

# AutoFinder

Assistant intelligent pour l'achat de voitures d'occasion basé sur RAG et filtrage de contraintes

<b>Auteur</b> : Barhadi Nadiir	<b>Travail</b> : Projet individuel
<b>Environnement</b> : Ubuntu Linux	<b>Language</b> : Python
<b>Framework</b> : Flask	<b>Exécution</b> : python app.py

## 1. Introduction

Les modèles de langage de grande taille (LLM) ont démontré une forte capacité à comprendre et générer du texte en langage naturel. Cependant, leur utilisation directe dans des applications de recommandation pose plusieurs problèmes majeurs : hallucinations, manque de contrôle sur les données utilisées, et difficulté à respecter des contraintes précises exprimées par l'utilisateur.

Le projet AutoFinder vise à répondre à ces limitations en proposant une application web intelligente capable d'assister un utilisateur dans l'achat d'une voiture d'occasion. L'objectif est de combiner la flexibilité du langage naturel avec la fiabilité d'un catalogue structuré, en s'appuyant sur une architecture Retrieval-Augmented Generation (RAG) et une base de données vectorielle.

## 2. Problématique et objectifs

### 2.1 Problématique

Un utilisateur exprime généralement ses besoins de manière floue ou partielle :

- "diesel automatique"
- "moins de 80 000 DHS"
- "Toyota récente"

**Les difficultés principales sont :**

- Interpréter correctement ces demandes en langage naturel
- Éviter que le modèle invente des voitures inexistantes
- Garantir que les recommandations respectent strictement les contraintes
- Maintenir des performances acceptables sur une machine locale aux ressources limitées

### 2.2 Objectifs

- Concevoir un assistant conversationnel pour l'achat de voitures d'occasion
- Intégrer une IA générative tout en contrôlant ses sorties
- Utiliser une base de données vectorielle pour la recherche sémantique
- Proposer un MVP fonctionnel avec interface web
- Assurer une architecture robuste et défendable d'un point de vue ingénierie

## 3. Vue d'ensemble de la solution

AutoFinder repose sur une architecture modulaire composée de :

- Frontend Web** (HTML / CSS / JavaScript)
- Backend Flask** (API REST)
- Pipeline IA contrôlé** :
  - Détection d'intention
  - Extraction de contraintes
  - Recherche sémantique (RAG)
  - Filtrage strict
  - Génération contrôlée via LLM local

**Principe fondamental** : Le LLM ne génère jamais de réponse sans être contraint par des données réelles issues du catalogue.

## 4. Architecture globale

### 4.1 Frontend

Deux interfaces principales sont proposées :

**Catalogue (/catalogue)**

- Affichage de toutes les voitures disponibles
- Filtres côté client (marque, prix, carburant, transmission, kilométrage)

**Chatbot (/chatbot)**

- Interface conversationnelle
- Historique maintenu côté client
- Communication avec le backend via requêtes POST JSON

### 4.2 Backend Flask

Le backend centralise toute la logique métier et IA.

**Routes principales :**

- / : page d'accueil
- /catalogue : catalogue des voitures
- /chatbot : interface du chatbot
- /chat : endpoint principal de dialogue IA

Le backend gère l'historique conversationnel, la détection d'intention, l'appel au moteur RAG, et la génération de la réponse finale.

## 5. Détection d'intention

Un détecteur d'intention léger et déterministe est utilisé pour router les requêtes.

**Intentions possibles :**

- smalltalk** : salutations, remerciements, messages généraux
- car\_search** : recherche de voiture
- other** : hors sujet

La détection repose sur des mots-clés ("diesel", "prix", "km", "automatique"... ) et des expressions régulières (présence de nombres + unités).

**Justification du choix :**

- Évite d'utiliser le LLM inutilement
- Réduit fortement le coût en ressources
- Rend le comportement du système prévisible et contrôlable

## 6. RAG et base de données vectorielle

### 6.1 Embeddings

**Modèle** : all-MiniLM-L6-v2 (SentenceTransformers)

Chaque voiture est décrite par un texte synthétique : marque, modèle, carburant, transmission, kilométrage, prix.

### 6.2 Base vectorielle

- Base utilisée** : ChromaDB
- Mode** : persistant sur disque
- Signature du dataset (mtime + taille) pour détecter les changements et reconstruire l'index si nécessaire

### 6.3 Recherche

Lors d'une requête utilisateur :

- Le texte est transformé en embedding
- Une recherche par similarité est effectuée
- Les métadonnées des voitures candidates sont récupérées

Le RAG est utilisé comme mécanisme de rappel (recall), et non comme décision finale.

## 7. Extraction de contraintes et filtrage

### 7.1 Extraction de contraintes

Un module dédié analyse le message utilisateur pour extraire :

- Carburant (diesel, essence, électrique)
- Transmission (automatique / manuelle)
- Budget maximum
- Kilométrage min/max
- Année minimale / maximale
- Marque

Cette extraction repose sur des expressions régulières, des dictionnaires de correspondance, et la normalisation des valeurs.

### 7.2 Filtrage strict

Après la recherche RAG, un filtrage déterministe est appliqué : toute voiture ne respectant pas les contraintes est éliminée, aucune interprétation n'est laissée au LLM.

**Rôle clé :**

- Le RAG apporte la pertinence sémantique
- Le filtrage apporte la précision et la fiabilité

## 8. Génération de réponse via LLM

### 8.1 Modèle utilisé

- Modèle local** : Optimus 7B (GGUF)
- Moteur** : llama\_cpp
- Température** : 0.4 pour réduire la créativité et les hallucinations

### 8.2 Contrôle du LLM

Le LLM reçoit uniquement les filtres extraits, un catalogue filtré, et l'historique utile.

**Des règles strictes sont imposées via le prompt système :**

- Interdiction d'inventer
- Obligation de référencer les IDs

## 9. Gestion du contexte conversationnel

**Problème identifié**

Le smalltalk ("bonjour", "ça va") pollue la recherche vectorielle et le prompt envoyé au LLM.

**Solutions mises en place**

- Réinitialisation du contexte lors du passage de smalltalk à car\_search
- Limitation stricte de l'historique envoyé au modèle
- Conservation uniquement des messages utiles

Ces choix **améliorent** : la pertinence du RAG, la stabilité du LLM, et les performances globales.

## 10. MVP et scénarios de test

### 10.1 Initialisation du système

Le système effectue plusieurs opérations au démarrage :

- Vérification et réutilisation de la base ChromaDB existante
- Chargement du modèle d'embeddings (all-MiniLM-L6-v2)
- Chargement du LLM local (Optimus 7B) avec warmup

**Console - Démarrage de l'application**

```
nadib@U-Nadib: ~/vscode_files/autofinder
$ cd vscode_files/autofinder/
$ source venv_autofinder/bin/activate
$ python app.py
[warnup] starting heavy loads in background...
[rag] voitures.json inchangé, réutilisation de la base Chroma.
[rag] opening Chroma collection... path=/home/nadib/vscode_files/autofinder/chroma_db
* Serving flask app - app
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
[rag] collection existing, embeddings réutilisés.
[rag] chroma ready | ms=129.0
[llm] loading model...
llama_context: n_ctx_per_seq (2848) < n_ctx_train (32768) -- the full capacity of the model will not be utilized
[llm] model loaded | ms=464.8
[warnup] done | ms=594.2
```

Figure 1 : Logs de démarrage montrant le chargement des modèles et l'initialisation de ChromaDB

### 10.2 Cas de test : Recherche sans résultats

Lorsque aucune voiture ne correspond aux critères de l'utilisateur, le système gère élégamment la situation en informant l'utilisateur et en suggérant d'ajuster ses critères.

**Requête : "je cherche une voiture avec un budget 100 dh"**

**Chatbot AutoFinder**  
Pose une question et trouve une voiture rapidement.

**Utilisateur: je cherche une voiture avec un budget 100 dh**

**Bot: Il n'y a aucune voiture dans le catalogue qui corresponde à cette recherche. Pour réinitialiser la conversation, veuillez modifier un critère ou ajuster votre budget. Par exemple, augmenter le budget ou changer la marque.**

Écrivez votre message...

Envoyer

Rafraîchir

Figure 2 : Console montrant l'extraction des contraintes et l'absence de candidats

```
127.0.0.1 - - [18/Jan/2026 21:02:41] "POST /chat HTTP/1.1" 200 -
[chat] request received | history_len=1
[chat] history=[{"role": "user", "content": "je cherche une voiture avec un budget 100 dh"}]
[filters] constraints={'carburant': 'essence', 'transmission': 'manuelle'}
[rag] search | k=5 query='je cherche une voiture avec un budget 100 dh'
[rag] where={'and': [{'carburant': 'essence'}, {'transmission': 'manuelle'}]}
[rag] candidates (top 5):
- ID: 14901 | Maruti alto[STD] | essence | manuelle | 73129 km | 18700 DHS | Options: climatisation, direction assistée
- ID: 14905 | Maruti alto[STD] | essence | manuelle | 73129 km | 18700 DHS | Options: direction assistée, vitres électriques
- ID: 14910 | Maruti alto[STD] | essence | manuelle | 73129 km | 18700 DHS | Options: direction assistée, vitres électriques, climatisation
- ID: 14921 | Maruti alto[STD] | essence | manuelle | 73129 km | 18700 DHS | Options: direction assistée, vitres électriques
- ID: 14923 | Maruti alto[STD] | essence | manuelle | 73129 km | 18700 DHS | Options: climatisation, direction assistée
[chat] calling LLM | max_tokens=500
[chat] LLM done | llm_ms=223199.2 total_ms=223719.5
127.0.0.1 - - [18/Jan/2026 21:04:43] "POST /chat HTTP/1.1" 200 -
```

Figure 3 : Réponse du chatbot suggérant d'ajuster le budget

### 10.3 Cas de test : Recherche avec résultats

Le système extrait correctement les contraintes (essence, transmission manuelle) et retourne les voitures correspondantes du catalogue.

**Requête : "je cherche une voiture essence et manuelle"**

**Chatbot AutoFinder**  
Pose une question et trouve une voiture rapidement.

**Utilisateur: je cherche une voiture essence et manuelle**

**Bot: 14901 | Maruti alto[STD] | essence | manuelle | 73129 km | 18700 DHS | Options: climatisation, direction assistée 14905 | Maruti alto[STD] | essence | manuelle | 73129 km | 18700 DHS | Options: direction assistée, vitres électriques 14910 | Maruti alto[STD] | essence | manuelle | 73129 km | 18700 DHS | Options: direction assistée, vitres électriques, climatisation 14921 | Maruti alto[STD] | essence | manuelle | 73129 km | 18700 DHS | Options: direction assistée, vitres électriques 14923 | Maruti alto[STD] | essence | manuelle | 73129 km | 18700 DHS | Options: climatisation, direction assistée**

Écrivez votre message...

Envoyer

Rafraîchir

Figure 4 : Console affichant les 5 candidats trouvés et les métriques du LLM

```
127.0.0.1 - - [18/Jan/2026 20:57:34] "GET / HTTP/1.1" 200 -
[chat] request received | history_len=1
[chat] history=[{"role": "user", "content": "je cherche une voiture essence et manuelle"}]
[filters] constraints={'carburant': 'essence', 'transmission': 'manuelle'}
[rag] loading embedding model...
[rag] embedding model loaded | ms=2433.4
[rag] search | k=5 query='je cherche une voiture essence et manuelle'
[rag] where={'and': [{'carburant': 'essence'}, {'transmission': 'manuelle'}]}
[rag] candidates (top 5):
- ID: 14901 | Maruti alto[STD] | essence | manuelle | 73129 km | 18700 DHS | Options: climatisation, direction assistée
- ID: 14905 | Maruti alto[STD] | essence | manuelle | 73129 km | 18700 DHS | Options: direction assistée, vitres électriques
- ID: 14910 | Maruti alto[STD] | essence | manuelle | 73129 km | 18700 DHS | Options: direction assistée, vitres électriques, climatisation
- ID: 14921 | Maruti alto[STD] | essence | manuelle | 73129 km | 18700 DHS | Options: direction assistée, vitres électriques
- ID: 14923 | Maruti alto[STD] | essence | manuelle | 73129 km | 18700 DHS | Options: climatisation, direction assistée
[chat] calling LLM | max_tokens=500
[chat] LLM done | llm_ms=223199.2 total_ms=223719.5
127.0.0.1 - - [18/Jan/2026 21:02:41] "POST /chat HTTP/1.1" 200 -
```

Figure 5 : Présentation des voitures Maruti alto correspondant aux critères

**Observations :**

- L'extraction de contraintes fonctionne correctement
- Le filtrage élimine toutes les voitures non conformes
- Le LLM présente uniquement les véhicules réels du catalogue
- Aucune hallucination détectée dans les réponses

## 11. Difficultés rencontrées et compromis

### 11.1 Coût en ressources

L'exécution locale d'un LLM et d'un moteur RAG est coûteuse en CPU et mémoire.

**Solutions adoptées :**

- Limitation du contexte
- Warmup du modèle au démarrage
- Détecteur d'intention heuristique
- Extraction de contraintes déterministe
- Filtrage strict avant génération

Ce choix améliore la robustesse et la prédictibilité du système.

## 12. Limites du système

- Extraction de contraintes basée sur regex (sensibilité aux formulations)
- Détection d'intention non probabiliste
- Pas de mémoire longue durée des préférences utilisateur
- Indexation limitée aux attributs principaux des voitures

## 13. Conclusion

---

Le projet AutoFinder démontre qu'il est possible de construire une application IA fiable et contrôlée en combinant intelligemment des modèles de langage, une base de données vectorielle et des mécanismes déterministes.

L'approche adoptée privilégie la fiabilité et la transparence plutôt que la sophistication pure, en reconnaissant les limites des ressources disponibles et en mettant en place des garde-fous appropriés pour garantir des recommandations précises et vérifiables.