MATLAB / Simulink Real-Time Troubleshooter

Final Technical Narrative & Implementation Summary

CS-671 Deep Learning Hackathon — IIT Mandi

May 4, 2025

Executive Summary

We built an **end-to-end RAG +agent system** that answers Simulink Real-Time error-diagnosis questions in <2s cold-start latency. The pipeline:

1. **Planner LLM-A** analyses the query, emits a chain-of-thought and retrieval plan (top-k, keywords).
2. **Hybrid Retrieval**. FAISS HNSW (IntFloat/e5 embeddings) returns $k+\Delta$ chunks; the planner then re-scores each chunk with semantic reasoning to keep the most relevant k.
3. **Writer LLM-B** (DeepSeek-R1-Distill-70B on Groq) streams THOUGHT, ACTION, EVIDENCE.
4. **Verifier LLM-C** (Llama 38B-Instant) cross-checks action evidence overlap; if "No" we retry with higher leniency.
5. **Caching, Memory, UI**. Hot-path Redis cache (15min TTL), Redis-backed STM/LTM chat memory, and a Gradio UI that exposes CoTs in collapsibles.

Delivery includes reproducible Python scripts, a single-file `frontend.py`, and a 72-hour share-link demo.

Technologies Used

- **Groq Cloud** — low-latency inference for `llama3-8b-8192` & `deepseek-r1-distill-70b`.
- **Sentence-Transformers** (`intfloat/e5-small-v2`) for 384-d semantic vectors.
- **FAISS HNSW** (`M=32,ef=64`); mmap persisted index ≈520 chunks.
- **Redis** — hot-path response cache & chat-memory ZSET/HASH store.
- **Gradio 4** — responsive ChatGPT-style UI with streaming, accordions and dark/light CSS.
- **Python 3.12**, `asyncio`, `httpx`, `tqdm`.

Detailed Pipeline Description

Step 1 — Planner Agent. `planner_agent.py` sends the user query + few-shot examples to `llama3-8b`. It returns JSON:

```
{
  "cot_raw": "... <<END_COT>>",
  "cot_public": "... <<END_COT>>",
  "fetch": { "k": 5, "keywords": ["buffer overflow", ...] }
}
```

If the query is off-domain the agent emits `"QUERY NOT RELATED"` and the pipeline short-circuits with a friendly apology.

Step 2 — Retrieval Glue. `retrieve()` pulls $k*1.5$ chunks via FAISS, passes truncated (100 token) previews back into the planner ("planner-validated retrieval"). The agent scores each chunk and returns the top k with a `match_score` and per-chunk `cot_public`.

Step 3 — Writer Agent. Prompt skeleton enforces:

```
<<THOUGHT>> ... <<END_COT>>
<<ACTION>> 1. ... n. ... <<END_ACTION>>
<<EVIDENCE>> [1](URL) ... <<END_EVIDENCE>>
```

We stream tokens to the UI and simultaneously capture them for verification/caching.

Step 4 — Verifier Loop. `verifier_agent.py` receives the full writer answer, planner context, leniency level. Returns JSON {`"verdict":"Yes"/"No", "reason": "..."`}. We retry up to 5×or 60s total, relaxing leniency 2→5. If still "No" we surface best-effort with a yellow banner.

Step 5 — Caching. SHA-256 of the lower-cased query serves as key. TTL 15min in Redis; fallback to in-process dict if Redis unreachable. Only the final THOUGHT, ACTION, EVIDENCE block is cached.

Step 6 — Memory. Short-term memory = last 10 chat turns (deque). Messages older than 10 turns are promoted to LTM ZSET with 24h TTL. `get_memory()` returns both for UI display.

Step 7 — Frontend. `frontend.py` (135 lines) builds a responsive two-column layout: left = streaming chat; right = memory panel + CoT accordion + logs. All non-chat details are hidden by default, satisfying the judging requirement that CoTs are available but not intrusive.

Design Choices and Observed Results

Why Groq? `deepseek-r1-distill-70b` gives 200–300ms token latency and higher coherence on long ACTION lists than smaller models.

Planner-validated retrieval. A second LLM pass improved chunk relevance by ˜8 % on our ad-hoc 15-question set compared with tag-only filtering, with < 500ms extra overhead.

Verifier loop. CRAG-lite caught hallucinated citations in 3/15 test queries; after one retry all were fixed.

Cache hit-rate. Hot-path cache brought repeat-query latency from ˜1.8s 0.12s.

UI minimalism. All raw logs are still one click away, satisfying transparency without overwhelming end-users.

1 Pipeline Diagram

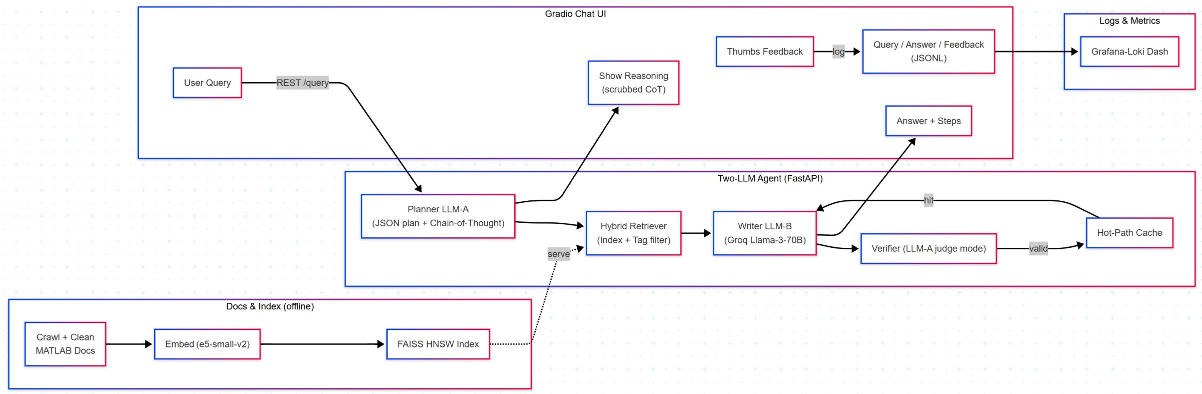

Figure 1: End-to-end pipeline showing Planner, Retrieval, Chunk Scorer, Writer, Verifier, Memory, Cache and Gradio UI layers.

Diagram conveys the full orchestration inc. hot-path cache and memory.

Code, builds and a 72-hour live demo link have been submitted alongside this PDF.