Student AI Hub — Workflow Appendix

Student AI Hub — Workflow Appendix

This appendix documents the technical workflow and tools used to build the initial foundational reference layer of the Student AI Hub. This describes the process used for the corpus-grounded reference sections, not all hub content. Future content may follow different workflows or be entirely human-authored.

What This Appendix Covers (and Doesn't)

- Documents the technical pipeline used to build the initial five locked reference sections (AI Basics, Using AI for School and Work, How Businesses Are Using AI, AI Tools You Might Use, Rules, Risks, and Ethics of AI).
- Explains the tools, artifacts, and decisions made during the initial build phase.
- Describes how sources were ingested, chunked, retrieved, and synthesized.
- Does not mandate that all future hub content must follow this workflow.
- Does not cover the human-authored sections (AI News That Matters, AI Resources at Penn State, AI by Smeal Major).

Inputs and Canonical Decisions

CSV Source Registry

The workflow began with a Google Sheet exported as CSV (`data/SAIH Content - Corpus v0.csv`). Each row represents one human-approved source with the following fields:

- **section**: One of eight predefined hub sections (e.g., "AI Basics", "Using AI for School and Work").
- **url**: The canonical URL to fetch.
- **source_type**: Classification such as "University / Official", "Explainer / Guide", "Research / Academic", etc.
- **relevance_note**: A brief human-written note explaining why the source belongs in its assigned section.
- **date_added**: The date the source was added to the registry (used for tracking, not publication date).

Locked Hub Sections

The following sections were predefined and locked before ingestion: - AI Basics - Using AI for School and Work - How Businesses Are Using AI - AI Tools You Might Use - Rules, Risks, and Ethics of AI - AI News That Matters (human-authored, not generated) - AI Resources at Penn State (human-authored, not generated) - AI by Smeal Major (human-authored, not generated)

Locked Source Types

The following source type classifications were predefined: - University / Official - Course / Training - Tool Documentation - Explainer / Guide - Case Study / Example - Research / Academic - News / Update

Ingestion & Extraction Tools

Core Tools Used

- **requests**: Python library for fetching HTTP content. Used with a custom User-Agent string and timeout handling.
- **urllib.robotparser**: Standard library module for parsing and enforcing `robots.txt` rules. Each domain's robots.txt was fetched and checked before attempting to crawl.
- **trafilatura**: Primary HTML text extraction library. Designed for robust content extraction from web pages, handling common HTML structures and ignoring navigation/ads.
- **beautifulsoup4**: Fallback HTML parser used when trafilatura failed or for extracting structured elements like headings (h1, h2).
- **pypdf**: Primary PDF text extraction library. Extracts text from PDF pages sequentially.
- **pdfplumber**: Optional fallback PDF extractor used when pypdf failed. Provides alternative parsing strategies for complex PDF layouts.

Behaviors Implemented

- **Rate limiting**: 1.25 seconds between requests to avoid overwhelming servers.
- **Failure logging**: All fetch attempts were logged with status codes, error messages, and blocked reasons. Failed or blocked sources were recorded but not retried.
- **No paywall bypassing**: If a page returned HTTP 200 but appeared to be behind a paywall (detected via keywords in response), it was marked as blocked.
- **Content type detection**: HTML vs. PDF was determined from Content-Type headers, with appropriate extraction methods applied.

Artifacts Produced

Snapshot Files (`data/corpus_snapshots/`)

One JSON file per source URL, named with a hash-based ID and timestamp. Each snapshot contains: - Original and final URLs (after redirects) - Retrieval timestamp and HTTP status - Extracted title, headings, and full text (when available) - Content hash (SHA256) for deduplication - Scrape status (success/partial/blocked/error) and blocked reason - Full CSV row metadata preserved in a `notes` field

Stable Snapshots (`data/corpus_snapshots_stable/`)

Renamed copies of snapshots using stable IDs based on URL hostname and hash. Enables consistent referencing across ingestion runs.

Run Logs (`data/runs/*.jsonl`)

JSONL files (one line per source) recording each ingestion run. Includes snapshot IDs, URLs, status codes, content hashes, and blocked reasons. Used for auditing and tracking what was attempted vs. what succeeded.

Corpus Index (`data/corpus_index_stable.json`)

Aggregated index of all sources with metadata, scrape status, word counts, and stable IDs. Sorted by section and title. Includes summary counts (success/blocked/error/partial).

Chunk Files (`data/chunks.jsonl`)

JSONL file with one chunk per line. Each chunk includes: - `chunk_id`: Stable identifier (`{stable_id}::c{index}`) - Source metadata (URL, title, section, source_type) - Character offsets (`char_start`, `char_end`) within the original full text - The chunk text itself (typically 1,200–1,800 characters)

Chunk Index (`data/chunk_index.json`)

Lightweight index summarizing chunking results: total chunks, average length, min/max lengths, and a list of chunk IDs with basic metadata for quick lookup.

Retrieval and Citation Approach

Chunk-Based Retrieval

Content was retrieved at the chunk level, not the document level. This allows precise citation and prevents over-generalization from entire articles.

Lexical Ranking (BM25)

Chunks were ranked using BM25-style lexical matching (implemented in `search_chunks_v2.py`). The core ranking algorithm is BM25, with custom heuristics layered on top:

Core BM25 Algorithm: - Preprocessing: lowercase conversion, tokenization on whitespace, punctuation stripping (except quotes preserved for phrase detection) - Document frequency calculation: counts how many chunks contain each unique token across the entire corpus - Inverse document frequency (IDF): computed using the standard BM25 IDF formula: `log((N - df + 0.5) / (df + 0.5) + 1)` where N is total chunks and df is document frequency - BM25 scoring: standard formula with parameters k1=1.5, b=0.75, applied to each query token's term frequency within a chunk

Custom Heuristics (Applied After BM25): - **Quoted phrase boost**: If the query contains quoted phrases (e.g., `"risk management framework"`), an additional +20 is added to the BM25 score for each chunk where the exact phrase appears in the chunk text (case-insensitive). - **Title token boost**: For tokens that appear in both the query and the chunk's title field, an additional BM25 component is computed and added to the base score. This effectively doubles the contribution of matching title tokens by adding their BM25 score once more.

Optional Query Expansion (Acronym/Alias Expansion): The system includes optional, limited acronym expansion rules: - If the query contains "NIST", the tokens "ai" and "rmf" are added to the query token set. - If the query contains "rmf", the tokens "risk", "management", and "framework" are added to the query token set.

These expansions are hardcoded aliasing rules, not semantic understanding. They are applied before BM25 scoring, so expanded tokens are scored using the same BM25 formula as original query tokens. This expansion is optional and limited to these specific acronym/alias mappings.

Citations by Chunk ID

Each paragraph in generated sections ends with citations in the format `(Source: chunk_id)`. This enables traceability back to the exact source text and retrieval moment.

"Insufficient Evidence" Behavior

If a requested section or claim could not be supported by available chunks, the system explicitly stated that evidence was insufficient rather than inferring or fabricating content.

Synthesis Workflow Used in Cursor

The content generation followed this iterative process:

1. **Generate**: AI-assisted draft using only chunks from the target section, following structure and tone guidelines.
2. **Audit source pressure**: Check for paragraphs relying heavily on single sources, especially for cautionary or corrective claims.
3. **Revise minimally**: Narrow language scope (e.g., "some researchers," "institutional guidance suggests") rather than expanding claims or adding new sources.
4. **Lock**: Final version marked as locked after human approval.

Main Audit Checks

- **Single-source cautionary claims**: No paragraph making a corrective, critical, or cautionary claim should rely on a single source. If multiple sources aren't available, language must be explicitly scoped.
- **Prescriptive tone**: Avoid "must," "should," "required" unless explicitly framed as institutional guidance or documented policy.
- **Hype/endorsement risk**: Avoid superlatives, inevitability framing, or implied tool recommendations. Use descriptive language instead.
- **Source balance**: Ensure no section feels over-dependent on one institution or source type.
- **Citation completeness**: Every paragraph must end with at least one chunk_id citation.

Known Limitations and Intentional Exclusions

Blocked Sources

Sources blocked by `robots.txt`, paywalls, HTTP 403/404, or other access restrictions were recorded with their blocked reason but not retried. No content was inferred for blocked sources.

Intentionally Human-Authored Sections

Three sections were explicitly excluded from automated generation: - **AI News That Matters**: Requires editorial judgment and timeliness that automated systems cannot provide. - **AI Resources at Penn State**: Requires institutional ownership and local context. - **AI by Smeal Major**: Requires curricular nuance and specialized academic guidance.

These boundaries are documented in section README files and respected by the synthesis workflow.

How This Can Be Rerun or Updated

The workflow is designed to be repeatable and updatable:

1. **Add source to registry**: Add a new row to the Google Sheet (or CSV) with section, URL, source_type, and relevance_note.
2. **Ingest**: Run `ingest_corpus.py` to fetch new URLs and create snapshots. Existing snapshots are preserved unless URLs change.
3. **Re-index**: Run `build_corpus_index.py` to regenerate the corpus index with new sources.
4. **Re-chunk**: Run `chunk_corpus.py` to regenerate chunks from updated snapshots.
5. **Re-synthesize affected section**: Regenerate the markdown for any section that includes new sources, using the updated chunk set.
6. **Re-audit**: Apply the same audit checks to revised sections before locking.

The stable ID system ensures that unchanged sources maintain consistent chunk IDs across runs, minimizing disruption to existing citations.