CASE STUDY

1. Executive Summary
   This case study examines a lightweight enhancement to a rule-based e-commerce chatbot: adding a keyword-driven FAQ fallback and a response cache to improve answer coverage and lower latency without complex ML. The approach targets common developer-round requirements and demonstrates measurable gains in resolved queries and faster repeat responses with simple, file-driven assets (CSV and JSON).
2. Business Context
   E-commerce support must instantly answer questions about shipping, returns, product availability, and pricing. Traditional AIML-style chatbots handle exact patterns but fail on paraphrasing and re-ask scenarios, hurting experience and conversion. A structured fallback and basic caching are established conversational design practices to sustain dialogue and speed service while remaining easy to maintain for small teams.
3. Problem Statement

- High “no match” rate when customer phrasing differs from predefined rules.
- Redundant processing for repeated questions increases wait time and compute usage.
- Need a beginner-friendly, low-cost improvement aligned with academic developer round deliverables.

4. Objectives

- Reduce “no match” outcomes via a curated FAQ fallback triggered on rule failure.
- Reduce latency for repeated queries using in-memory response caching.
- Maintain a simple, auditable codebase with CSV/JSON data inputs for transparent demos and viva defense.

5. Solution Overview
   The solution inserts two components into a classic rule pipeline:

- FAQ Fallback: On rule miss, compute token overlap between user query and FAQ questions; return the closest answer above a small threshold or guide the user with options, following fallback best practices (acknowledge, suggest, and offer buttons or a handoff path).
- Response Cache: Store normalized user queries with answers; on repeat, return instantly, mirroring prompt/response caching benefits documented in AI services and chatbot systems.

6. System Design

- Flow: Normalize → Cache lookup → Rules/AIML → Helpers (stock/total) → FAQ fallback → Cache store → Respond.
- Data: inventory.csv (product_id, name, price, stock_qty) for stock and totals; faq.json for fallback answers.
- Ops: Periodically review logs of fallback triggers and expand FAQs/patterns, a standard continuous-improvement loop in conversational design.

7. Implementation Details

- Tech: Python, CSV/JSON I/O; modular loader for data, router for steps, and simple tokenizer for overlap matching.
- Fallback algorithm: token-overlap scoring; configurable threshold; safe default message with options if no FAQ passes threshold.
- Cache: dictionary keyed by normalized text; optional TTL to refresh stale content; negligible memory footprint for coursework scale.

8. Sample Use Cases

- FAQ: “How long is shipping?” and paraphrases (“Does shipping take long?”) route to the same answer via fallback when rules miss.
- Stock Check: “Is wireless mouse in stock?” uses inventory.csv and returns availability; demonstrates real catalog linkage.
- Repeated Queries: Asking the same FAQ twice returns an instant response on the second attempt due to caching, aligning with prompt-caching performance guidance.

9. Evaluation Method

- Coverage: Percentage of test queries receiving a helpful answer; compare rule-only vs. rule+fallback using a 40–50 item test set derived from store FAQs and paraphrases.
- Latency: Average response time for repeated queries with and without cache across five runs, controlling for network/IO variance as feasible.

10. Results (insert measured values)

- Coverage lift: e.g., from 68% to 88% after enabling fallback; fewer dead-ends and improved task completion for paraphrased inputs.
- Latency reduction: e.g., repeat response time decreased by 40–60% with caching, consistent with reported gains in AI prompt caching scenarios.

11. Comparative Analysis

- Baseline AIML: fast for exact patterns; brittle on paraphrases; redundant work on repeats; typical of instructional e-commerce chatbot reports.
- Enhanced Bot: robust fallback keeps users in flow, and cache boosts responsiveness under repetitive demand; minimal code and infra overhead.