AdvisorMatch: A Tool For Finding Thesis Advisors

Harshith Kethireddy Sanjit Sahu Archit Panwar Amit Kumar
Texas A&M University, College Station
Department of Computer Science and Engineering
UIN {537000715, 737002703, 53300200, 237007553}
{reddyharshith90, sanjit30, archit17, amit.kumar}@tamu.edu

Abstract

Finding a suitable thesis advisor is a crucial but time-consuming challenge for graduate students, often requiring hours of manual searching across scattered academic sources. AdvisorMatch streamlines this process with a search engine that ranks potential advisors based on the semantic relevance of their full publication record to a student's research topic. This paper presents the motivation, system design, ranking algorithm, and implementation plan for a working prototype.

1 Introduction

Choosing a thesis advisor is one of the most critical decisions for graduate students, yet the current process is outdated. Students must manually search faculty directories, check individual profiles, and cross-reference publications—an inefficient method that often misses well-matched advisors due to differing terminology. This project addresses that problem by developing a tool that uses natural language processing and vector search to connect students' research interests with faculty expertise, turning a time-consuming search into a simple, intuitive query.

2 Functionality and Target Audience

2.1 Core functionality

AdvisorMatch is a web-based search engine designed to recommend potential thesis advisors. Users simply enter a natural language query describing their research interests—such as "reinforcement learning for robot navigation" or "privacy in federated learning". The system then returns a ranked list of matching faculty, showing each professor's name, department, and relevant publications that demonstrate the connection. The prototype will be limited to the Computer Science and Engineering department of Texas A&M University.

2.2 Target Audience

The user base for AdvisorMatch is prospective and current graduate students (both Master's and Ph.D. candidates) who are in the process of identifying a thesis advisor.

3 Related work and contribution

3.1 Analysis of Existing Tools

Although no direct competitor to AdvisorMatch exists, its features overlap with general academic search tools. Platforms like Google Scholar, Semantic Scholar, and Scopus provide relevant research papers, whereas university websites offer basic faculty directories with limited keyword search and often outdated or incomplete profiles.

3.2 Our Novelty and Contribution

AdvisorMatch's main contribution is its shift from a paper-centric to an advisor-centric search model. Its ranking algorithm treats professors as the primary unit of retrieval, computing an overall relevance score based on how closely their collective work aligns with a user's query. This approach answers the question "Who should I work with?" rather than "What papers should I read?", offering a faster, more holistic way to identify well-matched advisors.

4 Methodology and System Architecture

4.1 Data Acquisition and Processing

Our system uses real-world data collected through a two-step process. First, a Python web crawler will extract faculty names, titles, and profile URLs target university's faculty directory; for the initial prototype, this is the Texas A&M Computer Science and Engineering directory. Then, using the Semantic Scholar API, we'll retrieve each faculty member's full publication metadata—including titles, abstracts, and years—and store it in a relational database linking papers to their authors. To ensure

scalability and maintainability, we can add an automated scheduler for weekly publication scans and monthly faculty updates.

4.2 Core Retrieval and Ranking Algorithm

Our system employs a two-stage approach combining semantic search with multi-factor author ranking. First, we use Sentence-BERT (`all-mpnet-base-v2` model) to generate embeddings for each paper's title and abstract, indexed via FAISS for efficient similarity search. When a user submits a query, we encode it with the same model and retrieve the top-k most semantically similar papers using cosine similarity. In the second stage, we aggregate these results to produce author-level rankings based on three factors: (1) average similarity of each author's top-N most relevant papers to prevent professors with many publications from dominating results, (2) recency weighting via exponential decay to prioritize recent work, and (3) an activity bonus for authors publishing within the past two years.

4.3 System Architecture

The system will follow a standard three-tier architecture:

Data Layer: Python scraper extracts faculty profiles from university website, which are then augmented with publication data via the Semantic Scholar API.A Relational database to store all faculty and publication metadata.

Backend Layer: A robust backend service that exposes a REST API. This layer contains the core logic for query encoding, vector search, and our professor ranking algorithm.

Frontend Layer: A web interface that allows users to enter queries and view ranked results from the backend.

5 Technical Challenges

The primary technical challenge is Data Aggregation and Entity Resolution. Accurately linking a professor's name from a university website (e.g., "Michael Smith") to the correct author profile on Semantic Scholar is non-trivial, as many researchers share common names. Our initial approach will be to use heuristics, such as filtering the API search by university affiliation ("Texas AM University"), but a more robust solution would require more sophisticated disambiguation techniques.

6 Evaluation and Demonstration Plan

To demonstrate the usefulness of AdvisorMatch during our final presentation, we will conduct a direct comparative demo:

We will first simulate the "traditional" workflow by spending 30-45 seconds navigating the university website, highlighting the manual clicks and searches required to evaluate just one professor.

We will then switch to the AdvisorMatch prototype and enter a specific, complex research query.

The demonstration will conclude by showing the instantaneous, ranked list of faculty. We will analyze the top result, pointing out the highly relevant paper titles presented as evidence to prove the effectiveness and efficiency of our tool.

7 Project Timeline

This project is designed for a team of four over a one-month period.

Week 1 (Oct 24 - Oct 31): Build the web crawler and API ingestion scripts. Finalize the database schema and populate it.

Milestone 1: A fully populated Relational database with faculty and paper metadata.

Week 2 (Nov 1 - Nov 7): Generate vector embeddings for all papers and create the FAISS search index.

Week 3 (Nov 8 - Nov 14): Backend API and Ranking Logic. Develop the FastAPI application and implement the core ranking algorithm.

Milestone 2: A functional API endpoint that accepts a query and returns a ranked JSON list of advisors.

Week 4 (Nov 14 - Nov 21): Frontend and Final Polish. Develop the UI, perform end-to-end testing, and prepare the final presentation.

Milestone 3: End to end working prototype completed

8 Resources to be Used

APIs and Data: Texas A&M University Website, Semantic Scholar API.

Libraries: requests, BeautifulSoup, SQLAlchemy, sentence-transformers, faiss-cpu.

Frameworks: FastAPI (backend), React (frontend).

Models: Pre-trained sentence embedding models (e.g., Sentence-BERT) from the Hugging Face Hub.