1) Title

ParaCite: An Auto-Citation (In-text) Assistant for Academic Writing Using RAG, Vector Search, Reranking, and Metadata Indexing

---

2) Abstract

Academic writers frequently spend significant time searching for credible sources that support specific claims, then formatting and managing citations. We propose ParaCite, an auto citation assistant that suggests relevant papers from a curated knowledge base and returns properly formatted citations. ParaCite ingests a corpus of research papers using metadata and extracted text from PDFs, and benchmarks two retrieval methods for citation recommendation: keyword based retrieval using Best Matching 25 (BM25), which relies on weighted term occurrences, and Word2Vec based semantic matching scored with cosine similarity, with an additional hybrid method that combines both signals to improve top results. We also explore an agentic retrieval augmented generation approach that iteratively retrieves and refines candidate citations from the curated knowledge base. To keep retrieval fast and allow structured constraints, ParaCite also maintains a metadata index for fields like year, venue, and author, and supports hybrid retrieval that applies metadata filtering before vector based search. We will evaluate ParaCite using offline information retrieval metrics such as recall at K, mean reciprocal rank (MRR), and normalized discounted cumulative gain (nDCG) on an automatically constructed ground truth set from existing citation contexts, and we will run a small end to end study that rates the relevance, correctness, and usefulness of the suggested citations.

---

3) Introduction (including complexity)

Finding the right citation is more than keyword search because it is a semantic matching problem with real constraints like topical relevance, whether a source actually supports the claim, and basic quality signals such as recency and venue. Current workflows make writers repeatedly try queries, skim many results, and manually manage references, which is slow and error prone. ParaCite aims to automate this process by searching a curated collection of papers and recommending citations for a given claim or paragraph. The system will represent text using both keyword based retrieval, such as Best Matching 25 (BM25), and vector representations so it can compare a paragraph to paper titles and abstracts using similarity measures like cosine similarity. ParaCite will also use simple metadata filters, for example year, venue, or author, to narrow results and reduce obvious mismatches or duplicates, and then return the top suggestions with properly formatted citations. The work is challenging because it requires building a complete retrieval pipeline that can ingest papers and metadata, index them for fast

search, combine keyword and vector based scoring in a reliable ranking method, and produce results quickly. We will evaluate ParaCite by using existing citation contexts as queries and measuring retrieval quality with standard Information Retrieval metrics such as Recall at K, Mean Reciprocal Rank (MRR), and normalized Discounted Cumulative Gain (nDCG).

---

4) Summary of Proposed Work (half-page style, with a–d)

a) Software to implement

We will implement ParaCite, consisting of four core modules:

Paper ingestion pipeline:
Input: PDFs plus structured metadata such as title, authors, year, venue, DOI or URL, and keywords or abstract when available.
Processing: PDF text extraction, basic section segmentation, reference parsing when possible, and chunk creation using a simple strategy such as a sliding window or section based chunks.
Output: normalized paper records and text chunks with stable identifiers.

Indexing layer (hybrid search):
Vector index: store vector representations of chunks for semantic retrieval using similarity search.
Metadata index: store paper level fields to enable fast filtering by year, venue, author, and topic tags, and support lookups by DOI or title.
Optional baseline: keyword based retrieval such as Best Matching 25 (BM25) to support hybrid ranking.

Retrieval and ranking engine:
Given a user claim or paragraph, retrieve candidate chunks or papers using vector search and optional BM25, then rank candidates to produce top citation suggestions.
Return: top N papers with a supporting snippet, a confidence score, and a copy ready formatted citation in a standard style such as BibTeX, IEEE, or APA.

User interface:
A lightweight web interface or command line interface where a user pastes a claim or paragraph and receives suggested citations and formatted references.

Potential Add On: a simple filters panel for year, venue, and author backed by the metadata index.

b) Data to use

We will build a curated knowledge base of papers relevant to the course domain and our team's chosen topic area, targeting roughly 200 to 2,000 papers depending on scope and compute limits. For each paper, we will store extracted text from the PDF, lightweight section information when available, and chunked passages used for retrieval. We will also store bibliographic metadata including title, authors, year, venue, and DOI or URL, and we will keep the reference list when it can be reliably extracted to support linking and evaluation.

To create labeled data without manual annotation, we will automatically generate query to citation pairs from the knowledge base. We will extract sentences or short paragraphs from sections that commonly include citations, such as Introduction or Related Work, and treat the cited papers in those contexts as approximate ground truth labels. We will apply simple cleaning rules to remove cases that are too trivial, too ambiguous, or reference non academic sources so the evaluation set is more reliable.

---

c) Evaluation / testing plan

We will construct a dataset of citation context queries extracted from our knowledge base, where each query is a sentence or short paragraph and the label set is the paper or papers cited in that context. We will report Recall at K, Mean Reciprocal Rank (MRR), normalized Discounted Cumulative Gain (nDCG at K), and runtime latency statistics such as p50 and p95. We will compare against the following baselines: keyword only retrieval using Best Matching 25 (BM25), dense only retrieval using vector similarity search, dense retrieval with a stronger ranking stage, and a full hybrid pipeline that applies metadata filtering followed by dense retrieval and ranking.

We will run a small human evaluation within the team or class. Given short writing prompts or paragraphs, evaluators will rate the top suggested citations using a rubric covering relevance (does the source support the claim), correctness (is it the intended paper rather than a false match), and usefulness (perceived time saved and confidence). We will also validate citation formatting correctness by checking that generated BibTeX or APA or IEEE fields match the stored metadata.

We will implement unit tests to ensure parsing and indexing consistency, integration tests for the full retrieval pipeline from query to ranked citations, and regression tests on a fixed subset of documents to catch ranking or formatting changes over time.

d) Timeline and activities per student/team-member

Week	Member 1	Member 2	Member 3	Member 4	Member 5
Week 1	Finalize paper corpus selection; collect PDFs/metadata; define data schema and stable IDs	Implement PDF text extraction and chunking; basic section detection if available	Build BM25 baseline index and keyword retrieval API	Implement embedding generation for chunks; create initial vector index	Design evaluation protocol; start ground-truth construction from citation contexts; set up metrics code
Week 2	Improve ingestion robustness; deduplicate; normalize metadata (DOI/title matching)	Add reference parsing when feasible; improve chunking strategy; ingestion unit tests	Implement hybrid retrieval scaffold (BM25 + dense merge); add caching for repeated queries	Tune vector retrieval; similarity scoring; latency profiling	Build labeled dev set; run baseline evaluations; define human-eval rubric
Week 3	Implement metadata index for filters (year, venue, author) and lookup endpoints	Integrate pipeline end to end; add integration tests from ingestion to indexing	Implement ranking fusion strategy and scoring; add output formatting hooks	Implement reranking stage and evidence-snippet selection from abstracts/chunks	Error analysis on baseline results; refine ground-truth cleaning rules
Week 4	Integrate metadata pre-filtering with hybrid retrieval end to end	Improve parsing reliability on edge-case PDFs; create regression test subset	Build citation formatting module (BibTeX, APA, IEEE) using stored metadata; formatting tests	Calibrate ranking; suppress near-duplicates; add confidence scoring	Run full offline evaluation; generate result tables and plots
Week 5	Performance and scalability improvements; caching; index build scripts	Finalize ingestion pipeline and documentation; reproducibility checklist	Build CLI or lightweight web UI for paragraph input and citation output	Polish reranking and snippet justification; handle no-good-citation cases	Run human evaluation study; summarize findings; collect qualitative feedback

Week 6	Finalize dataset and indexes; freeze versions for final evaluation	Final testing pass; fix bugs; finalize integration and regression tests	Finalize UI and demo flow; package deliverable scripts	Finalize ranking thresholds; final latency report (p50/p95)	Compile final results; write report sections; prepare slides and demo script