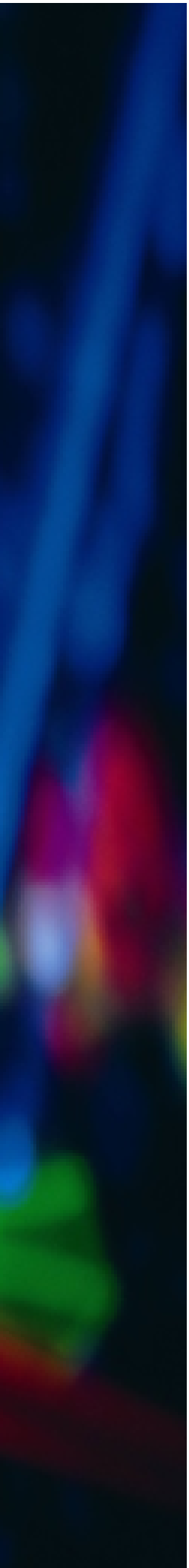


Développement de programme assisté par une IA local

Arman Behnam, Lyna Kammoun, Imène Zebiri

Université Paris Nanterre

15 Janvier 2024

Université Paris Nanterre

Contents

1 Introduction 2

2 Environnement de Travail 2

3 Réussites et avantages 2

4 Bibliothèques, outils et technologies 3

5 Descriptif de partie de code intéressante 3

6 Difficultés rencontrées 6

7 Répartition du travail 6

8 Conclusion 7

9 Mode d'emploi 7

1 Introduction

Notre passionnant projet repose sur l'idée ambitieuse de créer une intelligence artificielle, visant à générer des codes optimisés en réponse à des consignes utilisateur. Notre objectif ultime était de concevoir une interface conviviale permettant à tout utilisateur, même sans compétences approfondies en programmation, de recevoir des solutions de codage efficaces. Dans cette quête, nous avons intégré le modèle linguistique de pointe GPT-4 de OpenAI, pour guider les utilisateurs à travers divers langages de programmation.

Ainsi, notre choix de ce sujet n'était pas seulement guidé par la technicité, mais également par le désir de créer une solution pragmatique et innovante. Notre engagement à résoudre des problèmes concrets à travers des approches novatrices a été le moteur de notre participation à ce projet passionnant.

2 Environnement de Travail

Notre aventure de développement s'est déployée dans un environnement dynamique, alternant entre des séances individuelles chez nous et des rassemblements collaboratifs à la bibliothèque ou par appel vidéo. Cependant, notre parcours n'a pas été sans embûches, avec des défis inattendus dus à la compatibilité entre les systèmes MAC et WINDOWS, exigeant une adaptation continue des fonctionnalités pour assurer une expérience utilisateur homogène. Notre choix varié de langages pour la création du projet comprenait du CSS, JavaScript, HTML, Python, et TypeScript, avec Visual Studio Code en tant qu'environnement de développement.

La conception du projet s'est scindée en deux parties distinctes : la première partie dédiée au backend avec Flask, et la seconde axée sur le frontend avec Angular.

Tout au long de ce descriptif de notre projet, nous allons beaucoup parler de Backend et de Frontend. En effet le Frontend est ce que voit l'utilisateur, le code qui permet de créé l'interface etc... Et le Backend, vous l'aurez compris c'est tout ce qui touche à l'algorithme de la machine et aux différentes liaisons entre l'API et le Frontend.

3 Réussites et avantages

Au-delà de la flexibilité offerte par le choix de plusieurs langages de programmation, notre interface se distingue par sa capacité à répondre directement à la consigne utilisateur, éliminant ainsi la nécessité pour l'utilisateur de formuler explicitement un code optimisé. Cette fonctionnalité, élément clé de notre succès, simplifie significativement l'expérience utilisateur, la rendant accessible même aux novices en programmation.

L'avantage majeur réside dans la démystification du processus de codage. Notre IA propose des solutions optimales sans nécessiter une expertise préalable, contribuant ainsi à éliminer les barrières à l'entrée dans le monde de la programmation. Cette accessibilité accrue s'aligne sur notre objectif initial de rendre la programmation plus inclusive et compréhensible pour un public plus large.

En considérant le choix stratégique d'une interface épurée, nous avons mitigé les risques potentiels liés à des réponses excessivement longues ou à des problèmes de performance. Cette décision délibérée de simplicité s'est révélée cruciale pour garantir une expérience utilisateur fluide et sans heurts.

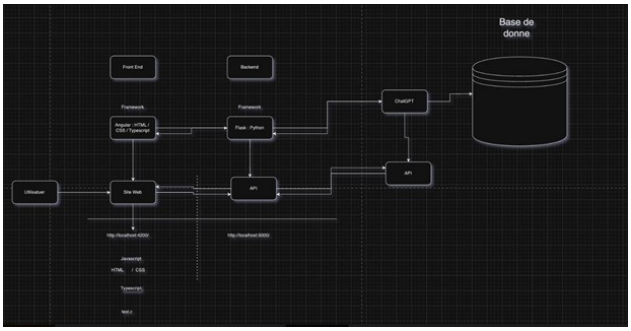
En résumé, les réussites de notre projet découlent de notre engagement à créer une solution accessible, pragmatique et novatrice. Les avantages offerts par notre interface démontrent son potentiel à transformer la façon dont les individus abordent la programmation, en simplifiant un processus autrefois complexe et intimidant.

4 Bibliothèques, outils et technologies

Le frontend a été sculpté avec Angular, fournissant une structure solide et une gestion fluide des composants. Le backend, quant à lui, s'appuie sur Flask, un framework Python, pour une gestion efficace des requêtes HTTP. L'IA tire sa puissance du modèle GPT-4 de OpenAI, spécialisé dans le traitement du langage naturel, capable de comprendre et générer des réponses en français.

Dans notre quête pour intégrer une intelligence artificielle puissante, nous avons initialement exploré Pinokio, une solution novatrice pour tirer parti de Chat GPT-4. Cependant, au fil de notre progression, nous avons découvert une API directe pour Chat GPT-4, simplifiant notre intégration et éliminant la nécessité de passer par Pinokio. Ce choix stratégique nous a permis de créer un lien direct entre notre interface web et l'API de Chat GPT-4, facilitant ainsi la communication fluide entre l'utilisateur et le modèle linguistique de pointe. Nous avons pu avoir accès à l'API grâce à un proche de Lyna qui utilise régulièrement, pour son travail d'ingénieur informatique, l'API de Chat GPT-4.

Voici notre schéma explicatif :



Cela souligne l'importance de rester adaptable et prêt à ajuster nos choix technologiques en fonction des découvertes et des évolutions du projet. Notre passage de Pinokio à l'API Chat GPT-4 a contribué à optimiser notre processus de développement et à améliorer la performance globale de l'IA dans notre interface utilisateur. Nous avons utilisé ce schéma pour communiquer entre nous et mettre nos idées aux clairs.

5 Descriptif de partie de code intéressante

Voici quelques passages de code que nous avions trouvés intéressants à détaillés :

La méthode sendMessage() dans la classe AppComponent orchestre l'envoi des messages utilisateur à l'IA et la gestion des réponses.

```
sendMessage() {
  // ... (code précédent)

  this.mainService
    .ask(finalRequest)
    .pipe(take(1))
    .subscribe((res) => {
      // ... (code suivant)
    });
}
```

Explication : Cette méthode encapsule le processus d'envoi de messages à l'IA, sollicitant une réponse et la traitant de manière appropriée.

La méthode ask() dans le service MainService gère l'envoi de requêtes à l'API Flask, la gestion des réponses, et contrôle les états de chargement.

```
ask(request: string): Observable<string> {
  // ... (code précédent)

  return this.http.post(apiUrl, requestBody, httpOptions).pipe(
    takeUntil(this.onDestroy$),
    map((value: any) => {
      return value['response'] as string;
    }),
    catchError((err: any) => {
      console.error(err);
      return 'Erreur lors de la récupération des informations.';
    }),
    finalize(() => {
      this.isLoading.next(false);
    })
  );
}
```

Explication : Cette méthode encapsule le processus de requête HTTP vers l'API Flask, la gestion de la réponse, et la mise à jour des états de chargement.

La méthode improveResponse() dans AppComponent affine la réponse initiale en sollicitant des optimisations ou des corrections.

```
improveResponse(request: string, response: string) {
  // ... (code précédent)

  this.mainService.improveResponse(request, response).subscribe({
    next: (value) => {
      // ... (code suivant)
    },
    error: (err) => {
      // ... (code suivant)
    },
  });
}
```

Explication : Cette méthode initie une demande d'amélioration de la réponse initiale en sollicitant des ajustements.

La fonction formatResponse() transforme la réponse de l'IA en un format lisible avec des balises HTML pour la mise en page.

```
formatResponse(res: string): string {
  // ... (code précédent)

  return res
    .replace(/\\n/g, '<br/>')
    .replace(/```(.*?)```/gs, function (match, p1) {
      // ... (code suivant)
    });
}
```

Explication : Cette fonction prend une réponse de l'IA, effectue des transformations pour rendre le texte plus lisible et formaté avec des balises HTML.

La méthode setActiveProgrammingLanguage() dans le service MainService permet de définir le langage de programmation actif.

```
setActiveProgrammingLanguage(language: string) {
  // ... (code précédent)

  if (this.programmingLanguages.includes(language)) {
    this.activeProgrammingLanguageSubject.next(language);
    localStorage.setItem('lang', language);
  }
}
```

Explication : Cette méthode gère le changement du langage de programmation actif, s'assurant que la sélection est valide et met à jour le sujet observable correspondant.

La fonction ask() dans le fichier app.py du backend reçoit les requêtes de l'interface frontend et interagit avec le modèle GPT-4.

```
@app.route("/ask", methods=["POST"])
def ask():
  data = request.get_json()
  user_input = data.get("message")

  completion = client.chat.completions.create(
    model="gpt-4",
    messages=[
      # ... (code précédent)
    ]
  )

  ai_response = completion.choices[0].message.content
  return jsonify({"response": ai_response})
```

Explication : Cette route Flask gère les requêtes POST de l'interface frontend, extrait le message utilisateur, et utilise le modèle GPT-4 pour générer une réponse.

La méthode ngOnDestroy() dans le service MainService s'occupe du nettoyage lorsque le service est détruit.

```
ngOnDestroy(): void {
  this.onDestroy$.next();
  this.onDestroy$.complete();
}
```

Explication : Cette méthode s'assure que les ressources du service sont correctement libérées pour éviter les fuites de mémoire lors de la destruction.

La fonction formatResponse() utilise des expressions régulières pour formater la réponse de l'IA, en remplaçant les retours à la ligne et en encadrant le code dans des balises HTML.

```
formatResponse(res: string): string {
  return res
    .replace(/\\n/g, '<br/>')
    .replace(/```(.*?)```/gs, function (match, p1) {
      var escapedCode = p1.replace(/</g, '&lt;').replace(/>/g, '&gt;');
      escapedCode = escapedCode.replace(
        /^(\w+)/,
        '<div class="code-language">$1</div>'
      );
      return '<pre><code>' + escapedCode + '</code></pre>';
    });
}
```

Explication : Cette fonction complexe utilise des expressions régulières pour formater la réponse en remplaçant les retours à la ligne et encadrer le code dans des balises HTML, offrant une sortie lisible et bien structurée.

Au cœur de notre code se trouve une étape particulièrement intrigante qui donne une dimension interactive unique à notre projet (donné par vous). Après que l'utilisateur a soumis une consigne de code, Chat GPT génère une réponse, mais ce n'est pas la fin de l'histoire. La fonction (NUK), présente à l'IA la consigne initiale avec le code généré.

Cette étape est cruciale pour l'optimisation du code. Cette boucle interactive, intégrée habilement dans notre code, définissant notre projet comme bien plus qu'une simple interaction avec une intelligence artificielle.

Fonctionnement : on entre une consigne de code précise à Chat GPT qui récupérer la réponse puis (NUK)* redonne la consigne + le code générer et demande si c'est ok, alors (si c'est ok) se demande comment l'améliorer...

6 Difficultés rencontrées

Au cours de notre périple, nous avons rencontré des défis significatifs, notamment l'harmonisation des fonctionnalités entre les environnements MAC et WINDOWS, nécessitant une adaptation constante. La coordination entre le frontend et le backend a également nécessité des ajustements, mais une communication transparente et une collaboration étroite ont permis de surmonter ces obstacles.

Des difficultés inattendues ont émergé, notamment la gestion des réponses générées par l'IA, parfois un peu longues, pouvant entraîner des ralentissements ou des problèmes de performance sur les ordinateurs. Cette complexité pourrait potentiellement conduire à des réponses incorrectes, soulignant ainsi les limites des capacités de traitement de l'IA dans le contexte de l'interface utilisateur. Ces complications ont nécessité une approche prudente dans la conception de l'interface pour éviter toute complication majeure. Par conséquent, nous avons opté pour une interface épurée, mettant l'accent sur la simplicité, pour minimiser les risques de complications lors de l'interaction avec l'IA.

De plus, nous avons rencontré un défi intéressant en conceptualisant la boucle pour optimiser le code. Initialement envisagée comme une boucle qui ne s'arrêtait que si le programme était optimisé au maximum, nous avons rapidement réalisé que cette approche pouvait potentiellement devenir trop complexe et en générale dès la septième tentative d'optimisation le code devenait faux.

En raison des contraintes de temps liées à nos cours et examens, nous avons opté pour une interface simple afin de prioriser le développement approfondi du code, qui s'est avéré être la phase la plus compliqué et exigeante du projet.

Afin de maintenir la convivialité de l'interface, nous avons décidé de limiter la boucle à un maximum de 5 répétitions. Cette solution équilibre la recherche d'une optimisation continue tout en évitant une complexité excessive qui pourrait dérouter les utilisateurs. Cette adaptation témoigne de notre engagement à trouver un équilibre entre l'optimisation du code et la simplicité d'utilisation de l'interface.

7 Répartition du travail

L'équipe a travaillé de manière cohésive, dans la répartition du travail, chaque membre de notre équipe a apporté une contribution unique, créant une synergie collaborative essentielle au succès du projet.

Lyna a été la force motrice du développement, plongeant profondément dans la logique et la structure du code. Son expertise technique a été cruciale pour la mise en place du backend et du frontend, garantissant une intégration harmonieuse des différentes composantes du projet.

Arman a assumé le rôle de rédacteur en chef, concentrant son énergie sur la création du document final.

Sa capacité à articuler clairement les concepts techniques en langage accessible a grandement contribué à la compréhension globale du projet.

Imene a dirigé l'effort sur la gestion de LaTeX, s'assurant que la documentation était non seulement technique mais également esthétiquement cohérente et professionnelle. Sa précision dans la mise en page a apporté une valeur ajoutée à la présentation globale.

Cependant, malgré ces rôles principaux, notre méthode de travail a encouragé la polyvalence. Chaque membre a eu l'opportunité de s'immerger dans divers aspects du projet, du codage à la rédaction et à la gestion des outils LaTeX. Cette approche a créé un environnement collaboratif où les compétences individuelles ont été partagées et développées.

En somme, la répartition du travail a été pensée de manière à capitaliser sur les forces individuelles tout en favorisant l'apprentissage mutuel. Chacun de nous a contribué à la réalisation globale du projet, démontrant ainsi une collaboration harmonieuse au sein de notre équipe.

8 Conclusion

En conclusion, notre projet d'IA pour l'optimisation de code représente une convergence réussie de technologies modernes, de méthodologies flexibles et d'une répartition des tâches efficace. L'interface résultante offre une solution intuitive et accessible, promettant une utilisation généralisée dans le domaine de la programmation. Notre engagement envers l'innovation et la résolution de problèmes complexes transparaît dans cette réalisation technologiquement avancée et collaborativement construite. Ce projet ouvre des perspectives passionnantes pour l'avenir de l'assistance à la programmation.

L'idée de développer une IA capable de comprendre les consignes utilisateur pour générer des solutions de codage optimisées a immédiatement suscité notre intérêt. Nous avons vu l'opportunité de rendre la programmation plus accessible à un public diversifié, y compris ceux qui n'ont pas une expertise approfondie en codage. Ce projet alliait la complexité technique à une application concrète, en aspirant à simplifier le processus de codage et à fournir des réponses optimales à partir de consignes simples.

9 Mode d'emploi

Dans un premier temps, nous allons activer le Backend grâce aux instructions suivantes :

- Extraire le fichier zip.
- Ouvrir le dossier Backend (De préférence VS Code).
- Installer Python (Python 3 de préférence)
- Créé l'environnement virtuel, tapez : python -m venv venv
- Activer python environnement virtuel, tapez : source venv/bin/activate
- Installer les packages Python, tapez : pip install -r requirements.txt
- Exécuter : Python3 app.py

Nous allons maintenant faire fonctionner notre Maître Code (désolé on n'avait pas d'inspiration :)) :

- Ouvrir le dossier Frontend (de préférence dans Visual Studio Code.)

- Installer nodejs.org
- Ouvrir le terminal
- Tapez : npm install
- Tapez : npm run start

Instructions pour la configuration du Backend :

1. Ouvrir le dossier Backend :
   - Accédez au dossier backend dans le répertoire de votre projet.
2. Installer Python :
   - Assurez-vous que Python 3 est installé sur votre système. Vous pouvez le télécharger depuis python.org.
3. Créer un environnement virtuel :
   - Exécutez `python3 -m venv venv` pour créer un environnement virtuel nommé 'venv'.
4. Activer l'environnement virtuel :
   - Sous Windows, exécutez `venv\Scripts\activate`.
   - Sous macOS/Linux, exécutez `source venv/bin/activate`.
5. Installer les packages Python requis :
   - Exécutez `pip install -r requirements.txt` pour installer toutes les dépendances listées dans le fichier `requirements.txt`.
6. Lancer l'application :
   - Exécutez la commande `python3 app.py` pour démarrer le serveur backend.

Instructions pour la configuration du Frontend :

1. Ouvrir le dossier Frontend :
   - Utilisez de préférence Visual Studio Code comme éditeur de code pour gérer vos fichiers frontend.
2. Installer Node.js :
   - Téléchargez et installez Node.js depuis nodejs.org.
3. Ouvrir l'éditeur de code :
   - Lancez votre éditeur de code et ouvrez le dossier frontend.
4. Installer les dépendances du projet :
   - Exécutez `npm install` pour installer tous les packages Node.js nécessaires listés dans votre fichier `package.json`.
5. Démarrer l'application Frontend :
   - Exécutez `npm run start` pour lancer le serveur de développement frontend.