**Option 1 : Communication via Flask et Node.js**

**Avantages :**

• **Flask (Python)** est particulièrement adapté pour interagir avec des API comme **Ollama**, notamment grâce à la richesse des bibliothèques Python pour la gestion des requêtes, du traitement de données, ou des modèles de machine learning.

• **Node.js** permet de gérer efficacement les interactions avec le frontend en temps réel grâce à son architecture orientée événements.

• Cette méthode permet de **découpler** les responsabilités : Flask gère les calculs ou traitements lourds, tandis que Node.js s’occupe du serveur web et de la distribution des données au frontend.

**Inconvénients :**

• Complexité accrue avec deux serveurs à maintenir (Flask pour le backend en Python et Node.js pour le serveur web).

• Latence potentielle dans la communication entre Flask et Node.js.

• Risque d’ajouter de la maintenance supplémentaire et de la gestion de l’interaction entre les deux serveurs.

**Recommandation :**

• Cette méthode est idéale si vous avez des **besoins complexes** côté backend (par exemple, pour des traitements LLM nécessitant Python), tout en gardant un serveur Node.js pour gérer les tâches frontend et la communication en temps réel.

**Option 2 : Tout faire en JavaScript avec Node.js**

**Avantages :**

• **Simplicité** : Un seul langage (JavaScript/TypeScript) à gérer à la fois pour le frontend et le backend. Cela réduit la complexité du projet.

• **Node.js** permet des performances élevées pour les applications à haute demande, comme les appels fréquents à des APIs.

• La communauté Node.js dispose de nombreuses bibliothèques qui facilitent le développement d’applications complètes.

**Inconvénients :**

• **Limitations de l’API Ollama** : Comme vous le soulignez, Ollama étant principalement orienté vers Python, utiliser Node.js pourrait rendre certains aspects du développement plus complexes, surtout si vous devez interagir avec des modèles LLM. Python reste plus riche en bibliothèques et outils pour gérer des LLM.

• **Gestion des LLM** : Le développement d’un backend LLM en JavaScript est moins mature et performant par rapport à Python.

**Recommandation :**

• Si votre application utilise **principalement JavaScript**, que vous ne faites pas de traitements trop lourds côté serveur et que l’API Ollama fonctionne bien en JavaScript, cette option peut fonctionner.

• Cependant, pour des tâches complexes comme la gestion des LLM, **Node.js pourrait être limité** par rapport à Python.

**Option 3 : Communication directe entre Flask et React**

**Avantages :**

• **Simplicité** : Une seule couche backend (Flask), directement connectée au frontend React. Il n’y a pas besoin de gérer deux serveurs distincts.

• **Efficacité** : Flask, en Python, est bien adapté pour interagir avec des API comme Ollama. Cela simplifie la gestion des LLM et réduit le besoin d’un serveur intermédiaire.

• **Moins de latence** : La communication directe entre Flask et React élimine les intermédiaires (Node.js), ce qui pourrait améliorer la vitesse de réponse pour l’utilisateur.

• **Réduction de la maintenance** : Vous n’avez qu’à maintenir un seul serveur backend (Flask).

**Inconvénients :**

• Flask n’est pas **orienté événements** comme Node.js, donc il pourrait être moins efficace pour des applications web qui nécessitent beaucoup de **temps réel** (par exemple, les websockets, les notifications instantanées, etc.).

• Cela pourrait rendre le **développement frontend** un peu plus complexe, car vous devrez probablement écrire plus de code pour la gestion des API RESTful directement entre Flask et React.

**Recommandation :**

• **Idéal** si vous souhaitez tirer parti des puissantes fonctionnalités Python pour gérer un **LLM** comme celui de l’API Ollama et que votre application n’a pas des besoins importants en termes de **communication en temps réel**. Flask est bien adapté pour des API RESTful et pour gérer des modèles de machine learning, tout en restant relativement simple.

# Proposition:

J’ai étudié en amont comment l’application fonctionnerait si nous communiquions directement avec l’API. Étant donné qu’il s’agit simplement d’une communication via des requêtes HTTP avec une API RESTful, il serait assez facile de l’intégrer directement avec Node.js en JavaScript. C’est pourquoi, en prévision de ce futur POC, je propose que nous travaillions avec deux serveurs : un serveur Node.js, et un serveur en Python qui simulerait le fonctionnement de l’API de ChatGPT en hébergeant notre modèle en local sur un serveur RESTful, pour lequel nous pourrions utiliser Flask. Cependant, il ne s’agirait pas ici d’une communication entre URLs où le serveur Flask aurait des droits d’écriture sur X et de lecture sur Y, et inversement pour le serveur Node.js.