Phase 3 Final Report: LLM Safety Research Portal

Taha Zakir March 2026

---

1. Introduction

This report describes the design, implementation, and evaluation of a Personal Research Portal (PRP) built over a corpus of 17 peer-reviewed papers on LLM jailbreaking and safety. The portal supports the full research workflow: asking questions, retrieving cited evidence, synthesizing findings into structured artifacts, and exporting results. It builds on the Phase 2 RAG pipeline by adding a web interface, research threads, artifact generation, export capabilities, and three stretch goal features.

The research question driving the corpus is: "How do current jailbreak attacks exploit LLM safety mechanisms, and how effective are existing defenses and evaluation benchmarks at measuring these vulnerabilities?" The corpus spans attack methods (PAIR, Crescendo, ArtPrompt, Agent Smith, Foot-in-the-Door, X-Teaming), defense mechanisms (SafeDecoding), evaluation benchmarks (HarmBench, SORRY-Bench, AgentHarm, BeaverTails, Aegis 2.0), risk frameworks (NIST AI RMF, Frontier AI Risk Management), and analyses of safety training failures (Jailbroken, Self-Jailbreaking).

---

2. Architecture

The system has four main components:

Ingestion Pipeline. PDFs are parsed using Docling, which extracts structured sections (Abstract, Methods, Results, etc.) from each paper. Sections are split into chunks using sentence-boundary splitting with a target of 1024 tokens and 100-token overlap. Chunks are embedded using google/embeddinggemma-300m (768 dimensions) and stored in ChromaDB with HNSW cosine similarity indexing. Each chunk carries metadata including source_id, section title, filename, year, type, and authors.

RAG Pipeline. When a user asks a question, the query is embedded with the same model using a task-specific prompt ("task: search result | query: ..."). ChromaDB returns the nearest chunks, which go through a source diversification step: we fetch 2x the requested number of chunks and cap at 3 per paper, so that synthesis queries draw from multiple sources rather than being dominated by one. The chunks and question are passed to Claude Haiku, which generates an answer with strict citation rules. Every claim must cite a specific [source_id, chunk_id]. The system prompt forbids fabricated citations and requires explicit flagging of missing or conflicting evidence.

Web Portal. A Streamlit application with seven pages provides the user interface. The pages are described in Section 4.

Caching Layer. Every Claude API call is intercepted by a caching function that computes a SHA-256 hash of the model name, system prompt, and user message. If a cached response exists, it is returned immediately. If REPLAY_MODE is enabled and no cache exists, the system raises an error. This allows the full evaluation suite to be reproduced without an API key.

The data flow end-to-end is: PDF files in data/raw/ are parsed into data/processed/ as JSON, chunked and embedded into data/vector_store/ (ChromaDB), queried by the RAG pipeline, and

the API responses cached in data/cache/. Logs go to logs/run_logs.jsonl. Generated artifacts go to outputs/.

---

3. Design Choices

ChromaDB over FAISS. FAISS would have been faster for pure vector search, but ChromaDB provides built-in metadata filtering. This was important because the portal supports filtering by year, author, and document type at query time. With FAISS, I would have had to build a separate metadata index and do post-filtering manually. ChromaDB also handles persistence automatically, so there is no need to save and reload index files.

embeddinggemma-300m. This is a free, open-source embedding model that runs locally. It produces 768-dimensional vectors and supports task-specific prompting (different prefixes for queries vs documents). The main trade-off is that it is smaller than commercial alternatives like text-embedding-3-large, which may affect retrieval quality on highly technical passages. However, it keeps the system free of embedding API costs and makes ingestion fully reproducible offline.

Chunking strategy. The 1024-token target with 100-token overlap was chosen as a balance between context length and retrieval precision. Smaller chunks would give more precise retrieval but lose surrounding context. Larger chunks would include more context but reduce the number of distinct results. Section-aware splitting means the chunker respects paper structure boundaries, so a chunk from the Methods section will not bleed into the Results section. The sentence-boundary constraint ensures no chunk ends mid-sentence, which would confuse both retrieval and generation.

Source diversification. For synthesis queries comparing multiple papers, standard top-k retrieval tends to return many chunks from the single most relevant paper. The diversification step caps each paper at 3 chunks and fills the remaining slots from other papers. This noticeably improved the quality of cross-source synthesis answers, though it can occasionally displace a highly relevant fourth chunk from a dominant source.

Two-model approach. Claude Haiku handles regular Q&A queries because it is fast and cheap, and the strict citation prompt keeps it on track. Claude Sonnet handles artifact generation (evidence tables, synthesis memos, gap analysis, disagreement maps) because these require more nuanced synthesis across many chunks. In practice, Haiku rarely misses citations on direct queries, but Sonnet produces noticeably more coherent long-form artifacts.

API caching. The caching key is a SHA-256 hash of the exact (model, system prompt, user message) triple. This means that if anything changes in the prompt or retrieved chunks, a new cache entry is created. The deterministic hashing ensures the same query with the same retrieval results always hits the same cache. The trade-off is that the cache only covers queries that have been run before. Interactive queries from the UI will not be cached unless the user has previously asked that exact question with the same retrieval results.

---

4. Portal Features

The portal has seven pages:

Search and Ask. The main query interface. Users type a question, optionally apply sidebar filters

(year, document type, author substring), and receive a cited answer. Metrics are shown (latency, chunks retrieved, sources cited). Each retrieved chunk is expandable to show the source text and cosine distance. Answers can be saved to a research thread.

Research Threads. File-based persistence (JSON) that stores query, answer, retrieved chunks, filters, and timestamp for each entry in a thread. Users can browse threads, rename them, or delete them. Threads accumulate across sessions.

Artifact Generator. Produces two artifact types. Evidence tables follow the schema: Claim, Evidence snippet, Citation [source_id, chunk_id], Confidence (HIGH/MEDIUM/LOW), and Notes. Synthesis memos are 800-1200 words with inline citations and a reference list. Both can be exported as Markdown, CSV (tables only), or PDF. The topic can be entered directly or derived from a research thread.

Gap Finder (stretch goal). Identifies what evidence is missing in the corpus for a given topic. The system first retrieves chunks and generates an answer, then makes a second LLM call analyzing what is missing. The output includes an evidence coverage summary, identified gaps (each with importance level and type), and suggested search queries to find the missing evidence. See outputs/gap_analysis_defense_mechanisms.md for an example.

Disagreement Map (stretch goal). Surfaces where sources agree and disagree on a topic. Retrieves chunks with high source diversity, then asks the LLM to categorize points of agreement (with strength ratings), points of disagreement (typed as methodological, empirical, definitional, or scope), and unresolved questions. See outputs/disagreement_map_attack_effectiveness.md for an example.

Corpus Explorer (stretch goal). Lets users browse all 17 papers with full metadata, filter by year, source type, venue, and topic tags. Each paper can be expanded to show its parsed sections and character counts. The page also shows a tag distribution summary across the corpus.

Evaluation Dashboard. Displays results from the 20-query evaluation suite. Shows aggregate metrics, per-query-type breakdown, and per-query drill-down with full answers and citation details.

---

5. Evaluation

5.1 Query Set Design

The evaluation set contains 20 queries split into three types:

- 10 direct queries that target a single paper's content (e.g., "What does ASR measure in HarmBench, and what are its known limitations?")
- 5 synthesis queries that require comparing across two papers (e.g., "Compare PAIR vs Crescendo attack strategies")
- 5 edge-case queries that test the system's handling of absent or limited evidence (e.g., "Does the corpus contain evidence about watermarking for detecting AI-generated prompts?")

Each query includes expected source IDs so that source recall can be computed automatically.

5.2 Metrics

Three metrics are computed programmatically:

- **Citation precision**: fraction of cited chunk IDs that actually exist in the vector store. Measures whether the model is citing real chunks.
- **Groundedness**: fraction of cited chunk IDs that were in the retrieved set for that query. Measures whether the model is citing chunks it was actually given.
- **Source recall**: fraction of expected source papers that appear in the retrieved chunks. Measures whether retrieval is finding the right papers.

5.3 Results

All 20 queries completed successfully. Aggregate metrics:

Metric	Score
Citation Precision	85%
Groundedness	85%
Source Recall	96.88%

Per-type breakdown:

Type	Count	Citation Precision	Groundedness
Direct	10	90%	90%
Synthesis	5	100%	100%
Edge-case	5	60%	60%

The synthesis queries scored highest because the source diversification ensures chunks from both expected papers are retrieved, and Haiku does a good job citing from each. Direct queries score well because the evidence is concentrated in one paper. Edge-case queries score lower because several correctly return "EVIDENCE MISSING" with no citations, which registers as 0% citation precision by the metric even though the behavior is correct.

5.4 Failure Cases

Failure 1: d07 (NIST framework query). Citation precision and groundedness both scored 0%. The model generated an answer about NIST AI RMF risk categories that included citations in the format [nist.ai.100-1, nist.ai.100-1_c28], but the citation extraction regex did not match the period in "nist.ai.100-1" as a valid source_id pattern. The answer itself was correct and grounded in retrieved chunks, but the metric failed to detect the citations due to the unusual source_id format containing periods. This is a metric limitation, not a generation failure.

Failure 2: e01 (watermarking query). Citation precision and groundedness scored 0%, but this is correct behavior. The corpus does not contain papers on watermarking for detecting jailbreak prompts, and the system correctly returned "EVIDENCE MISSING" with an explanation of what is covered instead. The evidence_handling metric confirms this was correctly flagged (correctly_flags_missing: true).

Failure 3: e05 (constitutional AI query). Same pattern as e01. The corpus does not discuss constitutional AI, and the system correctly flagged this as missing evidence. The 0% citation score is expected because there is nothing to cite.

These failure cases reveal that the citation precision metric underestimates the system's actual performance. Queries d07 and e01/e05 all produce correct behavior, but the metric penalizes them. If we exclude the intentionally citation-free edge-case responses and account for the regex issue on d07, the effective citation precision on queries where the system attempts to cite would be closer to 94%.

---

6. Generated Artifacts

The following artifacts are included in the outputs/ directory:

- **evidence_table_multi_turn_attacks.md**: Evidence table on multi-turn jailbreak attack strategies, with 8 rows covering Crescendo, FITD, X-Teaming, and GOAT findings.
- **evidence_table_safety_benchmarks.md**: Evidence table on LLM safety evaluation benchmarks, covering HarmBench, SORRY-Bench, AgentHarm, and Aegis 2.0.
- **evidence_table_Multi-turn_jailbreak_strategies_20260224_100810.md**: Earlier evidence table generated during Phase 2 testing.
- **synthesis_memo_Compare_the_methodology_of_'static'_jail_20260224_101252.md**: Synthesis memo comparing static and dynamic jailbreak methodologies.
- **gap_analysis_defense_mechanisms.md**: Gap analysis on defenses against jailbreak attacks, identifying missing evidence on formal verification methods, real-time detection systems, and cross-model transferability of defenses.
- **disagreement_map_attack_effectiveness.md**: Disagreement map on the effectiveness of automated red teaming, identifying methodological disagreements between GOAT's and X-Teaming's approaches and empirical differences in reported attack success rates.

Six research threads are also saved in outputs/threads/, containing accumulated query-answer-evidence sessions from development and testing.

---

7. Trust Behaviors

The system implements several trust-related behaviors:

- Every answer includes [source_id, chunk_id] citations that can be traced back to specific passages in specific papers.
- The system prompt explicitly forbids fabricating citations: "Do NOT invent or fabricate any citations. Only cite chunk IDs that appear in the context below."
- When evidence is missing, the system responds with "EVIDENCE MISSING" and explains what the corpus does cover instead, rather than hallucinating an answer.
- When evidence conflicts across sources, the system is instructed to flag the conflict with citations to both sides. The disagreement map feature makes this analysis more systematic.
- The gap finder explicitly identifies what is not in the corpus and suggests what to look for next.

---

8. Reproducibility

The system is designed to be fully reproducible without an Anthropic API key:

1. All dependencies are pinned in pyproject.toml and resolved in uv.lock.
2. Source PDFs are stored in data/raw/ (17 files).
3. The data manifest (data/data_manifest.json) contains full metadata for every source.
4. All Claude API responses are cached in data/cache/ (25 files covering all evaluation queries and sample artifacts).
5. Setting REPLAY_MODE=true causes the system to read from cache instead of calling the API.
6. The full evaluation suite runs in about 1 second in replay mode and produces identical metrics.
7. A single Makefile provides one-command targets: make setup, make ingest, make evaluate, make app.

---

9. Limitations

Retrieval is vector-only. There is no BM25 or keyword-based hybrid retrieval. This means exact keyword matches (like specific metric names or acronyms) may rank lower than semantically similar but lexically different passages. Adding BM25 as a second retrieval path with score fusion would likely improve precision on technical queries.

No reranking. Retrieved chunks go directly to the LLM without a cross-encoder reranking step. A reranker could improve the ordering of chunks before they reach the generation model, especially for queries where the most relevant chunk is not the nearest in embedding space.

Citation regex limitations. The citation extraction regex expects source_ids made of lowercase alphanumeric characters, underscores, and hyphens. Source IDs with periods (like nist.ai.100-1) are not fully captured, which causes the evaluation metrics to undercount valid citations. Fixing the regex would bring the reported metrics closer to the actual performance.

Corpus is static. The 17 papers were selected manually and cannot be expanded through the UI. A production system would need a way to add new papers, re-ingest, and update the vector store without starting from scratch.

PDF parsing imperfections. Docling handles most paper layouts well, but LaTeX equations, complex tables, and figures are either stripped or partially garbled during extraction. Papers that rely heavily on mathematical notation (like some defense mechanism papers) lose some technical detail in the parsed text.

Single-user design. Streamlit runs as a single-process server without authentication. Multiple users accessing the portal simultaneously could interfere with each other's session state.

---

10. Next Steps

Hybrid retrieval. Adding BM25 alongside vector search and fusing the scores would improve keyword-sensitive queries without losing semantic matching. Libraries like rank-bm25 integrate easily.

Cross-encoder reranking. Running a cross-encoder (like cross-encoder/ms-marco-MiniLM-L-6-v2) on the top 20 retrieved chunks before passing the top 8 to the LLM would improve generation quality at a small latency cost.

Query decomposition. For complex multi-part questions, breaking the query into sub-queries and retrieving separately for each would improve coverage. This would be especially useful for synthesis queries where each sub-question targets a different paper.

Fix citation regex. Updating the regex pattern to handle periods in source_ids would correct the evaluation metrics and give a more accurate picture of system performance.

Live corpus expansion. Adding a UI page to upload new PDFs, trigger ingestion, and update the vector store would make the system useful for ongoing research rather than a fixed snapshot.

Agentic research loop. Building a plan-search-read-synthesize loop where the system automatically decomposes a research question, retrieves evidence for each part, identifies gaps, and iterates until coverage is sufficient. The gap finder already provides the analysis; the next step is automating the iteration.

---

11. AI Usage Disclosure

Claude Code (Anthropic CLI, Claude Opus). Used for implementing the API response caching layer in generate.py, building the three stretch goal UI pages (gaps.py, disagreements.py, corpus.py), updating the data manifest schema, writing prompt templates for gap analysis and disagreement mapping, updating navigation and configuration files, and generating the report notes skeleton. All generated code was reviewed for correctness, tested locally, and adjusted where needed.

Claude API (Haiku and Sonnet). Used at runtime for generating RAG answers with citations (all 20 evaluation queries plus interactive queries), generating evidence tables and synthesis memos, and producing gap analysis and disagreement map outputs. Citation correctness and groundedness were evaluated programmatically. System prompts were refined based on output quality during development.