✅ Updated Project Structure

Car\_Clinic\_Project/

│

├── .github/

│ └── workflows/

│ └── main.yaml # GitHub Actions workflow

│

├── data/

│ ├── raw/ # Raw scraped Reddit data

│ ├── cleaned/ # Cleaned problem-solution pairs

│ ├── tagged/ # JSON with tags for problems & solutions

│ ├── embeddings/ # Vectorized Reddit issues & branches

│ └── branches/ # Static config or enriched data per branch

│

├── docs/ # Design, architecture, decision logs

│ ├── architecture.md

│ ├── embedding\_guide.md

│ └── branch\_matching.md

│

├── python\_scripts/

│ ├── reddit\_data\_extractor/ # Reddit scraping pipeline

│ │ ├── \_\_init\_\_.py

│ │ ├── config.py

│ │ ├── extractor.py

│ │ ├── extractor\_Base.py

│ │ ├── flow.py

│ │ ├── reddit\_client.py

│ │ ├── scraper.py

│ │ ├── utils.py

│ │ └── writer.py

│ │

│ ├── reddit\_data\_cleaner/ # ✅ LLM cleaning pipeline

│ │ ├── \_\_init\_\_.py

│ │ ├── flow.py

│ │ ├── cleaner.py

│ │ ├── llm\_cleaner.py # Standalone script for testing

│ │ ├── llm\_runner.py

│ │ ├── preprocessor.py

│ │ ├── postprocessor.py

│ │ └── utils.py

│ │

│ ├── tag\_generator/ # ✅ Tags from cleaned problem/solution

│ │ ├── \_\_init\_\_.py

│ │ ├── flow.py

│ │ ├── tagger.py # LLM- or rule-based tag extractor

│ │ ├── utils.py

│ │ └── constants.py

│ │

│ ├── embedding\_generator/ # ✅ Embedding logic (for issues + branches)

│ │ ├── \_\_init\_\_.py

│ │ ├── flow.py

│ │ ├── embedder.py # Embedding model wrapper (HF, SBERT, etc.)

│ │ ├── utils.py

│ │ └── config.py

│ │

│ └── branch\_recommender/ # ✅ Branch matching logic

│ ├── \_\_init\_\_.py

│ ├── flow.py

│ ├── matcher.py # Similarity logic (tags + embeddings)

│ ├── filters.py # Location, availability filters

│ ├── utils.py

│ └── ranker.py # Composite ranking engine

│

├── run\_pipeline.py # Entrypoint to test everything end-to-end

├── requirements.txt

└── .gitattributes

📊 Execution / Data Flow Diagram

flowchart TD

A[Reddit Scraper (Extractor)] --> B[Raw Posts + Comments (JSON)]

B --> C[LLM Cleaner Pipeline]

C --> D[Structured Problems & Solutions]

D --> E[Tag Generator (tagger.py)]

E --> F[Tagged Problems & Solutions (JSON)]

F --> G[Embedding Generator (embedder.py)]

G --> H[Problem Embeddings]

I[Branch Metadata (Skills, Tools, Tags)] --> J[Branch Embedding Generator]

J --> K[Branch Embeddings]

H --> L[Branch Recommender]

K --> L

F --> L

L --> M[Best Branch Recommendation]

subgraph Offline Phase

B

C

D

E

F

G

H

I

J

K

end

subgraph Online Use

L

M

End

**Now to not ruin the whole project: i want a comprehensive set of steps or phase to follow, Currently i finished the reddit\_data\_extractor and it runs perfectly on GitHub workflow autonomously. I want you to generate a comprehensive set of phases and steps in each phase so that i can tackle the project logically and safely**

Great! Since you've **successfully completed the reddit\_data\_extractor phase** and have it running on GitHub Actions, you're now at the perfect point to move forward in a structured and professional way without breaking the project.

Here's a **comprehensive project roadmap** broken down into **phases → sub-steps** that ensure logical progression, testability, modularity, and future-proofing:

**🧠 Car Clinic Smart Repair Advisor Roadmap**

**✅ Phase 1: Reddit Data Extraction [✔️ Completed]**

**Goal**: Fetch raw Reddit posts + comments and store them.

* Subreddit selection & config
* PRAW & HTTPX data extraction
* Pagination + API limits handling
* Metadata (timestamp, post\_id, comment\_count, etc.)
* Save to /data/raw/
* GitHub Actions integration
* Logging, retries, error handling

**🔍 Phase 2: Reddit Data Cleaning (LLM-Based)**

**Goal**: Extract structured (problem → solution) pairs using offline LLM.

**🧩 Sub-steps:**

1. **Preprocessing (preprocessor.py)**
   * Remove invalid/bot content
   * Text deduplication / normalizing
   * Ensure every post has valid top comment
2. **LLM Inference (llm\_runner.py)**
   * Load DeepSeek/other LLM (offline)
   * Use prompt templates for input
   * Generate JSON-like structure
3. **Postprocessing (postprocessor.py)**
   * Validate output format
   * Strip hallucinations / empty responses
   * Save clean outputs to /data/cleaned/
4. **Flow Orchestration (flow.py)**
   * Chain all steps together
   * Add CLI / test interface
   * Logging + error trapping

**🏷️ Phase 3: Tag Generator (Problem + Solution Tags)**

**Goal**: Convert cleaned data into meaningful tags.

**🧩 Sub-steps:**

1. **Tagging Model**
   * Use keyword extraction (rule-based extraction) or prompt-based tagging via LLM
   * Generate tags for both problems and solutions
   * Save tags in /data/tagged/
2. **Design Schema**
   * Format: { post\_id, problem\_tags: [...], solution\_tags: [...] }
   * Track confidence / source (LLM, rules, hybrid)
3. **Flow File**
   * tag\_generator/flow.py to orchestrate tagging
   * CLI or Prefect-ready

**🔢 Phase 4: Embedding Generation (Problem + Branch Embeddings)**

**Goal**: Enable semantic matching between Reddit posts and branches.

**🧩 Sub-steps:**

1. **Model Setup**
   * Use Sentence-BERT, Instructor-XL, fine-tuned domain model, or similar
   * Freeze model version for consistency + store version hash
2. **Problem Embeddings**
   * Vectorize cleaned Reddit problem + solution text
   * Store in /data/embeddings/problems/
3. **Branch Embeddings**
   * Manually tag mechanics & branches with expertise
   * Create descriptions like: "Branch A: BMW, brakes, engine"
   * Vectorize branch descriptions
   * Store in /data/embeddings/branches/
4. **Flow Orchestration**
   * embedding\_generator/flow.py
   * Auto skip already-embedded entries

**🧭 Phase 5: Branch Recommender System**

**Goal**: Recommend nearest & best-fit branch based on tags + semantic similarity.

**🧩 Sub-steps:**

1. **Tag-Based Matching (matcher.py)**
   * Compare problem tags with branch skill tags
   * Score based on overlap / weight
2. **Embedding Similarity**
   * Use cosine similarity between vectors
   * Combine with tag score
3. **Filtering Layer**
   * Location-based filtering (user coordinates vs branch)
   * Availability filter (hours, mechanics free, etc.)
4. **Ranking (ranker.py)**
   * Weighted combination of all metrics
   * Return top-N best matches
5. **Flow + CLI**
   * Input: JSON problem
   * Output: Branch recommendation
   * Optional: Include reasoning/logs

**🧪 Phase 6: Local & Integrated Testing**

**Goal**: Validate each phase independently before final orchestration.

* Unit test: each module separately
* End-to-end run on small batch
* Visualize outputs (tags, embeddings, matches)
* Add data samples to /docs/ for testing reference

**🔄 Phase 7: Prefect Orchestration**

**Goal**: Schedule or trigger the full pipeline in modular, testable chunks.

* Convert each flow.py to Prefect flows
* Add retry, logging, alerts
* Trigger chain:

nginx

CopyEdit

extractor → cleaner → tagger → embedder → recommender

**☁️ Phase 8: GitHub Actions & Deployment**

**Goal**: Automate key stages with GitHub Actions.

* Add LLM cleaning and taggen steps to the GitHub workflow
* Schedule runs (e.g. daily at 12:15 EGY time)
* Optional: Dockerize embedding & matching services for later deployment

**📘 Phase 9: Documentation & Finalization**

**Goal**: Ensure professional presentation and reproducibility.

* Maintain /docs/ for every module
* Record decisions: LLM choice, tag schema, model selection
* Include data samples, diagrams
* Update README with final pipeline

**🏁 Final Product: Features Recap**

* ✅ Scrape daily car repair-related issues from Reddit
* ✅ Clean using LLM into structured (problem-solution) form
* ✅ Generate tags and embeddings from issues
* ✅ Recommend optimal nearby Car Clinic branch
* ✅ Fully autonomous pipeline

**What if i will need to add a data augementation and translation phase to the set of steps: Data Augmentation • Paraphrasing: Use tools like NLPAug to create variations of queries. • Translation (if multilingual): Use back-translation for non-English queries. • Noise Injection: Simulate typos or slang (e.g., "car no go vroom" → "car won’t start").**

Great idea. Adding a **Data Augmentation & Translation phase** is especially useful if you want to:

* Improve **generalization** and **robustness** of the recommendation system.
* Support **multilingual queries**.
* Simulate **real-world Reddit-style noise** (typos, slang, abbreviations).
* Enrich a small dataset with more linguistic variations.

**✅ Updated Project Roadmap (with New Phase)**

We’ll place the new phase **after data cleaning** but **before tagging and embedding**, since augmentations must apply to clean data.

**🧬 Phase 3: Data Augmentation & Translation**

**Goal**: Enhance the cleaned (problem-solution) data with diverse linguistic variations.

**🧩 Sub-steps:**

1. **🔁 Paraphrasing**
   * Use NLPAug or ParrotParaphraser (offline preferred) to create paraphrased versions of:
     + Problem texts
     + Solution texts
   * Save both the original and paraphrased samples.
   * Optionally limit to 1–3 paraphrases per sample to control data volume.
2. **🌍 Back-Translation**
   * Translate cleaned data → another language → back to English.
     + Languages: Arabic, Spanish, etc.
     + Tools: MarianMT, M2M100, or EasyNMT (offline if needed).
   * Keep only high-quality back-translations (filter by semantic similarity).
   * Store augmented versions in /data/augmented/translated/
3. **🎛️ Noise Injection**
   * Add synthetic Reddit-style noise:
     + Typos, internet slang, emojis
     + Grammar loss (e.g., remove “a”, “is”, etc.)
   * Use NLPAug word swap, OCR noise, or your own ruleset.
   * Useful for robustness testing of tag generator and recommender.
4. **🗂️ Version Control**
   * Track origin of each variant (e.g., original, paraphrased, translated)
   * Store metadata like confidence, type, language
5. **🧪 Quality Check**
   * Spot-check random samples
   * Ensure semantic consistency with the original
6. **🧠 Flow File**
   * Add data\_augmenter/flow.py
   * CLI interface: python flow.py --mode=all|paraphrase|translate|noise
   * Test on 10 samples before scaling

**🔗 Impact on Later Phases:**

* **Tag Generator**: Now processes a richer and more diverse set of texts.
* **Embedding Generator**: Captures semantic nuances across different phrasings/languages.
* **Recommender**: More robust to user input variability.
* **Training Future ML Models**: More training data = better downstream models.

**🗂️ File Structure Changes**

bash

CopyEdit

Car\_Clinic\_Project/

├── python\_scripts/

│ ├── data\_augmenter/ # NEW

│ │ ├── \_\_init\_\_.py

│ │ ├── paraphraser.py # NLPAug or Parrot-based

│ │ ├── translator.py # Back-translation module

│ │ ├── noise\_injector.py # Simulate typos/slang

│ │ ├── utils.py

│ │ └── flow.py # Orchestrate augmentation

**📊 Updated Phase Sequence Overview**

markdown

CopyEdit

reddit\_data\_extractor

↓

reddit\_data\_cleaner

↓

data\_augmenter 🔁

↓

tag\_generator

↓

embedding\_generator

↓

branch\_recommender

Would you like:

* 📄 A starter code template for flow.py inside the data\_augmenter module?
* 📉 A new **data flow diagram** updated with this phase?
* ✅ Or should I also update the previous checklist version with this new phase?

The final structure of the project:  
Car\_Clinic\_Project/

│

├── .github/

│ └── workflows/

│ └── main.yaml # GitHub Actions pipeline

│

├── data/

│ ├── raw/ # Raw scraped Reddit data

│ ├── cleaned/ # Cleaned LLM-extracted problem-solution pairs

│ ├── augmented/

│ │ ├── paraphrased/ # Paraphrased versions of cleaned texts

│ │ ├── translated/ # Back-translated versions (multi-lang)

│ │ └── noisy/ # Noise-injected texts (slang, typos)

│ ├── tagged/ # Tags extracted from problems/solutions

│ ├── embeddings/ # Vectorized issues + enriched branches

│ └── branches/ # Static + enriched branch + mechanic data

│

├── docs/

│ ├── architecture.md

│ ├── embedding\_guide.md

│ ├── branch\_matching.md

│ └── augmentation\_strategy.md # Details about paraphrasing, translation, noise

│

├── python\_scripts/

│ ├── reddit\_data\_extractor/ # Reddit scraping pipeline

│ │ ├── \_\_init\_\_.py

│ │ ├── config.py

│ │ ├── extractor.py

│ │ ├── extractor\_Base.py

│ │ ├── flow.py

│ │ ├── reddit\_client.py

│ │ ├── scraper.py

│ │ ├── utils.py

│ │ └── writer.py

│ │

│ ├── reddit\_data\_cleaner/ # ✅ LLM cleaning & structuring

│ │ ├── \_\_init\_\_.py

│ │ ├── flow.py

│ │ ├── cleaner.py

│ │ ├── llm\_cleaner.py

│ │ ├── llm\_runner.py

│ │ ├── preprocessor.py

│ │ ├── postprocessor.py

│ │ └── utils.py

│ │

│ ├── data\_augmenter/ # ✅ Augmentation logic

│ │ ├── \_\_init\_\_.py

│ │ ├── flow.py # Unified entry for paraphrase + translate + noise

│ │ ├── paraphraser.py # NLPAug, Parrot, etc.

│ │ ├── translator.py # MarianMT, EasyNMT, etc.

│ │ ├── noise\_injector.py # Slang/typo simulation

│ │ ├── utils.py

│ │ └── config.py

│ │

│ ├── tag\_generator/ # ✅ Tag extraction

│ │ ├── \_\_init\_\_.py

│ │ ├── flow.py

│ │ ├── tagger.py

│ │ ├── utils.py

│ │ └── constants.py

│ │

│ ├── embedding\_generator/ # ✅ Embedding logic

│ │ ├── \_\_init\_\_.py

│ │ ├── flow.py

│ │ ├── embedder.py

│ │ ├── utils.py

│ │ └── config.py

│ │

│ └── branch\_recommender/ # ✅ Final recommendation system

│ ├── \_\_init\_\_.py

│ ├── flow.py

│ ├── matcher.py

│ ├── filters.py

│ ├── utils.py

│ └── ranker.py

│

├── run\_pipeline.py # Entrypoint: chain all phases

├── requirements.txt

└── .gitattributes  
  
  
  
  
  
  
  
The Execution / Data Flow Diagram  
[GitHub Workflow Trigger]

│

▼

┌────────────────────────────┐

│ reddit\_data\_extractor │

│ • Scrape Posts + Comments │

└────────────────────────────┘

│

▼

┌────────────────────────────┐

│ reddit\_data\_cleaner │

│ • LLM Problem-Solution │

└────────────────────────────┘

│

▼

┌────────────────────────────┐

│ data\_augmenter │

│ • Paraphrasing │

│ • Translation │

│ • Noise Injection │

└────────────────────────────┘

│

▼

┌────────────────────────────┐

│ tag\_generator │

│ • Extract tags from │

│ problems & solutions │

└────────────────────────────┘

│

▼

┌────────────────────────────┐

│ embedding\_generator │

│ • Vectorize problems & │

│ branch expertise │

└────────────────────────────┘

│

▼

┌────────────────────────────┐

│ branch\_recommender │

│ • Match user issue with │

│ best-fit nearby branch │

└────────────────────────────┘

│

▼

[Output]

┌────────────────────────┐

│ Structured CSV/JSON │

│ Post, Problem, Tags, │

│ Solution, Branch Match │

└────────────────────────┘

**Include any missing piece into this roadmap(e.g., augmenter and translator) and write it all out:**

The roadmap for now:  
**🧠 Car Clinic Smart Repair Advisor Roadmap (Enhanced & Modular)**

**✅ Phase 1: Reddit Data Extraction [✔️ Completed]**

**Goal**: Fetch raw Reddit posts + comments and store them.

* Subreddit selection & config
* PRAW & HTTPX data extraction
* Pagination + API limits handling
* Metadata (timestamp, post\_id, comment\_count, etc.)
* Save to /data/raw/
* GitHub Actions integration
* Logging, retries, error handling

**🧼 Phase 2: Reddit Data Cleaning (LLM-Based)**

**Goal**: Extract structured (problem → solution) pairs using offline LLM.

**🧩 Sub-steps:**

**Preprocessing**

* Remove invalid/bot content
* Text deduplication / normalizing
* Ensure every post has valid top comment

**LLM Inference**

* Load DeepSeek / offline LLM
* Use prompt templates for input
* Generate JSON-like structure

**Postprocessing**

* Validate output format
* Strip hallucinations / empty responses
* Save clean outputs to /data/cleaned/

**Flow Orchestration**

* Central flow.py for this phase
* CLI & test interface
* Logging + error trapping

**🌐 Phase 3: Augmentation & Translation (Optional/Enhancement Layer)**

**Goal**: Improve data generalization and support multilingual/local slang repair issues.

**🧩 Sub-steps:**

**Paraphrasing**

* Use tools like NLPAug, TextAttack, or offline LLM prompts
* Generate 1–3 paraphrased variations per cleaned entry

**Translation & Back-Translation**

* Translate problems to Arabic → English and back (or other relevant dialects)
* Use offline translation models or APIs (if online is acceptable)

**Noise Injection**

* Add common typos, slang, or shorthand (e.g., "no start" → "won't start")

**Flow Management**

* augmenter/flow.py and translator/flow.py for orchestration
* Output to new folder: /data/augmented/

**🏷️ Phase 4: Tag Generator (Problem + Solution Tags)**

**Goal**: Convert cleaned/augmented data into meaningful tags.

**🧩 Sub-steps:**

**Tagging Model**

* Prompt-based LLM or rule-based extraction
* Tag both **problem** and **solution** aspects
* Save to /data/tagged/

**Design Schema**

* Format: { post\_id, problem\_tags: [...], solution\_tags: [...] }
* Track confidence, LLM version, rules vs prompt, etc.

**Flow Integration**

* tag\_generator/flow.py
* Supports both CLI and Prefect-based invocation

**🔢 Phase 5: Embedding Generation (Issues + Branches)**

**Goal**: Enable semantic similarity matching between Reddit issues and branch profiles.

**🧩 Sub-steps:**

**Model Setup**

* Sentence-BERT, Instructor-XL, or fine-tuned domain model
* Freeze model + store version hash

**Reddit Embeddings**

* Use problem + solution full text
* Store in /data/embeddings/problems/

**Branch Embeddings**

* Convert mechanic expertise → tags → vector descriptions
* Examples: “Mercedes brakes engine diagnosis”, “Toyota electrical hybrid”
* Store in /data/embeddings/branches/

**Flow**

* embedding\_generator/flow.py
* Skips duplicates, auto checkpoints

**🧭 Phase 6: Branch Recommender System**

**Goal**: Match Reddit issue to the nearest and most relevant Car Clinic branch.

**🧩 Sub-steps:**

**Tag-Based Matching**

* Compare Reddit issue tags with branch mechanic expertise tags
* Score via Jaccard or weighted overlap

**Embedding Similarity**

* Compute cosine similarity between Reddit embedding and branch vectors
* Merge with tag-based scores for final relevance

**Filters**

* Location (user vs branch geolocation)
* Branch availability, working hours, mechanic status

**Ranker**

* Composite scoring algorithm (weights: tag match, embedding, proximity)
* Explainability: reason for recommendation (e.g., "matched 3 tags, 92% vector match, nearby")

**Flow**

* branch\_recommender/flow.py
* Outputs ranked branches with logs and reasonings

**🧪 Phase 7: Local & Integrated Testing**

**Goal**: Ensure correctness, modularity, and traceability of results.

* Unit tests for each submodule
* Sample run of all phases (small batch)
* Visualize: tag distribution, embeddings t-SNE/UMAP
* Store testing artifacts in /docs/test\_cases/

**🌀 Phase 8: Prefect Orchestration**

**Goal**: Create robust, fault-tolerant pipeline flows.

* Convert every flow.py into a Prefect flow
* Use Prefect Cloud/Server for logging & retry policies
* Define task dependencies:

nginx

CopyEdit

extractor

↓

cleaner

↓

augmenter / translator (parallelizable)

↓

tagger

↓

embedder

↓

recommender

**☁️ Phase 9: GitHub Actions & CI/CD**

**Goal**: Fully automate data pipeline steps on GitHub.

* Add augmentation and translation jobs to main.yaml
* Trigger workflows at 12:15 PM EGY Time (UTC+2)
* Use matrix builds for running in parallel
* Optional: Docker support for embedding & LLM tasks

**📘 Phase 10: Documentation & Finalization**

**Goal**: Ensure professional delivery and reproducibility.

* /docs/architecture.md, /branch\_matching.md, /embedding\_guide.md
* Include:
  + LLM prompt design
  + Tag schema
  + Data samples from each phase
  + Model selection decisions
  + Performance benchmarks
* Finalize:
  + README.md (visuals, usage, roadmap, contribution guide)
  + Data schemas and folder tree
  + Glossary (e.g., "Reddit Issue", "Mechanic Tag")

**🏁 Final Product Recap**

✅ Autonomous Reddit → Repair Branch Recommendation System  
✅ Cleaned + tagged + embedded data flow  
✅ Multilingual + slang support via augmentation  
✅ Fully orchestrated (Prefect + GitHub Actions)  
✅ Modular and testable architecture  
✅ Professional documentation, visualizations, and reproducibility

ROADMAP 3 (INCLUDING THE CHATBOT INTEGRATION AND BACKEND)

**✅ Phase 1: Reddit Data Extraction [✔️ Completed]**

**Goal:** Fetch raw Reddit posts + comments and store them.

* Subreddit selection & config
* PRAW & HTTPX data extraction
* Pagination + API limits handling
* Metadata (timestamp, post\_id, comment\_count, etc.)
* Save to /data/raw/
* GitHub Actions integration
* Logging, retries, error handling

**🔍 Phase 2: Reddit Data Cleaning (LLM-Based)**

**Goal:** Extract structured (problem → solution) pairs using offline LLM.

**🗂️ Sub-steps:**

* **Preprocessing (preprocessor.py)**
  + Remove invalid/bot content
  + Text deduplication / normalization
  + Ensure every post has a valid top comment
* **LLM Inference (llm\_runner.py)**
  + Load DeepSeek/other LLM (offline)
  + Use prompt templates
  + Output JSON-like structure
* **Postprocessing (postprocessor.py)**
  + Validate format
  + Strip hallucinations / empty responses
  + Save to /data/cleaned/
* **Flow Orchestration (flow.py)**
  + Chain steps
  + CLI / test interface
  + Logging + error handling

**🦑 Phase 3: Data Augmentation & Translation**

**Goal:** Improve LLM training and coverage for diverse queries.

* **Paraphrasing:** NLPAug / LLM-based rewording
* **Back-translation:** English ⇆ Non-English
* **Noise injection:** Typos, slang simulation
* Output stored in /data/augmented/

**🌿 Phase 4: Tag Generator (Problem + Solution Tags)**

**Goal:** Enrich cleaned data with semantic tags.

**🗂️ Sub-steps:**

* **Tagging Model:**
  + LLM-based or keyword rule engine
  + Extract tags for both problems and solutions
  + Store in /data/tagged/
* **Design Schema:**
  + { post\_id, problem\_tags: [...], solution\_tags: [...], source: "llm" }
* **Flow File:** tag\_generator/flow.py

**🔢 Phase 5: Embedding Generation (Problems + Branches)**

**Goal:** Enable semantic similarity between problems and branch expertise.

**🗂️ Sub-steps:**

* **Model Setup:** Instructor-XL / Sentence-BERT
* **Problem Embeddings:**
  + Encode cleaned problem + solution text
  + Store in /data/embeddings/problems/
* **Branch Embeddings:**
  + Encode branch expertise profiles (e.g., "BMW, brakes")
  + Store in /data/embeddings/branches/
* **Flow File:** embedding\_generator/flow.py
  + Auto-skip already embedded entries

**🗺️ Phase 6: Branch Recommender System**

**Goal:** Match user issue to most relevant repair branch.

**🗂️ Sub-steps:**

* **Tag Matching (matcher.py)**
  + Overlap score between problem & branch tags
* **Embedding Similarity**
  + Cosine similarity between problem + branch vectors
* **Filtering Layer:**
  + Location filtering (user coordinates)
  + Availability filter (mechanic open slots)
* **Ranking (ranker.py)**
  + Weighted hybrid score
  + Return top-N branches
* **Flow + CLI:**
  + JSON in, branch + explanation out

**🥺 Phase 7: Local & Integrated Testing**

**Goal:** Validate logic of all phases before production.

* Unit tests for each script
* Small-batch E2E test
* Visualize results: embeddings, tags, matches
* Sample records in /docs/

**🔄 Phase 8: Prefect Orchestration**

**Goal:** Automate flow chaining and error handling.

* Convert each flow.py to Prefect tasks
* Add retries, logs, email/Slack alerts
* Flow chain:

extractor → cleaner → augmenter → tagger → embedder → recommender

**☁️ Phase 9: GitHub Actions & Deployment**

**Goal:** CI/CD automation.

* LLM cleaning, tagging, and recommender in CI
* Daily schedule (e.g., 12:15 PM EGY time)
* Optional: Dockerize embedding and matching microservices

**📘 Phase 10: LLM Chatbot Engine**

**Goal:** Enable real-time chatbot for emergency and mechanic usage.

**🗂️ Sub-steps:**

* **Interface:** REST API for chatbot app integration
* **Prompt Router:** Classify query type (issue-only, branch lookup, both)
* **Retriever (RAG):** Retrieve top Reddit cases
* **Reasoner (LLM):** Generate solution from retrieved data
* **Branch Matcher:** Reuse Phase 6 logic
* **Chat Formatter:** Response = Solution + Suggested Branch + Reason
* **Fallback:** Handle low confidence and no-match cases

**🚪 Phase 11: Backend Integration (FastAPI)**

**Goal:** Deploy chatbot and recommender behind an API.

**🗂️ Backend API:**

* /chat/solve [POST]:
  + Input: { "query": "My BMW won't start", "location": "Cairo" }
  + Output: { "solution": ..., "branch": ..., "confidence": ... }
* /recommend/branch [POST]:
  + Input: problem/embedding/tags
  + Output: Top-N recommended branches

**🌐 Tech Stack:**

* FastAPI (main framework)
* Prefect triggers
* Dockerized modules
* Optional: Caching / Redis for frequent issues

**🔹 Final Product Features Recap**

* ✅ Scrape car repair-related issues from Reddit
* ✅ Clean with offline LLM into (problem → solution)
* ✅ Augment and translate to expand coverage
* ✅ Tag and embed issues + branches
* ✅ Recommend nearest matching branch
* ✅ Serve real-time chatbot solution + branch recommendation
* ✅ Fully automated and deployable system

The final file structure:  
car\_clinic\_chatbot/

│

├── README.md # Comprehensive overview of the project

├── requirements.txt # Main dependencies

├── .env # Environment variables (ignored via .gitignore)

├── .gitignore # Ignore sensitive & runtime files

├── Dockerfile # Containerize the app

├── docker-compose.yml # Optional orchestration of modules

├── main.py # FastAPI entry point

├── prefect\_flows.py # Central Prefect orchestration logic

├── llm\_prompt\_templates.py # Stores prompt templates for various LLM modules

│

├── config/

│ └── settings.py # Central configuration (paths, API keys, etc.)

│

├── data/

│ ├── raw/ # Raw Reddit data

│ ├── cleaned/ # Cleaned problem → solution pairs

│ ├── augmented/ # Paraphrased / translated versions

│ ├── tagged/ # Problem + solution semantic tags

│ ├── embeddings/

│ │ ├── problems/ # Vectorized problem descriptions

│ │ └── branches/ # Vectorized branch expertise profiles

│ └── examples.jsonl # Manually verified samples for debugging

│

├── extractor/ # Phase 1

│ ├── reddit\_scraper.py

│ ├── utils.py

│ └── flow.py

│

├── cleaner/ # Phase 2

│ ├── preprocessor.py

│ ├── llm\_runner.py

│ ├── postprocessor.py

│ └── flow.py

│

├── augmenter/ # Phase 3

│ ├── paraphraser.py

│ ├── backtranslator.py

│ ├── noise\_injector.py

│ └── flow.py

│

├── tag\_generator/ # Phase 4

│ ├── tagger.py

│ └── flow.py

│

├── embedding\_generator/ # Phase 5

│ ├── model\_loader.py

│ ├── generate\_embeddings.py

│ └── flow.py

│

├── recommender/ # Phase 6

│ ├── matcher.py

│ ├── ranker.py

│ └── flow.py

│

├── chatbot/ # Phase 10

│ ├── query\_classifier.py # Issue classification

│ ├── retriever.py # Context fetcher

│ ├── reasoner.py # LLM-based final answer

│ ├── branch\_suggester.py # Use recommender

│ ├── formatter.py # Output formatting

│ └── fallback.py

│

├── api/ # Phase 11 - FastAPI backend

│ ├── routes/

│ │ ├── chat.py

│ │ └── recommend.py

│ ├── models.py # Pydantic models

│ ├── dependencies.py # API deps and utilities

│ └── server.py # FastAPI app instance

│

├── tests/

│ ├── unit/ # Unit tests for each component

│ ├── integration/ # End-to-end tests

│ └── test\_data/

│

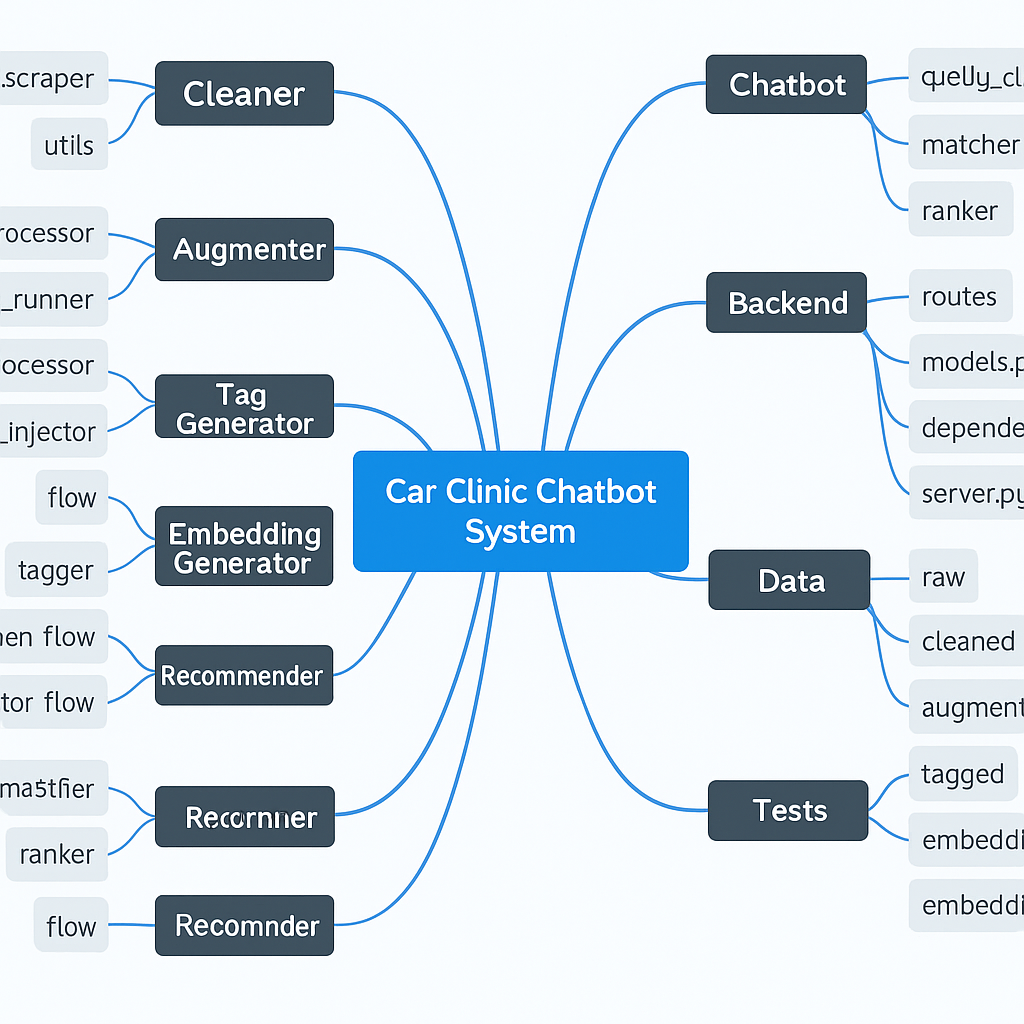
└── docs/

├── architecture\_diagram.png

├── dag\_flow.png

├── api\_schema.md

└── sample\_output\_examples.md



📊 **Data Flow Diagram (DFD)** – *Text Format*

User Query/Input

|

v

[ Chatbot Interface ]

|

v

[ Prompt Router ]

|

|--> (Classification)

| - Problem only

| - Branch lookup

| - Both

|

v

[ Retriever (RAG System) ]

|

v

[ Cleaned + Augmented Reddit Data ]

|

v

[ Reasoning LLM (Offline Model) ]

|

v

[ Problem → Solution Generator ]

|

v

[ Branch Recommender ]

| |

| |--> [Tag-Based Match]

| |--> [Embedding Similarity]

| |--> [Location Filter]

| v

| [Top-N Branch Suggestions]

|

v

[ Response Composer ]

|

v

Chatbot Output:

(Solution + Best Branch + Explanation)

⚙️ **Execution Flow Diagram** – *Textual Representation*

1. Reddit Data Pipeline

├── Extract Reddit data (Phase 1)

├── Clean using LLM (Phase 2)

├── Augment + Translate (Phase 3)

└── Store in /data/{raw,cleaned,augmented}

2. Data Enrichment

├── Tag generation (Phase 4)

└── Embedding generation for:

├── Problems (Phase 5)

└── Branch profiles (Phase 5)

3. Recommender Logic

└── Phase 6:

├── Tag Matching

├── Cosine Similarity (Embeddings)

└── Location-based filtering

4. Testing & Validation

└── Unit + E2E test suite (Phase 7)

5. Orchestration

└── Prefect DAG:

extractor ➝ cleaner ➝ augmenter ➝ tagger ➝ embedder ➝ recommender (Phase 8)

6. CI/CD Automation

└── GitHub Actions (Phase 9)

7. Chatbot Inference System

├── Prompt Routing

├── RAG over cleaned Reddit data

├── Reasoner LLM

└── Branch Matcher (reuse Phase 6) (Phase 10)

8. API Backend Integration

└── FastAPI App:

├── /chat/solve

└── /recommend/branch (Phase 11)