. 🎯 Project Goal and Scope

The primary goal of Plum-Bot is to simulate a plumber's initial diagnostic process using a knowledge base and logical inference.

- **Goal:** To diagnose a probable cause for a user-reported plumbing issue (symptom) within a specific fixture and provide a step-by-step resolution plan.
- **Scope:** The system's knowledge base is limited to a few common issues, specifically focusing on drainage clogs (e.g., P-trap blockage) and running toilets (e.g., flapper issues). Complex system-wide failures or issues requiring specialized tools are excluded or noted as needing professional help.

---

2. 🏗️ System Architecture and Implementation

Plum-Bot follows a standard expert system architecture, divided into three main components implemented across three separate Prolog files:

2.1. Inference Engine (main.pl)

The main.pl file serves as the system's controller and inference engine.

- **Initialization:** The start_diagnosis/0 predicate clears all dynamically asserted facts (symptom/2, fixture/1, user_confirms/1) from previous sessions.
- **Input Handling:** get_user_input prompts the user for the fixture and the main problem. It uses read(Variable) to accept user input as a Prolog atom and asserts these as dynamic facts (e.g., assertz(fixture(sink))). It also includes **conditional questioning**; for instance, if the fixture is a toilet with running_water, it asks a follow-up question about a sticking handle.
- **Diagnosis & Output:** It calls diagnose(Cause) to initiate the inference process. If a cause is found, it calls write_output(Cause) to format the results; otherwise, it reports that no diagnosis could be found.

2.2. Diagnostic Rules Knowledge Base (kb_rules.pl)

This file contains the logical rules that map symptoms and user confirmations to a probable cause.

- **Predicate:** The main rule is diagnose(Cause).
- **Inference Logic (Example):** Diagnosis relies heavily on Prolog's built-in unification and the principle of **Negation As Failure (NAF)**. For example:
    - diagnose(faulty_flapper) only succeeds if symptom(toilet, running_water) is true **AND** the chain issue is **NOT** the diagnosis (\+

diagnose_flapper_chain_issue). This prioritizes the simpler, more specific **flapper_chain_issue** diagnosis first.

- diagnose_flapper_chain_issue requires both the main symptom and the user's specific confirmation (user_confirms(handle_sticking)).

2.3. Solution Knowledge Base (kb_solutions.pl)

This file contains the recommended actions for each confirmed diagnosis.

- **Predicate:** recommend(Cause, Steps) maps a diagnosis atom (e.g., flapper_chain_issue) to a list of step-by-step instructions.
- **Output Formatting:** The list_steps/2 predicate in main.pl iterates over this list, formatting the steps with numerical indexing for clear instructions.

---

3. 🧪 Testing and Results

The system was tested with various input combinations to verify the correct application of the diagnostic rules.

3.1. Test Case 1: Clogged Drain

Input	**Expected Diagnosis**	**Actual Output**
Fixture: sink	clogged_p_trap	Correct. Rule applied: symptom(sink, slow_drain) and sink is not toilet.
Problem: slow_drain		

3.2. Test Case 2: Running Toilet (Chain Issue)

Input	**Expected Diagnosis**	**Actual Output**
Fixture: toilet	flapper_chain_issue	Correct. The system prioritized the specific chain diagnosis over the general faulty_flapper diagnosis.
Problem: running_water		
Handle Sticking? yes		

---

4. 📈 Conclusion and Future Enhancements

Plum-Bot successfully demonstrates the fundamental principles of an expert system, including knowledge representation using facts and rules, and backward-chaining inference to reach a conclusion.

4.1. Conclusion

The system provides accurate diagnoses within its defined knowledge scope and effectively links diagnoses to concrete, formatted action plans. The use of dynamic facts for user input and NAF for diagnostic prioritization are key strengths of the Prolog implementation.