Jules HIRTZ
Yann MIJATOVIC
Théo PINCHON
Alexandre PERROT

Mimir
Tuteur : Yann BONIFACE
Année 2023 - 2024

MIMIR

Récapitulatif 1ère itération

IUT nancy Charlemagne

Sommaire

Planning prévisionnel

- **Itération 1 (38h du 18/12 au 12/01)**

Création d'une application morte. (Jules / Théo)
Mise en place de la base de données mongo. (Jules / Théo)
Preuve de concept de l'IA : 2 membres (Yann / Alexandre)

Preuve de Concept de l'Intelligence Artificielle (PoC)

Pour la preuve de concept, nous avons tout d'abord commencé nos recherches sur le site huggingFace, qui répertorie des cours, des modèles d'IA et des librairies afin d'utiliser des IA.
Nous nous sommes penchés d'abord sur l'utilisation du modèle BERT. On pourra retrouver les conclusions sur le test des différents modèles d'IA testés dans la suite de cette partie.
Pour utiliser les modèles de huggingFace, nous nous sommes rendus compte que plusieurs paramètres influent beaucoup sur la sortie des modèles. Ces paramètres sont :

- input_ids : Les identifiants de jetons de la séquence d'entrée à partir de laquelle vous souhaitez générer du texte.
- max_length : Longueur maximale du texte généré
- num_return_sequences : Le nombre de séquences générées à retourner. Si par exemple je veux générer plusieurs questions à partir d'un contexte, j'augmenterais ce nombre au nombre de séquences désiré.
- no_repeat_ngram_size : La taille du n-gramme qui ne doit pas être répété dans le texte généré. Cela aide à éviter la répétition excessive de motifs dans le texte.
- top_k : Le nombre de mots les plus probables à considérer lors de la sélection du mot suivant. Seuls les mots avec les probabilités les plus élevées parmi ceux-ci sont pris en compte. (de la même manière que le kNN)
- temperature : Contrôle le niveau d'entropie lors de la sélection des mots. Une valeur plus élevée (par exemple, `temperature=1.0`) introduira plus d'aléatoire, tandis qu'une valeur plus basse (par exemple, `temperature=0.8`) rendra les choix plus déterministes.
- attention_mask : Un masque d'attention qui indique quels éléments de la séquence d'entrée doivent être pris en compte lors de la génération.
- do_sample : Un drapeau qui indique si l'échantillonnage stochastique doit être utilisé (`True`) ou si l'échantillonnage déterministe doit être utilisé (`False`).

Pour ce qui est des deux types d'échantillonnages :

- Déterministe (Greedy Sampling) : Dans cette approche, le mot avec la probabilité la plus élevée d'être le suivant est choisi à chaque étape de la génération. Cela signifie que le modèle sélectionne toujours le mot qui semble être le "meilleur" à chaque étape, basé sur les probabilités prédites.

- Stochastique (Top-k Sampling, Top-p Sampling, Nucleus Sampling) : Cette approche introduit de l'aléatoire dans le processus de sélection du mot suivant. Au lieu de choisir systématiquement le mot le plus probable, le modèle tient compte d'une distribution de probabilités plus large.
    - Top-k Sampling : Le modèle sélectionne le mot parmi les k mots les plus probables.
    - Top-p Sampling (Nucleus Sampling) : Le modèle sélectionne le mot parmi les mots dont la somme cumulative des probabilités atteint un seuil (probabilité cumulée supérieure à p).

En modifiant tous ces paramètres pour vérifier les différentes sorties des modèles avec un prompt simple (une question basique), puis des prompts avec un contexte plus complexe,, nous sommes arrivées à ces conclusions :

- BERT : Ce modèle a besoin de la réponse dans le contexte pour pouvoir générer la réponse. Le problème ici est que il faut le contexte et la question pour générer la réponse. On ne peut pas seulement se servir du contexte pour générer des questions et des réponses. RIEN N'EST GÉNÉRE , TOUT EST EXTRAIT (la réponse est trouvée dans le contexte)

- GPT-2 : L'IA ici ne donne pas de résultats explicites. On obtient des résultats aberrants quand je lui demande de générer les réponses d'une question simple. (mots inventés et dans une autre langue). Malgré les changements de paramètres, nous n'avons jamais pu obtenir des résultats concluants.

- T5 : Je voulais ici générer les questions pour un contexte donné. Malheureusement, le modèle me retourne des questions pas très logique (True , False , Tasty-Ultra-Natural-Facts-Not-For-Profit-Analyse-Question-Reporter-In-Transcription ……). Tout comme GPT-2, nous n'avons pas obtenu de résultat prometteur malgré les changements de paramètres.

- Vigogne : Le problème de ce modèle est qu'il utilise beaucoup de RAM et est assez long pour donner une réponse (15 - 20 minutes) avec 16 Go de RAM sans GPU. Cependant, c'est le seul modèle avec des résultats concluants, où nous obtenons une réponse cohérente en français.

Vigogne étant le modèle le plus prometteur mais trop lent, nous avons essayé d'utiliser une nouvelle technologie afin de faire tourner nos modèles : Ollama.

Le seul souci de Ollama est que le modèle Vigogne n'est pas disponible. Nous avons donc en premier lieu essayé un nouveau modèle d'IA récent prometteur qui fait beaucoup parler de lui en ce moment, Mistral.

D'après les tests réalisés sur ce modèle, nous nous sommes rendus compte de plusieurs choses :

- Les librairies d'huggingFace utilise la RAM et le disque local de la machine pour faire tourner ses IA, ce qui ralentit drastiquement leur exécution alors que Ollama se sert du CPU et du GPU, ce qui accélère énormément l'exécution.
- Avec Mistral, on obtient des résultats dans un laps de temps plus qu'acceptable (quelques secondes sans GPU) un résultat moyennement convaincant. En effet, ce modèle est "fine-tuned" en anglais, c'est-à-dire qu'il est plus fait pour parler en anglais (même si normalement le français ne doit pas être un problème pour lui). On obtient donc facilement des fautes de conjugaison, d'orthographe ou de syntaxe.

Les résultats de Mistral auraient pu suffir dans le cadre de notre PoC pour l'utiliser afin de générer des questions réponses, mais nous avons eu une idée. Nous nous sommes rendus compte que des modèles pouvaient être importés sur Ollama à l'aide d'un fichier .gguf, puis en définissant un "Modelfile", donnant un template au modèle. C'est-à-dire qu'on lui dit quel comportement adopter à chaque fois qu'un prompt lui est transmis. Nous avons pu donc importer un modèle de Vigogne dans Ollama, en lui définissant dans son template un format d'écriture de questions et de réponses afin de pouvoir récupérer le même format à chaque contexte transmis au modèle.

Pour la suite des tests, nous avons utilisé un serveur personnel avec une carte graphique, contrairement aux ordinateurs de l'IUT, et avec un CPU plus puissant, ce qui va drastiquement accélérer le traitement du modèle pour chaque prompt.

Avec Vigogne, nous nous rendons compte de plusieurs choses :

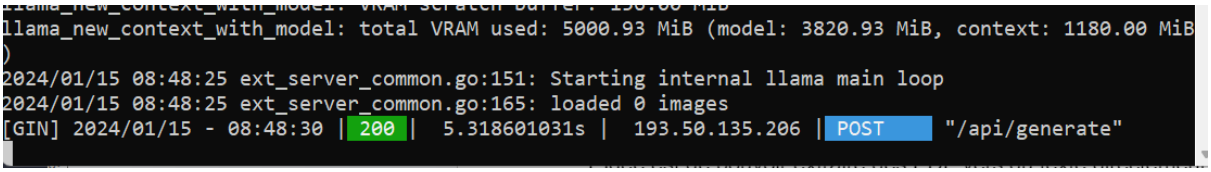
- Le template inséré au départ n'est pas toujours appliqué dépendamment de la taille du prompt (le modèle peut ne pas générer la bonne forme de questions/réponses ou simplement ne pas donner de questions/réponses)
- Si on ne précise pas un nombre de questions/réponses, le modèle en génère généralement une.
- Si le contexte est beaucoup trop long (texte d'un gros PDF), le modèle peut ne rien générer ou générer quelque chose qui ne respecte pas du tout le format de questions/réponses.
- Certains prompts vont le perturber. Il faut bien lui préciser son comportement dans le template ainsi que dans le prompt (indiquer la fin du contexte et le nombre de questions/réponses voulues à la fin)

Malgré ces petits défauts, nous obtenons presque à chaque fois des questions/réponses concluantes très rapidement (jusqu'à 10 secondes en utilisant le texte d'un PDF de 50 pages dans le prompt).

La PoC de l'IA est quasiment déjà valide, il faut juste vérifier que celle-ci est utilisable dans le cadre de notre application web. Il faut tout d'abord lier l'extracteur de PDF à notre modèle d'IA. Pour cela, nous allons nous servir de l'API de Ollama et de notre serveur personnel afin d'obtenir les sorties de notre modèle en tant que réponse JSON (cf. Extracteur PDF).

Il y'a plusieurs paramètres à préciser lors de notre requête POST vers l'API d'Ollama. Il faut préciser le nom du modèle utilisé, le prompt envoyé et le type de retour (stream). Dans notre cas, nous allons aussi préciser que la sortie sera au format JSON et préciser au modèle la forme du format JSON de sortie.

Après plusieurs requêtes POST tests, nous remarquons que le format JSON n'est pas toujours respecté par le modèle. Malgré tous nos efforts pour essayer de modifier ça, nous gardons quand même un taux de différence resultat attendu/ résultat réel assez élevé (⅓ d'après nos observations). Ceci dit, l'API de Ollama et le modèle sont quand même très rapide (voir ci-dessous).



On pourrait générer les questions une par une ce qui réduirait effectivement le taux de différence mais le problème resterait le même au final, puisque on a un taux plus faible mais sur plus de prompts. Il faut donc anticiper ce problème génératif à chaque réponse JSON du modèle pour pouvoir gérer ce cas (que faire ? regénerer ?).

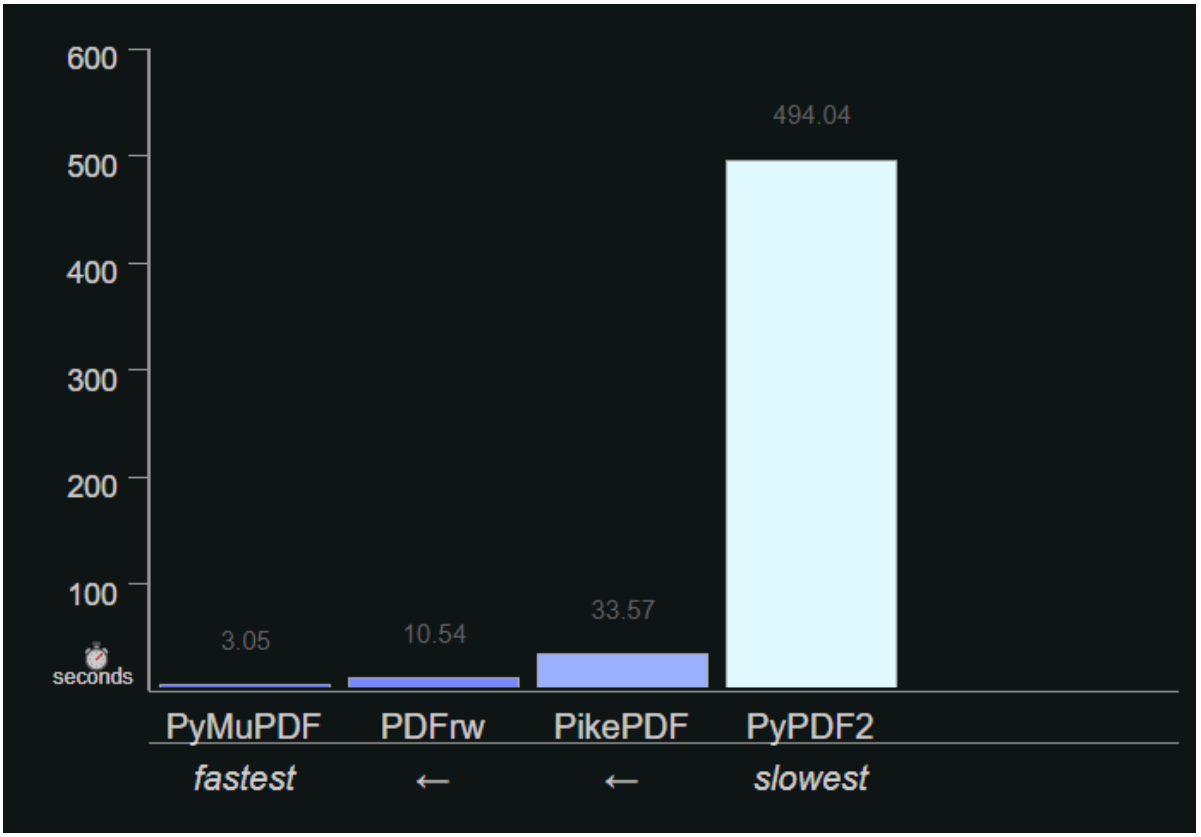
Extracteur de PDF vers du texte

L'idée est de pouvoir extraire des PDF vers du texte directement, et pour faire cela, nous avons utilisé le langage Python pour sa simplicité d'utilisation et pour ses nombreuses librairies disponibles pour faire la fonctionnalité.

Les librairies disponibles pour extraire du texte d'un PDF sont nombreuses dont : PyMuPDF, pikepdf, PyPDF2, pdfrw, pdfplumber.

Celle qu'on a sélectionné est PyMuPDF en raison de sa vitesse d'exécution et de compatibilité avec tout type de document PDF et la possibilité d'extraire des images (peut-être qu'on pourrait extraire des images pour les convertir en carte dans le futur)

Ci-dessous un graphique qui montre la vitesse d'extraction pour chacune des librairies.

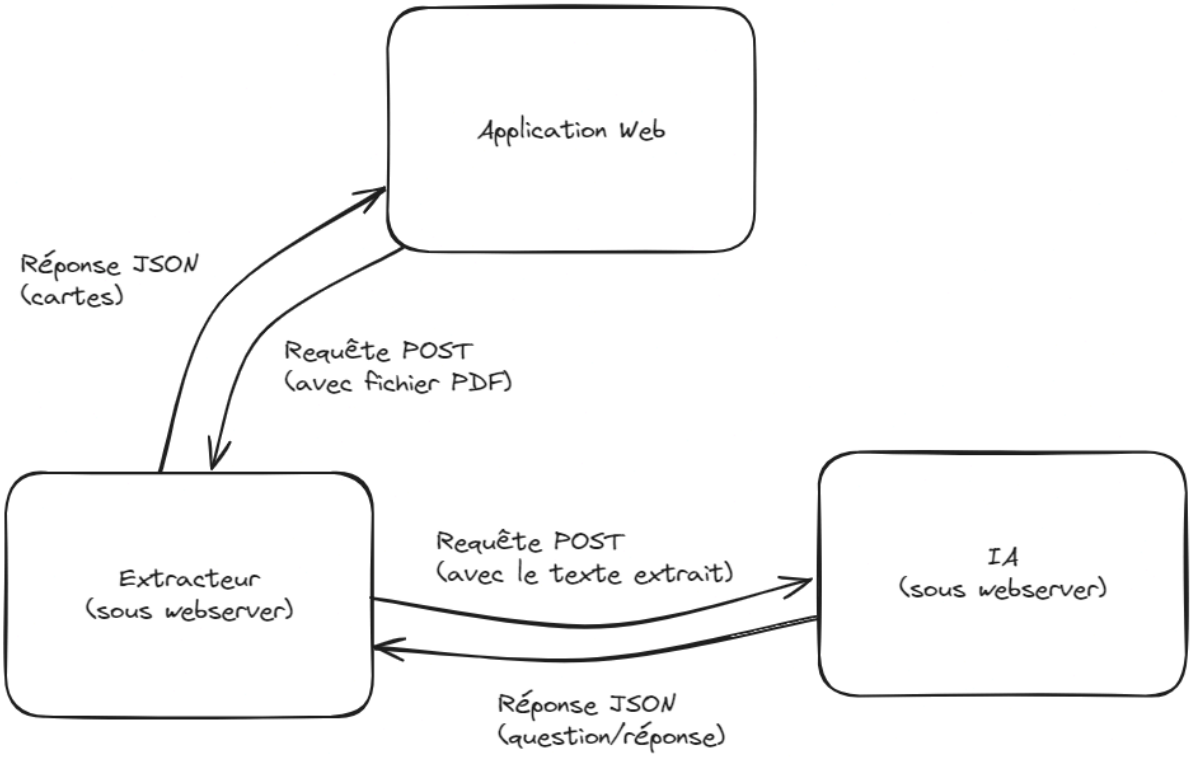


Après avoir sélectionné la librairie pour ce projet et l'avoir implémenté, nous avons déjà un cas concret où l'on peut extraire un PDF et qui nous retourne du texte, on a essayé sur plusieurs types de PDF de nos cours (espagnol, logique, optimisation…)

Les résultats sont concluants mais on remarque quelques bémol, mauvaise reconnaissance des caractères avec accent (‘e au lieu de é par exemple) et on a aussi remarqué que les en-tête ou pied de pages pouvait être embêtant en raison d'information inutile pour l'intelligence artificielle.

Ensuite, il fallait mettre en place un serveur Web pour pouvoir faire des requêtes HTTP vers notre application d'extracteur depuis l'application Web.

Voici un schéma qui représente notre infrastructure



Donc pour mettre en place le serveur Web pour l'extracteur de PDF, nous avons utilisé Flask (qui est un serveur Web léger en Python) qui permet de déployer rapidement un serveur Web

Nous avons donc mis en place un endpoint http://ip_extracteur_pdf/upload/
qui va prendre une requête POST et prend un fichier PDF en paramètre, après cela, le serveur Web va extraire le contenu du PDF avec PyMuPDF et va garder en mémoire le contenu texte pour enfin faire une requête vers le serveur Web de l'intelligence artificielle et qui retournera un JSON avec les questions/réponses. Enfin le serveur Web de l'extracteur retournera une réponse JSON avec toutes les questions/réponses sous le format d'une carte dans notre application Web.

Un exemple visuel du résultat

On choisit son PDF et on envoie :

Après extraction et génération des questions, le serveur Web nous renvoie :

{ "question1": "Qu'est-ce que l'algorithme de gradient descendante ?", "question2": "Comment choisir le pas d'apprentissage k dans l'algorithme de gradient descendante?", "question3": "Pourquoi la convergence peut être lente ou manquer si le pas d'apprentissage est trop petit ?", "question4": "Quelle est l'utilisation pratique de l'algorithme de gradient descendante ?", "question5": "Comment déterminer les limites de convergence pour l'algorithme de gradient descendante?" }

Cela conclut la première itération pour l'extracteur de texte. Les résultats sont prometteurs mais il faut régler le principal souci de réussir à extraire correctement les questions et réponses de l'IA et de le "formaliser" dans un format JSON. Car à chaque génération, l'IA change parfois la façon dont elle répond comme expliqué dans la partie précédente.

Architecture

Durant la première semaine d'itération du projet d'application de révision étudiante, l'architecture a été mise en place en utilisant Turbo Repo. Toutes les bibliothèques nécessaires ont été importées, y compris Tailwind, dotenv, mongo et mongoose. Ces bibliothèques ont permis de configurer l'environnement de développement, de gérer les variables d'environnement, et d'établir une connexion sécurisée avec la base de données MongoDB.

Base de donnée

La base de données MongoDB a été correctement configurée et connectée. Les données ont été vérifiées à travers Postman, confirmant ainsi que la connexion est opérationnelle. Le premier endpoint de l'API Next a également été créé pour démontrer la fonctionnalité de l'API. Ce premier endpoint a permis de valider que la connexion à la base de données fonctionne correctement, et a servi de base pour le développement ultérieur de l'API.

Application Web (Dead App)

Afin de simplifier la création de l'application ainsi que l'affichage des données statistiques, plusieurs librairies JavaScript ont été utilisées.

ShadCN est une librairie JS très populaire pour sa large galerie de composants de tous types allant du simple bouton au menu de navigation. Elle nous entre-autre servi à l'interactivité du partage de deck ainsi qu'à la disposition en carrousel des deck previews.

React-chartjs-2, comme son nom l'indique, est une librairie utilisant la librairie CharJs initiale afin de créer et manipuler efficacement des graphiques. Elle est grandement sollicitée dans la rubrique "statistiques" avec 3 de ses utilisations.

Après la forme, vient le contenu. Celui-ci est réparti sur 6 pages distinctes : login (connexion), contact, deck (réponse), new deck (création/modification), profil et statistiques. Chacune de ces rubriques respecte le fonctionnement react est utilise donc des composants dynamiques répertoriés dans un dossier parallèle. Par la suite, d'autres librairies devraient être ajoutées, notamment pour la rédaction des questions et réponses en utilisant du LaTeX (équation) et de l'UML (schéma).

Actuellement, l'application est en grande partie codée. Un retard sur les pages de statistiques et profil s'est fait remarquer, Ces dernières seront donc à finir lors de la prochaine itération.
De plus, la prochaine prévoit l'implémentation des classes métier User et Deck afin de définir leur structure et pouvoir les utiliser là où elles sont nécessaires. A celà s'ajoute la gestion des cartes entre chacune des pages ainsi que la résolution de certains bugs graphiques tels que le menu ou l'affichage de certaines images.

Récapitulatif

Pour cette première itération, nous avons pu compléter toutes nos tâches prévues et même au-delà sans trop de difficulté.

Nous avons pu réaliser la preuve de concept de l'IA, qui au départ semblait impossible, mais au bout de la première itération, nous avons réalisé qu'il était tout à fait possible d'implémenter la fonctionnalité d'extraction de PDF vers des cartes directement pour simplifier l'utilisation sur le processus de création (bien qu'il ne sera pas forcément pertinent pour chaque PDF).

On a pu également avancer très vite dans le développement du site Web en faisant toutes les pages en Front-end et nous allons pouvoir commencer à l'itération 2 l'implémentation de la logique dans l'application Web grâce à l'implémentation de la base de données.