1. Intent parsing and missing-information detection

We rely on fast, regex-based rules (rather than heavier ML models) to categorise user input into three high-level intents: calculations, drinkware product queries, and outlet information requests. During the same pass we check whether any required “slots” are missing. For example, a calculation must include a complete math expression; a product query needs a specific item such as “mug”; an outlet query must specify a location or the exact info desired (hours, phone, address). Whenever even one slot is missing, the system forces an ASK FOLLOW-UP action—this always outweighs executing the detected intent.

2. Action-selection decision tree

The planner applies a simple hierarchy:

1. If anything is missing → ASK FOLLOW-UP
2. Else if a calculation intent is present → CALL CALCULATOR
3. Else if it is a product query → CALL RAG (vector search)
4. Else if it is an outlet query → CALL TEXT2SQL
5. Otherwise → FINISH (end the conversation gracefully)

Conversation state (Initial, Processing, Waiting-for-Clarification, Completed) is passed through the context dictionary so future logic can branch on it. When a user asks several things at once, missing-slot checks still come first; otherwise the same priority order applies.

3. Action execution

All tool calls funnel through one execute() function, keeping routing and error-handling in one place.

- Calculator – We strip the natural-language prompt down to a math expression, allow only digits, +, −, *, /, dots, and parentheses, then evaluate with eval() while disabling built-ins.
- RAG endpoint – A small keyword lookup returns a short description of mugs, cups, bottles, etc., or a catalogue fallback.
- Text2SQL endpoint – Simple pattern matching answers hours, phone, or lists outlets by city; it defaults to a full directory if the query is vague.

4. Error handling and follow-ups

The bot never fails silently. Invalid math returns an apology plus guidance; unknown products trigger a catalogue suggestion; vague outlet questions prompt for a city or data

type. Follow-up wording is tailored to the intent (e.g., “Could you provide a complete mathematical expression?”).

5. Performance choices

All regexes are pre-compiled once at startup and the IntentExtractor is a module-level singleton, avoiding repeated compilation. Passing state explicitly in the context keeps the planner stateless and easily testable.

6. Integration architecture

planner.py holds purely stateless planning utilities, while agentic_bot.py maintains conversation memory and invokes the planner. Every turn is logged with timestamps, chosen action, detected intents, and any missing slots—useful for analytics or debugging.

7. Testing strategy

We unit-test every branch (ASK FOLLOW-UP, calculator, RAG, Text2SQL, and finish) and run full integration flows, including edge cases such as incomplete math or unknown locations. Current test suite: 41 cases, 100 % pass.