SAÉ – S6

Tableau Comparatif IAs & humain.

Membres :
Hocine Zared – Lacine Sanogo – Hichem Zenaini – Yacine Sellaoui

Phase 2 - Comparaison développement humain vs IA génératives

Prompt utilisé pour les 4 IA

“Je veux que tu développes une application complète qui génère un Diagramme de Voronoï

Cahier des charges :

- Lire un fichier texte contenant des points sous ce format (une paire par ligne) :

2,4

5.3,4.5

18,29

12.5,23.7

- Générer le diagramme de Voronoï à partir de ces points
- Visualiser le diagramme avec les points affichés par dessus
- Exporter le résultat en SVG et en image PNG
- Proposer une interface utilisateur conviviale pour charger le fichier

Exigences de qualité :

- Code bien structuré avec des fonctions séparées
- Docstrings sur chaque fonction
- Gestion des erreurs (fichier introuvable, format incorrect, moins de 2 points)
- Respect des bonnes pratiques PEP8
- Une suite de tests unitaires complète avec pytest couvrant toutes les fonctions
- Commentaires clairs dans le code

Contraintes techniques :

- Python uniquement
- Bibliothèques autorisées : numpy, matplotlib, tkinter, pytest
- Le code doit tourner sans erreur dès le premier lancement

Fournis le code complet, les tests séparés dans un fichier test_voronoi.py,

et explique brièvement les choix techniques que tu as faits”

Tableau comparatif

Critères	Phase 1 (sans IA)	Claude I.A	ChatGPT	Gemini	GrokCodeFast1
Temps de développement	5 jours (4-5 heures)	2 min	2 min	2 min	4 min
Temps de correction	5 jours	0 min	0 min	0 min	2 min
Code fonctionnel du 1er coup	NON	oui	Oui	Oui	Non (cahier des charges non respecté)
Qualité des tests unitaires	Aucun	Excellente	bonne	très bonne	très bonne
Compréhension du code produit	Totale	très claire	complexe	abstrait	très flou
Gestion des erreurs	Basique	excellente	bonne	très bonne	bonne
Qualité des commentaires	Naturels	Excellente	Bonne	Très bonne	Bonne

Conclusion

1. Comparaison Développement Humain vs IA

Ce projet nous a permis de comparer deux approches de développement très différentes. En phase 1, on a tout codé nous-mêmes sans aide, ce qui nous a pris beaucoup plus de temps mais on comprend vraiment chaque ligne de code qu'on a écrite. On a galéré sur certains points comme la normalisation des coordonnées mais au final on est capables d'expliquer tout ce qu'on a fait.

Ce qu'on retient c'est que l'IA est un bon outil pour aller vite quand on sait exactement ce qu'on veut. Mais sans avoir fait la phase 1 nous-mêmes, on n'aurait pas su écrire un bon prompt ni vérifier que le code produit était correct.

J'ai trouvé intéressant d'envoyer les 4 codes à une I.A (Gemini par exemple) afin qu'il garde le meilleur de chaque code et nous fasse un code ‘’final’’ respectant pleinement le cahier des charges ainsi que les bonnes pratiques d'un développeur je trouve ça intéressant .

2. Analyse de correction et Prompt Engineering

Sur toutes les I.A il n'y a que GROKCODEFAST1 qui ne nous à pas donné satisfaction car il a utilisé scipi -> Voronoi pour créer le diagramme, ce qui ne respecte pas notre cahier des charges : “Bibliothèques autorisées : numpy, matplotlib, tkinter, pytest” donc on a du lui dire :

“"from scipy.spatial import Voronoi" pourtant je t'ai dis : Bibliothèques autorisées : numpy, matplotlib, tkinter, pytest dans le cahier des charges””

Suite à cette remarque l'IA à supprimer cet ajout et nous à permit d'avoir un diagramme de Voronoï fonctionnel et respectant le cahier des charges.

En phase 2, on a remarqué que la qualité du résultat dépend beaucoup du prompt qu'on écrit, regardez par exemple quelque chose de tout bête, on a bien précisé à l'I.A qu'on voulait des fichiers tests nommés : “voronoi_test.py” ils ont respecté cette contrainte, mais comme nous n'avons rien dis par rapport au nom du main, on a des nom de main différents Gemini l'a appelé ‘’main.py’’ et les autres I.A ‘’voronoi_app.py’’ Avec un prompt précis et bien détaillé, les IA ont produit du code fonctionnel du premier coup, propre et avec des tests. Avec des prompts moins précis, il nous aurait fallu sans doute des corrections, néanmoins comme on à pu

voir avec Grok en utilisant le même prompt que pour les I.A lui à par contre omis un détail (plutôt important !) donc il faut toujours rester derrière l'I.A et ne pas l'utiliser bêtement. Ça nous a appris qu'utiliser une IA efficacement c'est aussi une compétence en soi et la qualité du premier prompt est très très primordiale, c'est ce qui permet de lui éviter d'aller dans tous les sens, et c'est d'ailleurs pour ça que le prompt engineering existe en soit.

3. Regard critique et risques identifiés

L'IA est un outil puissant, mais notre expérience met en lumière plusieurs risques majeurs :

Sans avoir fait la phase 1 nous-mêmes, nous n'aurions pas eu les connaissances requises pour rédiger un prompt robuste ni pour vérifier la validité de l'algorithme généré.

L'IA peut générer du code fonctionnel mais illisible pour l'humain, posant un réel problème de maintenabilité à long terme, si je veux améliorer le code je suis presque condamner de redemander à l'I.A d'ajouter la nouvelle fonctionnalité car moi, je n'aurais pas très bien compris le code.

Le coût environnemental : Utiliser l'I.A chaque jour dans un cadre professionnel ou scolaire, que des milliards d'utilisateurs fassent ça.. Nous allons faire face à un gros problème environnemental, prenons juste l'exemple de notre stratégie d'utiliser 4 IA différentes, puis de demander à Gemini de fusionner "le meilleur de chaque code" pour un résultat parfait , même si ça peut être très efficace techniquement ça n'est pas très éco-responsable ça peut être intéressant pour un développeur perfectionniste

Liens I.A :

Lien de la conv ''perfectionniste'' :
https://gemini.google.com/share/67a3dd51e54b
Lien conversation Gemini :
https://gemini.google.com/share/bbd0b2396b45
Lien conversation GrokFastCode 1 :
https://github.com/copilot/share/4839422e-01e4-8083-8140-7c0da496016c