des chemins passe par la vectorisation des WS pour le RAG, ce qui est à faire une seule fois pour tous les énoncés. L'autre chemin passe par l'établissement d'une conversation (une série de messages) avec le LLM. Puis, pour les deux chemins d'expérimentation, un prompt est créé avec l'énoncé de l'utilisateur. Les expériences sont exclusivement en anglais afin de ne pas biaiser les expérimentations.

Lors de l'utilisation des WS choisis via des énoncés, certains paramètres seront jugés optionnels et d'autres obligatoires. Si l'énoncé demande une liste d'objets sans spécifier comment il doit la recevoir, plusieurs réponses valides peuvent être possibles. Avec les paramètres optionnels il est possible de changer le nombre de résultats par page de résultat. L'usage de cette option n'est pas pénalisé par la métrique, sauf contre-indication explicite dans l'énoncé.

Configuration : Le LLM utilisé est Mistral 7b Instruct [4]. La version quantifiée à 4 bytes a suffisamment de connaissances et de compréhension pour répondre aux énoncés utilisés [5]. La taille maximale du prompt est fixée à 512 tokens (tout inclus), et cette taille de prompt n'est jamais dépassée dans nos évaluations. Il est encapsulé dans le projet PrivateGPT[15], et l'appel au modèle avec RAG (LlamaIndex[14]) ou 'chat' est effectué à travers des appels par API.

Afin de garantir la reproductibilité des expérimentations, la température des complétions est configurée à zéro (équivalent à *Greedy sampling*. Cela ne dégrade pas les performances de résolution de la tâche de raisonnement demandée[19]. L'architecture matérielle n'a aucune importance tant qu'elle peut charger le modèle, l'outil de procédure et l'environnement de développement. Cependant, les expérimentations sont nettement plus rapides si l'on peut disposer d'un GPU pour itérer plus vite. *PrivateGPT* permet d'exécuter les expériences et le développement sur un ordinateur portable tout en déportant le modèle sur une machine plus puissante. Par ailleurs, l'utilisation d'un modèle en open source avec inférence en locale nous permet de garantir la confidentialité des données requises par les activités de l'entreprise et/ou ses clients. Il n'est pas envisageable d'utiliser les modèles en ligne tels que ceux d'OpenAI, car les lois américaines(FISA) ne garantiraient aucune sécurité sur les données.

Une étape finale de filtrage des paramètres permet d'améliorer les scores et les distances. En effet, les hallucinations ont tendance à ajouter des paramètres qui n'existent pas dans les descriptions des

4. TheBloke/Mistral-7B-Instruct-v0.2-GGUF, mistral-7b-instruct-v0.2.Q4_K_M.gguf l'embedding est BAAI/bge-small-en-v1.5
5. https ://huggingface.co/blog/optimize-llm : 4-bit quantization allows the model to be run on GPUs such as RTX3090

services. Les complétions réalisées par le LLM pour les paramètres sont filtrées pour ne laisser que les paramètres qui existent réellement dans la description du service.

Premiers résultats : Avec le modèle Mistral-7b-Q4, la précision moyenne est d'environ *0,42* pour le RAG et *0,39* pour le mode Disc(ussion). Les résultats détaillés nous montrent que les formulations peuvent engendrer des différences de performances importantes. Comme attendu, les résultats sur l'ensemble d'apprentissage sont meilleurs. Quand le vocabulaire de l'énoncé s'éloigne de celui de l'API, les résultats sont moins bons.

Les prompts en one-shot learning donnent empiriquement de meilleurs précisions. Néanmoins, les erreurs restantes sont des hallucinations, des WS qui ne sont pas présents dans le contexte (et qui n'existent pas).

En effet, nous n'avons pas réussi, sur notre jeu d'énoncés d'apprentissage, à obtenir le score maximum. Parfois, le LLM ne choisit pas le bon WS mais un WS proche et va proposer un modèle de requêtes pseudo-SQL pour obtenir un paramètre requis par le mauvais WS. Évidemment, le modèle SQL est une hallucination de la complétion. Parfois, aucun de nos prompts ne permet au LLM de respecter un format reconnaissable de WS et de paramètres. La complétion est alors composée d'un texte détaillant l'usage de paramètres et valeurs qu'il faut utiliser. Le résultat de notre exemple du début, observé en sortie du llm :

```
{"action":"Post_monitoring_services_notifications",
"monitoring_service_id":"48658", "state":"ERROR",
"content":"storage is broken"}
```

6 Amélioration à base de distance d'édition

Usage de la distance d'édition : Nous ajoutons un post-traitement visant à réduire les hallucinations et les petites erreurs de rédaction des WS (snake-case vs camel-case, etc.). Ceci est réalisé via l'utilisation d'une distance d'édition entre le verbe déterminé après application du parser et en comparaison de l'ensemble de verbes possibles déterminé dans le contexte de la procédure (et donc du RAG ou de la Discussion). Le corpus de WS utilisé présente une bonne dispersion lexicale des WS. Sur la totalité des APIs manipulés (internes et externes à nos systèmes d'information), les verbes ont en moyenne une distance d'édition minimale de 59 et une médiane minimale de 37. Nous pouvons donc être globalement confiants que la sélection systématique du verbe le plus proche par distance d'édition conduira presque toujours à une amélioration des résultats. La distance entre les WS peut être liée à la volonté du développeur de conserver des API non confusantes.

Dans notre exemple après correction par distance de Leveinstein nous retrouvons l'attendu :

```
{"action":"Post_monitoringServices_notifications",
"monitoringServiceId":"48658", "state":"ERROR", "content":"storage
is broken"}
```

Usage d'un plus gros modèle : Nous avons reproduit nos expériences avec le modèle Mixtral 8x7b[9]. Il atteint la taille où il devient difficile de le faire fonctionner avec une configuration classique. Nous avons choisi ce modèle car il est l'un des meilleurs modèles open source en MoE[20][6]. Sa consommation en RAM est un peu plus de 24 Go, ce qui est trop pour notre matériel, mais via llama.cpp, il est possible de faire fonctionner le modèle sur le CPU, en utilisant LlamaIndex, il est possible de décharger 13 couches du modèle vers le GPU (sur les 33 couches redondantes du

modèle Mixtral). En utilisant un i7 12K (20 core) à 5 GHz, nous avons pu réaliser les expériences en seulement *3* fois plus de temps qu'avec le modèle Mistral 7b.

6.1 Résultats finaux

N° prompts	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
RAG	0.74	0.79	0.74	0.78	0.62	0.67	0.68	0.67	0.74	0.79	0.79	0.76	0.79	0.77	0.71
Disc	0.68	0.66	0.61	0.63	0.62	0.66	0.64	0.58	0.78	0.69	0.79	0.77	0.71	0.77	0.74
N° sentence group	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
RAG	0.79	0.8	0.75	0.76	0.76	0.74	0.69	0.8	0.68	0.73	0.77	0.74	0.7	0.63	0.79
Disc	0.79	0.59	0.77	0.71	0.68	0.8	0.68	0.61	0.63	0.68	0.78	0.68	0.74	0.62	0.61

TABLE 2 –
En haut : Moyenne des précisions des *150* énoncés pour chacun des *15* prompts pour chaque méthode. Les prompts n° 0 à 8 sont zero-shot. Les prompts n° 9 à 13 sont one-shot. Le prompt n° 14 est two-shot.
En bas : Moyenne des précisions des *15* groupes de *10* énoncés pour chaque méthode. RAG : le contexte des WS est chargé dans la base de donnée vectorielle du RAG. Disc : le contexte des WS est chargé comme une séquence de messages de Discussion avec le LLM. Mistral 7b Q4

	Précision		Temps de calcul	
Modèle	**RAG**	**Disc**	**RAG**	**Disc**
Mistral 7b	0,74	0,69	1h30	4h30
Mixtral 8x7b	0,74	0,72	4h30	14h
Llama3 8b	0,73	0,71	1h10	2h40

TABLE 3 – Performances en utilisant différents LLM. Tout les modèles ont été choisi dans leur forme GGUF, quantizé à 4 bytes. CPU-I7-12K, 64Go-DDR5, RTX3600

Le tableau 3 résume les précisions obtenues avec les différents modèles essayés. Le temps de calcul est véritablement un élément limitant dans les expérimentations, surtout avec les plus gros modèles. Les temps de calcul de ce tableau n'incluent pas les temps d'ingénierie des prompts. Nous pouvons constater que la précision est de *0,74* pour le RAG et de *0,69* pour le mode Discussion. Le modèle Mixtral ne montre pas des performances nettement supérieures. Par exemple, une erreur de complétion pour le modèle Mixtral pour l'énoncé "Retrieve all comments of current ticket." est d'indiquer que l'identifiant du ticket est "current_ticket_id", au lieu de simplement laisser ce paramètre manquant car non renseigné. Nous pouvons noter que le Mixtral a tendance à utiliser moins de paramètres optionnels, mais notre métrique ne fait pas de différence pour ce comportement secondaire.

7 Discussion et conclusion

Nous avons maintenant un résultat qui permet de transformer certains couplages forts entre actions / WS en des couplages faibles. Dans les cas d'échec, la sélection finale par l'utilisateur du bon WS après avoir saisi un énoncé permet non seulement d'obtenir la bonne documentation pour le WS choisi, mais aussi de permettre, dans un cas d'usage ultérieur, d'affiner la recherche du WS correspondant à une nouvelle version d'API. Cette précision de $0,74$ peut sembler plutôt modeste, mais la description en langage naturel est de toute façon indispensable et doit être réalisée. Cette précision reflète donc un gain fonctionnel lié à un début d'automatisation et une capacité partielle à mettre à jour les WS suite à des changements de version. L'utilisation d'un LLM remplace les CRF, évite d'avoir recours au fine-tuning sur les WS, ou d'ajouter des documents de description du métier, telle que la norme ITIL. Ces options restent toujours ouvertes pour accroître les gains ultérieurement.

Références

[1] James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11) :832–843, 1983.

[2] Daniela Castelluccia and Marina Mongiello. Modelling and verification of bpel business processes. In *MDB/MOMPES 2006*, pages 144–148, 2006.

[3] Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, Maël Primet, and Joseph Dureau. Snips voice platform : an embedded spoken language understanding system for private-by-design voice interfaces, 2018.

[4] Peter Dominey, Victor Paléologue, and Amit Kumar Pandey. Dominey et al. - 2017 - improving quality of life with a narrative companion. 10 2017.

[5] Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, and Randy Stafford. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

[6] Isobel Claire Gormley and Sylvia Frühwirth-Schnatter. Mixtures of experts models, 2018.

[7] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora : Low-rank adaptation of large language models, 2021.

[8] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

[9] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024.

[10] Stefan Karlsson, Robbert Jongeling, Adnan Causevic, and Daniel Sundmark. Exploring behaviours of restful apis in an industrial setting, 2023.

[11] John D. Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields : Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning*, 2001.

[12] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2020.

[13] AXELOS Limited and Stationery Office (Great Britain). *ITIL® Foundation, ITIL 4 Edition*. AXELOS global best practice. TSO (The Stationery Office), 2019.

[14] Jerry Liu. LlamaIndex, 11 2022.

[15] Iván Martínez Toro, Daniel Gallego Vico, and Pablo Orgaz. PrivateGPT, May 2023.

[16] Boris Motik, Peter F. Patel-Schneider, Bijan Parsia, Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, and Mike Smith. OWL 2 web ontology language : Structural specification and functional-style syntax. Last call working draft, W3C, 2008. (to be published, may be superseded).

[17] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla : Large language model connected with massive apis, 2023.

[18] Gregoire Pointeau, Solène Mirliaz, A.-L Mealier, and Peter Dominey. Learning to use narrative function words for the organization and communication of experience. *Frontiers in Psychology*, 12 :591703, 03 2021.

[19] Matthew Renze and Erhan Guven. The effect of sampling temperature on problem solving in large language models, 2024.

[20] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks : The sparsely-gated mixture-of-experts layer. In *ICLR (Poster)*. OpenReview.net, 2017.

[21] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama : Open and efficient foundation language models, 2023.

[22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35 :24824–24837, 2022.

[23] Stephen A. White. Using bpmn to model a bpel process, 2005.

[24] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React : Synergizing reasoning and acting in language models, 2023.

[25] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers, 2023.