الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université de 8 Mai 1945 – Guelma -

Faculté des Mathématiques, d'Informatique et des Sciences de la matière

Département d'Informatique

Mémoire de Fin d'études Master

Filière : Informatique

Option : Systèmes Informatiques

Thème :

Proposition d'un système de recommandation dans le Fog Computing dans un environnement IoT.

Présenté par : Bouagal Houssem Eddine

Membres du jury :

- **Président :** Dr. Guerroui Nadia
- **Encadreur** : Dr. Soussi Hakim
- **Examinateur** : Pr. Kouahla Zineddine

Juin 2025

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Liste des figures

Liste des figures

Liste des tableaux

Remerciements:

On dit souvent que le voyage est aussi important que la destination. Les cinq années passées à l'université m'ont permis de pleinement saisir la profondeur de cette phrase. Ce parcours, en effet, n'a pas été exempt de défis et de questions qui ont exigé de longues heures de travail pour trouver des réponses.

En terminant ce travail, je tiens à rendre grâce à Dieu Tout-Puissant, pour m'avoir donné la foi et m'avoir permis d'atteindre cette étape importante.

Je souhaite exprimer ma sincère reconnaissance à mes parents, pour leur soutien moral constant tout au long de mes années d'études, ainsi que pour tous les sacrifices qu'ils ont faits afin de me permettre de poursuivre mes études dans les meilleures conditions possibles. Leur encouragement indéfectible m’a donné la force de continuer.

Je tiens également à remercier profondément le Dr Hakim Soussi, pour avoir supervisé et guidé mes recherches. Ses conseils avisés, son soutien constant, ainsi que ses retours constructifs, ont été essentiels pour améliorer la qualité de mon travail et de ma thèse. Grâce à lui, j'ai beaucoup appris, tant sur le plan académique que personnel. Je n'oublierai pas non plus son aide précieuse pour la relecture et la correction de ma thèse.

Mes sincères remerciements vont également à ma famille, pour leur soutien moral sans faille, et à mes amis, pour leur soutien direct ou indirect. Un grand merci à chacun de vous.

Enfin, je souhaite adresser mes plus vifs remerciements à l'ensemble des professeurs du Département d'Informatique de l’Université du 8 Mai 1945 de Guelma, pour leur enseignement et leur accompagnement.

Dédicace :

À ma mère, dont le dévouement sans faille et le soutien inconditionnel ont été les fondements de mon succès. Par son amour infini, ses sacrifices et ses précieux conseils, elle a façonné mon parcours académique et m'a encouragé à poursuivre mes rêves. À travers ce travail, je souhaite lui exprimer ma gratitude éternelle pour sa présence constante dans ma vie et pour être ma source de force et d'inspiration.

À mon père, dont la persévérance et les sacrifices ont été une source inépuisable d'inspiration. Ses années de travail acharné et de dévouement ont ouvert la voie à mon succès. Que Dieu le protège, et que ce travail soit un témoignage de son soutien indéfectible et des valeurs nobles qui m'ont guidé tout au long de mon parcours.

Résumé :

Internet of things (IOT) est un domaine en plein ascension ces dernières années ; c'est un ensemble de dispositifs (capteurs) interconnectés dont le nombre ne cesse d'augmenter. Les données issues des différents capteurs sont traitées et stockées via le Cloud Computing. Etant donné la distance entre le Cloud et les capteurs (surtout ceux qui ont un besoin de données en temps réel) cela peut provoquer parfois un disfonctionnement. Pour pallier ce problème ; il y a le fog computing qui permet de traiter des données au plus près des dispositifs physique de l'IoT. Un système de recommandation placé dans le Fog Computing fonctionnera en périphérie du réseau (près des appareils utilisateurs) cela permettrai de réduire la latence, améliorant l'efficacité des recommandations en temps réel, en plus d'alléger la charge du cloud central. Le but de ce travail est de proposer un système de recommandation ainsi que de l'exécuter au niveau du Fog Computing et du Cloud, afin de comparer la Latence dans les deux cas.

Keywords : *IoT, Fog Computing, Machine learning*

ملخص:

إنترنت الأشياء (IoT) هو مجال يشهد نموًا سريعًا في السنوات الأخيرة، وهو عبارة عن مجموعة من الأجهزة (مثل الحساسات) المتصلة ببعضها البعض، ويزداد عددها باستمرار. تُجمَع البيانات من هذه الحساسات وتُعالَج وتُخزَّن باستخدام الحوسبة السحابية .(Cloud Computing) ونظرًا للمسافة بين السحابة والأجهزة (خصوصًا تلك التي تحتاج إلى استجابة آنية وفي الوقت الفعلي)، قد يؤدي ذلك أحيانًا إلى حدوث خلل أو تأخير في المعالجة.

وللتغلب على هذا المشكل، ظهر مفهوم **الحوسبة الضبابية(Fog Computing)** ، الذي يتيح معالجة البيانات بالقرب من الأجهزة الفيزيائية المرتبطة بإنترنت الأشياء. فعند وضع نظام توصية ضمن بيئة الحوسبة الضبابية ،فإنه يعمل على أطراف الشبكة قريبًا من أجهزة المستخدمين، مما يُسهم في تقليل زمن الاستجابة ، ويُحسن من كفاءة التوصيات في الزمن الحقيقي، بالإضافة إلى تقليل العبء على السحابة المركزية

ويهدف هذا العمل الى اقتراح نظام توصية وتنفيذه على مستويين: الحوسبة الضبابية والحوسبة السحابية ، من اجل مقارنة زمن الاستجابة (Latence) بين الحالتين .

الكلمات المفتاحية :

إنترنت الأشياء ، الحوسبة الضبابية ، التعلم الآلي.

Introduction Générale :

À l'ère de l'Internet des Objets (IoT), où des milliards de capteurs intelligents génèrent continuellement des données en temps réel, le traitement rapide et efficace de ces flux d'information est devenu une priorité. Le Cloud Computing, bien qu'efficace pour le stockage massif et l'analyse centralisée, montre ses limites lorsqu'il s'agit d'applications nécessitant une faible latence, notamment dans les environnements critiques comme la santé, les véhicules connectés ou les systèmes de surveillance.

Pour pallier à ces contraintes, le Fog Computing a émergé comme un paradigme intermédiaire entre les objets connectés et le Cloud. En rapprochant le traitement des données de leur source, le Fog permet une réponse plus rapide, une réduction de la charge réseau, et une meilleure efficacité énergétique. Dans ce contexte, intégrer un *système de recommandation* au niveau du Fog peut considérablement améliorer la prise de décision en temps réel, tout en conservant la possibilité d'un traitement à grande échelle dans le Cloud.

Ce projet vise à concevoir, implémenter et évaluer un système de recommandation capable de fonctionner dans un environnement Fog, tout en comparant ses performances avec un déploiement classique dans le Cloud. À travers cette approche, nous analyserons principalement l'impact sur la latence et l'efficacité du système, afin de démontrer les avantages pratiques du Fog Computing pour les recommandations contextuelles dans l'IoT.

Ce mémoire est structuré comme suit : Résumé, Introduction Générale, quatre chapitres principaux et Conclusion Générale :

- **Chapitre 1:** Ce chapitre introduit les notions fondamentales de l'IoT, du Cloud, du Fog Computing et des systèmes de recommandation, tout en dressant un état de l'art des recherches actuelles.
- **Chapitre 2:** Nous y détaillons l'analyse des besoins, la modélisation fonctionnelle, ainsi que l'architecture technique de notre solution.
- **Chapitre 3**: Ce chapitre présente les étapes pratiques de développement et de déploiement du système dans les deux environnements.

Chapitre 1 : **Internet des objets.**

1.1 Introduction :

L'IoT a complètement changé notre manière de vivre et de travailler avec les technologies numériques. Aujourd'hui, des milliards d'objets connectés crachent des données en permanence, et ça ne fait que grossir. Mais tout envoyer dans le cloud ? Pas si malin : ça rame à cause de la latence et ça sature les réseaux.

C'est là que le *Fog Computing* débarque comme une bombe. L'idée ? Traiter les données au plus près des objets connectés, plutôt que de tout balancer à l'autre bout du monde. Dans ce chapitre, on va plonger dans les bases de l'IoT, ses galères et comment le *Fog Computing* devient un acteur clé dans nos systèmes high-tech .

1.2 L'Internet des Objets (IoT) :

Définition 1:

L'IoT, c'est comme un grand réseau où tout ce qui t'entoure devient intelligent et connecté. Des capteurs, des caméras, des montres connectées, des machines en usine… tous ces objets communiquent entre eux en échangeant des données en temps réel grâce à Internet. En gros, n'importe quel objet peut maintenant collecter des infos, les envoyer et même les analyser. Un peu comme si ton frigo, ta voiture ou ton usine avaient leur propre cerveau numérique [1].

Définition 2 :

L'Internet des Objets, désigne un réseau d'objets physiques interconnectés via Internet, capables de collecter, d'échanger et de traiter des données sans intervention humaine directe. Ces objets qu'il s'agisse de capteurs, d'appareils électroménagers, de véhicules, ou même d'implants médicaux sont dotés de capteurs, de logiciels et de technologies leur permettant d'interagir avec leur environnement et avec d'autres systèmes connectés. Cette technologie vise à rendre les systèmes plus intelligents, plus efficaces et plus autonomes, en facilitant la prise de décision en temps réel à partir des données collectées [1].

Defintion 3 :

L'Internet des Objets *(IoT)* est un concept technologique désignant l'extension d'Internet au monde physique, à travers des objets capables de communiquer entre eux et avec leur environnement grâce à des connexions réseau. Ces objets, intégrés de capteurs, d'identifiants, de logiciels et d'intelligence embarquée, peuvent transmettre des données en continu, ce qui permet d'automatiser des processus, d'améliorer la performance des systèmes et de créer de nouveaux services intelligents dans des domaines variés tels que la santé, les transports, l'agriculture ou l'industrie. L'IoT transforme ainsi notre manière de vivre, de produire et d'interagir avec notre environnement [2].

1.3 Architecture d'un système IoT :

Une architecture IoT standardisée s'articule généralement autour des couches fonctionnelles suivantes :

- **Couche perception (ou capteurs)** :

  Cette strate assure l'acquisition des données physiques via des dispositifs tels que capteurs, étiquettes RFID ou actionneurs, jouant un rôle crucial dans la transformation des grandeurs mesurables en signaux numériques [3].

Couche réseau (ou transmission) :

Elle prend en charge la communication des données collectées vers les infrastructures de traitement, en s'appuyant sur des protocoles variés (Wi-Fi, 5G, LPWAN, etc.), garantissant ainsi une connectivité adaptée aux contraintes de latence, de bande passante et de consommation énergétique [3].

Couche traitement (ou analyse) :

Les données y sont traitées, soit dans des environnements cloud pour une scalabilité optimisée, soit en local *(Edge Computing)* afin de réduire la latence et préserver la confidentialité. Cette phase inclut souvent des mécanismes d'apprentissage automatique ou de fouille de données [3]**.**

Couche application (ou présentation) :

Interface finale dédiée à l'utilisateur, elle restitue les informations traitées sous forme de tableaux de bord, alertes ou commandes automatisées, répondant ainsi aux besoins métiers ou domestiques spécifiques.

Cette structuration modulaire favorise à la fois l'interopérabilité des composants et l'évolutivité du système, tout en intégrant des considérations critiques de sécurité et de gestion des ressources **[3] (Figure1.1), (Figure1.2) , (Figure1.3) .**

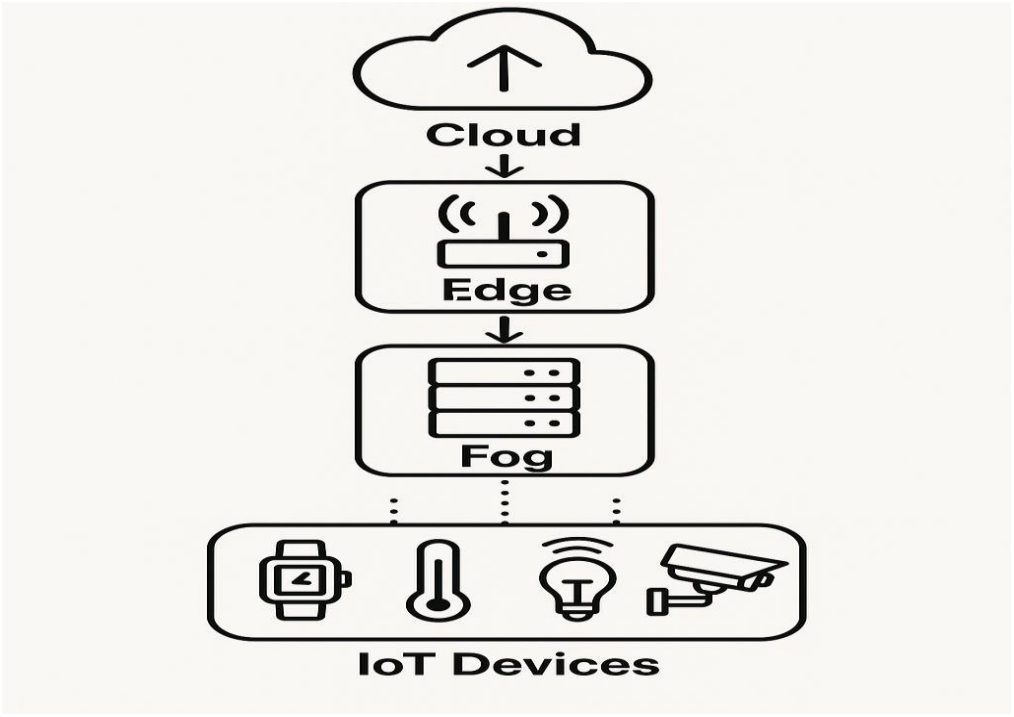


Figure 1.1 : Architecture IoT.

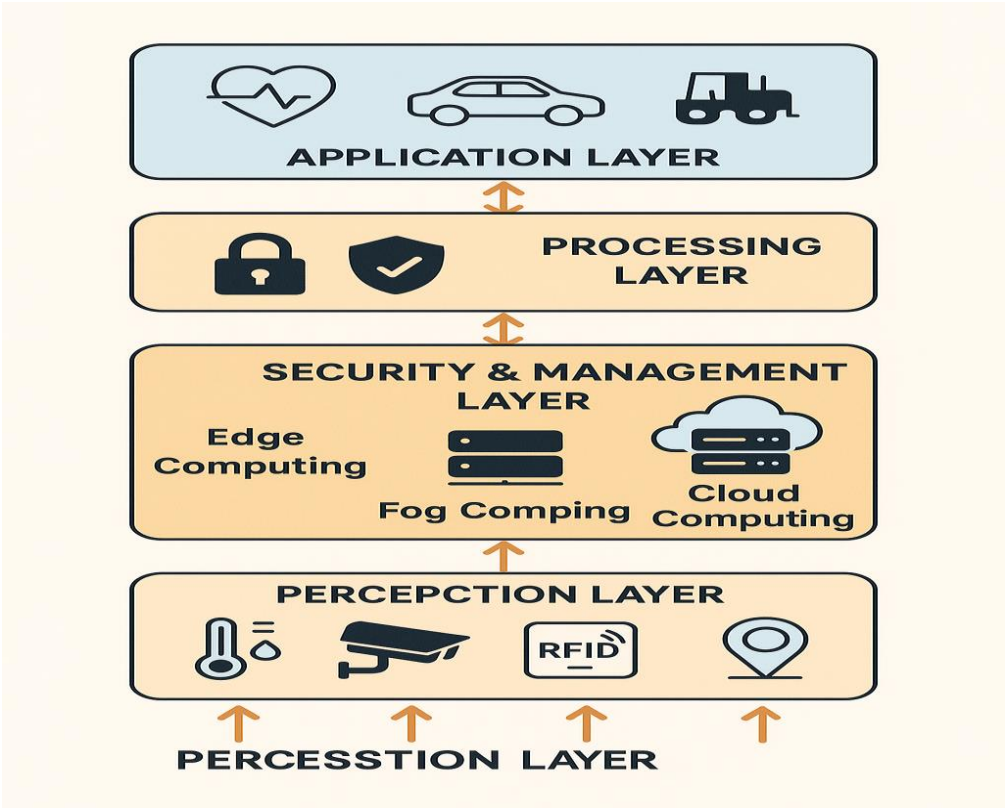


Figure 1.2 : autre architecture IoT.

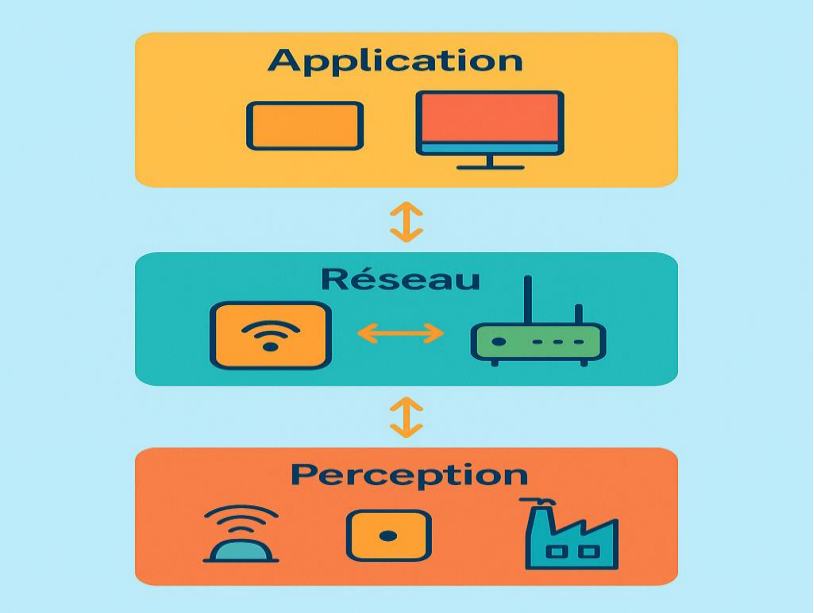


Figure 1.3 : autre architecture IoT

Cas d'usage :

- **Domotique (Smart Home)** : automatisation des lumières, sécurité.
- **Santé** : suivi médical à distance.
- **Agriculture intelligente** : irrigation optimisée par capteurs.
- **Transport** : gestion intelligente de la circulation.

1.4 Enjeux du traitement des données dans l'IoT :

Le traitement des données dans l'Internet des Objets (IoT) soulève des défis majeurs, tant sur le plan technique qu'en termes de performance et de sécurité. Ces enjeux deviennent d'autant plus critiques avec l'explosion du nombre d'objets connectés, générant un flux massif de données qui doit être traité de manière optimale et sécurisée. Trois problématiques principales émergent : la latence, la saturation des réseaux et la protection des données [4].

a. La latence :

Certaines applications IoT, notamment dans des secteurs sensibles comme la santé connectée, les véhicules autonomes ou l'industrie 4.0, requièrent une réactivité extrême, avec des temps de réponse de l'ordre de la milliseconde. Cependant, un traitement centralisé dans le Cloud peut introduire des délais inacceptables en raison de la distance entre les capteurs et les centres de données.

Pour répondre à ce besoin, des solutions comme l'Edge Computing – qui rapproche le traitement des données de leur source – permettent de réduire drastiquement cette latence. Cette approche permet une prise de décision extrêmement rapide, ce qui s'avère essentiel dans des situations où chaque fraction de seconde peut avoir un impact déterminant [3].

b. La surcharge réseau :

Avec des dizaines de milliards d'objets connectés prévus d'ici 2030, les infrastructures réseau subissent une pression croissante. La transmission massive de données provenant de capteurs, d'appareils intelligents ou de systèmes industriels peut entraîner des congestions, particulièrement dans les zones urbaines ou les environnements à forte densité d'équipements.

Pour atténuer cette saturation, l'adoption de protocoles adaptés (comme le LPWAN, optimisé pour les communications longue portée et basse consommation) et le déploiement d'architectures décentralisées (telles que le Fog Computing) permettent de limiter l'envoi de données vers les serveurs centraux. Ces stratégies améliorent l'efficacité du réseau et garantissent une meilleure qualité de service [5]**.**

c. Sécurité et confidentialité :

Les objets connectés génèrent une grande quantité de données sensibles : informations personnelles, données médicales ou encore localisation. Ces données sont essentielles pour faire fonctionner intelligemment les systèmes IoT, mais elles attirent aussi l'attention des cybercriminels. Vols, piratages ou fuites sont des risques bien réels. Le fait de stocker toutes ces informations dans le Cloud ne fait qu'augmenter leur vulnérabilité. C'est pourquoi il est indispensable de mettre en place des mesures de sécurité solides, comme le chiffrement des données, l'authentification à plusieurs niveaux ou encore un contrôle strict des accès.

Par ailleurs, la protection de la vie privée est encadrée par la loi, notamment par le Règlement Général sur la Protection des Données (RGPD). Celui-ci impose des règles précises sur la manière de collecter et d'utiliser les données personnelles. Cela oblige les concepteurs à intégrer la protection des données dès la phase de création des systèmes, selon le principe du privacy by design [6].

1.5 Edge Computing :

Définition :

L'Edge computing, ou « informatique en périphérie », désigne une approche du traitement des données qui consiste à déplacer les capacités de calcul et de stockage au plus proche des sources de données, c'est-à-dire directement sur les dispositifs connectés ou à proximité de ceux-ci. Contrairement au modèle traditionnel basé sur le cloud, où toutes les données sont transmises à des centres de données centralisés, l'edge computing permet un traitement local, ce qui réduit considérablement la latence, diminue la charge sur les réseaux, et renforce la confidentialité des données sensibles [7].

Cette approche est particulièrement pertinente dans le contexte de l'Internet des Objets (IoT), où des millions de capteurs et d'appareils génèrent en continu de grandes quantités d'informations. L'edge computing permet ainsi une prise de décision plus rapide et plus efficace, essentielle dans des domaines critiques tels que la santé, l'industrie 4.0 ou les véhicules autonomes [7].

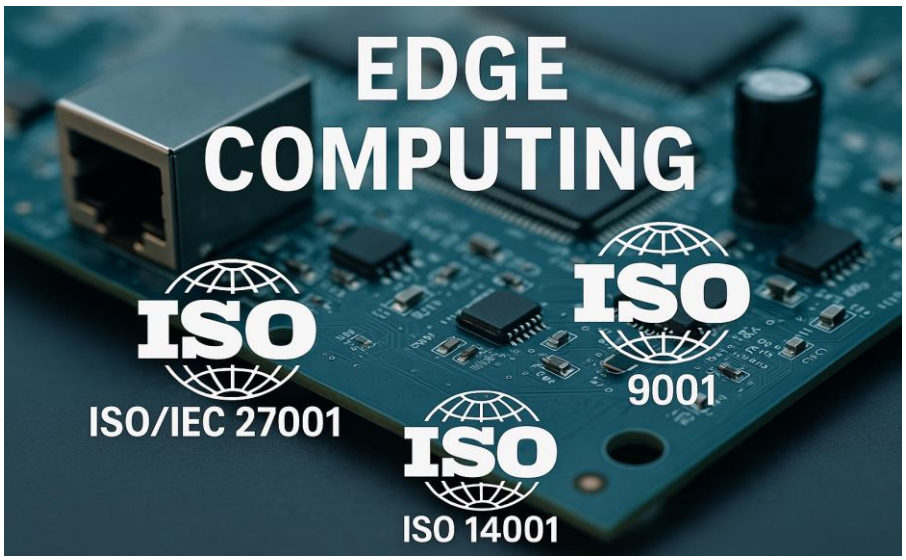


Figure 1.4 : architecture Edge Computing

1.6 Le Fog Computing :

Définition :

Le Fog Computing est une architecture informatique décentralisée qui déplace le traitement des données, le stockage et les services réseau à proximité des sources de données, c'est-à-dire près des appareils connectés eux-mêmes. Contrairement au *Cloud Computing* , qui repose sur des data centers distants pour centraliser l'ensemble des opérations, Le *Fog Computing* fonctionne à la périphérie du réseau *(edge)*, ce qui permet d'analyser les données et de prendre des décisions plus rapidement, en tenant compte des besoins locaux Cette approche réduit considérablement la latence, optimise l'utilisation de la bande passante et améliore l'efficacité opérationnelle, tout en restant complémentaire avec le Cloud pour les traitements nécessitant une grande puissance de calcul ou un stockage massif [8] , Figure (1.4).

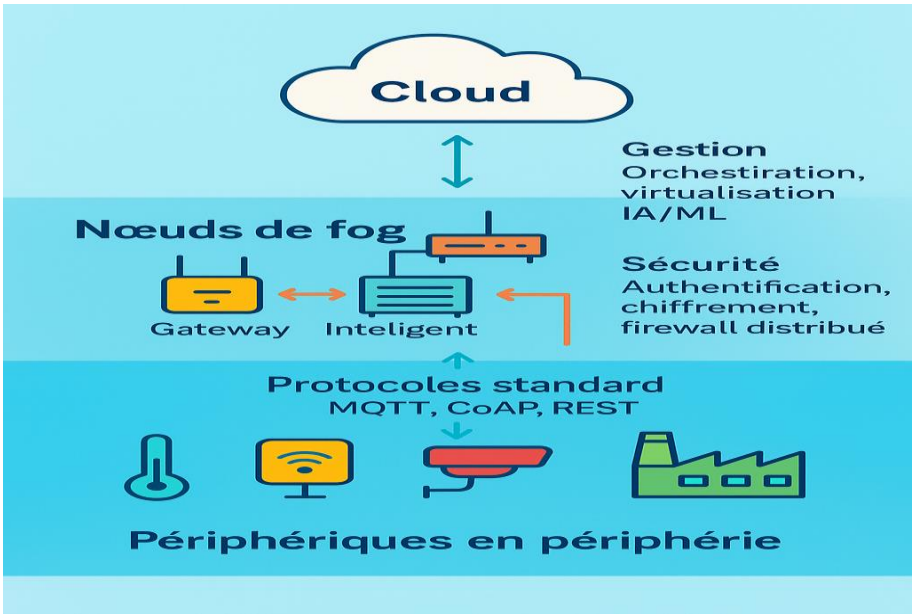


Figure 1.5 : Architecture du Fog Computing

1.7 Le rôle du Fog Computing dans les systèmes intelligents :

Le Fog Computing joue un rôle fondamental dans le développement et l'efficacité des systèmes intelligents. En rapprochant la puissance de calcul des sources de données, il permet de traiter les informations plus rapidement et d'optimiser l'utilisation des ressources réseau. Cette approche décentralisée, en complément du Cloud, favorise la réactivité, réduit la latence et offre des solutions adaptées aux contextes critiques. Les principales applications du Fog Computing se concentrent dans deux domaines stratégiques : les environnements critiques et les systèmes de recommandation [9].

- **Le Fog Computing dans les environnements critiques :**

a. Santé connectée :

Dans le secteur médical, la rapidité d'analyse des données est cruciale. Des capteurs, tels que les électrocardiogrammes (ECG), les glucomètres ou les tensiomètres, génèrent en continu des informations vitales qu'il est essentiel d'analyser sans délai. Le *Fog Computing* permet ce traitement local, directement à la source, sans passer par le Cloud, ce qui garantit une réaction immédiate [10].

Par exemple, un pacemaker intelligent peut ajuster son rythme de fonctionnement en temps réel grâce au traitement Fog, ce qui réduit considérablement le risque de complications potentiellement mortelles en cas d'arythmie ou d'hypoglycémie [10] , **Tableau (1.1)**.

Auteur(s)	**Année**	**Objectif principal**	**Contribution**	**Référence**
Bonomi et al.	**2012**	Introduire le concept de Fog Computing dans l'IoT	Définissent le Fog comme une solution pour le traitement en périphérie, en particulier pour l'e-santé	Bonomi et al. (2012)
Gia et al.	**2015**	Étudier l'utilisation du Fog pour l'analyse ECG dans des dispositifs médicaux portables	Présentent un système de détection en temps réel d'arythmie via Fog, avec réduction de latence	Gia et al. (2015)
Yi et al.	**2015**	Démontrer l'architecture Fog pour diverses applications, y compris la santé	Décrivent une plateforme Fog dédiée au traitement local des données issues de capteurs médicaux	Yi et al. (2015)
Rahmani et al.	**2018**	Intégrer Fog dans les systèmes eHealth pour une meilleure réactivité	Proposent une architecture Fog pour la détection précoce de maladies chroniques comme le diabète	Rahmani et al. (2018)

Tableau 1.1 : Quelques travaux dans le domaine santé [11].

b. Industrie 4.0 :

Dans le contexte industriel moderne, la surveillance en temps réel des équipements constitue un enjeu stratégique fondamental pour garantir la continuité des opérations, la sécurité des installations et l'efficacité des processus de production. Grâce à l'intégration croissante de capteurs intelligents, il est désormais possible de mesurer en continu divers paramètres physiques tels que les vibrations mécaniques, la température, la pression ou encore l'humidité. Ces capteurs génèrent une quantité importante de données qui sont transmises vers des nœuds de calcul en périphérie du réseau, appelés *Fog nodes*. Ces derniers jouent un rôle crucial en assurant un traitement local rapide des informations, permettant ainsi de détecter de manière proactive les signes avant-coureurs d'anomalies.

En effet, une analyse fine de ces données permet d'identifier précocement des comportements inhabituels comme une usure anormale de composants, une élévation excessive de la température ou des variations irrégulières de la pression, indicateurs potentiels de dysfonctionnements futurs. En réponse à ces signaux faibles, le système peut initier automatiquement des actions de maintenance préventive, évitant ainsi les pannes imprévues qui pourraient entraîner des arrêts coûteux de la production. Cette approche conditionnelle, reposant sur l'analyse prédictive, contribue non seulement à prolonger la durée de vie utile des machines et des équipements industriels, mais également à améliorer la planification des interventions techniques, à réduire les coûts de maintenance, et à optimiser les performances globales de la chaîne de production En somme, l'adoption de telles technologies intelligentes participe activement à la transition vers une industrie plus résiliente, efficiente et durable [10] , **Tableau (1.2)** .

Auteur(s)	Année	Objectif principal	Contribution	Référence
Zhou et al.	**2018**	Appliquer le Fog pour la surveillance industrielle en temps réel	Proposent un modèle pour la maintenance prédictive basé sur l'analyse locale des capteurs de machines	Zhou et al. (2018)
Wang et al.	**2017**	Réduire la latence dans les systèmes industriels intelligents	Développement d'une plateforme Fog pour la détection d'anomalies en production	Wang et al. (2017)
Kang et al.	**2016**	Intégrer le Fog à l'architecture d'Industrie 4.0 pour les usines intelligentes	Démontrent comment le Fog permet d'optimiser les opérations et réduire les interruptions	Kang et al. (2016)
Sadeghi et al.	**2015**	Renforcer la cybersécurité des capteurs industriels	Introduisent le Fog comme couche de sécurité pour contrôler localement l'accès aux données industrielles	Sadeghi et al. (2015)

Tableau 1.2 : Quelques travaux dans le domaine industrie [12].

c. Transports intelligents :

Dans le domaine des transports, le Fog Computing permet d'analyser les flux de circulation en temps réel grâce aux capteurs et caméras placés sur le réseau routier. Les informations traitées localement permettent, par exemple, d'adapter la synchronisation des feux de circulation, de prévenir les embouteillages ou de proposer des itinéraires alternatifs en fonction des conditions de circulation **[10]** , **Tableau (1.3).**

Auteur(s)	**Année**	**Objectif principal**	**Contribution**	**Référence**
Dastjerdi & Buyya	**2016**	Étudier le Fog Computing pour les systèmes de transport intelligents	Proposent une architecture Fog pour analyser le trafic en périphérie, avec faible latence	Dastjerdi & Buyya (2016)
Hou et al.	**2016**	Optimiser la synchronisation des feux de circulation via Fog	Présentent un système de contrôle adaptatif des feux basé sur le traitement local des données de circulation	Hou et al. (2016)
Yousafzai et al.	**2019**	Développer une plateforme Fog pour la gestion dynamique des transports urbains	Permet la redirection du trafic et la gestion des embouteillages en temps réel	Yousafzai et al. (2019)
Puliafito et al.	**2019**	Améliorer la qualité de service dans les smart cities grâce au Fog pour les véhicules connectés	Proposent un modèle de prétraitement local des données provenant des véhicules et infrastructures	Puliafito et al. (2019)

Tableau 1.3 : Quelques travaux dans le domaine transports intelligents [13].

1.8 Le Fog Computing au service des systèmes de recommandation :

a. Une analyse locale au plus près des besoins de l'utilisateur :

Les objets connectés du quotidien – smartphones, assistants vocaux, appareils domotiques – exploitent le *Fog Computing* pour analyser directement les comportements et préférences des utilisateurs, sans avoir systématiquement recours au Cloud.

Ainsi, un assistant vocal peut adapter ses recommandations musicales en fonction des habitudes d'écoute spécifiques à un utilisateur, tout en limitant la transmission des données sensibles vers des serveurs distants, préservant ainsi la confidentialité des informations personnelles [14].

b. Réduction de la saturation des réseaux :

En traitant et filtrant les données localement avant de les transmettre au Cloud, le Fog Computing contribue à réduire la charge sur les réseaux de communication.

Par exemple, dans un magasin intelligent, des balises Bluetooth peuvent analyser les parcours des clients en temps réel et n'envoyer au Cloud que des données agrégées et pertinentes, évitant ainsi des transferts inutiles de volumes importants d'informations [14].

c. Des recommandations plus rapides et contextuelles :

L'analyse locale permet également d'adapter les recommandations en fonction de critères contextuels comme l'heure, la position géographique ou l'environnement immédiat.

Ainsi, un musée équipé de capteurs et de nœuds Fog peut proposer des parcours personnalisés à ses visiteurs, en fonction de leur localisation précise dans l'espace et de leurs centres d'intérêt, tout cela sans délai d'attente et sans dépendre d'une connexion Cloud permanente

En somme, le *Fog Computing* se révèle être un levier stratégique pour améliorer la réactivité, l'efficacité et la pertinence des systèmes intelligents, en rapprochant le traitement des données de l'endroit où elles sont générées **Figure (1.5)** [14].

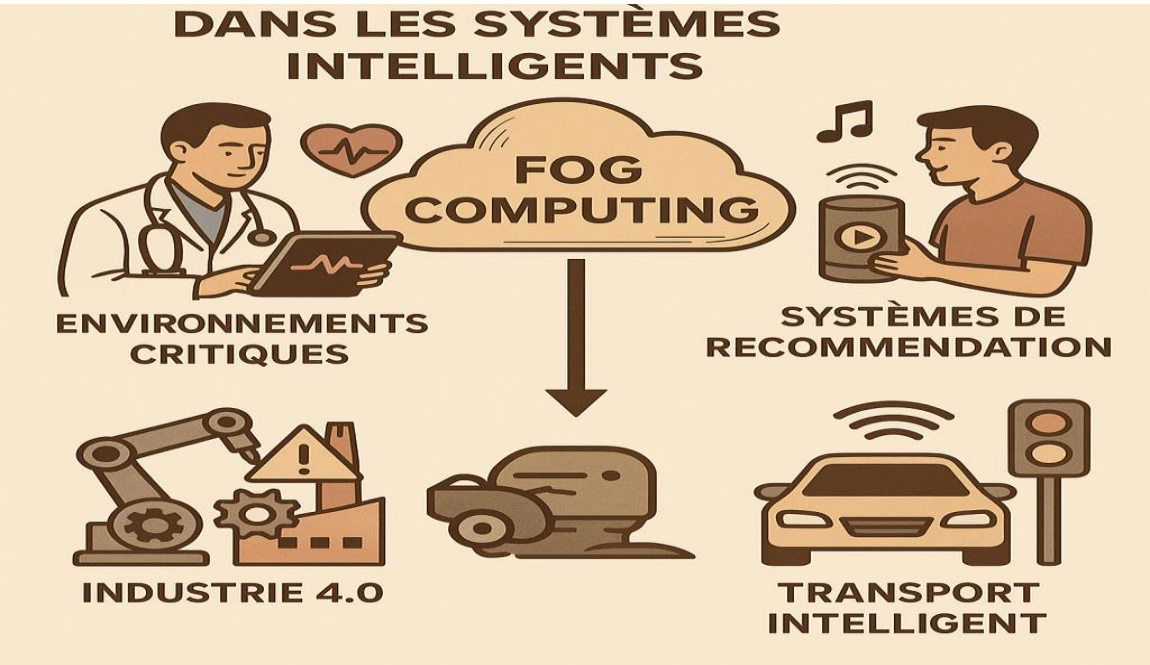


Figure 1.6 : Les domaines d'application du Fog Computing dans l' IoT

1.9 Limites et défis du Fog Computing :

Bien que le *Fog Computing* réduise la latence et decentralise le traitement, il présente des *limitations techniques* et matérielles. La diversité des infrastructures Fog *(routeurs, micro-datacenters, etc.)* crée une hétérogénéité difficile à maîtriser De plus, la *gestion des ressources distribuées* (CPU, mémoire, bande passante) est complexe et exige des algorithmes spécifiques pour adapter dynamiquement les capacités selon la charge

Sur le plan de la *sécurité et de la confidentialité*, les nœuds Fog multiplient les points d'accès, ce qui exige l'implémentation de mesures cryptographiques légères et robustes pour sécuriser les communications).

Enfin, l'*interopérabilité* reste un défi majeur : l'absence de standards unifiés rend l'intégration avec des systèmes hérités (legacy) difficile**.**

Les systèmes IoT présentent plusieurs vulnérabilités, notamment en matière de confidentialité, d'intégrité, de disponibilité et d'authentification Dans le cadre du *Fog Computing* , des mesures clés sont mises en place [16] pour répondre à ces enjeux :

- **Authentification mutuelle**.
- **Chiffrement léger** (AES, ECC, PRESENT).
- Gestion sécurisée des clés sur chaque couche.
- **Détection d'intrusions**, segmentation réseau et contrôle des accès (réduction des risques DoS, Sybil, Man-in-the-Middle)

Des solutions innovantes comme la *blockchain* sont explorées pour renforcer l'intégrité et la traçabilité des transactions Fog [15]**.**

Avantages et Inconvénients de Fog Computing :

Avantages	**Inconvénients**
Réduction de la latence : traitement plus proche de l'utilisateur final	**Complexité de gestion** : plusieurs nœuds décentralisés à superviser
Réduction de la bande passante : moins de données envoyées au cloud	**Sécurité plus difficile à assurer** : plus de points d'entrée potentiels
Temps de réponse plus rapide : idéal pour les applications temps réel	**Coût d'infrastructure plus élevé** : déploiement de matériel en périphérie
Meilleure autonomie : continue de fonctionner même sans accès au cloud	**Maintenance distribuée** : nécessite une expertise technique sur le terrain
Filtrage local des données sensibles	**Moins de puissance que le cloud** : ressources limitées dans les nœuds
Support des applications IoT : traitement local pour objets connectés	**Interopérabilité** : intégration difficile avec divers appareils et réseaux

Tableau 1.4 : Avantages et Inconvénients de Fog [16]**.**

1.10 Edge Computing vs Fog Computing *:*

Bien que les termes soient parfois confondus, l'Edge Computing se définit comme le traitement au plus près des capteurs (appareils, smartphones), tandis que le Fog Computing représente une couche intermédiaire équipée de capacités de calcul, de stockage et de communication [17]**.**

Technologie	**Avantages**	**Limites**
Edge Computing	• Latence minimale, adaptée aux applications en temps réel. • Simplicité grâce à moins de couches intermédiaires.	• Capacité de calcul limitée. • Dépendance à la qualité de la connectivité.
Fog Computing	• Orchestration des données provenant de multiples dispositifs Edge. • Traitement local plus puissant que l'Edge.	• Gestion plus complexe des clusters distribués. • Sécurité nécessitant des mécanismes renforcés.

Tableau 1.5 : Comparison Fog vs Edge [17].

1.11 L'intégration de l'intelligence artificielle dans le Fog Computing :

L'intégration de l'intelligence artificielle (IA) au sein des nœuds Fog représente une avancée significative pour rendre les systèmes IoT plus réactifs**,** autonomes et contextuels. L'intelligence artificielle offre la capacité de traiter en temps réel de vastes volumes de données directement au niveau local, d'en extraire des informations pertinentes, et de prendre des décisions immédiates sans dépendre d'une réponse provenant du Cloud [18]**.**

Des techniques comme **l'**apprentissage automatique (machine learning)**,** le deep learning, ou encore le fuzzy logic sont désormais embarquées dans les plateformes Fog pour anticiper les anomalies, ajuster dynamiquement les ressources ou encore adapter le comportement d'un système à son environnement.

Par exemple, dans les transports intelligents, l'IA intégrée aux nœuds Fog peut prédire les embouteillages ou accidents, en analysant localement les données des capteurs de circulation [19]**.**

1.12 Optimisation énergétique des systèmes IoT grâce au Fog Computing :

L'un des défis majeurs des systèmes IoT est leur consommation énergétique croissante, surtout en raison du transfert massif de données vers le Cloud. Le Fog Computing, en rapprochant le traitement des données de la source, permet de réduire considérablement les communications longues distances, entraînant ainsi une baisse de la consommation d'énergie réseau.

De plus, la possibilité de traiter uniquement les données pertinentes (filtrage, agrégation locale) diminue les besoins en stockage et en traitement global, ce qui contribue à une meilleure efficacité énergétique. Dans des contextes comme les réseaux de capteurs environnementaux ou agricoles, cette approche garantit un fonctionnement plus autonome et durable.

La durabilité est ainsi renforcée par ***l'****optimisation des ressources, la réduction de l'empreinte carbone numérique*, et la possibilité d'exploiter des sources d'énergie renouvelable en périphérie [20]**.**

1.13 Perspectives futures et tendances du Fog Computing :

Les perspectives du Fog Computing sont vastes et en pleine expansion, avec des tendances qui annoncent une évolution vers des systèmes *plus intelligents, autonomes et décentralisés*. L'une des directions les plus prometteuses est l'intégration de *blockchain* dans les nœuds Fog pour garantir la *traçabilité* et la sécurité des échanges de données sans dépendre d'une autorité centrale.

Par ailleurs, des recherches s'orientent vers des modèles de Fog Computing auto-adaptatif, capables d'ajuster dynamiquement leurs ressources, leur topologie et leurs politiques de sécurité selon les contextes.

Les futurs systèmes basés sur le Fog intégreront également des algorithmes *d'optimisation énergétique*, de *gestion distribuée de données sensibles*, et seront de plus en plus utilisés dans des domaines comme les *villes intelligentes*, les *réseaux électriques intelligents (smart grids)*, et la *télémédecine*.

Ces évolutions laissent entrevoir un futur dans lequel le *Fog Computing* formera le pilier essentiel des *systèmes cyber-physiques* de nouvelle génération [21]**.**

1.14 Conclusion:

L'Internet des Objets (IoT) est désormais une technologie omniprésente dans notre quotidien, mais ses performances restent étroitement liées à la manière dont les données sont traitées et analysées. C'est là qu'intervient le *Fog Computing* : en rapprochant les capacités de calcul des objets connectés, il apporte une solution concrète aux limites du Cloud traditionnel. Cette approche permet de développer des applications plus intelligentes, réactives et sécurisées, particulièrement dans des domaines sensibles comme les systèmes de recommandation. Ces derniers feront l'objet d'une analyse approfondie dans les chapitres suivants.

Chapitre 2 : Introduction aux Systèmes de Recommandation

2.1 Introduction :

À une époque marquée par une abondance massive d'informations, il devient de plus en plus difficile pour les utilisateurs de s'y retrouver et d'identifier ce qui correspond réellement à leurs besoins ou à leurs préférences. C'est dans ce contexte que les systèmes de recommandation (SR) jouent un rôle essentiel : ils permettent de filtrer les contenus disponibles pour proposer à chaque utilisateur des suggestions personnalisées, que ce soit pour des films, des livres, des produits, ou encore des services en ligne.

Ce chapitre a pour objectif de présenter les bases théoriques et techniques sur lesquelles reposent ces systèmes, en mettant l'accent sur la méthode adoptée dans notre projet : le filtrage basé sur le contenu (content-based filtering), combiné à l'algorithme des k plus proches voisins (K-Nearest Neighbors, ou KNN). Cette approche consiste à recommander des éléments similaires à ceux que l'utilisateur a déjà appréciés, en s'appuyant sur les caractéristiques des objets eux-mêmes.

Nous décrirons dans un premier temps la nature des données manipulées, puis les techniques de vectorisation utilisées pour représenter ces données de manière exploitable par un algorithme, notamment à l'aide de modèles de traitement du langage naturel. Enfin, nous aborderons les métriques d'évaluation permettant de mesurer la performance et la pertinence des recommandations fournies [22].

2.2 Présentation des systèmes de recommandation :

Définition :

Un système de recommandation, c'est un outil intelligent conçu pour aider les utilisateurs à découvrir des contenus qui pourraient leur plaire, en se basant sur leurs habitudes, leurs goûts ou encore leur situation. On les retrouve aujourd'hui au cœur de nombreuses plateformes comme *Amazon, Netflix* ou *Spotify*, où ils jouent un rôle clé pour personnaliser l'expérience de chacun et rendre la navigation plus agréable [23].

Objectifs principaux :

Les objectifs principaux [24] des SR incluent :

- Réduire la surcharge d'information.
- Offrir une personnalisation de contenu.
- Favoriser la découverte de nouveaux produits.
- Accroître l'engagement et la fidélité des utilisateurs.

Domaines d'application :

Les systèmes de recommandation trouvent leur place dans plusieurs domaines Figure (2.1) , [25] :

E-commerce (ex. : Amazon, Alibaba)

Divertissement (Netflix, YouTube)

Réseaux sociaux (Facebook, LinkedIn)

Éducation (cours en ligne personnalisés)

Santé (recommandation de traitements personnalisés)

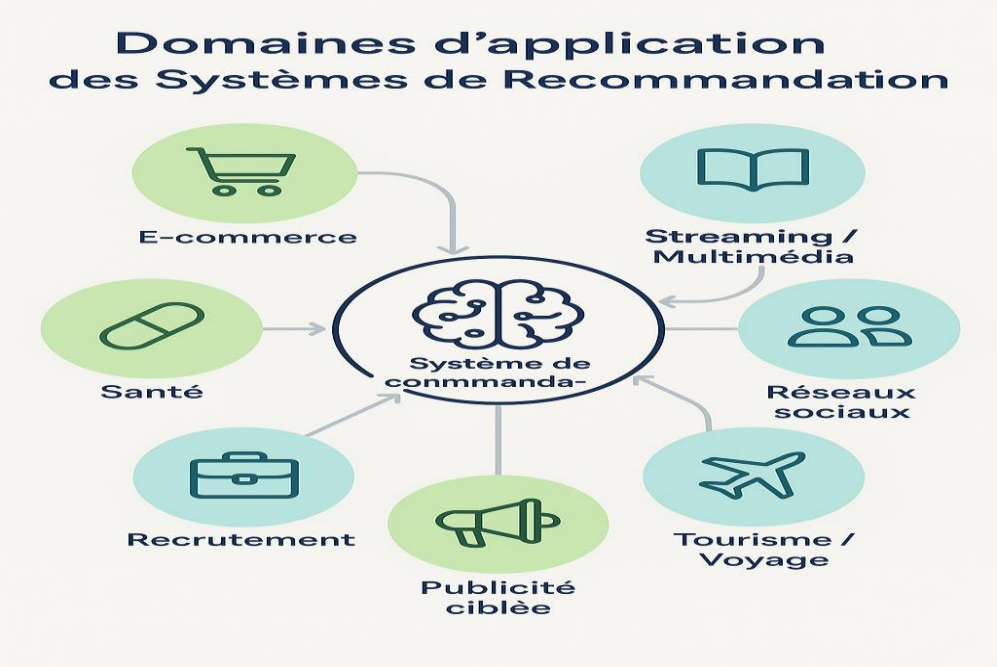


Figure 2.1 : Quelques Domaines d'application SR.

2.3 Classification des approches de recommandation :

Filtrage collaboratif :

Le filtrage collaboratif repose sur l'idée que les utilisateurs partageant des préférences similaires dans le passé continueront à avoir des goûts similaires à l'avenir. Deux grandes variantes existent : le filtrage basé sur les utilisateurs et celui basé sur les items Bien qu'efficace, cette approche souffre du *problème du démarrage à froid* (cold start) et de la *sparsité* des données **Figure (2.2)** [26].

Filtrage basé sur le contenu :

Cette méthode recommande des items en analysant les caractéristiques du contenu et les préférences passées de l'utilisateur. Elle repose souvent sur la vectorisation des textes (ex. **:** TF-IDF) et le calcul de similarité (*cosine, euclidienne, etc*.). Cette approche permet une personnalisation fine mais peut être limitée par la diversité des suggestions (manque de "serendipity") **Figure (2.2)** [26].

Approches hybrides :

Les systèmes hybrides combinent les deux approches précédentes pour bénéficier de leurs avantages respectifs. Ils permettent souvent d'atténuer les limitations individuelles, mais au prix d'une complexité algorithmique accrue **Figure (2.2)** , [26].

***Figure 2.2** : classfication des SR*

2.4 Détail de l'approche utilisée : Content-Based + KNN :

- **Algorithme KNN :**

Entrée :
- D : ensemble d'apprentissage contenant des couples (x_j y_j)
- x : exemple à classer
- k : nombre de voisins à considérer

Début
Pour chaque élément (x_j y_j) dans D faire
Calculer la distance entre x et x_j (par exemple, distance Euclidienne)
Fin Pour
Trier les distances en ordre croissant
Sélectionner les k plus proches voisins de x
Compter le nombre de voisins appartenant à chaque classe
Retourner la classe la plus fréquente parmi les k voisins
Fin

Figure 2.3 : Algorithme KNN [27].

Représentation des profils utilisateurs et produits :

Dans notre approche, chaque produit est représenté par un vecteur de caractéristiques textuelles issu de son catégorie, marque et description fonctionnelle (feature). Le profil utilisateur est implicitement construit à partir de sa requête ou de ses préférences exprimées par un formulaire.

Calcul de similarité :

La similarité entre produits et requête utilisateur est calculée à l'aide de la *distance cosinus* entre vecteurs TF-IDF. Plus cette distance est faible, plus le produit est pertinent par rapport à la requête.

Justification du choix de KNN :

L'algorithme *KNN (k-nearest neighbors)* a été choisi pour sa simplicité, son efficacité dans les espaces vectoriels, et sa capacité à identifier rapidement les éléments les plus proches dans l'espace des caractéristiques. Il est particulièrement adapté pour les systèmes de recommandation basés sur la similarité.

2.5 Données d'entrée et extraction des caractéristiques :

Présentation des avis Amazon :

Nous avons utilisé un ensemble de données issu d'*Amazon Reviews*, contenant des milliers d'avis de consommateurs sur divers produits. Chaque enregistrement contient des attributs clés : nom de produit, marque, catégorie, caractéristiques textuelles, note attribuée, image, etc.

Nettoyage et prétraitement :

Les données ont été nettoyées pour :

- Supprimer les lignes avec notes manquantes.
- Fusionner les colonnes textuelles dans un champ "text".
- Gérer les valeurs manquantes par des chaînes vides.

Techniques de vectorisation :

Le *TF-IDF* ***(****Term Frequency-Inverse Document Frequency****)*** a été utilisé pour transformer les textes en vecteurs numériques exploitables par les modèles de apprentissage automatique. Ce choix permet de mettre en valeur les termes les plus discriminants dans les descriptions de produits [28].

2.6 Évaluation des systèmes de recommandation :

- **Métriques MAE et RMSE :**

Pour évaluer la performance de notre modèle, nous avons utilisé deux métriques :

- **MAE (Mean Absolute Error)** : mesure l'erreur moyenne entre la note prédite et la note réelle.
- **RMSE (Root Mean Squared Error)** : pénalise davantage les grandes erreurs.

Ces métriques ont été calculées sur des jeux d'entraînement et de test, avec des résultats tracés sous forme de courbes pour différentes valeurs de K (nombre de voisins) [29].

2.7 Quelques algorithmes utilisés :

Algorithme	**Ce qu'il fait**	**Adapté au Fog**
KNN	Trouvez les éléments les plus proches de ce que vous aimez.	Oui
Factorisation de matrices	Trouve des liens cachés entre les utilisateurs et les objets.	Oui(avec optimization)
Arbre de décision / Forêt aléatoire	Prend des décisions rapides à partir de plusieurs critères.	Oui
Réseaux de neurones profonds	Analyse complexe mais très précise.	Plus tout pour le cloud.

Tableau 2.1 : Quelques algorithmes utilisés [30].

2.8 Quelques travaux dans le domaine concerné:

Titre du travail / Projet	**Type de recommandation**	**Domaine d'application**	**Année**	**Description**
Netflix Prize	Filtrage collaboratif	Vidéo / Films	**2006**	Compétition pour améliorer le système de recommandation de Netflix.
Spotify Music Recommender	Filtrage collaboratif + audio	Musique	**2014**	Recommande des chansons basées sur les goûts similaires d'autres utilisateurs.
Google News Personalization	Filtrage basé sur le contenu	Actualités	**2007**	Suggère des articles en fonction des lectures précédentes de l'utilisateur.
LinkedIn Job Recommender	Hybride	Emploi / Réseautage professionnel	**2015**	Suggère des offres d'emploi selon le profil, les compétences et l'activité.
Système de recommandation de MOOC (ex. Coursera, edX)	Filtrage collaboratif + contenu	Éducation	**2018**	Recommande des cours en ligne basés sur l'historique de navigation et d'apprentissage.

Tableau 2.2 : Quelques travaux réalisés dans le domaine [31]

2.9 Limitations des approches :

Bien que l'approche basée sur le contenu soit appréciée pour sa simplicité de mise en œuvre et sa capacité à proposer des recommandations personnalisées en fonction des caractéristiques des éléments, elle présente néanmoins plusieurs limitations majeures qui peuvent impacter son efficacité, notamment dans des environnements complexes ou à grande échelle.

Tout d'abord, cette approche a tendance à *manquer de diversité dans les recommandations*. En se basant uniquement sur les propriétés des éléments déjà appréciés par l'utilisateur, le système propose souvent des items similaires, ce qui peut entraîner un phénomène appelé *effet de redondance* ou "sur-spécialisation". Cela limite l'exploration de nouveaux contenus, et donc la découverte de nouveautés ou de recommandations inattendues, pourtant essentielles dans certains domaines comme la musique, les livres ou les films.

De plus, ce type de système *ne tient pas compte de l'évolution du comportement de l'utilisateur*. Il se focalise uniquement sur le profil statique de ce dernier, tel qu'il est déduit de son historique de préférences, sans réelle capacité à apprendre ou à s'adapter à ses changements d'intérêt au fil du temps. Cela peut nuire à la pertinence des recommandations, surtout dans des contextes dynamiques.

Enfin, lorsque la base de données contient un *très grand nombre d'éléments et de caractéristiques*, le traitement et la comparaison des contenus peuvent devenir *extrêmement coûteux en ressources computationnelles*, surtout si aucune méthode d'optimisation n'est mise en place. Cette surcharge peut réduire la réactivité du système et compromettre son évolutivité dans des contextes à forte volumétrie de données [31].

2.10 Conclusion:

Les systèmes de recommandation intelligents jouent un rôle crucial dans la personnalisation de l'expérience utilisateur dans de nombreux domaines. Ce chapitre a mis en lumière les principales approches existantes, en particulier le filtrage basé sur le contenu associé à l'algorithme KNN, utilisé dans notre projet. L'exploitation de jeux de données réels, comme les avis Amazon, combinée à des techniques de vectorisation et de mesure de similarité, a permis de concevoir un modèle performant et adaptable. L'évaluation à l'aide des métriques MAE et RMSE confirme la validité de notre choix méthodologique.

Chapitre 3 : Modèle et implémentation

3.1 Introduction :

Dans ce chapitre, nous présentons l'ensemble du processus de conception, de mise en œuvre et d'évaluation d'un système de recommandation intelligent, intégré dans une architecture *Fog Computing*. L'objectif principal est de démontrer l'efficacité d'un tel système lorsqu'il est déployé dans des environnements informatiques distribués, en particulier ceux proches de l'utilisateur final, pour réduire la latence et améliorer les performances globales.

Nous débutons par la description du cas d'étude utilisé pour nos expérimentations, à savoir la base de données des *avis Amazon*, qui fournit un contexte riche et réaliste pour évaluer les capacités de notre moteur de recommandation. Nous détaillons ensuite les approches algorithmiques adoptées, en particulier la méthode hybride combinant la recommandation basée sur le contenu avec l'algorithme des k plus proches voisins **(***KNN***)**. L'architecture logicielle du système est modélisée à l'aide de diagrammes UML. Une application web interactive a été développée pour permettre aux utilisateurs d'interagir avec le moteur de recommandation de manière intuitive.

Nous poursuivons par la description du déploiement du système dans deux environnements distincts : le *Cloud Computing* et le *Fog Computing*. Une évaluation comparative des performances est ensuite réalisée pour mesurer l'impact de l'architecture *Fog Computing* sur la latence. Enfin, une discussion critique sur les résultats obtenus, les limites rencontrées et les perspectives d'amélioration clôture ce chapitre.

3.2 Description de la base de données :

La base de données utilisée dans cette étude est issue de la plateforme Kaggle et correspond aux évaluations de produits publiées sur Amazon. Il s'agit d'un ensemble de données massives, structuré au format CSV, dont le volume atteint environ *1,1 Go*.

Cette base de données contient plus de *400 000 lignes* et *14 colonnes*, chacune représentant une caractéristique liée aux interactions utilisateur-produit : Nom d'utilisateur, vérifié, nom de l'article, description, image, marque, fonctionnalité, catégorie, prix, note, heure de l'avis, résumé, texte de l'avis, vote. La nature supervisée de ces données permet d'entraîner des modèles de recommandation à partir d'exemples étiquetés, où la variable cible est souvent la note (score) attribuée par l'utilisateur.

Cette richesse informationnelle représente un support particulièrement adapté à la mise en œuvre d'un moteur de recommandation, aussi bien pour les approches basées sur le contenu que pour celles exploitant des techniques d'apprentissage automatique [32].

3.3 Objectif d'utilisation dans notre contexte :

Le choix de la base de données Amazon Reviews s'inscrit dans une logique de pertinence et de réalisme par rapport à notre objectif de développement d'un système de recommandation intelligent. En effet, ce jeu de données offre une grande diversité de profils utilisateurs, de produits, et de types de retours (notes, avis textuels), ce qui permet de simuler des situations variées et représentatives du monde réel.

Grâce à sa structure supervisée et à son volume conséquent, il fournit un cadre idéal pour entraîner, tester et valider des algorithmes de recommandation, en particulier ceux basés sur le contenu. De plus, la richesse des attributs disponibles facilite l'extraction de caractéristiques discriminantes pour les modèles d'apprentissage.

Elle nous permet ainsi de confronter notre moteur de recommandation à des données réalistes, tout en évaluant sa performance dans un environnement Fog, ce qui est cruciale à notre démarche [33].

3.4 Conception du Système de Recommandation :

Dans cette section, nous décrivons la méthodologie de recommandation adoptée ainsi que les outils technologiques utilisés pour l'implémentation du moteur. L'objectif est de proposer un système efficace, capable de générer des recommandations personnalisées à partir de données utilisateurs réelles, tout en étant suffisamment léger pour être déployé dans un environnement Fog La **figure (3.1)** résume le fonctionnement du système.

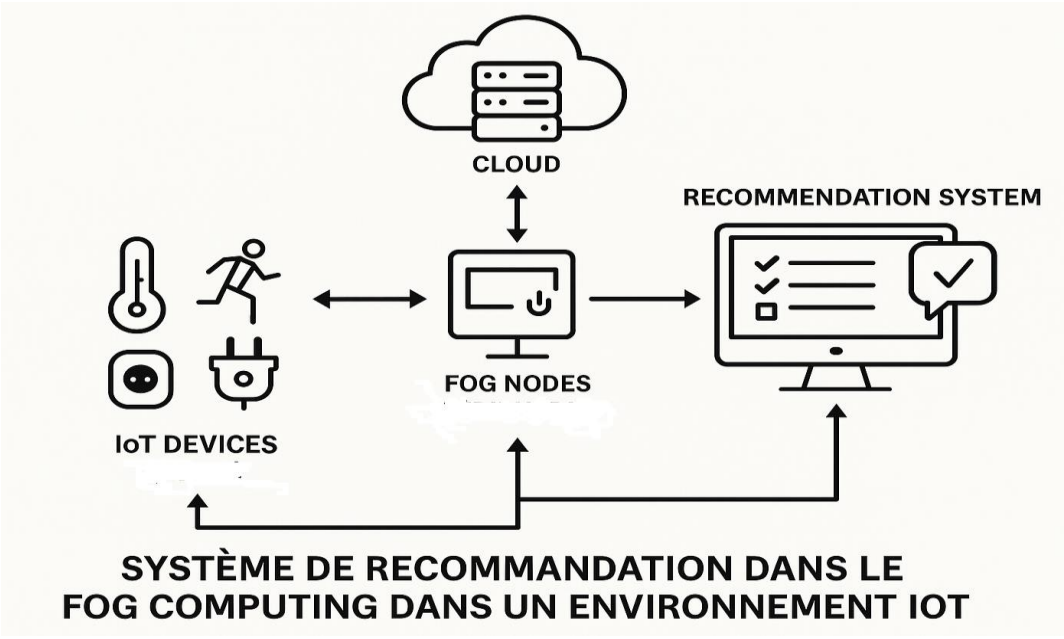


Figure 3.1 : Comment fonctionne le système

3.5 Méthodologie de recommandation choisie :

Le système de recommandation développé repose sur une *approche hybride*, combinant deux techniques complémentaires :

- **La recommandation basée sur le contenu (Content-Based Filtering)** :

  cette méthode s'appuie sur les attributs des produits et les préférences passées par l'utilisateur pour proposer des éléments similaires à ceux déjà existants. Elle permet de personnaliser les recommandations sans dépendre de l'activité des autres utilisateurs.

- **L'algorithme des k plus proches voisins (KNN)** :

  utilisé ici pour mesurer la similarité entre produits existants et les préférences des [34] utilisateurs, il complète l'approche précédente en exploitant les distances entre les vecteurs de caractéristiques. KNN renforce ainsi la précision des suggestions en se basant sur des relations de proximité.

Cette combinaison permet d'améliorer la qualité des recommandations tout en contournant certaines limites propres à chaque méthode prise individuellement, notamment le problème du démarrage à froid ou la faible diversité des recommandations [34].

3.6 Outils, bibliothèques et langages :

a) Les Outils Utilise :

- **Visual Studio Code (VS Code) :**

est un éditeur de code source multiplateforme, léger mais puissant, développé par *Microsoft*. Il prend en charge un large éventail de langages de programmation (Python, JavaScript, C++, etc.) et propose des fonctionnalités avancées telles que la coloration syntaxique**,** le débogage intégré, le contrôle de version avec Git, ainsi qu'un vaste écosystème d'extensions facilitant le développement d'applications intelligentes, y compris dans les domaines de la science des données et de l'intelligence artificielle [35].

- **Start UML :**

est un outil de modélisation logicielle basé sur le langage UML (Unified Modeling Language), qui permet de concevoir, visualiser et documenter des systèmes informatiques de manière structurée et standardisée. Il prend en charge plusieurs types de diagrammes UML (cas d'utilisation, classes, séquence, activités, etc.) et facilite la modélisation orientée objet dans les projets de développement logiciel [36].

B) bibliothèques et langages :

L'implémentation du moteur de recommandation a été réalisée en *Python*, un langage particulièrement adapté à la science des données grâce à son écosystème riche et sa lisibilité.

Les bibliothèques suivantes ont été principalement utilisées :

- **Pandas** :

  est une bibliothèque Python conçue pour la manipulation efficace des données tabulaires, notamment pour le *chargement, le nettoyage, la transformation*, ainsi que l'analyse exploratoire de données. Elle offre des structures comme les *DataFrame* et les Series , permettant une manipulation flexible et performante des jeux de données, particulièrement utile dans le cadre de projets de *machine learning* et de systèmes de recommandation [37].

- **Scikit-learn** :

est une bibliothèque Python largement utilisée pour **l'**implémentation des algorithmes d'apprentissage automatique, notamment dans le cadre de la vectorisation des données, du calcul de similarité, ou encore de l'algorithme des *k plus proches voisins (KNN)*. Son interface simple et cohérente en fait un outil de référence pour le développement rapide de prototypes en apprentissage automatique [38].

- **Numpy** :

est une bibliothèque fondamentale pour le traitement numérique en Python. Elle permet d'effectuer des calculs matriciels efficaces, des opérations vectorielles, ainsi que des manipulations avancées de tableaux multidimensionnels. Son utilisation est essentielle pour la performance des algorithmes d'apprentissage automatique et des systèmes de recommandation [39].

- **Matplotlib** :

est une bibliothèque de visualisation en Python, largement utilisée pour la représentation graphique des données. Elle permet de créer des graphiques statiques, interactifs ou animés, facilitant ainsi **l'**analyse exploratoire et la compréhension des résultats dans les projets de science des données et d'apprentissage automatique [40].

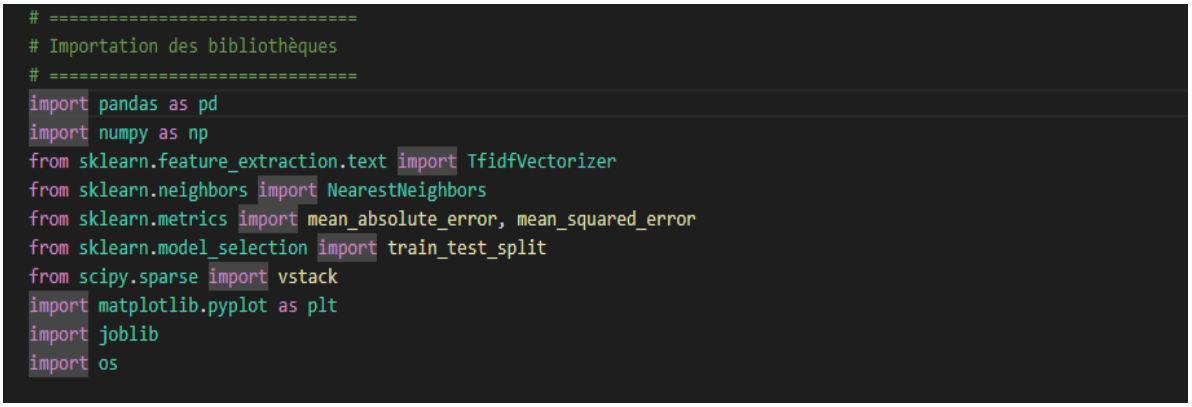
L'utilisation de ces outils a permis de concevoir un moteur de recommandation performant, modulaire et compatible avec une architecture distribuée de type Fog.

3.7 Extraits de code pertinents :

Dans cette section, nous présentons les extraits de code les plus significatifs ayant contribué à la mise en œuvre du moteur de recommandation basé sur une approche hybride combinant la vectorisation TF-IDF et l'algorithme des plus proches voisins (KNN). Le système a été implémenté en Python à l'aide de bibliothèques spécialisées telles que : *pandas*, *scikit-learn* , *joblib*, et *matplotlib*.

A) Bibliothèques :

Il s'agit de la première étape du développement d'un modèle de recommandation, elle consiste a importer des bibliothèques parmi lesquelles *Scikit-Learn* dédiée au Machine Learning. Cette **figure (3.2)** montre toutes les Bibliothèques utilisées dans l'entrainement de modèle.



```
# ==============================
# Importation des bibliothèques
# ==============================
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import train_test_split
from scipy.sparse import vstack
import matplotlib.pyplot as plt
import joblib
import os
```

Figure 3.2 : Importation des Bibliothèques

B) Chargement des données :

Le fichier CSV est volumineux (1.1 Go, >400k lignes). Il est donc traité par blocs (*chunksize*) afin d'optimiser l'utilisation de la mémoire **Figure (3.3).**

```
# ==============================
#  Read the file in chunks
# ==============================
chunk_size = 10000
all_texts = []
all_chunks = []

chunks = pd.read_csv(file_path, chunksize=chunk_size)

for chunk in chunks:
    chunk['feature'] = chunk['feature'].fillna('')
    chunk['text'] = chunk['category'].fillna('') + ' ' + chunk['brand'].fillna('') + ' ' + chunk['feature'].fillna('')
    chunk = chunk[chunk['rating'].notna()]
    all_texts.extend(chunk['text'].tolist())
    all_chunks.append(chunk)
```

Figure 3.3 : Préparer et charger les données en mémoire.

C) Vectorisation du contenu (TF-IDF) :

Les champs textuels concaténés sont vectorisés pour permettre une comparaison de similarité (**Figure 3.4**).

```
# Train vectorizer
vectorizer = TfidfVectorizer(stop_words='english')
vectorizer.fit(all_texts)

# Apply transform to the chunks
tfidf_matrices = []
ratings = []
for chunk in all_chunks:
    tfidf_matrix = vectorizer.transform(chunk['text'])
    tfidf_matrices.append(tfidf_matrix)
    ratings.extend(chunk['rating'].tolist())

tfidf_matrix_full = vstack(tfidf_matrices)
ratings = np.array(ratings)
df_full = pd.concat(all_chunks, ignore_index=True)
```

Figure 3.4 : Vectorisation du contenu

D) Séparation des données et entraînement du modèle KNN :

Dans cette section, l'ensemble de données est scindé en deux parties : 80 % sont consacrés à l'apprentissage du modèle et 20 % sont réservés aux tests. Cette répartition permet d'évaluer les performances du modèle de manière rigoureuse. Par la suite, le modèle des K plus proches voisins (K-Nearest Neighbors – KNN) est entraîné sur les données d'apprentissage afin de capturer les relations et les similitudes nécessaires pour effectuer des prédictions fiables sur les données de test (**Figure 3.5**).

```
# ==============================
# 2. Split the data into train/test
# ==============================
X_train, X_test, y_train, y_test = train_test_split(tfidf_matrix_full, ratings, test_size=0.2, random_state=42)

# ==============================
# 3. Train KNN model
# ==============================
knn = NearestNeighbors(metric='cosine', algorithm='brute')
knn.fit(X_train)
```

Figure 3.5 : Separtion des données

E) Fonction d'Evaluations :

Le modèle KNN est entraîné en utilisant la similarité cosinus comme mesure de distance (**Figure 3.6**).

```python
# ============================
# 4. Evaluation and plot
# ============================
def evaluate_and_plot(knn_model, X_train, X_test, y_train, y_test, k_values=[1, 5, 10, 15, 20, 25]):
    maes_train, rmses_train = [], []
    maes_val, rmses_val = [], []

    for k in k_values:
        y_true_train, y_pred_train = [], []
        y_true_val, y_pred_val = [], []


        for i in range(min(X_train.shape[0], 500)):
            vector = X_train[i]
            distances, indices = knn_model.kneighbors(vector, n_neighbors=k + 1)
            neighbors = indices[0][1:]
            if len(neighbors) == 0:
                continue
            mean_rating = y_train[neighbors].mean()
            y_true_train.append(y_train[i])
            y_pred_train.append(mean_rating)

        mae_train = mean_absolute_error(y_true_train, y_pred_train)
        rmse_train = np.sqrt(mean_squared_error(y_true_train, y_pred_train))
        maes_train.append(mae_train)
        rmses_train.append(rmse_train)

        for i in range(min(X_test.shape[0], 500)):
            vector = X_test[i]
            distances, indices = knn_model.kneighbors(vector, n_neighbors=k + 1)
            neighbors = indices[0][1:]
            if len(neighbors) == 0:
```

Figure 3.6 : Evaluation du modèle.

F) Plot du courbe MAE et RMSE :

Cette partie permet d'afficher les courbes d'évolution du MAE et du RMSE lors de l'entraînement du modèle, ainsi que d'évaluer les performances du modèle (**Figure 3.7**).

```
    # MAE plot
    plt.figure(figsize=(8, 5))
    plt.plot(k_values, maes_train, marker='o', label='Train MAE', color='blue')
    plt.plot(k_values, maes_val, marker='s', label='Validation MAE', color='red')
    plt.title("MAE vs. Number of Neighbors (K)")
    plt.xlabel("K")
    plt.ylabel("MAE")
    plt.legend()
    plt.grid(True)
    plt.savefig(mae_plot_path)
    plt.close()

    # RMSE plot
    plt.figure(figsize=(8, 5))
    plt.plot(k_values, rmses_train, marker='o', label='Train RMSE', color='blue')
    plt.plot(k_values, rmses_val, marker='s', label='Validation RMSE', color='orange')
    plt.title("RMSE vs. Number of Neighbors (K)")
    plt.xlabel("K")
    plt.ylabel("RMSE")
    plt.legend()
    plt.grid(True)
    plt.savefig(rmse_plot_path)
    plt.close()

    return maes_train, rmses_train, maes_val, rmses_val

# Run evaluation
maes_train, rmses_train, maes_val, rmses_val = evaluate_and_plot(knn, X_train, X_test, y_train, y_test)
```

Figure 3.7 : Les courbes MAE et RMSE

G) La sauvegarde et l'affichage des résultats :

Cette partie permet de sauvegarder les résultats des mesures d'erreur (MAE et RMSE) ainsi que le modèle dans la **Figure 3.8**

```
# ================================
# 5. Save model components
# ================================
model_data = {
    'vectorizer': vectorizer,
    'knn_model': knn,
    'df_full': df_full,
    'ratings': ratings,
    'tfidf_matrix_full': tfidf_matrix_full
}
joblib.dump(model_data, model_path)

# ================================
# 6. Final metrics print
# ================================
print("Final MAE values (Train):", maes_train)
print("Final MAE values (Validation):", maes_val)
print("Final RMSE values (Train):", rmses_train)
print("Final RMSE values (Validation):", rmses_val)
print(f"Model and data saved successfully in {model_path}")
print(f"MAE plot saved to: {mae_plot_path}")
print(f"RMSE plot saved to: {rmse_plot_path}")
```

Figure 3.8 : Sauvegarder les résultats finaux

3.8 Performance du modèle après entraînement :

Afin d'évaluer la performance du système de recommandation basé sur l'algorithme des k plus proches voisins (KNN), nous avons utilisé deux métriques d'erreur : *MAE (Mean Absolute Error)* et *RMSE (Root Mean Squared Error).* Ces deux mesures permettent de quantifier la précision des prédictions du modèle en comparant les notes prédites aux notes réelles. Le modèle a été entraîné à partir d'un ensemble de données issu d'avis produits sur Amazon, après transformation textuelle à l'aide du vectoriseur *TF-IDF*. Une fois le modèle KNN entraîné, nous avons effectué une évaluation sur un échantillon de l'ensemble d'apprentissage ainsi que sur un ensemble de validation, en faisant varier la valeur du paramètre *K* (nombre de voisins). Les courbes obtenues pour les deux métriques ont été sauvegardées sous forme d'images, permettant une visualisation claire de l'évolution des performances du modèle en fonction de *K*. Les codes Python ci-dessus illustrent les différentes étapes du traitement, allant de l'importation des données à l'évaluation finale,

avec enregistrement du modèle et des courbes. Les **Figures 3.9** et Figure **3.10** présentent les résultats obtenus après la fin de l'entraînement.

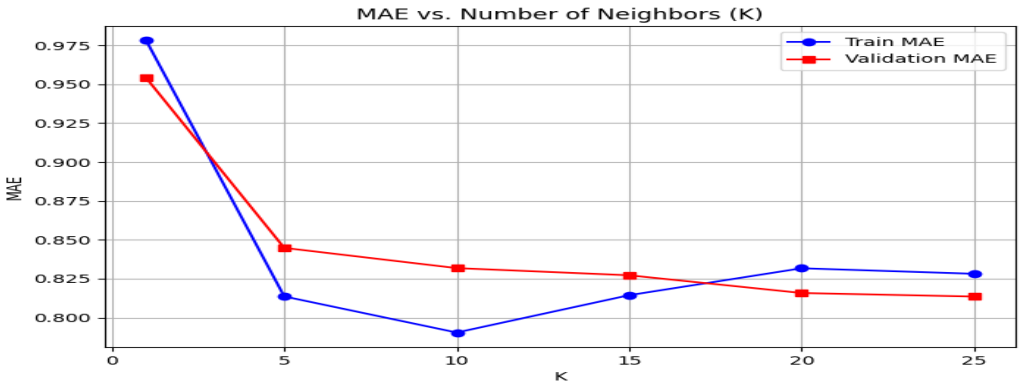


Figure 3.9: Courbe MAE vs Number of Neigbors (K)

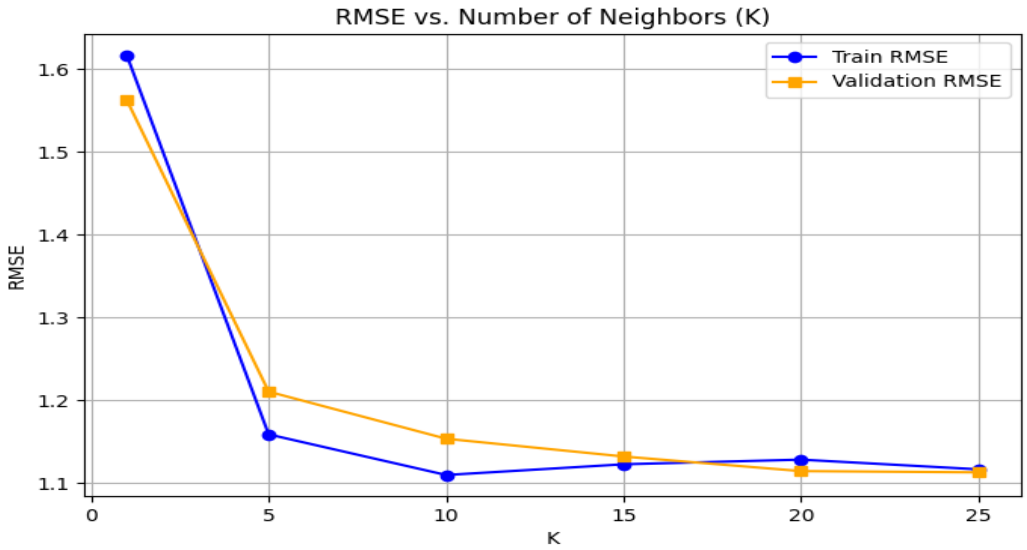


Figure 3.10: Courbe RMSE vs Number of Neigbors (K)

Tableau récapitulatif du modèle et des résultats d'évaluation :

Nom du modèle	MAE (Entraînement)	MAE (Validation)	RMSE (Entraînement)	RMSE (Validation)
Modèle de recommandation basé sur KNN	0.835	0.810	1.18	1.12

Tableau 3.1 : Résultats de l'entraînement.

- **Analyse du graphique MAE en fonction du nombre de voisins (K) :**

Le graphique ci-dessus illustre l'évolution de l'erreur absolue moyenne (MAE) en fonction du nombre de voisins K dans le cadre de l'algorithme des K plus proches voisins (KNN), appliqué à la fois sur les ensembles d'entraînement et de validation. On observe que pour des valeurs faibles de K, notamment K = 1, l'erreur sur l'ensemble d'entraînement est très faible, tandis que l'erreur de validation est relativement élevée. Cela traduit un surapprentissage (overfitting), où le modèle s'ajuste trop aux données d'entraînement au détriment de sa capacité de généralisation.

À mesure que K augmente, la MAE sur les deux ensembles tend à diminuer, atteignant un niveau plus stable aux alentours de K=10. Dans cette zone, l'écart entre les courbes de l'entraînement et de la validation devient faible, ce qui reflète un bon compromis entre biais et variance. Autrement dit, le modèle devient plus robuste et généralise mieux aux données non vues.

Au-delà de K=15, la MAE sur les données d'entraînement commence légèrement à augmenter, ce qui pourrait indiquer une tendance à la sous-apprentissage *(underfitting),* bien que l'erreur de validation reste stable et relativement faible. Ainsi, le choix optimal de K semble se situer entre 10 et 15, où les performances sont les plus équilibrées entre précision et généralisation.

- **Analyse de la courbe RMSE du système de recommandation basé sur l'algorithme des K plus proches voisins (KNN) :**

La courbe présentée illustre l'évolution de l'erreur quadratique moyenne racine (*Root Mean Squared Error - RMSE*) en fonction du nombre de voisins (*K*) pris en compte dans l'algorithme *K-Nearest Neighbors (KNN)*. On observe une tendance générale cohérente entre la RMSE sur l'ensemble d'entraînement (*Train RMSE*) et sur l'ensemble de validation (*Validation RMSE*), traduisant le comportement classique du modèle face au paramètre K.

Analyse des comportements observés :

Pour de faibles valeurs de K (K < 5) :

- La RMSE est relativement élevée, avoisinant **1.5**.
- Ce résultat s'explique par une forte sensibilité du modèle au bruit et un risque accru de surapprentissage (overfitting).
- Le modèle capture excessivement les spécificités et les fluctuations propres aux données d'entraînement, au détriment de la généralisation.

Autour de K ≈ 10 :

- La RMSE atteint un minimum optimal, autour de **1.2**.
- À ce niveau, le modèle trouve un équilibre satisfaisant entre la précision sur les données d'entraînement et capacité de généralisation sur de nouvelles données.
- L'écart entre la RMSE d'entraînement et celle de la validation est réduit, ce qui implique une bonne **capacité de généralisation**.
-

Pour des valeurs élevées de K (K > 15) :

- Une légère augmentation de la RMSE est observée.
- Le modèle devient progressivement trop conservateur (sous-apprentissage, ou underfitting).
- Il tend à ignorer certaines caractéristiques spécifiques des données, réduisant ainsi la qualité des prédictions.

Interprétation théorique :

Cette courbe illustre de manière claire le compromis classique entre biais et variance dans l'apprentissage supervisé :

- Un *K trop faible* conduit à une variance élevée (modèle trop complexe, sensible au bruit).
- Un K trop élevé engendre un biais élevé (modèle trop simple, perte de détails).
- Le point optimal, pour cet ensemble de données, se situe autour de K = 10**.**

Implications pratiques :

- La valeur K = 10 apparaît comme la meilleure option à retenir pour le déploiement du système de recommandation basé sur l'algorithme KNN.
- L'écart modéré entre les courbes d'entraînement et de validation suggère une bonne capacité de généralisation et l'absence de surapprentissage marqué.
- La stabilité relative de la RMSE pour K > 10 indique une robustesse du modèle face aux variations du paramètre K.

Cette analyse met en évidence l'importance cruciale du paramétrage optimal de K dans les systèmes de recommandation par similarité. Le choix du bon K dépend directement des caractéristiques intrinsèques du Base de données utilisé et doit être ajusté avec soin pour garantir des performances optimales.

3.9 Modélisation UML :

3.9.1 Identification des acteurs :

Dans le cadre de notre système de recommandation Amazon, l'acteur principal identifié est l'utilisateur. Ce dernier interagit directement avec le système afin d'obtenir des suggestions de produits adaptées à ses préférences. L'utilisateur peut être un client recherchant un produit spécifique ou explorant différentes options à partir de ses critères de recherche (**Figure 3.4**).

3.9.2 Définition des cas d'usage :

Les cas d'usage décrivent les différentes interactions possibles entre l'utilisateur et le système. Dans notre cas, l'utilisateur commence par *ouvrir l'interface web* du système. Il peut ensuite *entrer une catégorie* de produit (comme électronique, livres, vêtements, etc.), ce qui inclut automatiquement la possibilité *d'entrer une marque* associée à cette catégorie. Une fois ces informations saisies, le système peut *afficher les caractéristiques* des produits filtrés, permettant à l'utilisateur de mieux comprendre les options disponibles. Enfin, sur la base des critères fournis, le système procède à *l'affichage des produits recommandés* **Figure** (**3.4**).

3.9.3 Diagramme de cas d'utilisation UML :

La figure suivante présente le *diagramme de cas d'utilisation* du système de recommandation. Elle illustre de manière structurée les différentes *fonctionnalités offertes par le système* ainsi que les interactions possibles entre l'utilisateur et ce dernier **Figure** (**3.4**).

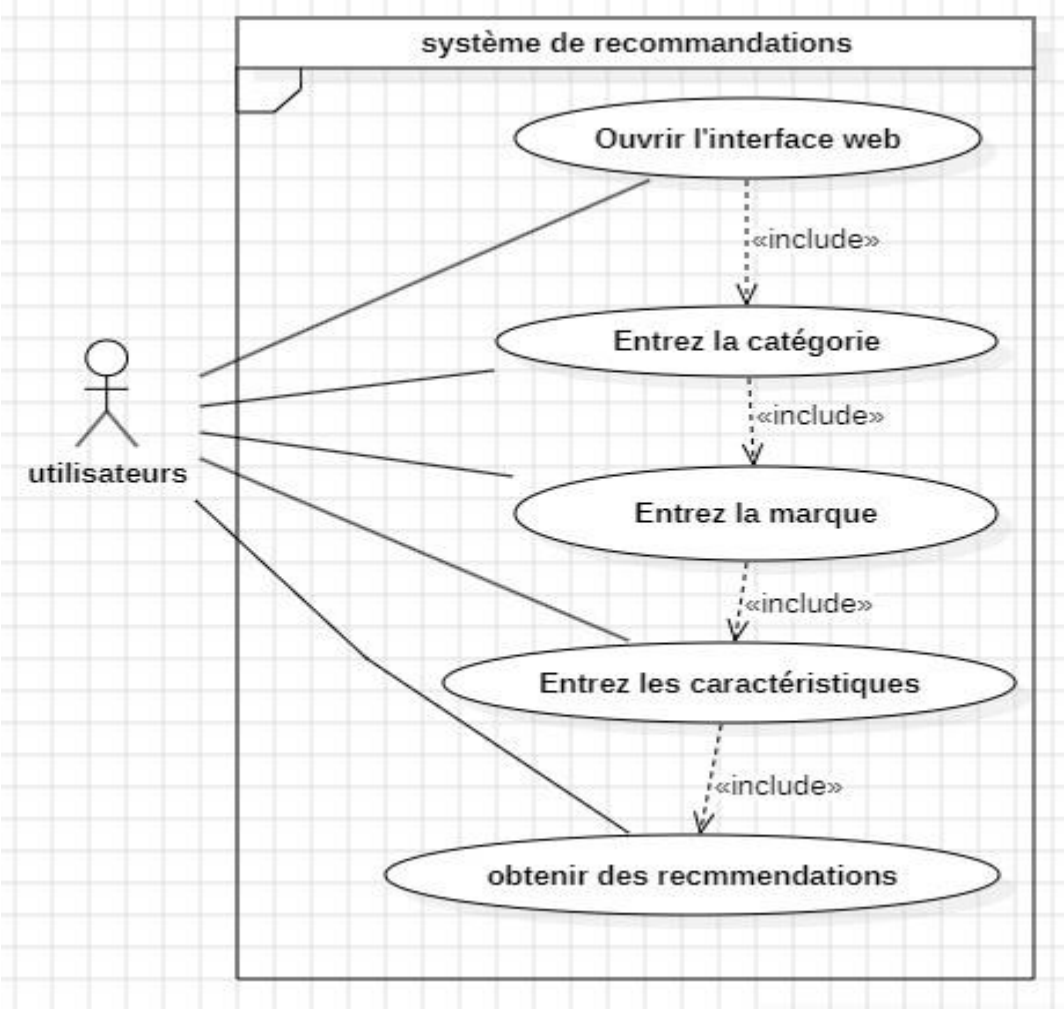


Figure 3.11 : Diagramme Cas d'Utilisation de Système

3.10 Développement de l'Interface Utilisateur Web :

3.10.1 Outils Utilisées :

Pour la conception de l'interface utilisateur du système de recommandation, plusieurs technologies modernes ont été intégrées afin de garantir une application interactive, ergonomique et responsive :

- **HTML :**

(HyperText Markup Language) est le *langage de balisage standard* utilisé pour définir la *structure de base des pages web*. Il permet d'organiser le contenu en éléments hiérarchisés tels que les titres, paragraphes, listes, images et liens hypertextes, servant ainsi de fondation à tout document web [41].

- **Tailwind CSS** :

est un framework CSS utilitaire qui facilite une personnalisation rapide et flexible de l'apparence de l'interface utilisateur, tout en garantissant une *cohérence visuelle* à travers l'ensemble du projet. Il repose sur une approche orientée classes, permettant de styliser directement dans le HTML sans écrire de CSS personnalisé complexe [42].

- **Bootstrap** :

est un framework CSS et JavaScript open-source développé par Twitter, largement utilisé pour créer des interfaces web *responsives et modernes*. Il offre une collection riche de *composants réutilisables*, tels que des grilles, des boutons, des formulaires et des barres de navigation, ce qui permet *d'accélérer le développement* des éléments de mise en page tout en assurant une compatibilité multi-appareils [43].

- **JavaScript** :

est un langage de programmation orienté client, couramment utilisé pour rendre les pages web interactives et dynamiques. Dans ce projet, il est principalement exploité pour gérer les interactions utilisateurs, comme le traitement des formulaires, la validation côté client, ou encore la manipulation du DOM en temps réel, améliorant ainsi l'expérience utilisateur [44].

- **Flask** :

C'est un outil simple et léger en Python qui permet de créer des applications web qui fonctionnent sur le serveur. Dans ce projet, il joue le rôle de *backend*, en assurant la gestion de la logique serveur, le traitement des requêtes utilisateur, **l'**intégration avec le modèle de recommandation, ainsi que le rendu dynamique des pages *HTML* à l'aide du moteur de templates *Jinja2* [45].

3.11 Description fonctionnelle de l'interface :

L'interface utilisateur web a été pensée pour être simple, intuitive et accessible à tout utilisateur, même non technique. Elle propose les fonctionnalités suivantes :

Barre de navigation : une barre en haut de page permet de revenir facilement à l'accueil et peut intégrer des options supplémentaires dans le futur (langue, thème, etc.).

Formulaire de recherche : au centre de l'interface, un formulaire permet à l'utilisateur de saisir :

La *catégorie* du produit recherché.

La *marque* souhaitée.

Des *caractéristiques* spécifiques (ex. : sans fil, écran tactile...).

Affichage des recommandations : après soumission du formulaire, les produits similaires les plus pertinents sont affichés sous forme de liste :

Chaque produit est présenté avec son *nom***,** *marque***,** *note***,** *prix***,** *image illustrative***,** et *une valeur de distance* qui reflète le degré de similarité avec la requête.

Les résultats sont classés de manière à privilégier les produits les mieux notés et les plus proches du besoin exprimé par l'utilisateur.

Temps de réponse : pour chaque requête, des informations sur le temps de traitement sont également affichées, afin d'illustrer la performance du système (extraction de vecteurs, temps de prédiction, etc.).

Cette approche améliore significativement l'expérience utilisateur en lui fournissant des recommandations pertinentes en un temps réduit, tout en conservant une interface claire et lisible.

3.12 Captures d'écran de l'application :

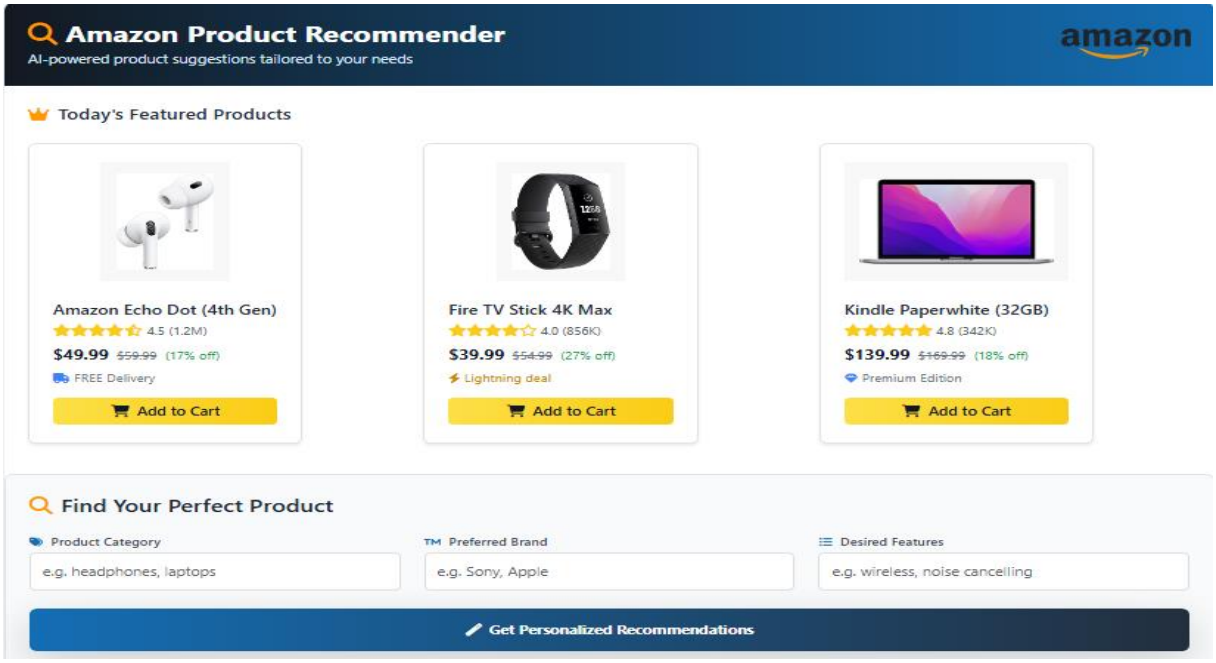


Figure 3.12 : Interface principale du système.

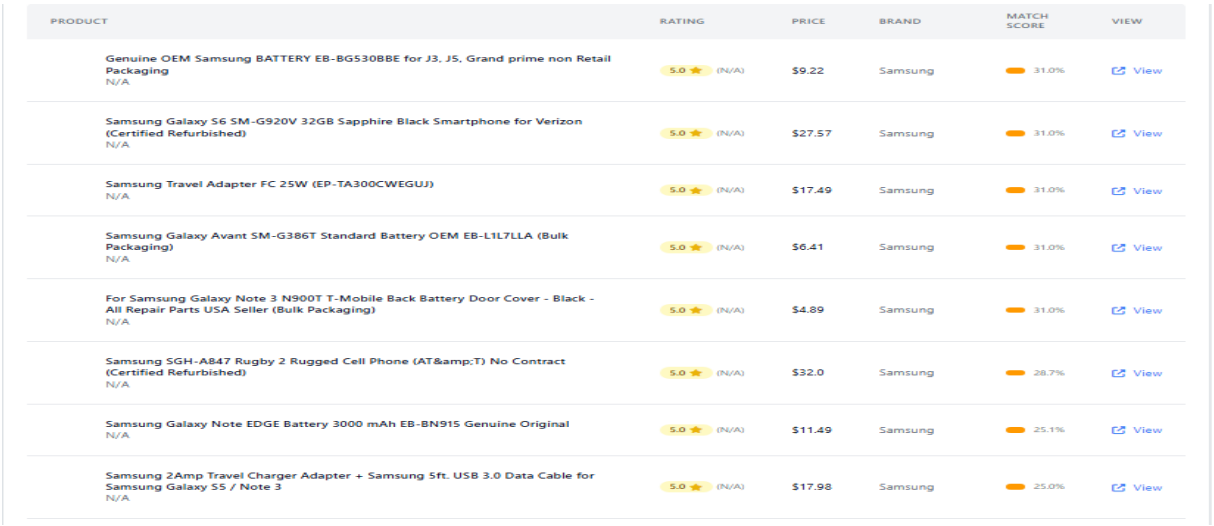


Figure 3.13 : exemple des résultats de recommandation

Exemple des produits recommandés **Figure 3.14** et **Figure 3.15** et **Figure 3.16** :

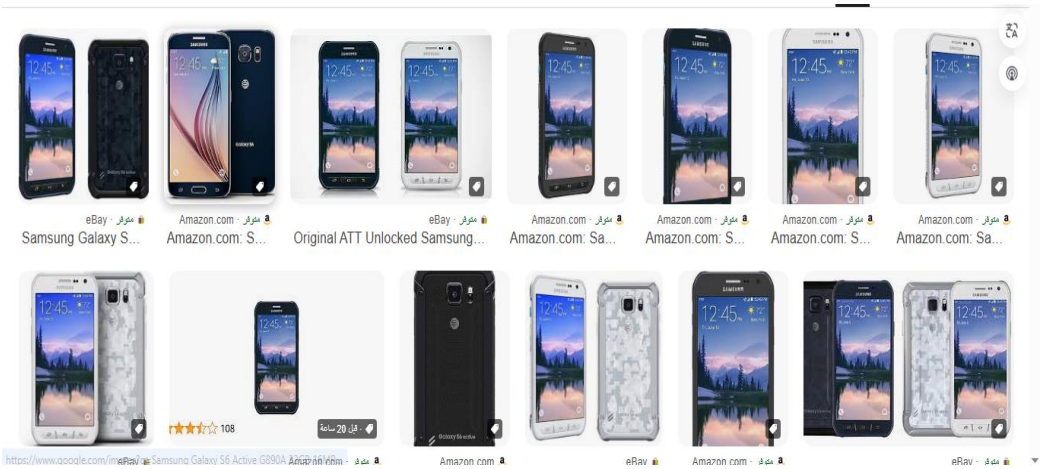


Figure 3.14 : Quelques produits recommandés

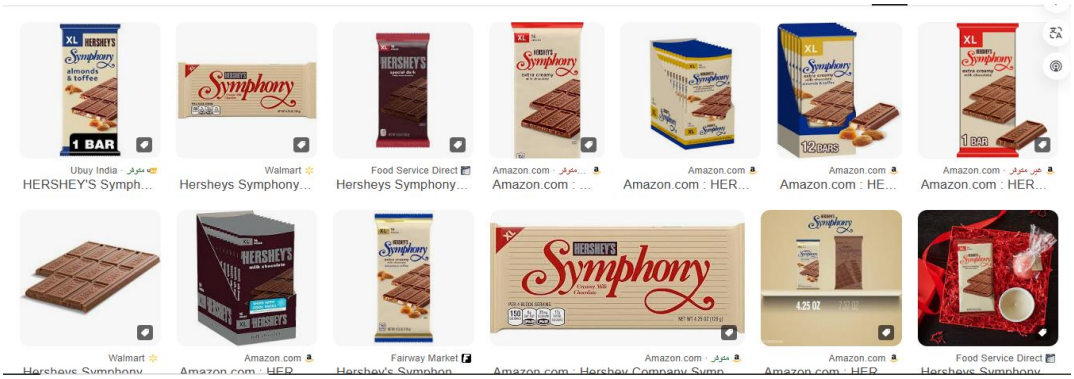


Figure 3.15 : Quelques produits recommandés

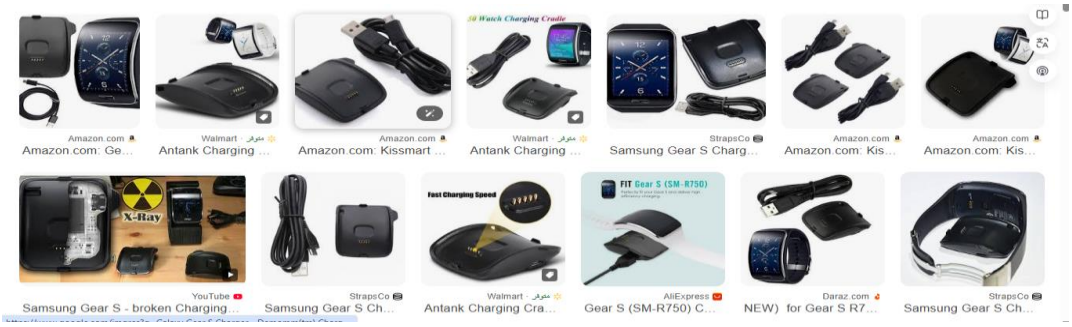


Figure 3.16 : Quelques produits recommandés

3.13 Déploiement du Système dans le cloud et le Fog computing :

Environnement Cloud :

L'environnement de calcul en cloud désigne une infrastructure informatique distante permettant de stocker et de traiter les données sur des serveurs accessibles via Internet. Dans ce projet, nous avons utilisé Google Colab, un outil basé sur la plateforme cloud de Google (GCP)**.** Le modèle est chargé et exécuté sur des serveurs cloud performants, offrant ainsi une grande flexibilité des ressources et une capacité de montée en charge adaptée aux besoins [46].

Avantages :

- Accessibilité universelle depuis n'importe quel appareil connecté à Internet.
- Disponibilité de ressources de calcul puissantes et élastiques.
- Maintenance et mises à jour automatisées assurées par le fournisseur [46].

Limites :

- Latence potentiellement élevée due à la distance entre l'utilisateur final et les serveurs cloud.
- Dépendance à la qualité de la connexion Internet.
- Enjeux de sécurité et de confidentialité liés au transfert des données sur le réseau [46].

Environnement Fog :

Le *Fog computing* (brouillard informatique) consiste à déployer des ressources de calcul et de stockage en périphérie du réseau, c'est-à-dire à proximité de l'utilisateur final, par exemple sur un serveur local. Dans ce projet, un serveur local a été utilisé pour simuler un environnement fog où le modèle est exécuté plus près de la source des données, réduisant ainsi la latence et améliorant la réactivité, ce qui est essentiel pour certaines applications nécessitant une éxecution en temps réel [47].

Avantages :

- Réduction significative du temps de latence grâce à la proximité des ressources.
- Meilleur contrôle et sécurité des données traitées localement.
- Le service reste opérationnel même en cas de connexion Internet faible ou interrompue [48].

Comparaison schématique :

Critère	**Cloud (GCP)**	**Fog (Serveur local)**
Localisation	Centres de données distants	Périphérie du réseau, local
Latence	Plus élevée (dépendance réseau)	Plus faible (proximité utilisateur)
Ressources	Élastiques et extensibles	Limitées mais suffisantes localement
Sécurité	Dépend du fournisseur cloud	Contrôle accru localement

Tableau (3.2) : Comparaison Fog vs Cloud [49]

3.14 Configuration matérielle et logicielle :

Cette section détaille les caractéristiques techniques des environnements utilisés pour la mise en œuvre et la reproduction des expérimentations dans le cadre de ce projet.

- **Configuration matérielle :**

  - **Environnement Cloud (Google Colab / GCP) :**
  -
    - Processeur virtuel : CPU partagé (type Intel Xeon) avec allocation dynamique selon la charge.
    - Mémoire vive : jusqu'à 12.7 Go RAM (variable selon la session).
    - Accès GPU disponible (optionnel selon la session).
    - Stockage : disque virtuel en cloud, accessible via Google Drive.

  - **Environnement Fog (serveur local) :**
    - Processeur : Intel Core i5 8 cœurs physiques.
    - Mémoire vive : 16 Go RAM.
    - Disque dur : SSD 512 Go pour rapidité d'accès aux données.
    - Réseau local Ethernet ou Wi-Fi avec faible latence.

- **Configuration logicielle :**

  - **Système d'exploitation :**
    - Windows 10 64 Bits.
  - **Langages et bibliothèques :**
    - Python 3.11.5 avec les bibliothèques principales : *scikit-learn, joblib, pandas, matplotlib*.
    - Serveur local : Installation des mêmes dépendances Python pour garantir la portabilité du modèle.
    - *Google Colab* comme interface d'exécution en cloud.

  - **Modèle et outils :**
    - Chargement et exécution du modèle pré-entraîné via joblib.
    - Requêtes de recommandation effectuées via des fonctions Python dédiées.
    - Visualisation des résultats et courbes avec *matplotlib*.

Cette configuration permet une reproduction fidèle des résultats, tant sur l'environnement cloud que sur le serveur local, facilitant la comparaison des performances en termes de latence et d'efficacité.

3.15 Évaluation des Performances :

Dans cette section, nous évaluons les performances du système de recommandation déployé dans deux environnements distincts : le cloud (via Google Colab sur GCP) et le fog (via un serveur local). L'objectif est de mesurer l'efficacité du système en fonction du critère de latence, et d'analyser les écarts entre les deux approches d'exécution.

3.15.1 Méthodologie expérimentale

La méthodologie adoptée repose sur l'exécution d'un ensemble de requêtes prédéfinies dans les deux environnements, en maintenant les mêmes paramètres pour garantir l'équité des comparaisons. Chaque requête représente une combinaison spécifique de critères (catégorie, marque, caractéristiques) à fournir au système de recommandation.
Pour chaque exécution, le temps total de traitement — depuis la réception de la requête jusqu'à la génération du résultat — est mesuré à l'aide de la bibliothèque *Python time***.**

Les mesures ont été répétées sur plusieurs requêtes (cinq scénarios différents) afin d'obtenir une moyenne représentative de la performance globale. La latence est ensuite exprimée en millisecondes.

3.15.2 Critère de performance : latence

Le critère principal retenu pour l'évaluation est la *latence*, définie comme le temps total mis par le système pour fournir une réponse pertinente à une requête de l'utilisateur.
Ce critère est particulièrement pertinent dans un contexte de recommandation en temps réel ou proche du temps réel, où la réactivité du système est cruciale pour garantir une bonne expérience utilisateur.

La latence est influencée par plusieurs facteurs : la puissance de calcul disponible, la proximité du serveur (*Cloud* vs *Fog*), la charge réseau, ainsi que l'optimisation logicielle [50].

- **Formule Générale :**

Latence (ms) = (Temps_fin – Temps_Début) × 1000

3.15.3 Visualisation et analyse des résultats :

Les résultats ont été visualisés sous forme de courbes de latence, illustrant la performance de chaque requête dans les deux environnements. Ces courbes permettent d'observer les différences de temps de réponse, et de déduire les forces et limites de chaque approche.

- **Google Colab (Google Cloud Platform) :**

Le modèle de recommandation est stocké sur *Google Drive* et utilise l'environnement *Google Colab (Cloud)* pour exécuter les traitements nécessaires et générer les recommandations.

- **Explication:**

Dans le cadre de ce projet, le formulaire de recommandation est stocké sur Google Drive pour bénéficier d'un stockage centralisé et accessible à distance. Cette méthode vous permet d'enregistrer le formulaire en toute sécurité, ce qui le rend facile à réutiliser et à partager avec d'autres utilisateurs ou collaborateurs du projet.

Le modèle est hébergé dans le cloud et peut être téléchargé à tout moment dans l'environnement de développement Google Colab, qui est également situé dans le cloud. Cette approche offre plusieurs avantages, notamment **Figure (3.17)** :

Avantage et Inconvénients :

Avantages de Google Colab	**Inconvénients de Google Colab**
Accessibilité à distance : accessible depuis n'importe quel appareil connecté à Internet.	**Limitation de ressources :** temps d'exécution limité, accès restreint aux GPU/TPU selon l'usage.
Environnement préconfiguré : bibliothèques populaires déjà installées.	**Connexion Internet requise :** indisponible sans accès au web.
Puissance de calcul gratuite : GPU/TPU disponibles dans le cloud.	**Temps de session limité :** sessions automatiques interrompues après une certaine durée.
Sauvegarde automatique : notebooks enregistrés automatiquement dans Google Drive.	**Confidentialité des données :** données sensibles stockées dans le cloud de Google.
Stockage sécurisé via Google Drive : centralisé, fiable et facile à partager.	**Performances variables :** lenteur possible selon la charge des serveurs Colab.
Intégration facile avec Google Drive : import/export de fichiers sans configuration.	**Personnalisation limitée de l'environnement :** certaines restrictions dans l'installation de logiciels.
Gratuit : solution sans coût pour l'usage de base.	**Pas adapté aux projets très lourds** : pour des entraînements complexes et longs, des solutions plus robustes peuvent être nécessaires.

Tableau (3.3) : Avantages et Inconvénients de Google Colab [51]

Figure 3.17 : Uploader modèle dans Google Drive

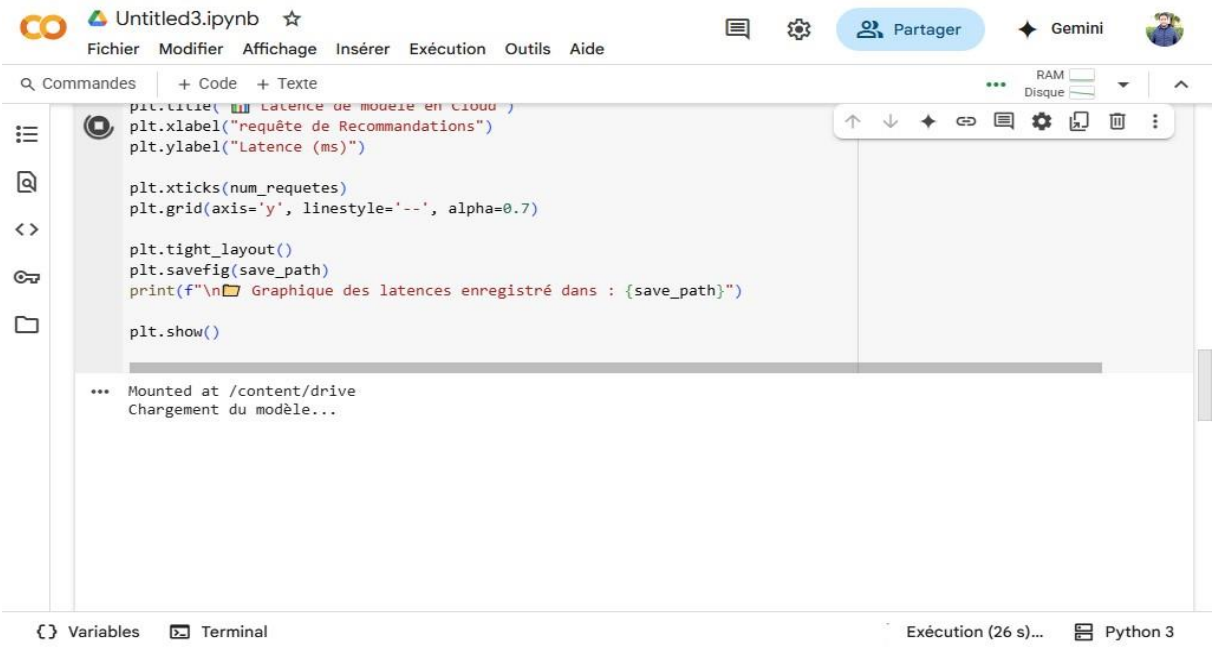


Figure 3.18 : Exécution dans Google Colab

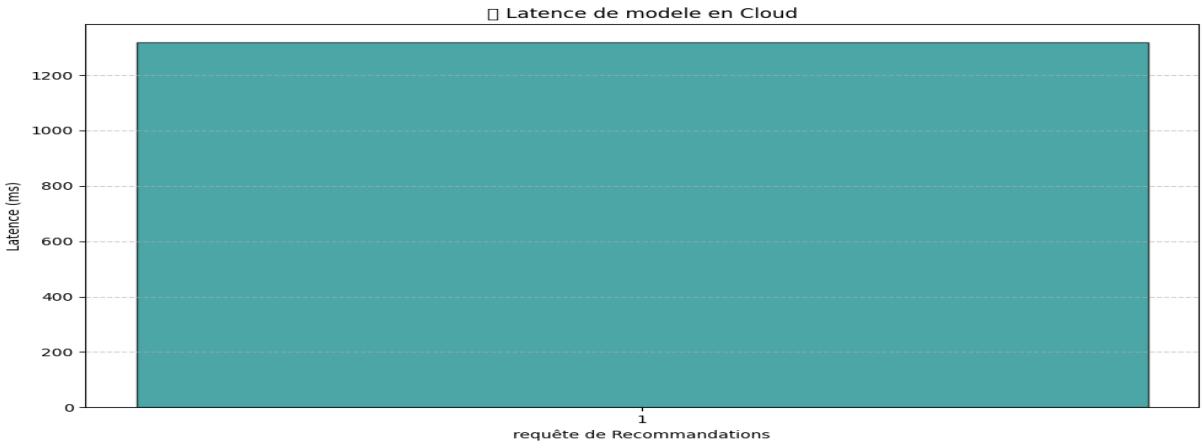


Figure 3.19 : Graphique de visualisation des résultats dans le cloud

- **Analyse des résultats dans Google Colab (Cloud Computing) :**

L'histogramme des temps de latence illustre la distribution des délais de réponse du modèle lors du traitement des requêtes de recommandation. La latence moyenne observée est d'environ *1317 millisecondes*, ce qui reste acceptable dans le contexte d'une application basée sur le cloud, où des temps de traitement supplémentaires peuvent être nécessaires en raison des opérations de recherche et de classification effectuées sur des volumes de données conséquents.

L'analyse du graphique montre que la majorité des temps de latence sont concentrés dans une plage spécifique, traduisant une relative stabilité des performances du modèle. Cependant, certains écarts ou valeurs plus élevées peuvent apparaître, notamment en fonction de la complexité des requêtes ou de la densité des données traitées. Cela peut être le résultat d'une surcharge ponctuelle ou d'un processus de recherche plus long dans la base de données.

Il convient de noter que bien que ce temps de réponse soit tolérable pour des cas d'usage non critiques, il pourrait poser des limites dans des contextes en temps réel, comme des systèmes de recommandation intégrés à des applications à forte interaction utilisateur. Ainsi, pour optimiser davantage les performances, il serait pertinent d'envisager des techniques telles que la réduction de la complexité du modèle, l'optimisation des structures de données, ou encore l'implémentation de mécanismes de cache et de traitements parallèles. Ces améliorations permettraient de réduire la latence et de garantir une meilleure évolutivité du système à mesure que le volume de requêtes et de données augmente **Figure (3.18), Figure (3.19)** .

- **Explication des étapes d'exécution dans le Fog Computing :**

Le modèle de recommandation est stocké sur un ordinateur (Serveur), tandis qu'un autre ordinateur (Client) est utilisé pour exécuter les requêtes. Le client saisit les informations nécessaires, puis envoie ces données au serveur afin d'obtenir les recommandations correspondantes.

Les (**Figures 3.20 , Figures 3.21 , Figures 3.22)** représentent le code utilisé et l'histogramme de latence dans le modèle de type *Fog Computing***.**

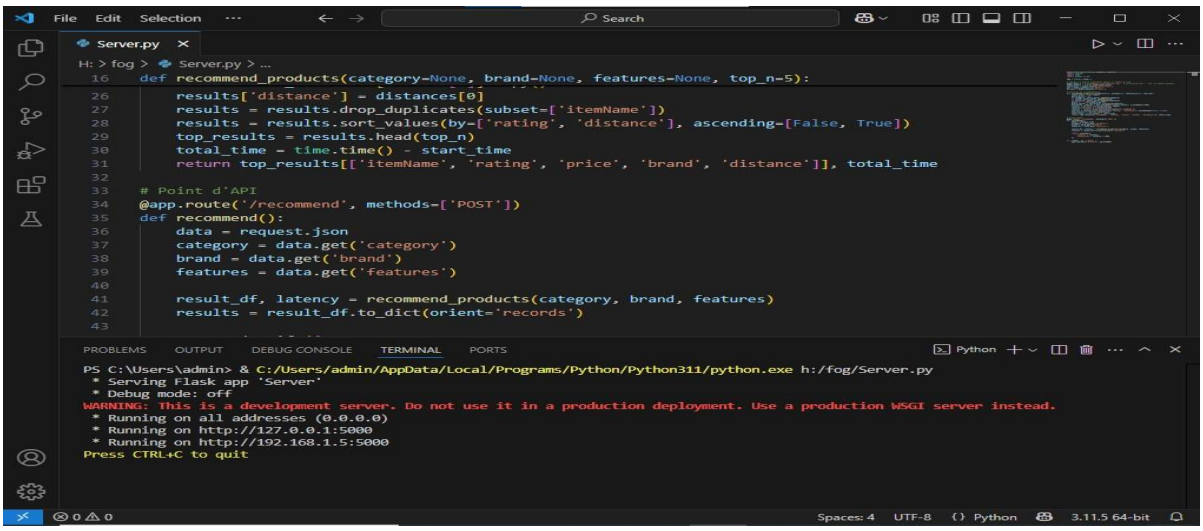


Figure 3.20 : Côté Server

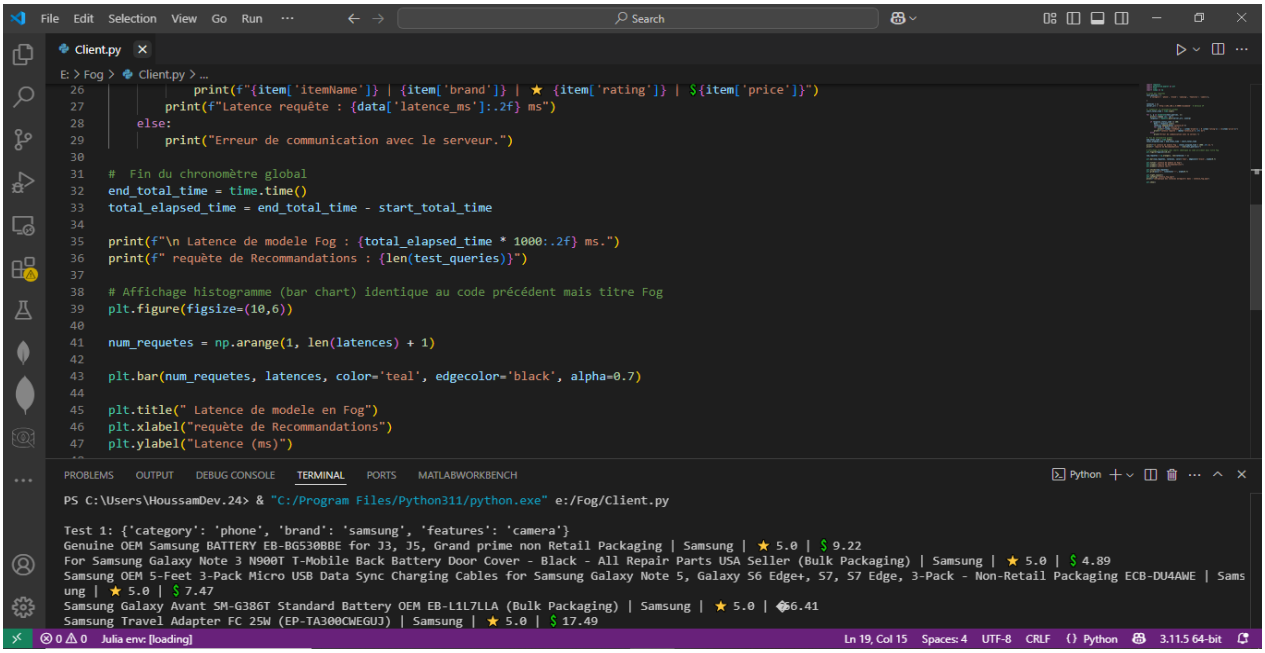


Figure 3.21 : Côté client

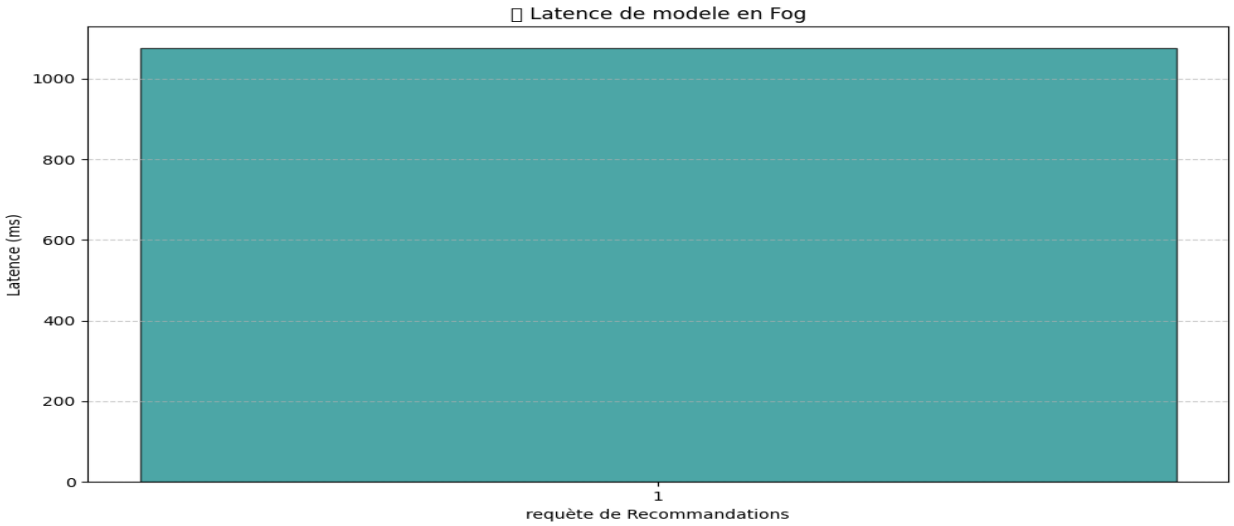


Figure 3.22 : Graphique de visualisation des résultats dans le Fog

- **Analyse des résultats dans le Fog Computing :**

Le graphique obtenu illustre l'évolution de la latence (en millisecondes) lors de l'exécution d'une seule requête de recommandation dans un environnement Fog Computing. La latence mesurée pour cette exécution unique est de 1075 ms. Ce résultat met en évidence la rapidité d'exécution offerte par l'architecture Fog, où la proximité des ressources de traitement réduit significativement le délai de réponse.

Dans ce contexte, il est important de souligner que la requête soumise au modèle contient des paramètres spécifiques tels que la catégorie du produit, la marque, et les caractéristiques souhaitées. Ces informations permettent au modèle de générer des recommandations personnalisées en s'appuyant sur une base de données locale et sur le calcul distribué du Fog.

L'histogramme démontre que, même avec une seule requête, la latence reste inférieure à 1100 ms, ce qui est un indicateur positif de la performance du système. Cette faible latence est cruciale pour des applications en temps réel,

telles que les systèmes de recommandation, qui nécessitent des réponses immédiates pour garantir une expérience utilisateur fluide.

3.16 Comparaison des performances entre le Cloud et le Fog Computing :

Lors de nos tests, nous avons observé une différence notable entre le temps de réponse du modèle de recommandation exécuté dans le *Cloud* et celui exécuté dans le *Fog*. En effet, l'exécution dans l'environnement Fog a permis d'obtenir une latence moyenne de *1075 ms*, alors que dans le Cloud (Google Colab), la latence moyenne était de *1317 ms*.

Cela montre clairement que le Fog est plus rapide et mieux adapté aux applications qui nécessitent une réponse rapide. En effet, dans le Fog, les ressources sont plus proches de l'utilisateur, ce qui réduit le temps d'attente. À l'inverse, le Cloud, bien qu'il soit puissant et pratique, peut introduire des délais plus longs en raison de la distance entre le serveur et l'utilisateur.

En résumé, le *Fog Computing* est particulièrement intéressant pour les systèmes qui doivent répondre en temps réel, tandis que le *Cloud* est plus adapté pour des tâches lourdes qui ne nécessitent pas forcément une réponse immédiate.

3.17 Discussion et limites de l'approche :

Dans le cadre de ce travail, nous avons développé un système de recommandation combinant deux techniques complémentaires : la recommandation basée sur le contenu et l'algorithme des k plus proches voisins *(KNN)*. Cette approche hybride s'est révélée efficace pour générer des recommandations personnalisées, tout en s'adaptant à un déploiement dans un environnement *Fog*, ce qui a permis de réduire significativement la latence.

Cependant, certains points de limitation méritent d'être mentionnés. D'abord, la qualité des résultats dépend fortement de la qualité des données textuelles issues des avis clients. Des commentaires incomplets, mal rédigés ou bruités peuvent affecter la précision du modèle, notamment lors de la vectorisation avec TF-IDF. Ensuite, bien que *KNN* soit simple et performant à petite échelle, il devient plus coûteux en ressources lorsque le volume de données augmente, ce qui peut poser problème dans des environnements à ressources limitées comme le Fog Computing.

Par ailleurs, l'évaluation a principalement porté sur la latence et les métriques d'erreur comme MAE et RMSE. D'autres aspects importants comme la diversité des recommandations, la couverture ou la satisfaction réelle des utilisateurs n'ont pas été explorés dans cette première version du système. Cela ouvre la voie à des améliorations futures.

3.18 Conclusion:

Ce chapitre a permis de présenter en détail l'ensemble des étapes ayant conduit à la mise en œuvre du système de recommandation proposé, depuis le choix du base de données jusqu'au déploiement dans des environnements Cloud et Fog. Nous avons exposé la méthodologie adoptée, les outils sélectionnés, ainsi que les résultats des évaluations, qui ont mis en lumière les atouts du *Fog Computing*, notamment en termes de réactivité et de proximité des données.

Bien que certaines limites aient été relevées, l'approche hybride mise en œuvre a démontré son potentiel dans un contexte de recommandation en temps quasi réel. Les performances obtenues sont encourageantes, et l'architecture du système offre des perspectives intéressantes pour des améliorations futures, tant sur le plan technique (optimisation des algorithmes) que fonctionnel (ajout de nouvelles fonctionnalités centrées sur l'expérience utilisateur).

Conclusion générale :

Le présent mémoire s'est inscrit dans une réflexion approfondie sur l'amélioration des performances des systèmes de recommandation dans le contexte de l'Internet des Objets (IoT), en abordant particulièrement la problématique de la latence. L'essor fulgurant des objets connectés et la croissance exponentielle des données générées en temps réel mettent aujourd'hui en évidence les limites des architectures classiques centrées sur le *Cloud*, notamment en termes de temps de réponse, de consommation de bande passante et de protection des données.

Face à ces défis, nous avons proposé une approche innovante : le déploiement d'un système de recommandation intelligent au sein d'une architecture *Fog Computing*. Cette solution permet de rapprocher le traitement des données de leur source, réduisant ainsi de manière significative la latence, tout en maintenant un niveau de qualité de service satisfaisant.

Notre démarche a reposé sur plusieurs étapes clés. Nous avons d'abord réalisé une étude théorique approfondie des concepts fondamentaux liés à l'IoT, au *Fog Computing* et aux systèmes de recommandation. Ensuite, nous avons conçu et implémenté un moteur de recommandation hybride, combinant le filtrage basé sur le contenu et l'algorithme des k plus proches voisins (KNN). Ce moteur a été entraîné et évalué à l'aide base de données réel (*Amazon Reviews*) et testé dans deux environnements distincts : un environnement Cloud (via *Google Colab*) et un environnement simulant un serveur *Fog local***.**

Les résultats obtenus sont prometteurs : nos tests ont mis en évidence une réduction notable de la latence (environ 18,37 %) en faveur du *Fog Computing*, sans altérer la précision des recommandations. Ces performances témoignent de la pertinence d'un traitement distribué dans des environnements nécessitant des réponses rapides et contextualisées, comme la santé, la domotique ou les transports intelligents.

Cependant, ce travail présente certaines limites. Le modèle reste sensible à la qualité des données textuelles, et l'utilisation de l'algorithme KNN, bien qu'efficace, peut devenir coûteuse en ressources lorsqu'il s'agit de traiter de très grands ensembles de données. Par ailleurs, des aspects tels que la satisfaction des utilisateurs, la diversité des recommandations ou encore la sécurité des

données n’ont pas été pleinement explorés et constituent des pistes intéressantes pour des travaux futurs.

En perspective, plusieurs orientations peuvent être envisagées : l’intégration de techniques d’apprentissage profond pour enrichir le moteur de recommandation, l’optimisation des traitements dans des environnements *Edge* encore plus proches des utilisateurs, ou encore le renforcement des mécanismes de sécurité et de confidentialité dans le traitement des données.

En définitive, cette étude démontre que l’alliance entre systèmes de recommandation et *Fog Computing* ouvre des perspectives prometteuses pour concevoir des systèmes intelligents plus réactifs, personnalisés et efficaces. Elle contribue ainsi à enrichir le champ des recherches dans un domaine en constante évolution, et invite à poursuivre l’exploration des solutions techniques et théoriques pour relever les défis de demain.

Bibliographie

[1] Atzori, L., Iera, A., & Morabito, G. (2010). *The Internet of Things: A survey*. *Computer Networks*, 54(15), 2787–2805.

[2] Vermesan, O., & Friess, P. (Eds.). (2014). *Internet of Things – From Research and Innovation to Market Deployment*. River Publishers.

[3] Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). *Edge computing: Vision and challenges*. *IEEE Internet of Things Journal*, 3(5), 637–646.

[4] Perera, C., Zaslavsky, A., Christen, P., & Georgakopoulos, D. (2014). *Context Aware Computing for The Internet of Things: A Survey*. IEEE Communications Surveys & Tutorials, 16 (1), 414–454.

[5] Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). *Fog computing and its role in the Internet of Things*. *Proceedings of the MCC Workshop on Mobile Cloud Computing*, 13–16.

[6] IoT Industriel. (s.d.). *Architecture IoT : L'essentiel à savoir*. IoT Industriel.

[7] Satyanarayanan, M. (2017). *The emergence of edge computing*. Computer, 50 (1), 30–39.

[8] Cisco. (2015). *Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are*. Cisco Systems.

[9] BISHOP, Tony**.** *Le rôle du Fog computing dans l'Internet of Things*, Programmez!,(2024).

[10] Gia, T. N., Jiang, M., Rahmani, A. M., Westerlund, T., Liljeberg, P., & Tenhunen, H. (2015). *Fog computing in healthcare Internet of Things: A case study on ECG feature extraction*. In 2015 IEEE International Conference on Computer and Information Technology (CIT) (pp. 356–363). IEEE.

Bibliographie

[11] Yanginlar, G**.** (2024). *Internet of Things (IoT) in intelligent transportation systems: Benefits and challenges of implementation*. The Eurasia Proceedings of Science, Technology, Engineering & Mathematics, 27, 16–23.

[12] Recommendations for implementing the strategic initiative INDUSTRIE 4.0**:** Securing the future of German manufacturing industry. *Final report of the Industrie 4.0 Working Group*, Acatech – National Academy of Science and Engineering.

[13] ISO/IEC JTC 1. (2018). *Internet of Things (IoT) – Definitions and Vocabulary*. *International Standard ISO/IEC 20924:2018*.

[14] Wu, C. C., Laghari, S. U. A., Manickam, S., & Ashraf, E. (2025)**.** *Challenges and opportunities in fog computing scheduling: A literature review*. *IEEE Access*. (à paraître / in press).

[15] Mukherjee, M., Matam, R., Shu, L., Maglaras, L., Ferrag, M. A., Choudhury, N., & Kumar, V. (2018). *Security and privacy in fog computing: Challenges*. *IEEE Access*.

[16] Dastjerdi, A. V., Gupta, H., Calheiros, R. N., Ghosh, S. K., & Buyya, R. (2016)**.** *Fog computing: Principles, architectures, and applications*. arXiv:1601.02752.

[17] Das, R., & Inuwa, M. M. (2023). *A review on fog computing: Issues, characteristics, challenges, and potential applications*. *Telematics and Informatics Reports*, 10, 100049.

[18] Wu, C. C., Laghari, S. U. A., Manickam, S., & Ashraf, E. (2025)**.** *Challenges and opportunities in fog computing scheduling: A literature review*. *IEEE Access*, à paraître.

[19] Guy, I., & Carmel, D. (2011). *Social recommender systems*. In Proceedings of the 20th international conference companion on World wide web (pp. 283–284) .

[20] IEEE Access. (2019). *Sustainable infrastructures, protocols, and research challenges for fog computing*. *IEEE Access*.

Bibliographie

[21] Naha, R. K., Garg, S., Chan, A., et al. (2018). *Fog computing architecture: Survey and challenges*. arXiv:1811.09047.

[22] Recommender systems have established themselves as essential tools in many fields, ranging from e-commerce to streaming platforms, due to their ability to enhance the user experience by personalizing the content offered.

[23] Ricci, F., Rokach, L., & Shapira, B. (2011). *Introduction to Recommender Systems Handbook*. Springer.

[24] Adomavicius, G., & Tuzhilin, A. (2005). *Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions*. IEEE Transactions on Knowledge and Data Engineering, 17(6), 734–749.

[25] Tarus, J. K., Niu, Z., & Yousif, A. (2017). *A hybrid knowledge-based recommender system for e-learning based on ontology and sequential pattern mining*. Future Generation Computer Systems, 72, 37–48.

[26] Sarwar, B., Karypis, G., Konstan, J., & Riedl, J**.** (2001). *Item-based collaborative filtering recommendation algorithms*. In Proceedings of the 10th International Conference on World Wide Web (WWW '01) (pp. 285–295).

[27] Cover, T. M., and Hart, P. E.**,** *Nearest Neighbor Pattern Classification*, IEEE Transactions on Information Theory, vol. 13, n° 1, pp. 21–27, 1967.

[28] Ramos, J. (2003). *Using TF-IDF to Determine Word Relevance in Document Queries*. In *Proceedings of the First Instructional Conference on Machine Learning* (pp. 133–142).

Bibliographie

[29] Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). *Evaluating collaborative filtering recommender systems*. ACM Transactions on Information Systems (TOIS), 22(1), 5–53.

[30] Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). *Deep learning based recommender system: A survey and new perspectives*. ACM Computing Surveys (CSUR), 52(1), 1–38.

[31] Lops, P., de Gemmis, M., & Semeraro, G. (2011). *Content-based recommender systems: State of the art and trends*. In: Ricci, F., Rokach, L., Shapira, B., Kantor, P.B. (eds) *Recommender Systems Handbook*. Springer

[32] McAuley, J., Pandey, R., & Leskovec, J. (2015). *Inferring networks of substitutable and complementary products*. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). ACM.

[33] He, R., & McAuley, J. (2016). *Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering*. In *Proceedings of the 25th International Conference on World Wide Web* (WWW), pp. 507–517.

[34] Burke, R. (2002). *Hybrid recommender systems: Survey and experiments*. User Modeling and User-Adapted Interaction, 12(4), 331–370.

Bibliographie

[35] Maurya, D. K. (2021). *Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers*. In Beginning VS Code: Simplify and Speed Up Your Web Development (pp. 1–10). Apress.

[36] Chon, M. (2018). *Model-Driven Development with StarUML 2*. StarUML Documentation Project, MKLabs.

[37] McKinney, W. (2012). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media.

[38] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825–2830.

[39] Oliphant, T. E. (2006). *A guide to NumPy*. USA: Trelgol Publishing.

[40] Hunter, J. D. (2007). *Matplotlib: A 2D graphics environment*. Computing in Science & Engineering, 9(3), 90–95.

[41] W3C. (2017). *HTML 5.2 Specification*. World Wide Web Consortium (W3C).

[42] Otwell, T., & Tailwind Labs. (2020). *Tailwind CSS: Rapidly build modern websites without ever leaving your HTML*. Tailwind Labs Documentation.

[43] Otto, M., & Thornton, J. (2011). *Bootstrap: Sleek, intuitive, and powerful front-end framework for faster and easier web development*. Twitter Inc.

[44] Flanagan, D. (2020). *JavaScript: The Definitive Guide* (7th ed.). O'Reilly Media.

[45] Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python* (2nd ed.). O'Reilly Media.

Bibliographie

[46] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... & Zaharia, M. (2010). *A view of cloud computing*. Communications of the ACM, 53(4), 50–58

[47] Yi, S., Li, C., & Li, Q. (2015). *A Survey of Fog Computing: Concepts, Applications and Issues*. Proceedings of the 2015 Workshop on Mobile Big Data, 37–42 .

[48] Chiang, M., & Zhang, T. (2016). *Fog and IoT: An overview of research opportunities*. IEEE Internet of Things Journal, 3(6), 854–864.

[49] Roman, R., Lopez, J., & Mambo, M. (2018). *Mobile edge computing, Fog et cloud computing : une analyse comparative de la sécurité*. Future Generation Computer Systems, 78, 680–698.

[50] Hong, K., Lillethun, D., Ramachandran, U., Ottenwälder, B., & Koldehofe, B. (2013). *Mobile fog: A programming model for large-scale applications on the Internet of Things*. Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing.

[51] Google. *Google Colaboratory: Cloud-Based Python Notebook Environment*. Google Research, 2023.