Phase 3 Final Report: Personal Research Portal (PRP)

Project: Personal Research Portal for the City Digital Twins domain

1. Overview

This repository delivers a Personal Research Portal that supports a complete workflow from research question to grounded synthesis. The portal ingests a domain corpus, retrieves evidence, generates answers with resolvable inline citations, produces exportable research artifacts, saves research threads, and logs evaluation runs for reproducibility and auditing.

Domain focus: city digital twins, urban AI, and digital twin enabled decision support for urban planning and management.

2. Research framing

Main question:
How do city digital twins enable evidence-backed decision support for urban planning and management?

The evaluation query set in `src/eval/query_set.json` operationalizes this into direct questions, cross-source synthesis questions, and edge cases where the correct behavior is to surface missing evidence and propose an actionable next retrieval step.

3. Portal workflow and UI

The Phase 3 deliverable includes a working Streamlit app in `src/app/app.py` (launched via `python run_app.py`) with a research workflow that mirrors how a student would actually use a corpus.

Typical flow:

- Ask a question in **Search & Ask** and choose retrieval options (hybrid retrieval, reranking, top-k).
- Inspect the returned **answer** plus the **retrieved evidence cards**. Each evidence card shows `(source_id, chunk_id)`, a snippet, and resolved metadata from the manifest.
- Save the run as a **research thread**. The saved thread stores query, retrieved chunks, answer, citations used, and retrieval configuration.
- Generate **research artifacts** from the saved thread (evidence table, annotated bibliography, synthesis memo, disagreement map).
- Export artifacts to **Markdown, CSV, PDF, and BibTeX**, then download or commit them under `outputs/exports/`.
- Run the evaluation set in the **Evaluation** page (baseline, enhanced, or compare) and review the resulting report file under `outputs/eval/`.

The UI also exposes optional metadata filters (publication year range, source types) so users can narrow retrieval by corpus facets while keeping citations resolvable.

4. System architecture

The system is organized into four layers.

Data layer

- `data/raw/`: stored PDFs for the corpus
- `data/processed/`: extracted text and chunk JSON per source, plus `all_chunks.json`
- `data/embeddings/`: FAISS index, BM25 index, embedding caches, retriever metadata

- `data/data_manifest.csv` : source metadata and file paths for citation resolution

Core RAG layer

- `src/rag/rag_system.py` : retrieval, fusion, reranking, answer generation, citation validation, and query logging

Product layer

- `src/app/app.py` : Streamlit portal with ask, sources, threads, artifacts, export, and evaluation view
- `src/app/thread_manager.py` : thread persistence and retrieval
- `src/app/artifact_generator.py` : evidence table, annotated bibliography, synthesis memo, disagreement map, and gap finder utilities
- `src/app/export_manager.py` : export to Markdown, CSV, PDF, and BibTeX

Evaluation layer

- `src/eval/evaluator.py` : runs the fixed query set, computes metrics, saves CSV and JSONL logs, and writes Markdown summary reports

5. Corpus profile and citation resolvability

Corpus snapshot for this Phase 3 submission:

- Sources: 22 total (years 2021 to 2025)
- Source types: journal_article: 11; arxiv_preprint: 10; technical_report: 1
- Chunks: 2512 total (per source: min 35, max 196, avg 114.2)

The citation contract is explicit and testable.

- Citation surface form: `(source_id, chunk_id)`
- Resolution path: `source_id` resolves through `data/data_manifest.csv` , and `chunk_id` resolves through processed chunk JSON under `data/processed/` .

The manifest schema includes the required metadata fields and local paths (`raw_path` , `processed_path`) so every citation can be traced back to a stored artifact.

6. Ingestion pipeline

The ingestion pipeline in `src/ingest/ingest_pipeline.py` performs:

- PDF extraction using `pdfplumber`
- cleaning and normalization
- section detection using heading regex
- section-aware chunking with overlap
- writing per-source chunk files plus a global `all_chunks.json`

Chunking uses a character-budget sliding window with overlap. When section headers are detected, chunking is constrained to section spans to reduce topic drift inside chunks.

7. Retrieval design and RAG core

Retrieval and grounding are implemented in `src/rag/rag_system.py` . The system builds two retrievers and fuses them when hybrid mode is enabled.

7.1 Vector retrieval (FAISS)

Embeddings use `SentenceTransformer("all-MiniLM-L6-v2")`. The FAISS index is `IndexFlatL2`, which is reliable and parameter-free for a moderate corpus size. Distances are converted to a similarity-like score using `1 / (1 + distance)` for ranking and logging.

Caching is implemented under `data/embeddings/` and guarded by `retriever_meta.json`, which stores a chunk-count and SHA1 hash of `all_chunks.json`. When ingestion changes chunks, indexes rebuild automatically.

7.2 Lexical retrieval (BM25)

BM25 is implemented with `rank_bm25.BM25Okapi`. Each chunk is tokenized via `lower().split()`. This intentionally simple tokenizer keeps dependencies and runtime minimal, and it helps on acronym-heavy queries.

7.3 Hybrid fusion (Reciprocal Rank Fusion)

Hybrid retrieval combines vector and BM25 via Reciprocal Rank Fusion (RRF). Each retriever returns a larger candidate list, then ranks are fused by:

```
score(d) = w_vec / (rrf_k + rank_vec(d)) + w_bm25 / (rrf_k + rank_bm25(d))
```

Default weights are equal, and `rrf_k` is set to 60. This choice avoids score calibration between BM25 and vector similarity and tends to preserve items that rank well in either modality.

7.4 Candidate filtering and reranking

Before reranking, the retriever filters reference-like chunks using heuristics (section name, DOI density, bracket citation density, URL density). This reduces the chance that retrieval returns bibliographies that match keywords but provide little explanatory evidence.

The enhanced configuration supports LLM reranking with Gemini. It reranks a capped candidate set (up to 40) using short previews and returns an ordered list. The final evidence set is truncated to `top_k_after_rerank` (default 5) to bound generation prompt size and align evaluation context with what the generator actually used.

7.5 Metadata filtering

The portal supports optional year and source type filters. Retrieval applies these filters early by restricting candidate chunks to manifest-backed allowed `source_id`s, so citations remain resolvable even when filters are active.

8. Generation, grounding, and citation enforcement

8.1 Evidence packing and budgets

Evidence is packaged as `(source_id, chunk_id)` followed by a chunk snippet. Generation enforces three budgets:

- maximum evidence chunks
- maximum characters per chunk
- maximum total evidence characters

These budgets prevent prompt blowups and improve responsiveness in the UI.

8.2 Strict citation policy and validation

The generator is instructed to end each claim sentence with citations and to use only citations from the retrieved set. After generation, the system validates:

- citation format

- membership in the allowed set
- per-sentence trailing citations

When violations are found, a repair rewrite pass runs, asking the model to rewrite using only the allowed citations.

When retrieval returns no usable evidence (including when metadata filters eliminate all candidates), the system returns a conservative "no evidence found" response and suggests next retrieval steps.

9. Research threads, artifacts, and export

9.1 Thread persistence (research history)

Threads are stored under `outputs/threads/` as JSON and indexed in `outputs/threads/threads_index.json`. Each thread captures:

- query, timestamp, answer
- full retrieved chunk objects (text, section, offsets, scores)
- citations used, retrieval configuration, prompt version, and run id

Example saved thread:

- `outputs/threads/d52996c0-4abe-463d-a5b2-f5844fd20ed8.json`

9.2 Artifact generator

Artifacts are generated in `src/app/artifact_generator.py` from a thread's answer and retrieved evidence, with schemas aligned to the course specification.

Implemented artifact types:

- **Evidence table**: Claim | Evidence snippet | Citation | Confidence | Notes
- **Annotated bibliography**: 8 to 12 sources with claim, method, limitations, why it matters
- **Synthesis memo**: target 800 to 1200 words with inline citations and a manifest-backed reference list
- **Disagreement map**: a structured view of tensions or competing claims surfaced from retrieved evidence (stretch)

Evidence table confidence is derived from normalized retrieval scores so it remains deterministic and auditable.

9.3 Export formats and example outputs

Exports are handled by `src/app/export_manager.py` and written to `outputs/exports/`.

Formats:

- Markdown: memo and thread exports
- CSV: evidence table and bibliography
- PDF: memo and evidence table (ReportLab rendering with wrapping)
- BibTeX: bibliography export (stretch goal)

Concrete example artifacts generated from the portal:

- Evidence table: `outputs/exports/artifacts_20260219_213627_evidence_table.csv` and `.pdf`
- Synthesis memo: `outputs/exports/artifacts_20260219_213627_synthesis_memo.md` and `.pdf`
- Bibliography: `outputs/exports/artifacts_20260219_213627_bibliography.csv` and `.bib`
- Thread export: `outputs/exports/thread_d52996c0-4abe-463d-a5b2-f5844fd20ed8_20260219_214135.md`

A representative evidence-table row (from the CSV) shows the intended traceability:

- Claim: "A significant function of an urban digital twin is the 3-Dimensional visualisation of the city or region."
- Citation: `(Apeldoorn2025, chunk_97)`
- Evidence snippet is taken verbatim from the cited chunk and truncated for display.

10. Evaluation on the full 22-query set

Evaluation is implemented in `src/eval/evaluator.py` and runs the fixed query set at `src/eval/query_set.json`. The Streamlit **Evaluation** page can run baseline, enhanced, or a baseline versus enhanced comparison.

For this submission, the summary numbers below come from:

- `outputs/eval/eval_report_compare_20260219_225105.md`
- raw logs: `outputs/eval/eval_runs_baseline_20260219_225105.jsonl` and `outputs/eval/eval_runs_enhanced_20260219_230030.jsonl`

Query set composition for this run:

- query types: direct: 11; edge_case: 5; synthesis: 6
- difficulty labels: easy: 2; medium: 11; hard: 9

System configuration snapshot for the enhanced run:

- generator model: gemini-2.5-flash
- retrieval config: k=10, k_raw=60, hybrid=True, reranking=True, top_k_after_rerank=5, vector_weight=0.5, bm25_weight=0.5

10.1 Metrics

Groundedness is judged against retrieved evidence on a 1 to 4 scale. Citation precision checks whether citations match retrieved chunks. Answer relevance is scored on a 1 to 4 scale based on whether the answer addresses the question.

10.2 Summary results (22 queries)

Metric	Baseline	Enhanced	Delta
Groundedness (avg /4)	2.91	3.91	1.00
Citation precision (avg)	99.43%	99.65%	0.22%
Answer relevance (avg /4)	3.41	3.55	0.14

Interpretation:

- Groundedness improves sharply in the enhanced configuration. Twenty of twenty-two queries score 4 in groundedness in the enhanced run.
- Citation precision remains near 100 percent in both configurations due to strict allowed-citation constraints and post-generation validation.
- Relevance improves slightly overall. A small group of questions remain relevance-limited because the corpus does not contain highly specific evidence for the requested concept, so the system responds with the closest available evidence plus a limitation note.

10.3 Impact by query type

Query type	Count	Groundedness baseline	Groundedness enhanced	Delta	Citation precision baseline	Citation precision enhanced	Delta

Query type	Count	Groundedness baseline	Groundedness enhanced	Delta	Citation precision baseline	Citation precision enhanced	Delta
direct	11	2.91	3.91	1.00	100.00%	100.00%	0.00%
synthesis	6	3.17	4.00	0.83	97.92%	98.72%	0.80%
edge_case	5	2.60	3.80	1.20	100.00%	100.00%	0.00%

10.4 Representative cases and what they reveal

Case 1: Q09 (data pipeline architectures)

- Enhanced groundedness is 3 and relevance is 2. The answer correctly states that the corpus does not explicitly describe common pipeline architectures, then summarizes adjacent evidence about data streams and system components. The groundedness shortfall comes from one sentence where "sensor data" is included but cited to a chunk that only mentions analytics and machine learning.

Design implication:

- Add a claim-to-evidence alignment check per sentence, and prefer citing chunks whose text explicitly contains the key entity in the claim.
- For architecture-pattern questions, add query decomposition that first retrieves "pipeline architecture" passages, then falls back to "data integration, data streams, IoT ingestion" passages with an explicit gap statement.

Case 2: Q18 (standardized APIs)

- Enhanced groundedness is 3 with a judge note about slight imprecision: the evidence attributes urban digital twin modeling to an information model, while the answer attributes it directly to the API.

Design implication:

- Add an "attribution precision" constraint in the generator prompt for standards and APIs. Require the model to distinguish API, information model, and standard specification roles.

Case 3: Q11 (citation precision < 100 percent)

- Enhanced citation precision is reduced by one invalid citation parse: `[('Building', 'Urban domains')]`. This string is not a model-invented source. It is a parser artifact caused by parentheses that contain a comma but are not citations.

Design implication:

- Tighten citation parsing regex to require `chunk_\d+` for `chunk_id`, and constrain `source_id` to known manifest ids. This eliminates false positives like `("Building", "Urban domains")` and `("e.g.", "specific methodologies")`.

Case 4: Coverage-limited relevance (Q04, Q20, Q21)

- These queries score groundedness 4 but relevance 2. The answers are faithful and cite correctly, yet the corpus evidence is broad or discusses limitations rather than giving concrete practices or techniques (for example, social media ingestion, specific visualization techniques, or specific human-centered interface practices).

Design implication:

- Improve missing-evidence messaging when the retrieved evidence is adjacent but not directly answering the asked scope. Add a structured gap statement and suggested targeted retrieval actions.

10.5 A note on evaluation context trimming

In the baseline run, the generator retrieves 10 chunks and may cite any of them. The evaluation judge uses a smaller evidence budget, so it can miss cited chunks and mislabel them as fabricated when they are present in the run log.

A concrete example is Q03:

- baseline evidence sources: Alkhateeb2023(4), Abdelrahman2025(2), Barbie2023(1), Lei2023(1), WEF2022(1), White2021(1)
- enhanced evidence sources: Barbie2023(2), White2021(2), Alkhateeb2023(1)

Because the enhanced configuration truncates to 5 chunks after reranking, the judge sees nearly the same evidence set that the generator used, which reduces false hallucination flags.

Recommended change:

- Always include cited chunks in the evaluation context passed to the judge, then fill remaining budget with top-ranked non-cited chunks.

11. Limitations

11.1 Retrieval and indexing limitations

- Character-based chunking yields variable token counts across chunks, which can impact embedding focus and prompt packing efficiency.
- Section detection is heuristic. Papers with nonstandard formatting can fall back to simple chunking, reducing topical coherence.
- `IndexFlatL2` scales linearly with corpus size. It is stable for the current corpus but will slow down as sources grow.
- BM25 tokenization via whitespace splitting misses punctuation and morphological variants.

11.2 Grounding and citation limitations

- Citation enforcement verifies format and membership in retrieved chunks, yet it does not fully guarantee claim-level semantic support.
- Citation parsing currently treats any parenthetical phrase with a comma as a candidate citation, creating rare false positives (Q11, Q12).

11.3 Evaluation limitations

- LLM judging introduces noise and potential bias.
- Evidence trimming in the judge can create false negatives when the generator cites chunks outside the trimmed set.

12. Next steps

12.1 Retrieval improvements

- Replace character-based chunking with token-aware chunking that respects sentence boundaries and keeps stable token budgets.
- Improve BM25 tokenization with punctuation normalization, optional stemming, and domain token handling for acronyms.
- Normalize embeddings and use cosine similarity or inner product with a normalized FAISS index.
- Add query rewriting or decomposition for architecture-pattern and synthesis queries, then merge evidence across sub-queries.

12.2 Stronger grounding checks

- Implement per-sentence claim-to-cited-evidence similarity checks. Flag sentences where cited text does not contain the key entity or contradicts the claim.
- Add evidence strength signals, including redundancy across sources and section-based weighting (Methods and Results preferred for technical claims).

12.3 Fix citation parsing and evaluation context

- Tighten citation parsing to match only `(source_id, chunk_\d+)` where `source_id` is in the manifest.
- In evaluation, always inject cited chunks into the judge context, then add additional top chunks up to the budget.

12.4 Product workflow enhancements

- Add click-to-highlight evidence spans using chunk offsets.
- Add a reading list and tagging system tied to manifest metadata.
- Add artifact templates that can regenerate outputs with different prompts while keeping evidence fixed.