UNIVERSITÉ DU LUXEMBOURG

Introduction to Generative AI

TruthMeterAI: Report

2025-12-07

Team data

Team number: 13
GitHub: Oleksandr-MB/GenAI (please check the README file for setup instructions)

Student number	First name	Last name
0231297359	Yaraslaw	Akhramenka
0231364350	Adelaide	Danilov
0230901309	Oleksandr	Marchenko Breneur

1 Introduction

TruthMeterAI is an end-to-end factuality analysis framework that converts arbitrary input text into structured, evidence-backed claim assessments. It identifies candidate factual spans, retrieves relevant Wikipedia snippets, evaluates the claims with a locally hosted instruction-tuned LLM, and aggregates the per-claim decisions into an overall verdict.
The system exposes both a CLI and a Flask-based web UI so it can be embedded in automation (CLI) or used interactively for detailed inspection.

2 System Architecture

The overall workflow, defined in `pipeline.py`, combines four major components:

1. Span Extraction (`keyword_extractor.py`)
2. Evidence Retrieval (`wiki_fetcher.py`)
3. LLM Fact Judging (`fact_checker.py`)
4. Aggregation and Output (`pipeline.py`, `schemas.py`)

Evidence is cached per unique entity to avoid duplicate Wikipedia requests, and the resulting `FactCheckResult` objects are shared between the CLI and Flask UI.

3 Component Descriptions

3.1 Span Extraction

Implemented in `keyword_extractor.py`. spaCy identifies named entities and syntactic anchors (subjects, head nouns, sentence roots). Up to `max_spans` unique ranges (default 32) are extracted, deduplicated, and wrapped as `Span` objects defined in `schemas.py`. When a sentence lacks entities, noun anchors ensure at least one representative span.

3.2 Evidence Retrieval

Implemented in `wiki_fetcher.py`. Evidence retrieval uses a hybrid approach:

- Wikipedia2Vec nearest-neighbor lookup to find related article titles,
- Fallback lexical search via the Wikipedia API with title scoring,

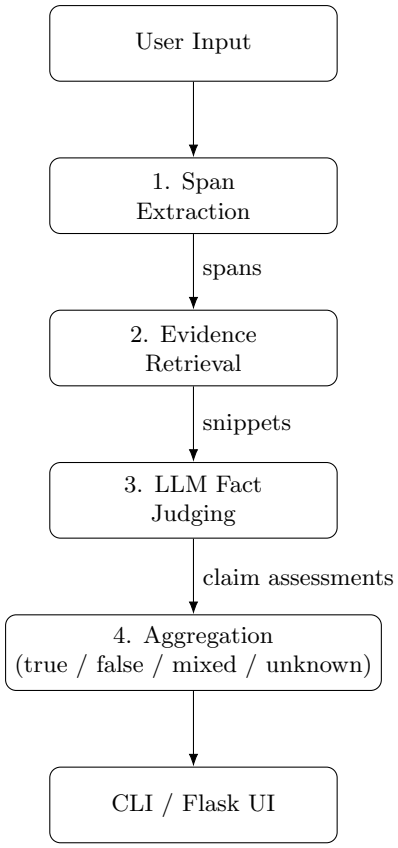

Figure 1: High-level TruthMeterAI architecture.

- Extraction of section-aware snippets by tracking heading stacks,
- TF-IDF ranking between the full user text and snippet candidates.

Top snippets are returned as `EvidenceChunk` objects (including page IDs, URLs etc.)

3.3 LLM Fact Judging

Implemented in `fact_checker.py`. For each sentence in the original text:

1. Collect the sentence and the shared evidence snippets (trimmed to configured limits).
2. Construct a strict prompt requiring `LABEL`, `EVIDENCE`, `CONFIDENCE`, `EXPLANATION`.
3. Query the local Hugging Face checkpoint (default `Qwen/Qwen3-4B-Instruct-2507`).
4. Parse the response (JSON first, regex fallback) and gracefully default to `uncertain` if parsing fails or evidence is missing.

Supported labels: `SUPPORTED`, `CONTRADICTED`, `UNCERTAIN`, `OUT_OF_SCOPE`.

3.4 Aggregation and Final Verdict

Aggregation lives in `pipeline.py` (method `_summarize_overall`). Rules:

- Any contradicted claim with confidence $\geq 0.5 \Rightarrow$ **false**,
- All claims supported with confidence $\geq 0.8 \Rightarrow$ **true**,
- Supported + uncertain/out-of-scope mix $\Rightarrow$ **mixed**,
- Otherwise (or no claims) $\Rightarrow$ **unknown**.

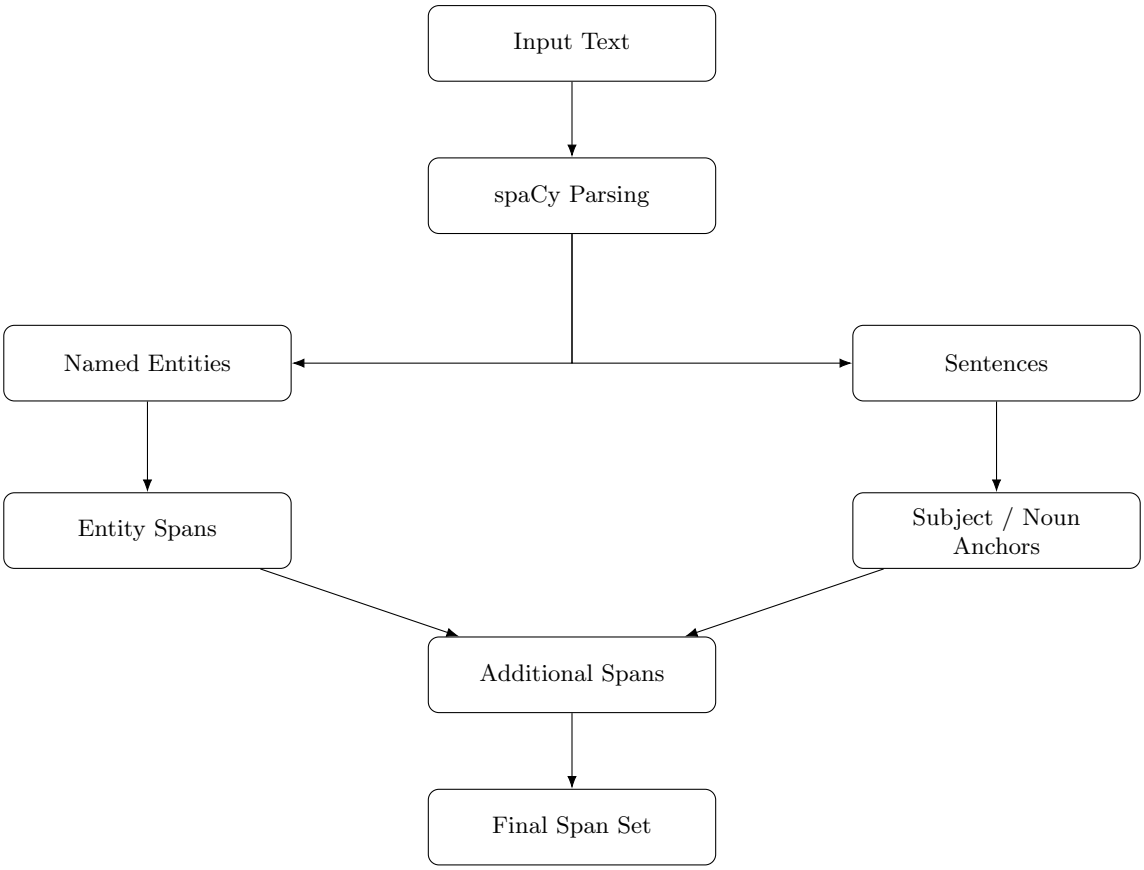

Figure 2: Span detection logic using entities and syntactic anchors.

4 Interfaces

CLI

The CLI entry point (`cli.py`) performs single-command evaluation:

```
python -m TruthMeterAI.cli "London is the capital of France."
```

It prints a JSON-serialized `FactCheckResult` containing the text, per-claim assessments, and overall verdict metadata.

Flask Web Application

Implemented in `web_app.py`. Features include:

- Text input form with a blocking overlay during inference,
- Verdict banner with overall label and average claim confidence,
- Rendered original text plus a table listing label, confidence, explanation, and evidence for each claim,
- A deduplicated list of consulted Wikipedia articles with outbound links.

5 Configuration and Extensibility

Configuration lives in `config.py`. Developers can adjust:

- LLM model name (`llm_model_name`),
- spaCy model / language (`spacy_model_name`, `language`),

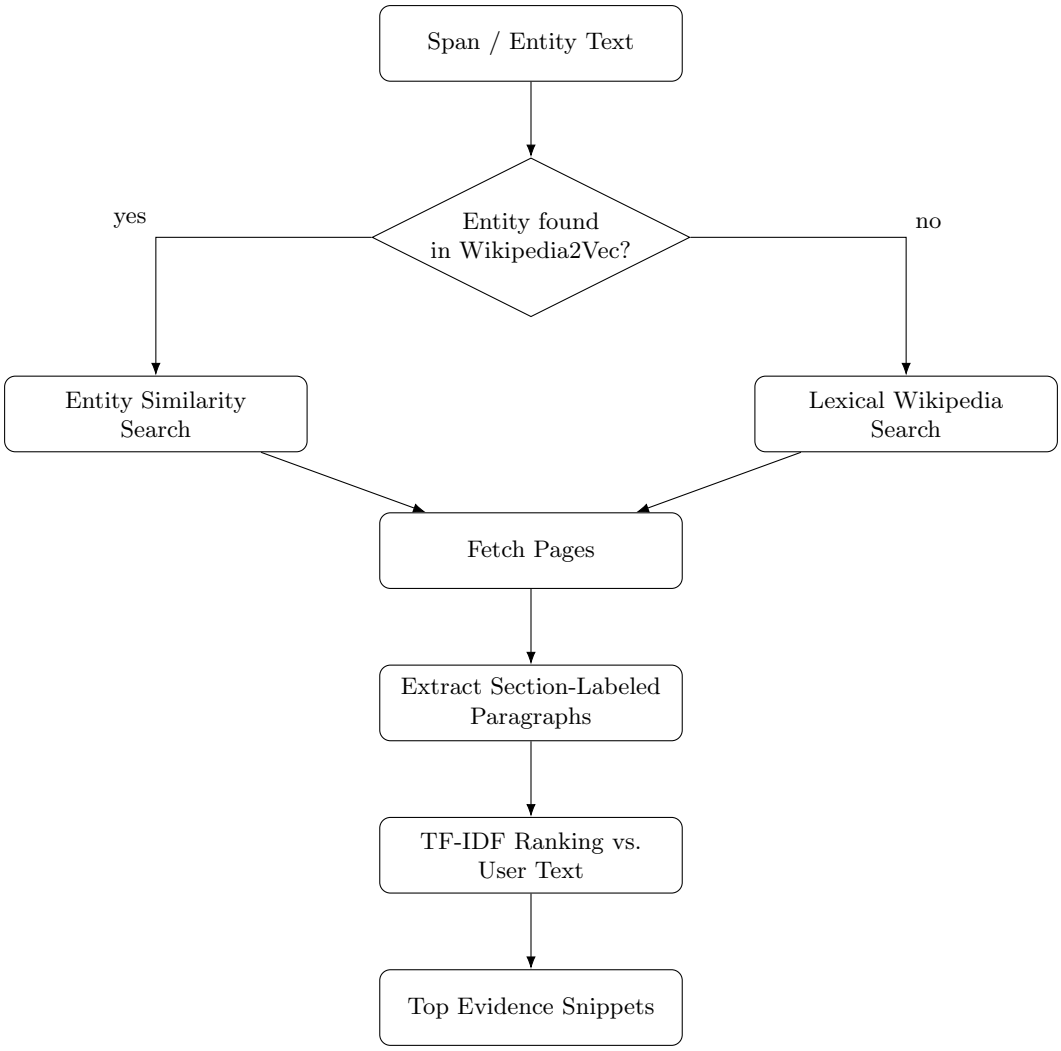

Figure 3: Evidence retrieval flow for each entity.

- Retrieval depth (`max_pages_to_fetch`, `max_snippets_per_span`),
- Span and snippet budgets (`max_spans`, `max_chars_per_snippet`).

Since every entry point instantiates `Pipeline` with the same dataclass config, swapping models or heuristics applies to both CLI and Flask UI. Retrieval backends, prompt, or LLMs can be replaced by inserting alternative classes.

6 Demonstration Examples

Example 1 - Simple Contradiction
Claim: "London is the capital of France."
The `KeywordExtractor` model extracts "London" as the primary factual span and retrieves the associated Wikipedia article; the evidence clearly states that London is the capital of the United Kingdom, not France; thus the LLM marks the claim as CONTRADICTED with high confidence, producing a FALSE overall verdict.
Screenshot: Fig 5

Example 2 - Mixed Factuality Across Sentences
Claim: "London is a large city in the UK. It is located on the Danube."
The pipeline processes each sentence independently but shares retrieved evidence.
Sentence (1) is correctly labelled SUPPORTED.
Sentence (2) contradicts the evidence and is labelled CONTRADICTED.

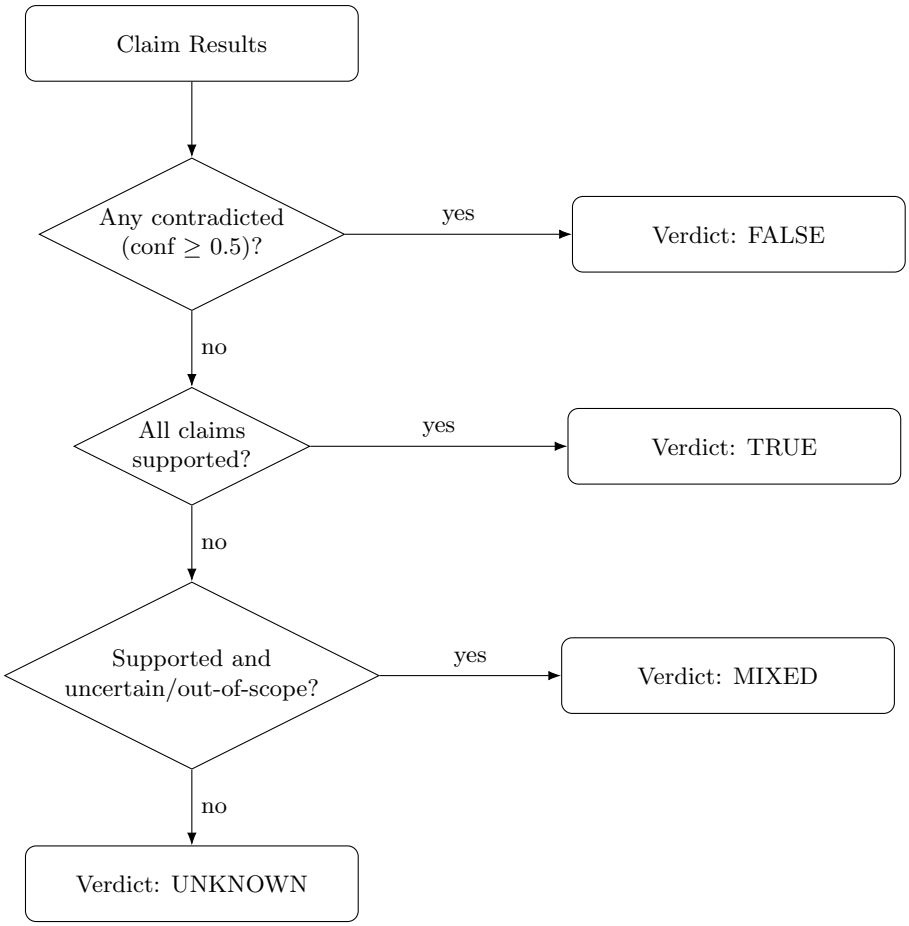

Figure 4: Decision logic for the overall verdict.

Since at least one sentence is contradicted with sufficient confidence, the system returns an overall FALSE verdict.
Screenshot: Fig 6

Example 3 — Retrieval of Deeply Hidden Evidence
Claim: "Obama has left 106 soldiers in Somalia."
The referenced fact appears in a non-prominent subsection of the Barack Obama article.
The retrieval module successfully identifies the relevant section—Legacy and Recognition, paragraph 7—and extracts the appropriate snippet.
Given the correct evidence, the LLM labels the claim as SUPPORTED, and the system outputs a TRUE verdict.
Screenshot: Fig 7

7 Limitations

- Wikipedia-only evidence constrains coverage for niche domains.
- The default compact LLM may misjudge subtle or ambiguous claims.
- Extraction capped at 32 spans (keywords), so it can miss details in very long texts.
- First-run latency depends on downloading the LLM weights and Wikipedia2Vec model; internet access is required.

8 Conclusion

TruthMeterAI is a minimalistic fact-checking system, lightweight enough to be hosted on an average consumer-grade PC. The described design choices make the system transparent, reproducible, and easy to experiment with.
While the current implementation is limited in scope; potential extensions could include richer retrieval methods, more capable models, or improved heuristics for span detection. Such enhancements would allow the pipeline to handle more complex or domain-specific factuality tasks for example by allowing general web-search requests or finetuning on the selected domain-specific data, such as medicine or law.
Overall, TruthMeterAI fulfills its goal as an educational example: a small, understandable framework that showcases the core components of automated fact checking.

Appendix

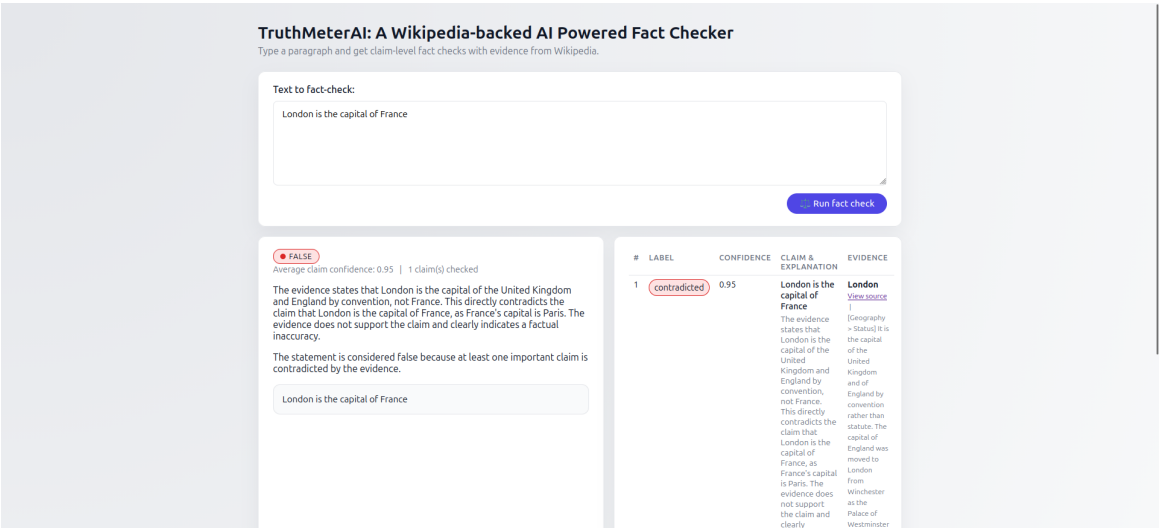

Figure 5: Trivial False Claim

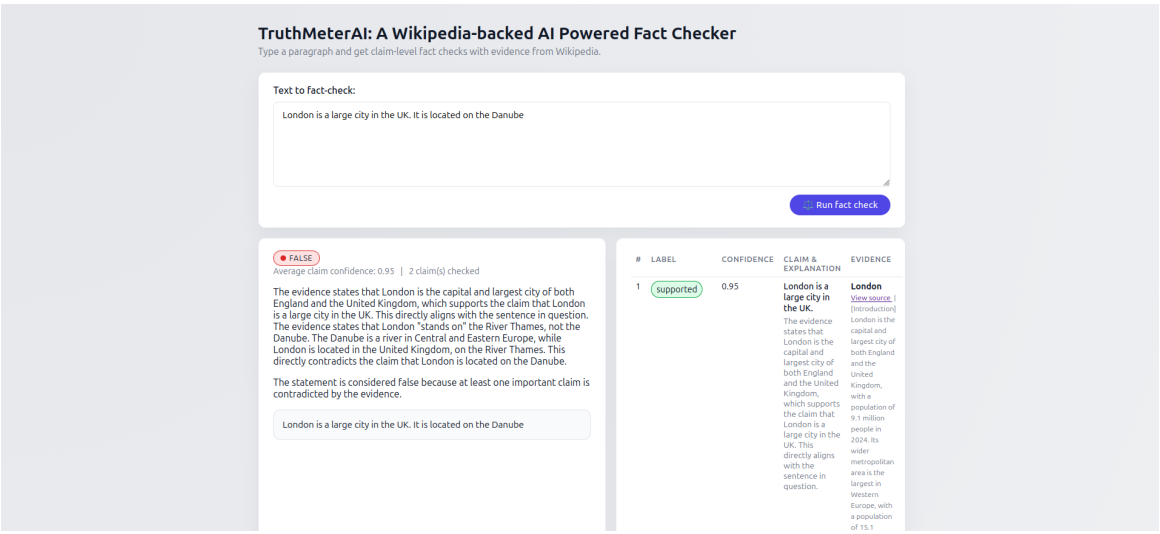

Figure 6: Compound False Claim

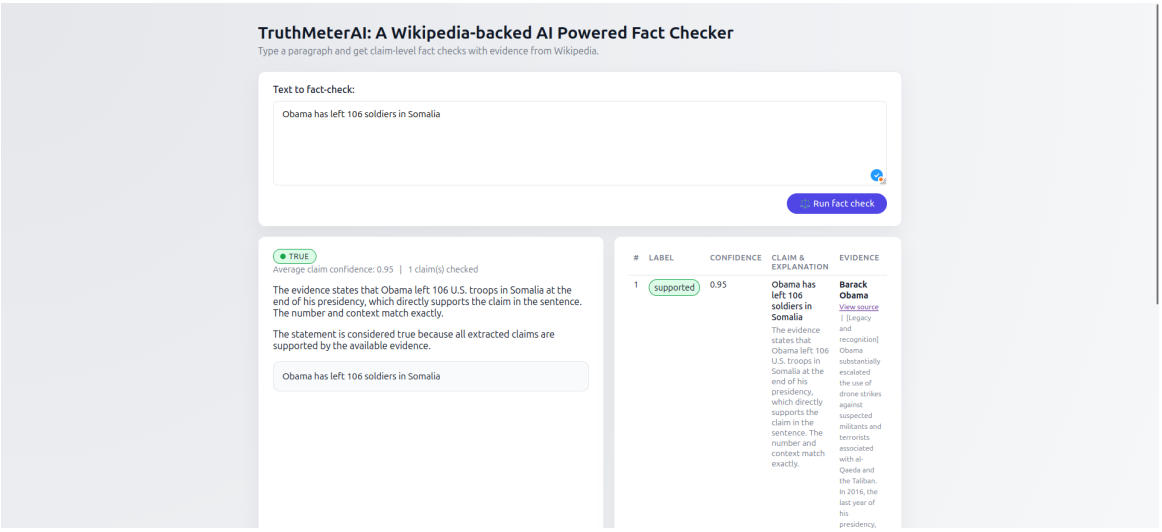

Figure 7: Obscure True Claim