**🧠 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 (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 output format
  + Strip hallucinations / empty responses
  + Save clean outputs to /data/cleaned/
* **Flow Orchestration (flow.py)**
  + Chain all steps together ( Central flow.py for this phase)
  + Add 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/

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

**🗂️ Sub-steps:**

* **Tagging Model:**
  + LLM-based or keyword rule engine ( Use keyword extraction (rule-based extraction) or prompt-based tagging via LLM)
  + Extract or generate tags for both problems and solutions
  + Save tags in /data/tagged/
* **Design Schema:**
  + { post\_id, problem\_tags: [...], solution\_tags: [...], source: "llm" }
* Track confidence / source (LLM, rules, hybrid): Track confidence, LLM version, rules vs prompt, etc.
* **Flow File:**
  + tag\_generator/flow.py to orchestrate tagging
  + Supports both CLI and Prefect-based invocation

**🔢 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

**🧩 Detailed 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"
   * Examples: “Mercedes brakes engine diagnosis”, “Toyota electrical hybrid”
   * Vectorize branch descriptions
   * Store in /data/embeddings/branches/
4. **Flow Orchestration**
   * embedding\_generator/flow.py
   * Auto skip already-embedded entries
   * Skips duplicates, auto checkpoints

**🗺️ 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: Compare problem tags with branch mechanic expertise tags
* Score based on overlap / weight: Score via Jaccard or weighted overlap
* **Embedding Similarity**
  + Cosine similarity between problem + branch vectors: Compute cosine similarity between Reddit embedding and branch vectors
* Combine orMerge with tag-based scores for final relevance
* **Filtering Layer:**
  + Location filtering (user coordinates): Location-based filtering (user coordinates vs branch)
* Availability filter (Branch availability, working hours, mechanic status, etc.)
* **Ranking (ranker.py)**
* Weighted hybrid score: Weighted combination of all metrics : Composite scoring algorithm (weights: tag match, embedding, proximity)
  + Return top-N branches
  + Explainability: reason for recommendation (e.g., "matched 3 tags, 92% vector match, nearby")
* **Flow + CLI (** branch\_recommender/flow.py)**:**
  + Development Input: Tags in JSON for problem and the branch + explanation out
  + Deployment Input: JSON problem
  + Output: Branch recommendation
  + Optional: Include reasoning/logs( Outputs ranked branches with logs and reasonings)

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

**Goal:** Validate each phase independently before final orchestration: Ensure correctness, modularity, and traceability of results.

* Unit tests for each script
* End-to-end run on small batch
* Small-batch E2E test
* Visualize results/outputs: embeddings, tags, matches
* Sample records in /docs/
* Store testing artifacts in /docs/test\_cases/

**🌀 🔄 Phase 8: Prefect Orchestration**

**Goal:** Automate flow chaining and error handling: Schedule or trigger the full pipeline in modular, robust, testable chunks.

* Convert each flow.py to Prefect tasks
* Add retries, logs, email/Slack alerts OR Use Prefect Cloud/Server for logging & retry policies
* Flow chain (Trigger Flow):

extractor → cleaner → augmenter → tagger → embedder → recommender

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

**Goal:** CI/CD automation: Fully automate data pipeline steps on GitHub Actions.

* Add LLM cleaning, tagging, and recommender steps to the GitHub workflow or CI
* Daily schedule (e.g., 12:15 PM EGY time)
* Use matrix builds for running in parallel

Optional: Dockerize embedding and matching and LLM microservices or tasks

**📘 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

**📘 Phase 12: 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
* 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: Features Recap**

**🔄 Autonomous Data Pipeline**

* **✅ Fully automated from daily Reddit scraping to repair branch recommendation.**
* **✅ End-to-end orchestration using Prefect and GitHub Actions for scheduled and event-driven runs.**

**🧠 Data Understanding & Structuring**

* **✅ Scrapes car repair-related Reddit posts and top comments daily.**
* **✅ Cleans and extracts structured problem → solution pairs using an offline LLM.**
* **✅ Applies data augmentation (e.g., paraphrasing, slang injection, back-translation) for robustness and multilingual support.**

**🏷️ Semantic Understanding & Representation**

* **✅ Generates issue tags to summarize mechanical topics.**
* **✅ Converts problems and branch specialties into vector embeddings for similarity comparison.**

**🧭 Smart Repair Branch Recommendation**

* **✅ Suggests the optimal and nearest Car Clinic branch based on:**
  + **Tags and embeddings of the extracted issue.**
  + **Specialties and location of each branch.**

**💬 Real-Time Interaction (Future-Ready)**

* **✅ Structured data enables powering an emergency chatbot assistant for:**
  + **Suggesting on-the-spot fixes.**
  + **Recommending where to go.**

**🧩 Software Design & Deployment Readiness**

* **✅ Modular and testable Python codebase with reusable components.**
* **✅ Clean file structure separating phases: extraction, cleaning, augmentation, tagging, recommendation.**
* **✅ Prepares data and modules for seamless backend integration.**
* **✅ Professional-quality deliverables:**
  + **📚 In-depth documentation**
  + **🧠 Mind map, 📊 data flow diagrams**
  + **⚙️ CI/CD workflows and orchestrated execution**

**File Structure**

✅ **Phase 1: Reddit Data Extraction (Scraping)**  
• 🔁 **Inputs**:  
  o List of subreddits  
  o Reddit API credentials (via PRAW)  
  o Configs (e.g., how many posts, which filters)

• ⚙️ **Inside**:  
  o Fetch top daily/weekly posts with comments  
  o Remove posts with no comments  
  o Filter spam/bot content  
  o Save results in /data/raw/ as JSON or CSV

• 🎯 **Purpose**:  
  To collect relevant raw text data (real-world issues and discussions) for downstream LLM processing.

• 🔁 **Used again in**:  
  Phase 2 (Cleaning) → The raw data is the LLM’s input  
  Phase 10 (Re-training or evaluation for LLMs)

• 📤 **Outputs**:  
  /data/raw/reddit\_posts\_with\_comments.json

✅ **Phase 2: Reddit Data Cleaning (LLM-Based)**  
• 🔁 **Inputs**:  
  o Raw Reddit posts + top comments  
  o Offline LLM (e.g., DeepSeek)  
  o Prompt templates

• ⚙️ **Inside**:  
  o Preprocessing: Remove invalid/bot content, deduplicate text  
  o LLM Inference: Extract structured (problem → solution) pairs  
  o Postprocessing: Clean JSON output, remove hallucinations, format correctly

• 🎯 **Purpose**:  
  Converts noisy Reddit discussions into clean, structured problem–solution data

• 🔁 **Used again in**:  
  Phase 3 (Data Augmentation)  
  Phase 4 (Tag Generation)  
  Phase 5 (Embedding Generation)  
  Phase 10 (LLM Chatbot response generation)

• 📤 **Outputs**:  
  /data/cleaned/cleaned\_problems\_solutions.json

🦑 **Phase 3: Data Augmentation & Translation**  
• 🔁 **Inputs**:  
  o Cleaned (problem → solution) pairs  
  o Augmentation tools (NLPAug, LLM prompts, etc.)  
  o Offline/online translation models

• ⚙️ **Inside**:  
  o Paraphrasing: Reword entries using LLM or augmentation tools  
  o Translation: Back-translate to improve variation (e.g., Arabic ⇄ English)  
  o Noise Injection: Introduce typos, slang for robustness  
  o Save to /data/augmented/

• 🎯 **Purpose**:  
  Increases LLM training robustness by simulating diverse query styles

• 🔁 **Used again in**:  
  Phase 5 (Embedding Generation — optional)  
  Phase 10 (Chatbot fallback understanding)

• 📤 **Outputs**:  
  /data/augmented/augmented\_problem\_solution.json

🌿 **Phase 4: Tag Generator (Problem + Solution Tags)**  
• 🔁 **Inputs**:  
  o Cleaned or augmented problem–solution pairs  
  o LLM or keyword-based tag engine  
  o Tagging schema

• ⚙️ **Inside**:  
  o Extract semantic tags from problems and solutions  
  o Track confidence, model/rule version  
  o Save structured tags to /data/tagged/

• 🎯 **Purpose**:  
  Summarize key mechanical topics for each case for easier matching

• 🔁 **Used again in**:  
  Phase 6 (Recommender System — tag matching)  
  Phase 10 (Chatbot response explanation)

• 📤 **Outputs**:  
  /data/tagged/tagged\_problems\_solutions.json

🔢 **Phase 5: Embedding Generation (Problems + Branches)**  
• 🔁 **Inputs**:  
  o Cleaned (or augmented) problem–solution texts  
  o Manually labeled branch expertise descriptions  
  o Sentence-BERT / Instructor-XL model

• ⚙️ **Inside**:  
  o Encode problem–solution text to embeddings  
  o Encode branch expertise profiles into vector space  
  o Store in /data/embeddings/problems/ and /data/embeddings/branches/

• 🎯 **Purpose**:  
  Enables semantic similarity search for smart recommendations

• 🔁 **Used again in**:  
  Phase 6 (Embedding similarity matching)  
  Phase 10 (RAG + recommendation logic in chatbot)

• 📤 **Outputs**:  
  /data/embeddings/problems/\*.npy  
  /data/embeddings/branches/\*.npy

🗺️ **Phase 6: Branch Recommender System**  
• 🔁 **Inputs**:  
  o Tagged problem records  
  o Problem + branch embeddings  
  o Branch metadata (location, availability)

• ⚙️ **Inside**:  
  o Compute tag overlap score  
  o Compute cosine similarity between embeddings  
  o Combine scores with location & availability filter  
  o Return top-N branches with explanation

• 🎯 **Purpose**:  
  Recommends best branch for issue using hybrid semantic and tag matching

• 🔁 **Used again in**:  
  Phase 10 (Chatbot output)  
  Phase 11 (API /recommend/branch)

• 📤 **Outputs**:  
  /data/recommendations/recommended\_branches.json

🧪 **Phase 7: Local & Integrated Testing**  
• 🔁 **Inputs**:  
  o All prior modules (flows and outputs)  
  o Sample configs + test Reddit records

• ⚙️ **Inside**:  
  o Run unit + integration tests  
  o Validate JSON output schema  
  o Visualize embeddings, tag accuracy, matching precision  
  o Store testing artifacts in /docs/test\_cases/

• 🎯 **Purpose**:  
  Ensure correctness, traceability, modularity before full automation

• 🔁 **Used again in**:  
  Phase 8 (Prefect workflows)  
  Phase 12 (Documentation samples)

• 📤 **Outputs**:  
  /docs/test\_cases/\*  
  Validation logs, charts, small-batch results

🌀 **Phase 8: Prefect Orchestration**  
• 🔁 **Inputs**:  
  o Individual flow.py scripts from each phase  
  o Prefect config, error/retry policies

• ⚙️ **Inside**:  
  o Convert scripts to Prefect tasks  
  o Chain flows via TriggerFlow logic  
  o Add logging, alerts, retries

• 🎯 **Purpose**:  
  Enable robust automation, reusability, and modular flow chaining

• 🔁 **Used again in**:  
  Phase 9 (CI/CD)  
  Phase 11 (Scheduled API backends)

• 📤 **Outputs**:  
  Full DAG-like orchestrated run of all phases

☁️ **Phase 9: GitHub Actions & Deployment**  
• 🔁 **Inputs**:  
  o Prefect flows  
  o GitHub Actions workflow YAML  
  o Environment vars & secrets

• ⚙️ **Inside**:  
  o Schedule workflows via GitHub Actions  
  o Optionally run parallel flows with matrix builds  
  o Dockerize heavy models or microservices

• 🎯 **Purpose**:  
  Turn your logic into a production CI/CD workflow with reproducibility

• 🔁 **Used again in**:  
  Phase 11 (APIs can reuse CI logic)  
  Daily automation for Reddit-to-Recommendation pipeline

• 📤 **Outputs**:  
  CI/CD logs, Docker images, deployment artifacts

📘 **Phase 10: LLM Chatbot Engine**  
• 🔁 **Inputs**:  
  o Cleaned problems  
  o Tags  
  o Embeddings  
  o Recommender logic  
  o RAG datasets (Reddit examples)

• ⚙️ **Inside**:  
  o Route queries: classify as branch lookup or solution request  
  o Use retriever to find similar Reddit cases  
  o Use LLM to generate helpful response  
  o Attach recommended branch

• 🎯 **Purpose**:  
  Answer car repair questions + direct users to nearby experts in real-time

• 🔁 **Used again in**:  
  Phase 11 (API routes)

• 📤 **Outputs**:  
  Structured chat response JSON: solution, branch, explanation

🚪 **Phase 11: Backend Integration (FastAPI)**  
• 🔁 **Inputs**:  
  o Chatbot logic  
  o Recommender module  
  o API schema + endpoints

• ⚙️ **Inside**:  
  o Expose /chat/solve and /recommend/branch endpoints  
  o Input: query, location  
  o Output: JSON with response, recommendation, confidence

• 🎯 **Purpose**:  
  Make system accessible via frontend app or internal dashboard

• 🔁 **Used again in**:  
  Phase 12 (Docs: API specs, usage)

• 📤 **Outputs**:  
  Running FastAPI server  
  API response examples for documentation

📘 **Phase 12: Documentation & Finalization**  
• 🔁 **Inputs**:  
  o All codebase + results  
  o Visuals, samples, model notes

• ⚙️ **Inside**:  
  o Organize documentation under /docs/  
  o Create mind maps, diagrams, JSON examples  
  o Write usage guides and contribution info  
  o Glossary, model versioning, schemas

• 🎯 **Purpose**:  
  Ensure reproducibility, easy onboarding, and professional delivery

• 🔁 **Used again in**:  
  — Final presentation  
  — Maintenance or handover

• 📤 **Outputs**:  
  /docs/README.md, /docs/diagrams/, /docs/test\_cases/, /docs/prompt\_design/

**📈 Future Roadmap**

* Push cleaned data to BigQuery / Snowflake
* Airflow / Prefect DAG integration
* LLM finetuning on automotive domains
* Issue tracker integration (labeling GitHub issues with LLM)
* Hugging Face Space for interactive demo

|  |
| --- |
| car\_clinic\_chatbot/ |
| │ |
| ├── .github/ |
| │ └── workflows/ |
| │ └── main.yaml # GitHub Actions workflow for CI/CD + scheduling |
| │ |
| ├── README.md # Project overview, setup instructions, usage, etc. |
| ├── requirements.txt # Main dependencies |
| ├── Dockerfile # Containerizes the entire project |
| ├── docker-compose.yml # Optional multi-service orchestration |
| ├── .env # Environment variables (excluded via .gitignore) |
| ├── .gitignore # Ignore sensitive and temporary files |
| │ |
| ├── main.py # Entry point for serving the FastAPI app |
| ├── prefect\_flows.py # Centralized Prefect orchestration runner |
| ├── llm\_prompt\_templates.py # Reusable prompt templates for LLMs |
| │ |
| ├── config/ |
| │ └── settings.py # Central configuration: paths, API keys, options |
| │ |
| ├── scripts/ # For one-off runs (e.g., test a flow locally) |
| │ ├── run\_cleaning\_locally.py |
| │ ├── generate\_embeddings\_once.py |
| │ └── dev\_debug\_utils.py |
| │ |
| ├── logs/ # Optional local logging (unless using Prefect Cloud) |
| │ └── pipeline.log |
| │ |
| ├── notebooks/ # Jupyter notebooks for EDA, LLM testing, prototyping |
| │ ├── llm\_prompt\_tuning.ipynb |
| │ ├── sample\_reddit\_analysis.ipynb |
| │ └── tagging\_experiments.ipynb |
| │ |
| ├── data/ |
| │ ├── raw/ # Raw scraped Reddit posts |
| │ ├── cleaned/ # Structured (problem → solution) cleaned data |
| │ ├── augmented/ # Augmented versions: translations, paraphrases |
| │ ├── tagged/ # Semantic tags (diagnosis, components, etc.) |
| │ ├── embeddings/ |
| │ │ ├── problems/ # Embeddings of user issues |
| │ │ └── branches/ # Embeddings of Car Clinic branch expertise |
| │ └── examples.jsonl # Verified labeled examples for development/debugging |
| │ |
| ├── python\_scripts/ |
| │ ├── reddit\_data\_extractor/ |
| │ │ ├── \_\_init\_\_.py |
| │ │ ├── config.py |
| │ │ ├── extractor.py |
| │ │ ├── extractor\_Base.py |
| │ │ ├── flow.py |
| │ │ ├── reddit\_client.py |
| │ │ ├── scraper.py |
| │ │ ├── writer.py |
| │ │ └── utils.py |
| │ │ |
| │ ├── reddit\_data\_cleaner/ |
| │ │ ├── \_\_init\_\_.py |
| │ │ ├── flow.py |
| │ │ ├── cleaner.py |
| │ │ ├── llm\_cleaner.py # Test-cleaner with a sample prompt locally |
| │ │ ├── llm\_runner.py |
| │ │ ├── preprocessor.py |
| │ │ ├── postprocessor.py |
| │ │ └── utils.py |
| │ │ |
| │ ├── data\_augmenter/ |
| │ │ ├── \_\_init\_\_.py |
| │ │ ├── flow.py |
| │ │ ├── paraphraser.py |
| │ │ ├── translator.py |
| │ │ ├── noise\_injector.py |
| │ │ ├── utils.py |
| │ │ └── config.py |
| │ │ |
| │ ├── tag\_generator/ |
| │ │ ├── \_\_init\_\_.py |
| │ │ ├── flow.py |
| │ │ ├── tagger.py |
| │ │ ├── constants.py |
| │ │ └── utils.py |
| │ │ |
| │ ├── embedding\_generator/ |
| │ │ ├── \_\_init\_\_.py |
| │ │ ├── flow.py |
| │ │ ├── embedder.py |
| │ │ ├── utils.py |
| │ │ └── config.py |
| │ │ |
| │ └── branch\_recommender/ |
| │ ├── \_\_init\_\_.py |
| │ ├── flow.py |
| │ ├── matcher.py |
| │ ├── filters.py |
| │ ├── utils.py |
| │ └── ranker.py |
| │ |
| ├── chatbot/ # Chat interface backend logic |
| │ ├── query\_classifier.py # Classifies the type of user issue |
| │ ├── retriever.py # Retrieves relevant past cases or FAQs |
| │ ├── reasoner.py # Final reasoning LLM logic |
| │ ├── branch\_suggester.py # Branch recommendation from embeddings |
| │ ├── formatter.py # Formats chatbot responses |
| │ └── fallback.py # Handles edge cases or failure modes |
| │ |
| ├── api/ # FastAPI backend serving recommendations |
| │ ├── routes/ |
| │ │ ├── chat.py # Chat endpoint |
| │ │ └── recommend.py # Branch suggestion endpoint |
| │ ├── models.py # Pydantic request/response models |
| │ ├── dependencies.py # Shared dependencies or security |
| │ └── server.py # FastAPI app entrypoint |
| │ |
| ├── tests/ |
| │ ├── conftest.py # Shared pytest fixtures |
| │ ├── unit/ |
| │ │ ├── test\_cleaner.py |
| │ │ ├── test\_embedder.py |
| │ │ └── ... |
| │ ├── integration/ |
| │ │ ├── test\_end\_to\_end\_pipeline.py |
| │ │ └── ... |
| │ └── test\_data/ |
| │ ├── sample\_raw.json |
| │ └── expected\_cleaned.json |
| │ |
| └── docs/ # Visuals and documentation |
| ├── architecture\_diagram.png |
| ├── dag\_flow.png |
| ├── embedding\_guide.md |
| ├── branch\_matching.md |
| ├── api\_schema.md |
| └── sample\_output\_examples.md |