Winnow: Software requirements specification

Ayan Das

March 9, 2025

Contents

1 Introduction

1.1 Purpose of this Document

The purpose of this Software Requirements Specification (SRS) is to provide a comprehensive description of the requirements, scope, constraints, and functionality for a *Python-based project* that aggregates large collections of bibliographic records from various sources (journals, conferences) and systematically filters them based on content relevance. This document follows a structure inspired by IEEE 29148 (which superseded IEEE 830) standards for software requirements.

The system's main goal is:

- To parse large volumes of BibTeX data scattered across multiple directories.
- To augment these records with missing abstracts from a variety of external services.
- To *reliably and thoroughly filter* references that lie at the intersection of *Physics* (especially *statistical physics*) and *Machine Learning (ML)*.
- To produce a *single curated BibTeX file* that can be used by researchers with minimal misses (i.e., minimal false negatives).
- To do all of the above in a *production-grade* codebase with robust logging, error handling, progress tracking, and a well-defined plugin-like interface for custom scrapers.

This document serves multiple audiences, including the project's developers, testers, prospective maintainers, external integrators, and stakeholders (e.g., research labs, libraries) who need to understand the system's capabilities and constraints.

1.2 Scope

The scope of this project includes:

a) Parsing BibTeX entries stored in a large directory structure, where each directory corresponds to a venue (journal or conference) and each file corresponds to a specific year.

b) Validating and normalizing the input data, applying a set of invariants to ensure consistent structure.

c) Augmenting missing abstracts using multiple external APIs/services (e.g., Semantic Scholar, OpenAlex, arXiv, Google Scholar, etc.), with fallback priority ordering.

d) Applying a *broad* textual filtering strategy based on two conceptual lexicons: a physics lexicon and a machine-learning lexicon.

e) Optionally applying advanced text classification to further refine the set of relevant references.

f) Providing a *unified interface* (i.e., plugin-like extension points) for specialized scrapers that can be integrated without major modifications to the rest of the pipeline.

g) Managing extensive logging, error-handling, progress tracking, and usage analytics in a production environment.

The system is primarily intended for *researchers working at the intersection of physics and machine learning*, but the overall design and code structure will be general enough that it can be repurposed to other domains (e.g., *biology and ML*, *finance and ML*, or any domain intersection). We will implement *configurable lexicons* and *pluggable scrapers* so that the pipeline can be adapted with minimal code changes.

1.3 Definitions, Acronyms, and Abbreviations

BibTeX
A reference management tool and file format used for describing and listing bibliographic items.

ML Machine Learning.

Abstract Augmentation
The process of retrieving missing abstracts for references from external services.

Lexicon
A set of domain-specific keywords or key phrases used for textual filtering.

API
Application Programming Interface.

Pipeline
A connected series of processing steps or modules forming a workflow.

Filtering Intersection
The method of ensuring that references simultaneously mention relevant keywords from *both* physics and ML, or that are determined relevant via advanced text classification.

SRS
Software Requirements Specification.

SAS
Software Architecture Specification.

SDS
Software Design Specification.

IEEE 29148
International standard providing guidance for SRS documents.

1.4 References

- IEEE 29148 (superseding IEEE 830): *Systems and software engineering – Life cycle processes – Requirements engineering.*
- ISO/IEC/IEEE 42010: *Systems and software engineering – Architecture description.*
- Additional references may include official documentation for:
  - **Semantic Scholar API** - `https://api.semanticscholar.org/`
  - **OpenAlex API** - `https://docs.openalex.org/`
  - **arXiv API** - `https://arxiv.org/help/api/index`
  - **Google Scholar** (no official public API, often requires HTML parsing or third-party libraries).

2 Overall Description

2.1 Product/Project Background

Many researchers need a *comprehensive* yet *focused* reference dataset for cross-disciplinary work. In fields like *statistical physics* and *machine learning*, references are *voluminous* and *highly fragmented* across different journals and conferences. Most existing solutions either:

1. Provide general reference management but lack advanced domain-based filtering or abstract retrieval.

2. Are domain-specific solutions that do not scale or adapt well to new domains.

Therefore, this project addresses the *specialized need* of reliably obtaining *all relevant references in a chosen domain intersection* while ensuring minimal false negatives and providing a flexible, maintainable codebase.

2.2 User and Stakeholder Descriptions

- **Primary Users (End-User Researchers)**: Researchers and graduate students working at the *intersection of physics and ML*. They expect to:
  - Provide a large corpus of '.bib' files.
  - Configure the pipeline (e.g., specify their domain intersection, external services).
  - Receive a curated '.bib' file of relevant references with *augmented abstracts*.
- **Domain Librarians or Reference Managers**: Librarians at research institutions who manage domain-specific bibliographies and want to unify them. They might also create specialized scrapers for *high-impact journals*.
- **System Administrators / DevOps**: Responsible for deploying and maintaining the system in a *production environment*. They care about logging, error handling, resource utilization, and system integration.
- **Software Developers / Plugin Implementers**: Developers who extend the pipeline with *additional scraper plugins* for specialized or newly emerging journals. They require a robust *API interface* or plugin template to do so *without rewriting the entire system*.

2.3 System Context

The system is a *command-line or library-based tool* that can be run on a local workstation or in a server environment. It may be integrated with:

- **File System**: The directories containing '.bib' files.
- **Internet / External APIs**: For abstract retrieval.

- **Configuration Files or CLI Arguments**: Where the user can specify domain lexicons, concurrency level, logging details, etc.

Users might also run it in an automated pipeline with **continuous integration** or **scheduled tasks** to keep their curated references up to date.

2.4 Constraints and Assumptions (High-Level)

- **Large Data Volume**: The system must handle tens or hundreds of thousands of references efficiently.
- **Incomplete Data**: Many references *lack* abstracts, titles may have LaTeX formatting, and years/venues might be mis-specified. The system must robustly handle or log anomalies.
- **External API Reliability**: Abstract retrieval services might be slow, rate-limited, or fail intermittently. Our system must *gracefully degrade* or *re-try* as appropriate, logging errors clearly.
- **Python 3.x Environment**: We assume the tool is written in Python 3.x.
- **Minimal Misses**: The system should emphasize *inclusive* first-pass filtering, then refine with text classification to reduce false positives.

2.5 General Dependencies

- **BibTeX Parsing Libraries** (e.g., `bibtexparser` or `pybtex`) for reading/writing '.bib' files.
- **HTTP Libraries** (e.g., `requests`) for external API calls.
- **Database or Key-Value Store** (optional) for caching results from external queries.
- **Logging Frameworks** (e.g., Python `logging` + structured logs).
- **Natural Language Processing Libraries** (e.g., `nltk`, `spaCy`, or `scikit-learn`) if advanced text classification is used.

3 Functional Requirements

This section outlines the *functional requirements* that the system must fulfill, typically described as user stories or use cases.

3.1 Use Cases / User Stories

3.1.1 Use Case 1: Parsing and Validating BibTeX

- **Actors:** Researcher, Librarian
- **Description:** The user provides a directory structure containing multiple '.bib' files. The system scans the directories, *reads and parses each file*, and creates an internal representation (e.g., Python objects) of each reference.
- **Goal:** Correctly load and validate each reference, capturing fields like `title`, `author`, `year`, `venue`, `abstract` (if present), and the *raw* BibTeX entry for output later.
- **Preconditions:**
  - The user has the '.bib' files in a structured directory layout.
  - The system is configured with the correct path or base directory.
- **Postconditions:**
  - Each reference is loaded into an internal data structure.
  - Validation logs are generated for missing fields or structural anomalies.
- **Exceptions:**
  - If a '.bib' file is corrupt or unreadable, the system logs an error and continues with the rest.
  - If critical fields are missing (e.g., `title`), the reference is flagged or partially omitted.

3.1.2 Use Case 2: Abstract Augmentation

- **Actors:** Researcher, Librarian, Third-Party Services
- **Description:** The user wants the system to *enrich references that have missing abstracts*. The system calls one or more external APIs (Semantic Scholar, OpenAlex, arXiv, Google Scholar, etc.) in a prioritized sequence. If one fails or returns incomplete data, it tries the next.
- **Goal:** Maximize the number of references that end up with an *abstract* field.
- **Preconditions:**

  - The system has network connectivity and valid credentials (if needed) for the external APIs.
  - The references have `title` or some other identifying metadata that can be used to query external services.

- **Postconditions:**
  - A large portion of references without an abstract are augmented with newly retrieved abstracts.
  - A fallback log is created detailing how many references were successfully enriched, from which service, how many failed, etc.

- **Exceptions:**
  - Service rate limits or unavailability: The system logs an error, retries after a backoff, or moves on to the next service.
  - Invalid or partial matches: The system might retrieve the *wrong* abstract. Therefore, there must be logic to verify that the matched reference is likely correct (e.g., compare title similarity).

3.1.3 Use Case 3: Keyword-Based Filtering

- **Actors:** Researcher, Librarian
- **Description:** The system uses two broad *lexicons* (physics and ML) to filter references in a first pass. It checks whether each reference's title/abstract *intersect* the lexicons (i.e., it must contain at least one physics term and at least one ML term).
- **Goal:** Quickly eliminate references that are obviously not relevant to the intersection domain.
- **Preconditions:**
  - A curated or user-configurable physics lexicon is available (with synonyms, partial matches, wildcard expansions).
  - A curated or user-configurable ML lexicon is available.
- **Postconditions:**
  - Each reference is flagged as *candidate* or *non-candidate*.
  - Non-candidate references are excluded from further advanced processing (except for auditing or sampling).
- **Exceptions:**
  - If an abstract is unavailable, only the title is used, which may lead to false negatives. The system logs the limitation.

3.1.4 Use Case 4: Advanced Text Classification (Optional)

- **Actors:** Developer, Researcher
- **Description:** A module that can train or load a text classifier (e.g., scikit-learn, spaCy, or a large language model) to refine the candidate set from the first pass, removing false positives.
- **Goal:** Further filter references to ensure high precision *without* sacrificing recall (minimizing misses).
- **Preconditions:**
  - A labeled dataset or a small set of *true positive* and *true negative* examples is available to train or calibrate the classifier.
  - The user has selected or configured a classification approach in the system's configuration.
- **Postconditions:**
  - The system outputs a final *refined* list of references, with fewer irrelevant ones.
- **Exceptions:**
  - Missing or low-quality training data leading to poor classification performance. Must log model performance metrics.

3.1.5 Use Case 5: Custom Scraper Plugins

- **Actors:** Plugin Developer, System Administrator
- **Description:** A developer wants to implement a *specialized abstract scraper* for a certain high-impact journal or a subscription-only database. The system provides an interface or base class that the developer can implement.
- **Goal:** Streamline integration of new scrapers with minimal overhead, ensuring data transformation and error logging are consistent with the pipeline.
- **Preconditions:**
  - The developer has relevant credentials or access tokens for the target journal API or webpage.
- **Postconditions:**
  - The new scraper seamlessly fits into the augmentation stage for references that match the relevant domain.
- **Exceptions:**
  - The specialized journal's API is undocumented or changes. This should be logged and flagged during integration tests.

3.1.6 Use Case 6: Final Merging and Export

- **Actors:** Researcher, Librarian
- **Description:** The pipeline merges *all surviving references* (i.e., those not excluded) back into a single '.bib' file or a set of '.bib' files.
- **Goal:** Present a unified, curated bibliography that has minimal misses and includes any newly added or corrected abstracts.
- **Preconditions:**
  - All earlier stages (parsing, augmentation, filtering) have completed.
- **Postconditions:**
  - A final file (or set of files) in valid BibTeX format is written to the user-specified output path.
  - The user can directly import or cite from this curated file in their LaTeX or reference manager.
- **Exceptions:**
  - Permission issues or disk space issues during the write operation. The system logs these errors and may fallback to partial export.

3.2 Detailed Functional Requirements

3.2.1 FR-1: Directory-Based Discovery

- **Description:** The system must recursively search through a user-specified root directory to find ‘.bib‘ files.
- **Rationale:** Large reference collections are structured by {venue}/{year}.
- **Criteria:** Must handle nested directories. Must skip non-BibTeX files or directories lacking ‘.bib‘.

3.2.2 FR-2: Configurable Invariant Checks

- **Description:** The system shall enforce certain *invariants* (e.g., every entry must have a `title` field, certain fields cannot be empty, etc.).
- **Rationale:** Ensuring partial data is recognized early.
- **Criteria:** The user can enable or disable specific invariants in a config file. Violations generate structured warnings or errors.

3.2.3 FR-3: Abstract Retrieval with Multiple Services

- **Description:** The system must attempt to retrieve abstracts from multiple sources in a priority order (Semantic Scholar, OpenAlex, arXiv, Google Scholar, etc.).
- **Rationale:** Minimizing the chance of an empty abstract.
- **Criteria:** Each external service is tried in turn; if successful, the system stops calling further services for that reference. Must log how many calls succeeded or failed per service.

3.2.4 FR-4: Textual Filtering (Lexicon Intersection)

- **Description:** The system must parse the *combined text* of `title + abstract` (when available) for each reference, scanning for presence of at least one *physics term* and at least one *ML term*.
- **Rationale:** Quick broad pass to reduce the dataset size.
- **Criteria:**
  - Must support case-insensitive matching.
  - Must support wildcards or partial matches for morphological variants.
  - Must allow a user to *edit* or *extend* the lexicon files easily.

3.2.5 FR-5: Advanced Classification (Optional Module)

- **Description:** The system can optionally load or train a classification model to refine the candidate set from FR-4.
- **Rationale:** Reduce false positives, preserve high recall.
- **Criteria:** The classification module should run *after* the broad intersection filter. It must produce a confidence score for each reference. The user can specify a threshold for acceptance or rejection.

3.2.6 FR-6: Custom Scraper Interface

- **Description:** Provide an interface or base class that can be extended by new scrapers.
- **Rationale:** Users or devs might want to add specialized scrapers (e.g., for *Nature Physics* or *Physical Review E* where a custom approach might exist).
- **Criteria:**
  - The interface must define standard methods like `get_abstract(title, authors, doi)` or a similar signature.
  - Return types, logging approach, and error handling must be consistent with the rest of the pipeline.

3.2.7 FR-7: Logging, Error Handling, and Reporting

- **Description:** The system must produce *structured logs* containing:
  - Timestamps
  - Event severity (INFO, WARNING, ERROR, CRITICAL)
  - Stage of the pipeline
  - Possibly the ID of the reference in question
- **Rationale:** For debugging and auditing.
- **Criteria:** Must gracefully handle partial failures (e.g., a single reference fails augmentation) without stopping the entire pipeline.

3.2.8 FR-8: Final BibTeX Generation

- **Description:** The system must produce a single or multiple '.bib' file(s) containing the filtered references with any newly added or updated abstract fields.
- **Rationale:** To allow the user immediate usage in their reference manager or LaTeX.
- **Criteria:**
  - The resulting '.bib' must preserve standard BibTeX formatting.

- The user can specify output location or a naming convention (e.g. `ml_physics_all.bib`).
- The pipeline should also preserve the original BibTeX keys or generate new ones only when necessary.

4 Non-Functional Requirements

4.1 Performance Requirements

- **NF-PERF-1: Scalability**
  The system must handle thousands to potentially hundreds of thousands of references in a *single run* without timing out or consuming excessive memory.

- **NF-PERF-2: Concurrency**
  Abstract augmentation calls to external APIs can be *parallelized* or *asynchronous* to speed up the overall runtime.

- **NF-PERF-3: Efficiency**
  The text filtering operation (searching for lexicon terms) should be efficient, ideally using compiled regex or other optimized string search algorithms.

4.2 Security Requirements

- **NF-SEC-1: API Credentials**
  If API keys or credentials are required for external services, they must not be exposed in logs or version control. They must be stored securely (e.g., environment variables or config files with appropriate permissions).

- **NF-SEC-2: Sanitization**
  When dealing with user-provided data (like manually added references or special plugin configurations), the system must sanitize inputs to avoid injection vulnerabilities (particularly if user data is appended to a shell command or similar).

4.3 Usability Requirements

- **NF-USAB-1: CLI / Configuration Simplicity**
  The system should expose an easy command-line interface with descriptive help text. Command-line arguments or a single config file can be used to control the pipeline steps.

- **NF-USAB-2: Documentation**
  Clear documentation (online or embedded docstrings) describing each configuration option, how to extend the system, etc.

4.4 Maintainability & Reliability Requirements

- **NF-MR-1: Modular Code Structure**
  Each stage (parsing, augmentation, filtering, classification, exporting) should be in a separate Python module or set of modules for clarity and maintainability.

- **NF-MR-2: Automated Testing**
  Comprehensive unit and integration tests, ensuring that updates or new plugins do not break existing functionality.

- **NF-MR-3: Graceful Degradation**
  If one external service is unavailable or fails, the system should log and move to the next service (for abstract augmentation), rather than terminate the pipeline.

4.5 Reliability Requirements

- **NF-REL-1: Recovery from Partial Failures**
  If an external API fails mid-run, the system should still complete partial augmentation for other references. A dedicated *re-run* mode might be provided to retry failed references.

4.6 Extensibility Requirements

- **NF-EXT-1: Domain Lexicon Customization**
  Users can easily replace the physics or ML lexicons with their own domain terms.

- **NF-EXT-2: Plugin Mechanism for Scrapers**
  The system includes guidelines and base classes for implementing new scrapers.

4.7 Internationalization

Not a major concern unless the user sets or provides references in multiple languages. The architecture should not inherently prohibit multi-language references, but it is not a primary focus here.

5 System Interface Requirements

5.1 User Interface (CLI or Library)

- The system will have a **Command-Line Interface (CLI)**. Example usage:

```
python aggregator.py --root_dir path/to/bib/files \
                     --config path/to/config.yaml \
                     --output curated.bib \
                     --enable-classifier
```

- Alternatively, a library approach can provide Python functions such as:

```
from aggregator import run_pipeline

run_pipeline(
  root_dir="path/to/bib/files",
  output_file="curated.bib",
  config="path/to/config.yaml",
  ...
)
```

5.2 External API Interfaces

- **Semantic Scholar API:**
  - Endpoint: `https://api.semanticscholar.org/v1/paper/`
  - Rate Limits: *TBD, often free tier is 100 requests per 5 minutes or similar.*
  - Required Field: *Title* or *DOI* or *arXiv ID*.
- **OpenAlex API:**
  - Endpoint: `https://api.openalex.org/works`
  - Format: JSON response.
  - Query fields: Title, authors, etc.
- **arXiv API:**
  - Endpoint: `http://export.arxiv.org/api/query`
  - Commonly uses search by *title* or *author* or *arXiv ID*.
- **Google Scholar:**
  - No official public API. Often requires HTML scraping with potential for *captcha* or blocking.
  - *We must approach with caution and user disclaimers.*

5.3 Internal Plugin Interface

- The system will expose a base Python class, e.g.:

```
class AbstractScraper:
    def __init__(self, config):
        ...
    def fetch_abstract(self, ref) -> Optional[str]:
        ...
```

- Developers can subclass and implement `fetch_abstract()` to integrate with a specialized service.
- The pipeline automatically calls each registered scraper in a chain or fallback order.

6 Data Requirements

6.1 BibTeX Input Format

- The input is primarily '.bib' files.
- The pipeline must handle *BibTeX entries of type* `@article`, `@inproceedings`, `@book`, etc.
- Minimal fields to read: `title`, `author`, `year`, `abstract (optional)`, `doi (optional)`, `url (optional)`, and the *complete raw entry text*.

6.2 Augmented Data / Metadata

- **Abstract Field:** If missing, must be appended or updated after augmentation.
- **Confidence Scores:** If classification is enabled, a *confidence or probability* might be stored in an auxiliary field.
- **Lexicon Terms Matched:** Optionally, the system can store which physics or ML terms triggered the candidate flag, for auditing.

6.3 Output Format

- A single unified '.bib' file that merges all references, rewriting or adding `abstract` fields, if found.
- The user can specify whether to overwrite or append to existing '.bib'.

6.4 Archival or Backup Requirements

- System logs and partial results (e.g., an intermediate CSV of references with newly found abstracts) may be stored for reference.
- A stable *cache* (e.g., a local database or JSON files) of retrieved abstracts might be used to avoid re-querying external APIs.

7 Constraints and Assumptions

7.1 Technical Constraints

- **Python Version**: The code is targeted for Python 3.8+.
- **External Dependencies**: For advanced text classification, the system might rely on `scikit-learn` or `spaCy`, which must be installed.
- **Command-Line Environment**: The typical usage is from a UNIX-like shell or from within an IDE that can run command-line scripts.

7.2 Organizational Constraints

- The system is developed and maintained by a small team; the code should be well-organized to facilitate easy onboarding.
- Licensing constraints for external libraries and APIs must be respected (e.g., usage limits, attribution requirements).

7.3 Assumptions

- The user has legitimate access to the external services for abstract retrieval.
- The user's local machine or server has enough disk space to store logs and any caches of abstracts.
- The references are primarily in English or contain English titles/abstracts. Non-English references might be partially processed, but text classification might degrade in accuracy.

8 Acceptance Criteria / Validation

8.1 Criteria

Each major functional requirement (FR-1 through FR-8) has a set of acceptance criteria that must be validated. Examples:

- **AC-1 (Parsing)**:
  - Provide a known set of '.bib' files with 500 references total. The system should parse them all and generate logs.
  - The system must not crash on any valid '.bib' file.
  - The number of references parsed must match the expected count.
- **AC-2 (Augmentation)**:
  - Provide 100 references missing abstracts, with known DOIs. The system must retrieve an abstract for at least 80% of them (assuming the external APIs have the data).
  - The system must produce an *enrichment report* showing success/failure for each external API call.
- **AC-3 (Filtering)**:
  - Provide a test dataset with a known distribution of physics+ML intersection papers. The system must identify at least 95% of them as *candidates* in the first pass.
  - The system must *exclude* references that do not match the intersection (or keep them *only* for logging/auditing).
- **AC-4 (Classification)**:
  - If classification is enabled, test with 50 *true positives* and 50 *true negatives*. The system must achieve a *precision* of at least 90% at a threshold that yields *recall* of at least 90%.
- **AC-5 (Final Output)**:
  - The generated final '.bib' file must be valid BibTeX (linted by a standard tool).
  - Newly added `abstract` fields must appear in the final file.
  - The user can confirm the presence of certain known references in the curated file.

8.2 Sign-off Process

1. **Requirement Traceability**: Each test scenario references the FR or NFR it addresses.

2. **Stakeholder Review**: The final acceptance testing is reviewed by at least one domain expert (e.g., a physics+ML researcher) who checks that the coverage is correct and that relevant references indeed appear in the final curated file.

3. **Final Approval**: Once the acceptance criteria are met or exceed the minimum thresholds, the system is considered ready for release.

9 Appendices

9.1 Glossary

Abstract Augmentation
: The process of adding or updating the `abstract` field in a BibTeX entry by querying external services.

Lexicon Intersection
: Filtering method requiring at least one physics keyword *and* one ML keyword in the reference's textual fields.

Plugin / Scraper
: A software component that implements a standard interface to retrieve data from a specialized source.

Reference
: A single bibliographic item in the system's internal representation or final '.bib' file.

9.2 Change History

Date	Version	Change
2025-03-09	0.1	Initial draft of the SRS.

9.3 References to Supporting Documentation

- A separate *Software Architecture Specification (SAS)* document that will detail the architecture-level decisions and diagrams.
- A *Software Design Specification (SDS)* that will describe each module, data structures, and class-level details.