


Libellage d'Algorithmes par l'Intelligence Artificielle

42003524 : Ermanno di Giulio

Double licence L2 2024-2025

 - Lien du GitHub

1 Résumé

Le projet consiste en la recherche d'une intelligence artificielle capable de déterminer la nature d'un algorithme de tri implémenté par une fonction en C. Les objectifs principaux sont de tester différents modèles d'IA opensource à l'aide d'un script Python, ainsi que de développer des prompts optimisés pour guider au mieux ces IA. Les modèles ont été évalués en fonction de leur précision à reconnaître les algorithmes parmi un ensemble de fonctions, en utilisant des métriques quantitatives. Au final, j'ai pu conclure que falcon3:7b était la plus performante parmi ceux testés pour ces applications.

2 Introduction

L'évaluation des algorithmes en informatique repose traditionnellement sur des méthodes manuelles ou automatisées, mais ces approches ont des limites. Les tests manuels sont chronophages et difficiles à étendre, tandis que les tests automatisés peuvent manquer de flexibilité, surtout face à des implémentations non standards. Dans ce contexte, l'Intelligence Artificielle (IA), notamment via le Machine Learning (ML) et le Traitement du Langage Naturel (NLP), offre une solution intéressante pour améliorer l'analyse des algorithmes.

Le Machine Learning permet à des modèles d'apprendre à partir de données, tandis que les modèles de Langage de Grande Taille (LLM) comme ChatGPT sont conçus pour comprendre et générer du texte, y compris du code source. Ce projet utilise des modèles qui se basent sur ces principes pour identifier des algorithmes de tri dans des fonctions en C, en automatisant ainsi une tâche traditionnellement manuelle.

Cependant, la diversité des modèles d'IA entraîne des variations dans les réponses, dues à leurs objectifs et conceptions différents. Certains modèles

sont plus adaptés à des tâches générales de langage, d'autres à des analyses techniques. Ces différences influencent la précision et la cohérence des résultats.

Ainsi, avec la démocratisation de l'IA dans ces dernières années - notamment par l'outil Ollama - j'ai tenté d'évaluer plusieurs modèles sur leur capacité à déterminer quel algorithme est implémenté par une fonction en C.

3 Objectifs

- Écrire un script Python pour automatiser les tests.
- Choisir des modèles d'IA opensource.
- Optimiser les prompts pour chaque modèle.
- Enregistrer les résultats d'exécution dans un fichier Excel et quantifier les performances.
- Comparer les performances des IA en termes de précision, temps de traitement, et cohérence des réponses.
- Déterminer l'IA la plus adaptée pour l'étiquetage des fonctions.

4 Choix des Technologies

4.1 Langage Python

J'utilise Python comme langage de script pour son efficacité dans le traitement des données et sa compatibilité avec sa bibliothèque pour l'interaction avec Excel `openpyxl`. Les réponses des LLMs sont enregistrées dans un tableau Excel.

4.2 Ollama

La librairie d'Ollama [1] offre une grande variété de modèles d'IA performants dont les suivants, qui seront analysés :

| Modèle | Paramètres | Caractéristiques |
|---------------|------------|---|
| llama 3.1 [2] | 8B | bon en langue et culture générale |
| llama 3.2 [3] | 3B | respecte assez bien le prompt |
| mistral [4] | 7B | bon en code et en langue |
| falcon3 [5] | 7B | bon en sciences, maths et programmation |

Table 1: Modèles et leurs caractéristiques affichées dans la librairie

Le nombre de paramètres est mesuré en B pour "billions" (milliards en anglais).

4.3 ChatGPT pour le Support au Développement

ChatGPT est utilisé en complément pour m’assister dans des tâches variées, telles que :

- Étiquetages manuel en amont de quelques fonctions pour préparer les tests.
- Écriture du script Python pour effectuer les tests.
- Réviser le prompt donné aux IA.

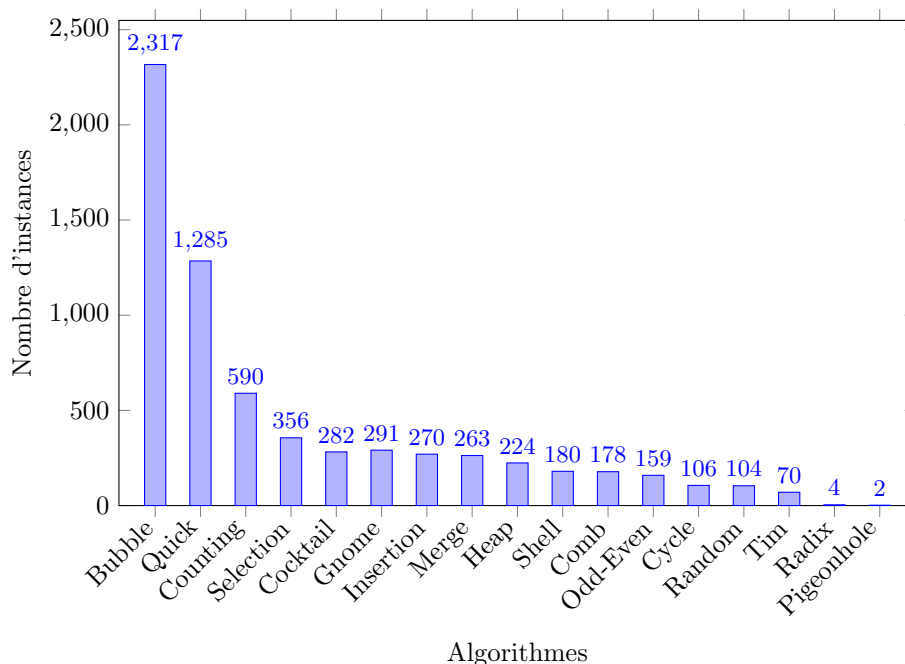
4.4 Bibliothèques Utilisées

- `openpyxl` [6] : Interaction avec Excel
- `sqlite3` : Stockage et gestion des fonctions à tester.
- `langchain-ollama` [7] : Intégration avec Ollama pour l’étiquetage des fonctions par les IA.
- `os` : Gestion des fichiers.
- `time` : Mesure du temps d’exécution

5 Méthodologie

5.1 Données traitées

La base de données des fonctions est proposée par le Prof. Delbot. Elle contient un grand nombre de fonctions écrites en C par des étudiants notamment dans le cadre de travaux dirigés. Les données sont représentées sur 4 colonnes : l’identifiant (ID), l’algorithme implémenté, le nom de la fonction et le code de la fonction. La répartition du type de fonction est montrée par le graphique suivant :



Ces fonctions tentent d'implémenter l'algorithme de tri décrit par leur nom, mais dans beaucoup de cas ces implémentations sont fallacieuses, d'où l'intérêt de ces données.

Dans le cadre de mon projet, j'en ai sélectionné 5 de chaque type (où cela est possible). Ces fonctions n'étaient pas (pour la plupart) encore étiquetées, donc la première partie de ce projet de recherche était l'étiquetage manuel de ces fonctions [8]. Les fonctions choisies sont généralement assez variées en termes de succès d'implémentation.

Malheureusement certaines fonctions n'apparaissent que très peu de fois, telles que `sort_pigeonhole` (2) ou `sort_radix` (4), tandis que d'autres ont un nombre d'instances très élevé telle que `sort_bubble` (2317). Ainsi les éventuels résultats des tests approfondis ne seront pas toujours autant fiables.

5.2 Les Étapes

5.2.1 Écriture du script Python

L'écriture du script Python s'est articulée autour de trois objectifs principaux : l'intégration avec les modèles d'intelligence artificielle, la gestion structurée des données dans un fichier Excel et la répétabilité des tests. Voici les principales étapes suivies pour sa conception.

Extraction des fonctions depuis la base de données Les fonctions à tester étaient stockées dans la base de données SQL fournie par le Prof. Delbot.

La première étape du script consistait à établir une connexion à cette base pour extraire les données nécessaires. Une requête SQL avec la bibliothèque `sqlite3` permettait de récupérer les identifiants des fonctions et leur contenu en langage C. Cette structure garantissait une compatibilité avec les tests et offrait une flexibilité pour ajuster les ensembles de données.

```
# Récupérer le code de la fonction à partir de la base de données
cursor.execute(
    "SELECT * FROM algorithms WHERE id = ?",
    (function_id,)
)
function_instance = cursor.fetchone()
algorithm_code = function_instance[3]
```

Intégration avec les modèles d'IA [9][10][11] La bibliothèque `langchain-ollama` a été utilisée pour interagir avec les modèles d'intelligence artificielle de la librairie Ollama, comme `llama3.2` ou `falcon3`. Chaque fonction extraite de la base était transmise à l'IA sous forme d'un prompt. Les réponses des modèles étaient ensuite récupérées, prêtes à être enregistrées.

```
# Construire le prompt pour Ollama
prompt = prompt_template + f"{algorithm_code}"
# Exécuter le modèle
response = llm.invoke(prompt)
```

Gestion des résultats dans Excel Afin de conserver les résultats des tests de manière ordonnée, le script exploitait la bibliothèque `openpyxl`. Les données étaient structurées de façon à inclure :

- L'identifiant unique de la fonction.
- L'algorithme attendu (référence manuelle).
- Les réponses générées par chaque modèle IA, placées dans des colonnes successives pour faciliter la comparaison.

Le script gérait également l'ajout automatique des colonnes et évitait les doublons en vérifiant si un ID de fonction existait déjà dans le fichier.

```
# Ajouter une nouvelle ligne
new_row = [function_id, expected_result] + [""] * (column_index - 3) + [response]
sheet.append(new_row)
```

Automatisation et robustesse Pour garantir la répétabilité des tests, le script comprenait des mécanismes de vérification. Par exemple, si une fonction avait déjà été testée, elle n'était pas envoyée de nouveau au modèle, ce qui optimisait les temps d'exécution. De plus, le prompt utilisé était enregistré une

seule fois, dans une cellule dédiée du tableau Excel pour assurer une traçabilité des tests effectués.

Avec ce script, il a été possible d'automatiser les tests sur une grande variété de fonctions, tout en maintenant un historique détaillé des résultats. Cette approche a posé les bases d'une analyse comparative rigoureuse entre les différents modèles d'intelligence artificielle.

5.2.2 Formatage et génération des prompts

llama3.2 Il s'agit du premier modèle que j'ai testé. L'élaboration d'un prompt pour llama3.2 était donc assez fastidieuse car je devais m'adapter au fonctionnement de l'IA pour la première fois. J'ai commencé par des prompts assez simples tel que :

Determine the nature of the algorithm implemented by the following function

Mais je me suis vite rendu compte que ce n'était pas assez précis, notamment pour la format de la réponse. J'ai donc écrit un prompt plus long et plus élaboré pour mieux guider ce modèle. Cependant, un prompt plus long peut compliquer la compréhension par l'IA, donc j'ai essayé de le réduire au sens culinaire du terme pour en avoir une version claire, aussi concise que possible et qui soit respectée. Au final, j'ai déterminé que ce prompt était le plus adapté :

You will see a function written in C that tries to implement an algorithm. If the algorithm corresponds to the function's name, respond only with the algorithm name (for example, 'Bubble Sort') and nothing else. If more than one algorithm is being implemented or if you see more than one function being defined, please respond with 'other'. If the code implemented by the function doesn't correspond to the function's name (for example the function's name is 'tri_bulle' (Bubble Sort) but it implements Bogo Sort), please answer only and only with 'other'. This means that I don't want to see you responding with anything else than either 'other' or the generic English name of the algorithm if it is consistent with the name of the function.

llama3.1 J'ai initialement utilisé le même prompt pour llama3.1 que celui pour la version 3.2, mais les réponses étaient souvent incorrectes. J'ai donc modifié la requête pour mieux correspondre au fonctionnement du 3.1. J'ai trouvé cette requête :

Identify the algorithm implemented by the following function code. Ignore the function names, comments, and identify only based on the structure and operations. Respond with the exact name of the algorithm only, and nothing else. If you believe that the functions

don't represent any known algorithm, answer with 'other'. Be careful, because a function could look like a specific algorithm, but could miss one line which would change the whole nature of the algorithm.

qui donnait des résultats plus corrects mais le format de réponse n'était pas du tout respecté. J'ai donc décidé de garder le même que pour llama3.2.

mistral Au premier abord, mistral a été jusqu'à ce moment-là le modèle d'IA le plus prometteur, aussi bien en termes de format que de justesse des résultats avec le même prompt que les IA précédentes. J'ai tout de même optimisé la requête en changeant remplaçant la dernière phrase par la suivante:

This means that I don't want to see you responding with anything else than the bare minimum, either 'other' or the generic English name of the algorithm if it is consistent with the name of the function.

falcon3 En ce qui concerne falcon3, il n'y a pas eu de changements au niveau du prompt.

Toutefois, la persistance des réponses différentes que "other" m'ont conduit à assouplir ma méthode en acceptant toute autre réponse différente que la fonction donnée.

5.3 Exécution des tests

Pour éviter de perdre trop de temps et de me retrouver avec une exécution entière du programme, le programme affiche chaque réponse une par une du modèle d'IA. Ainsi je peux rectifier d'éventuels problèmes en temps voulu. Voici un aperçu d'une exécution :

```
→ projet_recherche python3 main.py
Entrez le nom de la feuille où insérer les données : mistral-latest
Function ID: 184898, Expected: BubbleSort, Model Response: Bubble Sort
Function ID: 592, Expected: other, Model Response: Quick Select
Function ID: 62, Expected: BubbleSort, Model Response: Bubble Sort
Function ID: 105371, Expected: other, Model Response: Tri à bulle
Function ID: 165940, Expected: BubbleSort, Model Response: Bubble Sort
Function ID: 55874, Expected: CocktailSort, Model Response: Cocktail Sort
Function ID: 56136, Expected: CocktailSort, Model Response: 'Cocktail Sort'
Function ID: 43714, Expected: other, Model Response: Cocktail Sort
Function ID: 54547, Expected: CocktailSort, Model Response: Cocktail Sort
Function ID: 61954, Expected: other, Model Response: Cocktail Sort
```

Figure 1: Exemple d'exécution.

5.4 Métriques d'évaluation

Afin d'évaluer rigoureusement les performances des modèles, plusieurs métriques ont été établies pour analyser différents aspects clés. Ces métriques garantissent

une approche systématique et objective dans l'interprétation des résultats.

5.4.1 Analyse binaire de la nature de l'algorithme

La première métrique, qui sera la plus importante dans notre analyse finale, repose sur une analyse binaire permettant d'évaluer la capacité de l'IA à identifier correctement la nature de l'algorithme en question. Plus précisément, il s'agit de vérifier si, pour une fonction donnée, l'IA est en mesure d'associer correctement le nom de l'algorithme correspondant. Si l'implémentation est correcte, l'IA doit répondre par le nom de l'algorithme. Dans le cas contraire, elle peut produire une autre réponse, qu'il s'agisse du terme "autre", du nom d'un autre algorithme, ou de toute autre formulation erronée. Cette évaluation vise à mesurer la probabilité qu'une réponse correcte soit générée dans ce cadre strictement binaire.

On calcule donc le pourcentage de réponses correctes selon ces critères avec la formule :

$$P_{\text{correctes}} = \frac{N_{\text{correctes}}}{N_{\text{total}}} \times 100$$

5.4.2 Temps d'exécution

Le temps d'exécution représente un indicateur essentiel pour apprécier la performance pratique des modèles. Pour chaque algorithme testé, la durée moyenne d'exécution est calculée à l'aide de la bibliothèque `time`, garantissant une mesure précise et standardisée. La durée moyenne est donc calculée, pour chacune des IA, avec la formule suivante :

$$T_{\text{moyen}} = \frac{\sum_{i=1}^n T_i}{N_{\text{tests}}}$$

Cette métrique permet ainsi de comparer les modèles entre eux en termes d'efficacité opérationnelle.

5.4.3 Cohérence des réponses

La cohérence des réponses constitue un autre aspect critique de l'évaluation. Cette métrique s'intéresse à la stabilité des réponses fournies par l'IA à travers plusieurs itérations du même test. Une IA cohérente devrait fournir les mêmes réponses dans le même ordre pour des cas identiques. Pour mesurer cette cohérence, on évalue la diversité des réponses : on calcule combien de réponses différentes ont été générées par l'IA pour une même fonction au fil des tests :

$$C_{\text{réponses}} = \frac{\sum_{i=1}^n V_i}{N_{\text{tests}}}$$

Avec V_i le nombre de réponses différentes par fonction

Plus $C_{\text{réponses}}$ est proche de 1, plus le modèle d'IA est cohérent.

Par exemple, si une même fonction suscite deux réponses distinctes pour chacun des 5 tests, cela indique une variabilité de 0.4.

5.5 Défis et contretemps

Au cours du projet, plusieurs défis et contretemps ont émergé, affectant la progression et l'efficacité des tests. Ces obstacles ont été résolus de manière itérative, mais ils ont eu un impact considérable sur le calendrier et les résultats obtenus.

5.5.1 Améliorations progressives du script

Le script Python a nécessité plusieurs révisions pour répondre correctement aux exigences du projet. Une des premières difficultés a été le formatage des feuilles Excel : les résultats des tests étaient mal alignés et il était difficile de comparer les performances des IA de manière uniforme. Après quelques ajustements dans l'organisation des colonnes et des lignes, j'ai réussi à obtenir une présentation plus claire et lisible des résultats. De plus, certaines bibliothèques Python ont montré des comportements inattendus, nécessitant des mises à jour et des tests répétés pour assurer la compatibilité avec la version la plus récente de `openpyxl`.

5.5.2 Rectifications du prompt

L'un des défis majeurs rencontrés a été que, même après avoir soigneusement conçu les prompts, certaines IA ne respectaient pas systématiquement ces instructions. Cela a été particulièrement frustrant lors de l'exécution des tests, car il était difficile de reproduire des résultats cohérents. Par exemple, certains modèles ignoraient les détails spécifiques du prompt, donnant des réponses trop détaillées ou au contraire trop vagues. Malgré les ajustements apportés aux formulations, la plupart des IA semblaient s'écarter de manière significative de ce qui était demandé.

- **Exemple 1 :** Dans de nombreux cas, au lieu de donner simplement le nom de l'algorithme, le modèle fournissait une description détaillée, ce qui n'était pas nécessaire et augmentait considérablement le temps de traitement.
- **Exemple 2 :** Le modèle tendait parfois à interpréter le nom de l'algorithme de manière erronée, comme dans le cas de "Bubble Sort", qui a été parfois désigné par "Bulle" en français, ou encore par "tri à Bulles." dans d'autres cas, ce qui nuisait à la cohérence des résultats.

Pour contourner cela, plusieurs itérations ont été nécessaires pour affiner les prompts, mais même ainsi, les modèles n'ont pas toujours respecté exactement les consignes.

5.5.3 Choix des IA

Un autre défi a été de devoir éliminer certaines IA en raison de leur incapacité à respecter les prompts ou de leur précision insuffisante dans les réponses. Un de ces modèles d'IA était le modèle CodeLlama[11] 7B qui n'a pas donné de

résultats fiables après plusieurs tentatives et a été retiré de la liste des modèles utilisés. Ensuite, les modèles gemma[12] et yi-coder[13], étaient à premier abord les moins performants dû à leur incapacité totale à respecter les consignes et correctement identifier les algorithmes. Les performances de certaines IA se sont avérées trop hétérogènes pour être prises en compte de manière fiable dans l'analyse comparative. Le retrait de ces modèles a demandé un temps important, mais a permis de se concentrer sur les IA les plus prometteuses.

5.5.4 Latence d'exécution des tests

Les tests d'étiquetage des algorithmes ont montré des latences d'exécution importantes, notamment avec le modèle mistral, qui nécessitait un temps d'attente significatif avant de donner une réponse, surtout en comparaison avec d'autres modèles comme ChatGPT ou même llama3.1. Ces latences ont augmenté le temps global de traitement des tests, affectant ainsi l'efficacité du projet.

5.5.5 Hétérogénéité des réponses

Enfin, une des difficultés majeures rencontrées a été l'hétérogénéité des réponses fournies par les IA. Même après avoir ajusté les prompts et les formats d'entrée, les résultats restaient souvent incohérents. Cela a été illustré dans l'Exemple 2, ce qui rendait difficile l'évaluation systématique de la performance des modèles. De plus, certaines réponses étaient excessivement détaillées, incluant des explications sur l'algorithme, tandis que d'autres se contentaient du nom, mais sans préciser les variantes possibles de l'implémentation.

6 Avancement et Résultats

6.1 Analyse binaire

Les résultats des modèles en termes d'analyse binaire sont résumés dans le tableau suivant :

| Modèle | Score |
|----------|-------|
| Llama3.1 | 0.679 |
| Llama3.2 | 0.629 |
| Mistral | 0.664 |
| Falcon3 | 0.712 |

Table 2: Résultats de l'analyse binaire des modèles

6.2 Temps d'exécution

Les temps d'exécution des différents modèles sont présentés dans le tableau suivant :

| Modèle | Temps d'exécution (secondes) |
|----------|------------------------------|
| Llama3.1 | 1982.18 |
| Llama3.2 | 672.57 |
| mistral | 2575.83 |
| Falcon3 | 2030.59 |

Table 3: Temps d'exécution des modèles

6.3 Cohérence des réponses

Les valeurs de cohérence des réponses pour chaque modèle sont données dans le tableau suivant :

| Modèle | Cohérence des réponses |
|----------|------------------------|
| Llama3.1 | 1.279 |
| Llama3.2 | 1.206 |
| Mistral | 1.162 |
| Falcon3 | 1.647 |

Table 4: Cohérence des réponses des modèles

7 Discussion

Les résultats obtenus mettent en lumière des différences notables entre les modèles analysés selon les trois principales métriques : l'analyse binaire, le temps d'exécution et la cohérence des réponses. Cette section explore ces divergences en profondeur afin de tirer des enseignements significatifs pour l'utilisation et l'amélioration future des IA testées.

7.1 Analyse binaire

Les scores d'analyse binaire montrent que falcon3 se distingue légèrement avec un taux de précision de 71,2 %. Ce modèle semble avoir une meilleure capacité à associer les noms d'algorithmes aux fonctions données, bien que les autres modèles, comme llama3.1 et mistral, offrent des performances relativement compétitives. Cependant, ces résultats doivent être mis en perspective avec la cohérence des réponses et le temps d'exécution pour une évaluation globale.

7.2 Temps d'exécution

Le temps d'exécution varie considérablement entre les modèles, avec des durées allant de 672,57 secondes pour llama3.2 à 2575,83 secondes pour mistral. Ce dernier, bien que performant en termes d'analyse binaire, est désavantagé par sa latence élevée. Cela souligne un compromis clé entre précision et rapidité,

où falcon3 et llama3.1 offrent un équilibre acceptable. Il est important de noter que tandis que llama3.1, mistral et falcon3 ont été testés sur des versions avec des nombres de paramètres similaires (7-8 milliards), llama3.2 n'était proposé qu'avec un nombre de paramètres bien plus bas (3 milliards).

7.3 Cohérence des réponses

La cohérence des réponses, mesurée par la stabilité des réponses à travers plusieurs tests, révèle des résultats intéressants. falcon3 se distingue avec un score de 1,647, indiquant une variabilité relativement élevée dans ses réponses, ce qui réduit sa fiabilité dans des scénarios nécessitant des réponses uniformes. En revanche, mistral et llama3.2 montrent une plus grande uniformité, ce qui augmente leur fiabilité dans certains cas d'utilisation.

7.4 Défis et implications

Les difficultés rencontrées, telles que les variations dans les réponses dues aux prompts ou la latence élevée, soulignent les limites actuelles des modèles d'IA. Ces problèmes rappellent l'importance d'une conception rigoureuse des prompts et d'une infrastructure optimisée pour réduire les temps de traitement. Ces aspects pourraient être encore affinés pour garantir des performances constantes et adaptées à des applications spécifiques.

En ce qui concerne le prompt notamment, il pourrait être intéressant pour des recherches futures de laisser plus de flexibilité aux IA, sans les encadrer sur le format de réponse. En effet, non seulement est-il possible que les IA réussissent à mieux analyser les algorithmes en ayant plus de liberté de "réflexion", mais aussi qu'un prompt plus court aurait un impact positif sur l'étiquetage de ces derniers.

8 Conclusion

L'étude comparative des modèles d'IA, incluant falcon3, llama3.1, llama3.2 et mistral, révèle des forces et des faiblesses distinctes selon les métriques établies. falcon3 émerge comme le modèle le plus équilibré, offrant une bonne précision binaire, un temps d'exécution acceptable, mais moins cohérent que les autres modèles. Cependant, les compromis entre rapidité et précision restent une considération cruciale, particulièrement pour des applications nécessitant des réponses rapides ou des volumes élevés de données.

Les enseignements tirés de cette analyse ouvrent la voie à des recherches futures. En particulier, les modèles tels que granite3.1-dense, QwQ et athene-v2 pourraient être inclus dans des travaux ultérieurs pour élargir la comparaison. De plus, des approches expérimentales pourraient être envisagées pour réduire la latence des modèles et améliorer la gestion des prompts afin de garantir une uniformité accrue.

En fin de compte, cette étude contribue à une meilleure compréhension des capacités et des limitations des modèles d'IA actuels, posant les bases pour des travaux visant à optimiser leur efficacité et leur fiabilité dans des contextes pratiques variés.

References

- [1] Ollama, *Library*. [En ligne]. Disponible : <https://ollama.com/library>.
- [2] Meta, llama3.1 [En ligne]. Disponible : <https://ollama.com/library/llama3.1>.
- [3] Meta, llama3.2 [En ligne]. Disponible : <https://ollama.com/library/llama3.2>.
- [4] Mistral AI, mistral [En ligne]. Disponible : <https://ollama.com/library/mistral>.
- [5] Technology Innovation Institute (TII), falcon3 [En ligne]. Disponible : <https://ollama.com/library/falcon3>, Accédé : décembre 2024.
- [6] OpenPyXL, Tutorial [En ligne]. Disponible : <https://openpyxl.readthedocs.io/en/stable/tutorial.html>.
- [7] LangChain, OllamaLLM [En ligne]. Disponible : <https://python.langchain.com/docs/integrations/llms/ollama/>.
- [8] GeeksforGeeks, Sorting Algorithms [En ligne]. Disponible : <https://www.geeksforgeeks.org/sorting-algorithms/>.
- [9] Ollama, ollama [En ligne]. Disponible : <https://github.com/ollama/ollama>.
- [10] Medium, How to Run Llama-3.1 locally in Python using Ollama, LangChain [En ligne]. Disponible : <https://emmakodes.medium.com/how-to-run-llama-3-1-locally-in-python-using-ollama-langchain-331c9984a4b5>.
- [11] Manjushsh, Configuring Ollama and Continue VS Code Extension for Local Coding Assistant [En ligne]. Disponible : <https://manjushsh.github.io/local-code-completion-configs/>.
- [12] Meta, codellama [En ligne]. Disponible : <https://ollama.com/library/codellama>.
- [13] Google DeepMind, gemma [En ligne]. Disponible : <https://ollama.com/library/gemma>.
- [14] Yi-Coder, yi-coder [En ligne]. Disponible : <https://ollama.com/library/yi-coder>.