Exploring the Utility of Large Language Models in Improving the Precision of Narrator Name Extraction

Ala Kaymaram
Boston University
Boston, USA
kaymaram@bu.edu

Ibrahim Akar
New York University
Boston, USA
abeakar@gmail.com

Shawn Nassabi
New York University
New York, USA
san7522@nyu.edu

Abstract

The precise extraction of hadith narrator names is a critical task in computational hadith scholarship. This paper explores the utility of Large Language Models (LLMs) in automating and enhancing the process of hadith narrator name extraction. We present an analysis of existing methodologies, highlighting their limitations in handling the complexities of Arabic language nuances and the intricate relationships between narrators. By leveraging the contextual understanding and generative capabilities of LLMs, we propose a novel framework that integrates advanced natural language processing techniques with domain-specific knowledge. Our experiments demonstrate improvements in accuracy and efficiency compared to traditional machine learning methods.

1 Related Work

Computational techniques are increasingly being applied to the study of hadith literature, with a key area of focus being isnad analysis, which investigates the chain of narrations. Saloot et al. provide a foundational comparative analysis of Hadith classification methods prior to the advent of large language models Saloot et al. (2016). Their study re-implemented and evaluated various traditional machine learning techniques, including Naive Bayes, Support Vector Machines, and Neural Networks, on a constructed dataset from Sahih Al-Bukhari. The work establishes a quantitative baseline for Hadith classification, with a focus on matn (content) classification, to the exclusion of the structural analysis of the isnad (chain of narrators) and the use of modern deep learning and transformer models.

In the domain of isnad analysis, Alam and Schneider's work has systematically applied Social Network Analysis (SNA) to the narrator chains of Sahih Bukhari Alam & Schneider (2020). Their research models the isnad as a social graph and employs well-established centrality measures, such as PageRank, Betweenness centrality, and Degree Centrality to quantitatively identify influential narrators and key transmission hubs. This approach moves beyond simple textual analysis to provide novel insights into the propagation patterns, pinpointing central figures and revealing geographical shifts in knowledge transmission over time. Sumaira Saeed et al. also apply Social Network Analysis (SNA) to Hadith isnad, but their work uniquely focuses on the complete text of Sahih Muslim, a larger dataset from previous studies Saeed et al. (2022). A key contribution of their work is a proposed custom ranking formula that combines the number of narrations with the number of connections a narrator possess, which they validate with a Spearman correlation test. This research also resulted in a developed tool for Hadith network analysis, providing researchers with a practical system. While the study effectively uses centrality measures to identify influential narrators and provides a valuable comparison with the Sahih Bukhari network, it shares a common limitation with similar works: it does not incorporate the deep domain knowledge of Usul al-Hadith and is still reliant on manual data curation due to the lack of clean, digital Rijali sources.

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

Another approach has been the application of knowledge graphs. Building on the concept of knowledge representation, Kamran et al. present SemanticHadith, an ontology-driven knowledge graph designed to formally model and publish the Hadith corpus. This work is a pioneering effort in creating a unified, machine-readable representation of Hadith by defining a comprehensive ontology of core structural concepts, including narrators and their chains, and publishing six major Hadith collections in a standardized RDF format Kamran et al. (2023). This approach's primary strength lies in its ability to enable semantic search capabilities and facilitate knowledge discovery through structured data. However, the methodology heavily relies on a manually curated, rule-based ontology design and does not integrate more advanced machine learning techniques for entity extraction or validation on a large scale. This work serves as a foundational step toward a more integrated digital representation of Islamic knowledge, but its reliance on formal modeling limits its immediate utility for dealing with unstructured hadith corpus data. In graph-based representations, the main method involves encoding narrator names as nodes and their relationships as edges and one approach relies on rule-based processing. We review three prime examples.

[1] Relies on a rule-based pre-processing pipeline to identify and extract narrator entities and their connections, using a small dataset of 18 Hadith texts Al-Absi et al. (2024). Instead of relying on traditional graph-embedding techniques that tend to produce a cluttered "single cluster" for highly interconnected networks, the authors propose a unique layout that places narrators on concentric rings based on their chronological generation. This visual approach effectively translates the abstract, textual Isnad into a clear, interactive visual tool that can help users, including non-specialists, understand the flow of Hadith transmission across time. While this work demonstrates the practical application of graph models for Hadith classification, it is limited by its small-scale dataset and its dependence on manual, rule-based extraction, which could struggle with the complexities of larger, more varied datasets and various classical Arabic naming conventions.

[2] Another knowledge-graph-based approach with a focus on visualization is KASHAF, a knowledge-graph-based search engine that applies novel techniques for Hadith analysis, directly addressing the limitations of traditional, manual methods Shafie (2021). This approach frames narrator names as nodes and their relationships as edges, a representation that is then used for Hadith text classification. The authors' methodology relies on a rule-based pre-processing pipeline to identify and extract narrator entities and their connections, using a small dataset of 18 Hadith texts. While this work demonstrates the practical application of graph models for Hadith classification, it is limited by its small-scale dataset and its dependence on manual, rule-based extraction, which could struggle with the complexities of larger, more varied datasets and authentic, classical Arabic naming conventions.

[3] Takes a very domain-specific rule-based approach to mapping and visualizing hadith isnad chains, which makes it well adapted to Arabic and hadith stylistics, while staying transparent Azmi & Bin Badia (2010). However, its heavy reliance on hand-crafted rules, exact string matching, and manually built name variant lists prevents it from serving as a robust algorithmic pipeline. Without fuzzy matching, embeddings, or integration with Rijali sources, the trees it produces can show structure but can't offer deeper insights into narrator reliability.

2 Methodology

Our methodology combines large language model (LLM) capabilities with a custom API pipeline to extract the chain of narrators (sanad) from Arabic hadith text. We designed a modular FastAPI-based system that ingests hadith data in either plain text or CSV format, preprocesses it into manageable chunks, and calls LLM endpoints to identify and return the ordered list of narrators. This system was built for scalability and reproducibility, allowing us to run controlled experiments over large datasets while managing inference costs. We conducted our experiments using the Kaggle Hadiths dataset from the Islam & AI repository, which contains 34,441 ahadith from major Sunni collections. Each record includes the Arabic text of the hadith (text_ar) and a corresponding chain_indx that maps to narrator metadata in the companion Kaggle Rawis dataset. This structure enables objective evaluation: extracted narrators from the LLM output are compared against ground-truth narrator names derived from the dataset's chain_indx field. Evaluation was carried out using a dedicated testing script that samples hadiths from the 34,441-record dataset. To achieve at least a 95% confidence level with a ±5% margin of error, we computed the minimum required sample size using Cochran's formula with finite population correction, resulting in a threshold of approximately 380 hadiths per test run. All experiments therefore used sample sizes of 380 or greater. The testing script queries our

API in batches and stores structured outputs for reproducibility. Performance was measured using narrator-level accuracy (correctly extracted narrators divided by the total number of extractions), unique narrator level accuracy (correctly printed unique names divided by the total number of unique names), as well as full sanad accuracy (the proportion of hadiths where the complete narrator list was extracted perfectly). We ran multiple experiments both with and without fixed random seeds: fixed seeds ensured fair comparisons between models, while varied seeds allowed for robustness testing over different data subsets.

In the following subsections, we first provide a detailed Dataset Description, outlining the structure, sources, and suitability of the Kaggle Hadiths and Kaggle Rawis datasets for sanad extraction research. We then present the System Architecture, describing the FastAPI-based backend, file ingestion process, text preprocessing, and batch processing design. The LLM Pipeline subsection explains how prompts are constructed, how the extraction process is run, and how post-processing standardizes narrator names. Next, the Evaluation Procedure details our statistically grounded sampling approach, experiment configurations, and API query orchestration. Matching & Verification describes our hybrid approach for aligning extracted narrators with ground truth, combining deterministic token-subset matching with LLM-assisted fuzzy matching, along with special handling for Prophet mentions. Finally, the Metrics subsection defines the performance measures used and explains why these collectively provide a comprehensive assessment of extraction quality.

2.1 Dataset Description

The dataset, kaggle_hadiths_clean.csv, was sourced from the following GitHub repository: https://github.com/islamAndAi/QURAN-NLP/tree/master . It contains 34,441 ahadith from the following books: Sahih Bukhari, Sahih Muslim, Sunan Abi Da'ud, Jami' al-Tirmidhi, and Sunan an-Nasa'i. Each record contains information according to the following columns:

- source: The name of the book the hadith is sourced from.
- chapter_n: The chapter number.
- chapter: The name of the chapter.
- chain_indx: Contains a list of indices that are used to identify the corresponding narrators that make up the hadith's sanad from the accompanying kaggle_rawis.csv file.
- text_ar: Contains the hadith in Arabic language.
- text_en: Contains the hadith in English language.

The accompanying database, kaggle_rawis.csv, was also sourced from the same GitHub repo. It contains various information regarding 24,326 narrators. The most significant columns for the purpose of this study are scholar_indx (used to match scholars to ahadith) and name. The vast quantity of ahadith present in the kaggle_hadiths_clean.csv dataset, as well as the detailed, structured content make it highly suitable for this study. The variety of narrators present in the kaggle_rawis.csv file makes it a great resource for testing model performance since it exposes the LLM to a wide range of names with different levels of complexity.

2.2 System Architecture

The backend API was built using the Python and FastAPI. The API exposes two endpoints:

- "/upload": For extracting sanad from .txt files containing ahadith.
- "/upload-csv/": For extracting sanad from .csv files containing ahadith.

The files are parsed to extract each hadith from the list of ahadith that they contain. The list of ahadith are then provided to the extract_sanad_batch function in the openai_client.py file which makes an OpenAI API call for each hadith to extract the sanad. The LLMs are supplied with a catered system prompt and are instructed to produce structured output according to the following PyDantic model to ensure consistent results:

- source: String
- chapter_no: String

- hadith_no: String
- sanad: List of strings containing the narrator's names in Arabic.
- sanad_sentence: The full segment of the hadith that contains the sanad.
- sanad_english: The list of narrator's translated to English.
- sanad_sentence_english: The full segment of the hadith that contains the sanad, translated to English.

2.3 LLM Based Data Extraction

The core of our pipeline is the extract_sanad_batch function, which structures prompts and queries the LLM to extract sanad information from hadith text. Each hadith entry is passed to the function along with its metadata (source, chapter_no, hadith_no, and text_ar). These fields are formatted into a structured prompt that instructs the model to return only the narrators along with the sanad sentence. When English output is enabled, the model also provides English translations of both the narrators and the sanad sentence. The prompt design embeds explicit guidance on distinguishing narrators in the transmission chain from other individuals mentioned in the hadith text. For example, the system prompt includes illustrative cases where figures like Aisha are narrators (and should be included) versus cases where she is quoted as part of the hadith content (and should be excluded). This disambiguation is essential for accurate sanad extraction. Additional instructions also encourage the model to resolve relative references (e.g., "his father") into explicit names where possible. The function uses OpenAI's structured output parsing (client.responses.parse) to enforce JSON output matching predefined Pydantic schemas, ensuring consistent results. To increase robustness, calls are wrapped in an exponential backoff retry mechanism, which automatically retries failed API calls up to three times. This design allows the pipeline to process batches of hadiths efficiently while maintaining accuracy and minimizing error propagation.

2.4 Evaluation Procedure

We evaluated sanad extraction accuracy using a dedicated framework (test.py) applied to samples from the Kaggle Hadiths dataset. Each experiment used at least 380 hadiths—often more for robustness—selected either with fixed seeds for reproducibility or randomly for generalization. Sampled texts were processed through the API in batches with retry and exponential backoff handling. Extracted sanad lists were then aligned with ground-truth narrators (from chain_indx and Kaggle Rawis metadata), and both results and mappings were stored as JSON snapshots for reproducibility. Verification employed a two-stage alignment: deterministic token-subset matching for minor text variations, followed by an LLM-based fallback producing categorical judgments (YES, NO, or PROPHET). The "Prophet" label excluded references to the Prophet Muhammad (PBUH) from metrics. This hybrid verification captured both exact and fuzzy matches, including aliases, transliteration variants, and bin/ibn differences.

2.5 Metrics

We evaluated model performance using three complementary metrics tailored to sanad extraction.

- Narrator-level accuracy measures the proportion of correctly extracted narrators out of all narrators output by the model. This reflects how often each extracted name corresponds to a valid ground-truth narrator.
- Unique narrator-level accuracy considers only distinct names, measuring the proportion of correctly extracted unique narrator names relative to the total number of unique names produced. This prevents repeated correct extractions from inflating the results.
- Full sanad accuracy captures the stricter case where the entire chain of narrators is extracted exactly as it appears in the ground-truth dataset. This metric assesses whether the model can reconstruct the sanad faithfully and in full, beyond partial correctness.

Together, these three measures provide a balanced view of performance, capturing both per-name precision and the global fidelity of complete chain reconstruction.

References

Al-Absi, H. R., Kurup, D. G., Daoud, A., Schneider, J., Zaghouani, W., Al Marri, S. M. H., and Ait Mou, Y. Al-ahadeeth: A visualization tool of the hadiths' chain of narrators. In *International Conference on Human-Computer Interaction*, pp. 3–8. Springer, 2024.

Alam, T. and Schneider, J. Social network analysis of hadith narrators from sahih bukhari. In *2020 7th International Conference on Behavioural and Social Computing (BESC)*, pp. 1–5. IEEE, 2020.

Azmi, A. and Bin Badia, N. itree: Automating the construction of the narration tree of hadiths (prophetic traditions). In *2010 International Conference on Natural Language Processing and Knowledge Engineering*, pp. 1–6. IEEE, 2010. doi: 10.1109/NLPKE.2010.5587810.

Kamran, A. B., Abro, B., and Basharat, A. Semantichadith: An ontology-driven knowledge graph for the hadith corpus. *Journal of Web Semantics*, 78:100797, 2023.

Saeed, S., Yousuf, S., Khan, F., and Rajput, Q. Social network analysis of hadith narrators. *Journal of King Saud University-Computer and Information Sciences*, 34(6):3766–3774, 2022.

Saloot, M. A., Idris, N., Mahmud, R., Ja'afar, S., Thorleuchter, D., and Gani, A. Hadith data mining and classification: a comparative analysis. *Artificial Intelligence Review*, 46:113–128, 2016.

Shafie, O. A. Kashaf: A knowledge-graphs approach search-engine for hadith analysis & flow-visualization. Master's thesis, Hamad Bin Khalifa University (Qatar), 2021.

A Technical Appendices and Supplementary Material

Table 1: Summary of sanad extraction experiments across different LLM configurations.

Exp. #	LLM Model	Narrator Acc.	Seed	Sample Size
1	GPT-4o-mini	0.82	100	380
2	GPT-4o	0.86	100	360
3	GPT-4o-mini	0.86	250	480
4	GPT-4o	0.87	250	430
5	GPT-4o-mini	0.83	23	380
6	GPT-4o-mini	0.84	36	390
7	GPT-4o-mini	0.84	56	410
8	GPT-4o-mini	0.84	97	440
9	GPT-4o-mini	0.84	74	430
10	GPT-4o-mini	0.84	40	430
11	GPT-4o-mini	0.84	44	430
12	GPT-4o	0.86	96	460
13	GPT-4o	0.86	84	430
14	GPT-4o-mini	0.85	65	400
15	GPT-4o-mini	0.85	7	500
16	GPT-4o-mini	0.86	97	400
17	GPT-4o-mini	0.85	62	460
18	GPT-4o-mini	0.83	13	450
19	GPT-4o-mini	0.84	39	491
20	GPT-4o-mini	0.84	99	500
21	GPT-4o-mini	0.85	92	470
22	GPT-4o-mini	0.86	86	500
23	GPT-4o-mini	0.84	12	470
24	GPT-4o-mini	0.84	39	481
25	GPT-4o-mini	0.85	0	470

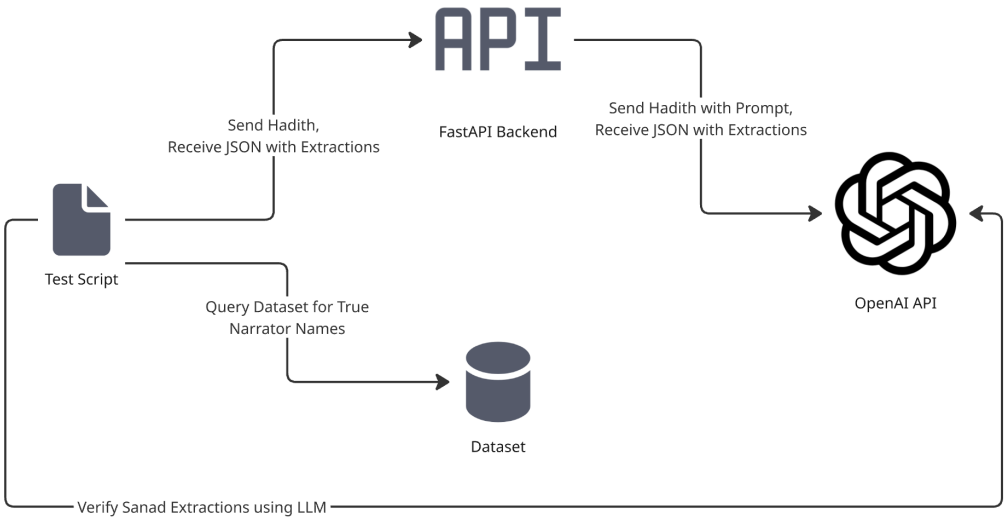

Figure 1: Architecture diagram showcasing the flow of data between the different components of the project.