

Projet 3 – Explorer une technologie à l'aide de l'IA

Yassine Adibe

Technologies explorées

Ce projet porte sur l'intégration de trois technologies complémentaires pour créer une application conversationnelle d'IA :

- **OpenAI API** : Interface pour accéder aux modèles de langage GPT (gpt-4o-mini, gpt-4o, gpt-3.5-turbo)
- **LangChain** : Framework Python pour construire des applications avec gestion avancée de mémoire conversationnelle
- **Streamlit** : Framework pour créer des interfaces web interactives en Python

Application développée

Une mini application web conversationnelle avec les fonctionnalités suivantes :

- **Interface** avec un design de style Windows Vista
- **Streaming des réponses Streamlit** pour l'affichage progressif en temps réel
- **4 modes de conversation LangChain** : Général, Tuteur, Analyste, Créatif
- **Gestion de mémoire avancée** avec InMemoryChatMessageHistory
- **Export multi-format** en TXT, Markdown et JSON
- **Comptage précis des tokens utilisées** avec tiktoken

Principales découvertes

OpenAI API

Streaming : Implémenté avec `stream=True`, retourne des chunks au lieu d'une réponse complète. L'affichage progressif améliore considérablement l'expérience utilisateur car on voit la réponse se construire mot par mot.

Gestion des coûts : L'utilisation de `gpt-4o-mini` (~\$0.15 par million de tokens) permet de minimiser les coûts. Le projet complet a coûté 8 USD afin de remplir la balance.

Comptage des tokens : Le streaming ne retourne pas les tokens directement. L'utilisation de `tiktoken` (bibliothèque OpenAI) permet un comptage précis au lieu d'estimer avec "4 caractères ≈ 1 token".

Streamlit

Gestion d'état : `st.session_state` permet de maintenir des données entre les interactions. Streamlit réexécute tout le script à chaque interaction, donc l'état de session est important.

Composants natifs : `st.chat_message` et `st.chat_input` simplifient grandement la création d'interfaces conversationnelles comparé à d'autres frameworks.

CSS personnalisé : Streamlit utilise des composants Baseweb difficiles à styler. L'utilisation de sélecteurs `[data-testid]` via l'inspecteur de navigateur est nécessaire pour un contrôle fin du design.

LangChain

Architecture moderne : LangChain a migré vers une nouvelle architecture. L'ancien `langchain.chains` est déprécié - il faut utiliser `langchain_core` avec `ChatPromptTemplate` et `RunnableWithMessageHistory`.

Gestion de mémoire : `InMemoryChatMessageHistory` avec un store global permet de gérer plusieurs sessions. Chaque conversation a un `session_id` unique pour retrouver son historique.

Templates de prompts : En modifiant le prompt système, on change complètement le style de réponse. Par exemple, le mode tuteur explique avec des exemples concrets, tandis que le mode analyste organise l'information en points clés.

Trade-off streaming vs mémoire : Il est difficile de combiner le streaming avec LangChain. J'ai donc créé une checkbox qui permet de choisir entre streaming (OpenAI direct) ou LangChain (mémoire avancée).

L'apport de l'IA dans le processus

Utilisation stratégique

L'IA (Claude d'Anthropic) a été utilisée comme partenaire de développement tout au long du projet, avec un total d'environ 20 heures de travail réparties sur 3 parties.

Exemples de prompts efficaces

1. **"Comment implémenter le streaming dans Streamlit avec OpenAI?"**
Code avec stream=True et affichage progressif
2. **"Recréer le style CSS du site <https://frutigeraeroarchive.org/>"**
CSS complet style Windows Vista avec glassmorphism et gradients
3. **"Créer un ChatManager avec LangChain pour gérer plusieurs modes"**
Classe complète avec templates et mémoire
4. **"Créer un système d'export multi-format"**
Fonctions TXT, JSON, Markdown avec download buttons

Code généré vs code adapté

- Streamlit : 60% généré, 40% adapté (correction bug assistant_response, full_response)
- Design CSS : 50% généré, 50% personnalisé (ajustements couleurs, effets glow)
- Export : 30% généré, 70% codé/ajusté (timestamps, formats)
- LangChain : 80% généré, 20% adapté (migration imports obsolètes)

Débogage avec l'IA

Plusieurs bugs ont été résolus rapidement :

- Erreur "assistant_response is not defined" : Solution en 2 minutes (variable renommée)
- Slider avec fond vert : Identification de la classe CSS problématique
- ModuleNotFoundError langchain.chains : Migration vers langchain_core

Approche critique

Tout code généré a été lu, compris, testé et adapté. L'IA a accéléré le développement, mais la compréhension technique et la validation restent essentielles.

Réflexion critique

Points forts du projet

- **Interface professionnelle unique** : Le design Windows Vista se démarque des interfaces IA génériques
- **Architecture modulaire** : Le code est réutilisable et maintenable (chat_manager.py séparé)
- **Fonctionnalités complètes** : Au-delà d'un simple chatbot (4 modes, export, stats, streaming)
- **Documentation** : README, journal de bord, commentaires de code

Limites techniques

- **Mémoire simple avec LangChain** : Tous les messages sont gardés en mémoire, ce qui augmente les coûts sur de longues conversations.
- **Absence de persistance** : Les conversations sont perdues au rechargement de la page.
- **Pas de support de documents** : Impossible d'uploader un PDF ou un fichier pour que l'IA discute de son contenu.
- **Streaming incompatible avec LangChain** : Compromis, l'utilisateur choisit entre les deux modes.

Considérations éthiques

Sécurité : Les conversations sont envoyées aux serveurs externes de OpenAI, donc les utilisateurs doivent être conscients que leurs données transitent par des tiers.

Coûts : Le coût estimé est affiché en temps réel dans l'interface. Pour ce projet académique, 8 USD de balance est très raisonnable.

Dépendance à l'API : L'application nécessite une connexion internet constante et dépend d'un service tiers (OpenAI).

Potentiel et améliorations futures

Court terme :

- RAG (Retrieval Augmented Generation) pour chat avec documents
- Persistance des conversations dans une base de données
- Optimisation de la mémoire avec ConversationSummaryMemory

Long terme :

- Supports multi-utilisateurs avec authentification
- Support vocal avec micro
- Déploiement cloud (Sur Azure par exemple)
- Dashboard analytics avancé avec visualisations

Conclusion

J'ai pu explorer trois technologies qui se complètent bien : OpenAI API pour les modèles de langage, LangChain pour gérer la mémoire conversationnelle, et Streamlit pour l'interface web. Utiliser l'IA comme partenaire de développement a vraiment aidé dans mon projet. J'ai pu avancer beaucoup plus vite en posant des questions précises et en obtenant du code fonctionnel rapidement. Par contre, j'ai fait attention à comprendre le code généré et de le tester afin de pouvoir l'adapter à mes besoins.

Liens et informations

Dépôt GitHub : <https://github.com/Yassou12321/projet3AI>

Coût total : 8 USD (OpenAI API)

Technologies : Python 3.12, Streamlit 1.51.0, OpenAI API, LangChain 1.1.3