# NOVELVERIFIED.AI

Technical Report & System Documentation

**TEAM STRAWHATS**

1. Bhagyesh Magar
2. Kaustubh Gawade
3. Pramey Pawar
4. Pramay Wankhede

**Track:** KDSH 2026 – Track A (Pathway Integration)

**Date:** January 12, 2026

# TABLE OF CONTENTS

## 1. Abstract

NovelVerified.AI is an automated verification system designed to validate hypothetical character backstories against the extensive, unstructured text of full-length novels (100k+ words). Developed for the **KDSH 2026 Track A** challenge, the system leverages the **Pathway framework** for scalable document ingestion and a proprietary **Dual-Perspective Anti-Bias Reasoning Engine** to mitigate LLM hallucinations. By decomposing claims into atomic facts and independently seeking both supporting and contradicting evidence, the system produces calibrated verdicts with high precision. This report details the 7-agent architecture, the integration of Pathway for real-time data handling, and the methodology used to distinguish narrative signals from noise.

## 2. Problem Statement & Track Requirements

The core task involves verifying whether a specific claim about a character (e.g., "Dantès was a Bonapartist conspirator") is **CONSISTENT (1)** or **CONTRADICTED (0)** by the source text.

This challenge presents two main technical hurdles:

1. **Context Volume:** Novels often exceed 100,000 words, making direct LLM processing impossible without lossy compression or truncation.
2. **Hallucination & Bias:** Standard RAG systems are prone to "sycophancy," where the model biases towards confirming the user's query rather than objectively verifying it.

### 2.1 Track A Compliance

To meet Track A requirements, **Pathway** (>=0.4.0) is utilized as the backbone for the data ingestion layer. We employ Pathway's connector capabilities to handle the reading, processing, and structured export of narrative data, ensuring a robust and scalable foundation for the pipeline.

## 3. System Architecture

The solution is architected as a modular **7-Agent Pipeline**. This design ensures separation of concerns between data ingestion, retrieval, and complex reasoning.

### 3.1 The 7-Agent Pipeline

#### *Agent 1: Ingestion Agent ( `ingestion_agent.py` )*

Integrates the **Pathway framework**. It connects to the `Data/` directory, reads raw text files, and splits them into chunks of 1000 tokens with 100-token overlaps. Crucially, it assigns **Temporal Tags** (EARLY, MID, LATE) to every chunk based on its position in the narrative, enabling time-aware retrieval downstream.

#### *Agent 2: Embedding Agent ( `embedding_agent.py` )*

Converts the processed chunks into vector embeddings using `sentence-transformers` . It builds a persistent **FAISS index** ( `index/faiss.index` ) optimized for CPU-based similarity search.

#### *Agent 3: Claim Parser ( `claim_parser.py` )*

A data normalization agent that reads the raw `train.csv` and `test.csv` inputs. It standardizes the claims into a JSON Lines format ( `claims.jsonl` ), ensuring a consistent schema for the reasoning engine.

#### *Agent 4: Retriever Agent ( `retriever_agent.py` )*

Performs **Temporal-Aware Retrieval**. Unlike basic semantic search, this agent queries specific narrative slices (e.g., "Search for evidence of Dantès' imprisonment in the 'EARLY' slice"). It also generates counterfactual queries to proactively search for contradictory evidence.

#### *Agent 5: Reasoning Agent ( `reasoning_agent_local.py` )*

The core logic engine, powered by **Ollama (Mistral 7B)** or Claude API. It executes the 4-Stage Anti-Bias protocol (detailed in Section 4) to derive a binary prediction. It includes robust JSON repair logic to handle malformed LLM outputs.

#### *Agent 6: Dossier Writer ( `dossier_writer.py` )*

Generates human-readable Markdown reports ( `dossiers/` ). These documents link specific text excerpts to constraint violations, providing auditability for every decision made by the AI.

#### *Agent 7: Aggregator Agent ( `results_aggregator.py` )*

Compiles the individual JSON verdicts into the final `output/results.csv` file, strictly adhering to the KDSH submission format (Story ID, Prediction, Rationale).

## 4. Methodology: Anti-Bias Reasoning

A key innovation of NovelVerified.AI is the **Dual-Perspective Anti-Bias Reasoning Engine**. Standard LLM prompting often results in false positives. To combat this, Agent 5 employs a 4-stage process:

> ***The 4-Stage Protocol:***
>
> 1. ***DECOMPOSE:*** *Break the claim into atomic sub-claims (e.g., "He was a sailor" + "He lived in Marseille").*
> 2. ***EVALUATE SUPPORT:*** *Prompt: "Actively search for evidence that SUPPORTS this claim."*
> 3. ***EVALUATE CONTRADICTION:*** *Prompt: "Actively search for evidence that CONTRADICTS this claim."*
> 4. ***SYNTHESIZE:*** *Compare confidence scores using calibrated thresholds.*

### 4.1 Calibrated Thresholds

We enforce strict numerical thresholds to determine the final verdict:

- **CONTRADICTION_THRESHOLD = 0.6:** A high bar is set for contradiction. The system must find substantial negative evidence to override the default assumption of consistency.
- **STRONG_SUPPORT_THRESHOLD = 0.5:** A moderate bar for support, acknowledging that narrative evidence can be subtle.

This logic prevents the "over-eager" behavior typical of GenAI models, ensuring that a claim is only marked as 0 (Contradicted) if there is clear, retrieval-backed evidence against it.

# 5. Implementation & Technology Stack

## 5.1 Core Technologies

| Component | Technology | Role |
|---|---|---|
| **Framework** | Pathway (>=0.4.0) | Data Ingestion & Orchestration (Track A) |
| **Vector Store** | FAISS (CPU) | High-performance similarity search |
| **LLM Backend** | Ollama (Mistral 7B) | Local inference & reasoning |
| **API Layer** | Flask | REST API for dashboard control |
| **Frontend** | React 18 + Tailwind | User Interface & Visualization |

## 5.2 Execution Modes

The system supports multiple runtime configurations:

- **Full Pipeline (CLI):** `python run_all.py --local --clean` runs the entire ETL and reasoning process headless.
- **Dashboard Mode:** A Flask-React architecture allows users to upload files, monitor progress via a real-time status bar, and visualize generated dossiers.
- **Stage-Specific:** The pipeline allows resuming from specific stages (e.g., `--start-from reasoning` ) to save time during debugging.

## 6. Engineering Challenges & Troubleshooting

During development, several specific challenges were addressed:

### *JSON Parsing Errors*

Local models (like Mistral) often output conversational filler alongside JSON. We implemented a regex-based "JSON Repair" utility in `reasoning_agent_local.py` that extracts the valid JSON block and discards the surrounding text.

### *Ollama Connectivity*

The pipeline requires a running Ollama instance. Common connection failures are handled by checking the Ollama service status before the batch processing begins.

### *Dependency Management*

Pathway requires a specific Python environment. We utilize a strictly versioned `requirements.txt` and a `setup.sh` script to ensure the environment is correctly configured (Python 3.10+).

## 7. Appendix: Directory Structure

The project follows a rigorous structure to maintain modularity:

```
StrawHats_KDSH_2026/
├── agents/
│   ├── ingestion_agent.py       # Pathway integration
│   ├── embedding_agent.py       # FAISS indexing
│   ├── claim_parser.py          # CSV parsing
│   ├── retriever_agent.py       # Temporal retrieval
│   ├── reasoning_agent.py       # Claude API logic
│   ├── reasoning_agent_local.py # Ollama logic (Anti-Bias)
│   ├── constraint_types.py      # Type definitions
│   ├── dossier_writer.py        # Report generation
│   ├── results_aggregator.py    # CSV output generation
│   ├── pathway_store.py         # Pathway document store
│   └── utils.py                 # Utilities
├── flask_api/                   # Backend API
│   ├── app.py                   # Server entry point
│   ├── pipeline.py              # Process orchestration
│   └── pipeline_api.py          # API endpoints
├── frontend/                    # React Dashboard
├── Data/                        # Source Texts
├── output/                      # Final Results
├── run_all.py                   # CLI Orchestrator
└── requirements.txt             # Dependencies
```

## 8. Conclusion

NovelVerified.AI successfully demonstrates a scalable, accurate approach to long-form narrative verification. By integrating **Pathway** for robust data engineering and employing a **Dual-Perspective Reasoning** strategy, the system effectively distinguishes between actual narrative events and hallucinated support. The modular architecture allows for easy extensibility, making it a robust solution for the KDSH 2026 Track A challenge.