
KDS HACKATHON
2026
REPORT

Technical Report: Backstory Consistency Verification System

1. Overall Approach

Core Objective

The **Backstory Consistency Verification System** is a retrieval based Natural Language Processing (NLP) framework designed to automatically assess whether a given backstory claim is **semantically consistent or contradictory** with respect to reference novel texts. The system works by retrieving relevant passages from primary literary sources and comparing them against the input claim using semantic similarity and reasoning techniques.

It produces a **binary classification output Consistency (1)** or **Contradiction (0)** along with **verbatim textual evidence** extracted from the source material to justify the decision.

This system follows **Track A: System Reasoning with NLP methods and Generative AI (GenAI)** to perform semantic understanding, contextual reasoning, and evidence-based verification.

Problem Statement and Methodology

The system addresses the following problem: **Given a backstory claim and a primary novel text, determine whether the claim is factually consistent with the content of the novel**.

To solve this, the system adopts a **multi-stage retrieval-and-reasoning pipeline** designed to handle long-form literary texts while ensuring accurate semantic comparison and explainability.

The approach consists of **four core components**:

1. **Text Chunking**: The complete novel text is segmented into **overlapping, fixed-length word-based chunks**. This strategy mitigates document-length limitations imposed by embedding models while preserving **contextual continuity across chunk boundaries**, ensuring that important narrative details are not lost during segmentation.

2. **Semantic Embedding**: Both the generated text chunks and the input backstory claim are transformed into **high-dimensional semantic vector representations** using a pre-trained **Sentence-Transformer model (all-MiniLM-L6-v2)**. These embeddings capture contextual meaning beyond surface-level lexical similarity, enabling robust semantic comparison in embedding space.

3. **Evidence Retrieval**: Using **cosine similarity scoring**, the system computes similarity scores between the claim embedding and each chunk embedding. It then retrieves the **top-k most semantically relevant chunks** from the novel, prioritizing passages that exhibit the highest semantic alignment with the backstory claim. These retrieved excerpts serve as potential supporting or refuting evidence.

4. **Consistency Decision**: A **threshold-based verification mechanism** evaluates the maximum similarity score obtained from the retrieved chunks against a **tuned similarity threshold**. If the score exceeds the threshold, the claim is classified as **consistent (1)**; otherwise, it is labeled as **contradictory (0)**. The retrieved top-k chunks are presented alongside the decision, providing **transparent and explainable evidence** for the system's verdict.

Design Rationale

What makes the **Backstory Consistency Verification System** actually usable in the real world is that it's not a "black box." We've designed it to be transparent and tunable through two main features: its decision-making logic and its evidence-backed results.

1. **The Power of Tuneable Decisions:** First, we use a **threshold-based decision rule**, which is a fancy way of saying we can fine-tune how strict or lenient the system is. Because we've optimized these settings through empirical testing, you aren't stuck with a one-size-fits-all approach. Instead, you get a controllable "precision-recall trade-off." If your project requires 100% accuracy and you don't mind missing a few minor details, you can tighten the threshold. If you'd rather catch every possible inconsistency—even at the risk of a few false alarms—you can loosen it. This level of interpretability is huge for developers who need to understand *why* a system is flagging a specific claim.
2. **Showing the Receipts:** Second, we've prioritized **evidence retrieval**. The system doesn't just give you a "yes" or "no"; it pulls verbatim text directly from the source to support its predictions. This is a game-changer for transparency. By providing the exact quotes, we allow for easy human validation. If the AI flags something, a person can quickly glance at the retrieved evidence to confirm if it's a genuine error or a misunderstanding of context. It turns the verification process from a guessing game into a clear, auditable workflow.

2. Handling Long Context

Strategy for Managing Extended Text

Literary novels are often far longer than what neural models can process in a single pass due to token and memory limitations. To handle this challenge, the system is designed to efficiently work with extended texts without losing important narrative details.

Chunking Strategy

To make long novels manageable, the text is divided into **fixed-size, overlapping word-based chunks**. The overlap between consecutive chunks ensures that important contextual information especially details that span across paragraph or sentence boundaries is preserved.

- **Benefit**: Enables processing of arbitrarily long texts while maintaining local semantic context.

Retrieval-Based Prioritization

Rather than processing the entire novel end-to-end, the system employs a retrieval step that dynamically selects only the top-k most relevant chunks relative to each backstory claim.

- **Top-k Retrieval**: By sorting chunks by cosine similarity and selecting only the highest-ranked excerpts, the system concentrates analysis on informationally relevant portions.
- **Efficiency**: Reduces computational overhead and memory consumption while focusing decision-making on salient evidence.

Two-Stage Pipeline

Our **Two-Stage Pipeline** is the secret sauce that keeps the system fast and accurate, even when we're dealing with massive amounts of text. Think of it as a "search then verify" approach:

1. **Coarse Filtering (The Quick Scan):** Instead of reading every single word deeply, the system starts with a high-speed scan. It uses **cosine similarity** to compare the claim against all available text chunks, instantly pulling the "Top-K" most relevant snippets. It's like using a search engine to find the right pages before you start reading.
2. **Fine-Grained Decision (The Deep Dive):** Once we have those specific snippets, the system applies **threshold-based logic** to make the final call. This is where the heavy lifting happens analysing the filtered evidence to give a definitive consistency verdict.

By splitting the work this way, we can process a 1,000-page novel without having to cut the text short or lose track of critical details. You get the best of both worlds: high-level accuracy and impressive computational speed.

3. Distinguishing Causal Signals from Noise

Signal Identification Methodology

Meaningful patterns genuine consistency or contradiction signals are distinguished from noise through semantic similarity scoring and evidence-based validation.

Semantic Embedding as Signal Amplification

- **Semantic Space Alignment**: Sentence-Transformer models encode both the backstory claim and novel passages into the same semantic space. In this space, texts with similar meanings are positioned closer together even if they use different words.
- **Noise Reduction**: Superficial term matching is inherently avoided; only semantically meaningful alignments produce high similarity scores. Synonymous expressions and paraphrases naturally register as signals rather than noise.

Threshold-Based Filtering

- **Similarity Thresholding**: The system applies a learned threshold on maximum cosine similarity scores retrieved from the top-k chunks. Similarities below this threshold are classified as insufficient evidence (contradiction), while those above it indicate consistency.
- **Empirical Optimization**: The threshold is tuned on labeled training data (train.csv) by sweeping across candidate values and selecting the threshold that maximizes accuracy on known labels.

Top-k Evidence Validation

- **Multi-Evidence Aggregation**: By retrieving top-k chunks (rather than a single best match), the system reduces susceptibility to outlier noise. Consensus among multiple high-scoring matches strengthens confidence in the decision.
- **Explainability**: Returned verbatim excerpts provide human-verifiable evidence, enabling manual inspection and filtering of false positives caused by semantic artifacts.

Heuristic Filters

The system inherently filters irrelevant data through:

- **Cosine Similarity Metric**: Normalized vectors and cosine distance naturally downweight less relevant content.
- **Book-Specific Matching**: Claims are matched against the correct novel source (via book_name column), preventing cross-novel contamination.

4. Key Limitations and Failure Cases

Known Constraints and Assumptions

Semantic Embedding Limitations

- **Out-of-Domain Concepts**: If a backstory claim references entities, events, or concepts not present or only obliquely mentioned in the novel, the embedding model may fail to identify genuine signals due to vocabulary or conceptual gaps.
- **Implicit vs. Explicit Information**: The system cannot reliably infer implicit plot details or unstated context. Claims about events not directly narrated in the novel may be classified as contradictions despite being logically consistent with the narrative.

Threshold-Based Decision Rule

- **Single Threshold Rigidity**: A fixed threshold applied uniformly to all claims may be suboptimal for claims with varying semantic similarity distributions. Claims requiring nuanced judgment (e.g., temporal ambiguities, partial truths) may be misclassified.
- **Threshold Overfitting**: Tuning thresholds on training data risks overfitting to artifacts in the labeled dataset, reducing generalization to novel claims.

Chunking-Induced Information Loss

- **Boundary Fragmentation**: Critical information spanning chunk boundaries may be split across separate chunks, reducing within-chunk semantic coherence and potentially lowering retrieval effectiveness.
- **Fixed Chunk Size**: A predetermined chunk size does not adapt to varying information density across the novel, potentially creating inappropriately sized segments.

Model and Data Dependencies

- **Embedding Model Bias**: Sentence-Transformers (`all-MiniLM-L6-v2`) has inherent biases toward certain domains and writing styles. Performance may degrade on literary texts outside the model's training distribution.
- **Limited Training Data**: Accuracy depends on the size and quality of labeled training data (train.csv). Sparse or noisy labels directly undermine threshold tuning quality.
- **Cross-Novel Generalization**: The system requires labeled examples for each novel to reliably tune novel-specific thresholds. Applying thresholds trained on one novel to an untrained novel may yield poor performance.

Failure Scenarios

No matter how advanced a system is, there's always a "break point" where things don't go exactly as planned. If we're going to use the **Backstory Consistency Verification System** effectively, we have to talk about its "blind spots."

It isn't just about the system being "wrong" it's about understanding the specific scenarios where the logic might trip up so we can prepare for them. Here is a deep dive into the failure scenarios we need to keep on our radar.

1. **The "Synonym Trap" (Semantic Drift):** Computers are getting better at understanding meaning, but they still get hung up on vocabulary. **Semantic Drift** happens when a claim uses totally different words than the source text, even if they mean the same thing.
   For example, if the novel describes a character as "overwhelmed by a sense of melancholic longing," but the claim says he was "unhappy about the past," the system might not see a strong enough connection. Because the mathematical distance between those specific word embeddings is too wide, the system might give it a low similarity score and mistakenly flag it as a contradiction simply because it didn't "recognize" the phrasing.
2. **The "Not" Problem (Negation and Logic):** This is a classic hurdle in Natural Language Processing. Embedding models are great at identifying **topics**, but they can be surprisingly bad at **logic**.
   If a novel says, "John went to the store," and a claim says, "John *never* went to the store," a basic embedding model might actually see these as very similar because they both involve "John," "going," and "store." The system might fail to weigh the word "never" heavily enough, leading it to mark a flat-out lie as "consistent" because the topical context matches so well. Complex logical structures (like "if/then" statements or double negatives) only make this harder for the AI to parse reliably.
3. **The Strict Librarian (Temporal and Quantitative Precision):** The system is a bit of a perfectionist when it comes to numbers and dates. We call this the **Precision Problem**.
   In a story, if a character spends 10 days in the mountains, but a claim says they were there for 11 days, the system will likely flag that as a hard contradiction. To a human reader, that one-day difference might be a rounding error or totally irrelevant to the plot. To the system, however, 10 does not equal 11. It treats small, functionally minor discrepancies with the same severity as massive plot holes. This lack of "nuance" means you might get a lot of "false alarms" over tiny details that don't actually hurt the story's consistency.
4. **The Hardware Tax (Scalability Constraints):** As the project grows, so do the demands on your machine. There are two main parts to this:
   - **Computational Overhead:** Every time you add a new chapter or a new set of claims, the work increases linearly. If you're trying to process an entire series of massive novels or run a huge batch of claims all at once, you're going to notice a significant "lag" or latency. It's not an infinite resource.
   - **Memory Requirements:** Storing "embeddings" (the mathematical maps of the text) takes up a lot of space. For instance, using a model like all-MiniLM-L6-v2 creates 384-dimensional vectors for every single chunk of text. If you have a 500,000-word novel broken into tiny pieces, those thousands of high-dimensional vectors start to eat up RAM very quickly.
5. **The "Trust Me" Factor (Limited Explanations):** Finally, there is the issue of **Interpretability**. While the system is great at showing you the "Receipts" (the Top-K evidence chunks), it doesn't actually explain its "thought process."

If the system makes a "borderline" decision meaning the score was right on the edge of the threshold it won't tell you why it leaned one way or the other. It doesn't explain why it chose a specific threshold or why it found one piece of evidence more "convincing" than another. This makes it tough for a human user to debug the system or understand why it's failing on specific edge cases. You're left with the evidence, but you still have to do the heavy lifting to figure out the "why."

Scalability and Accuracy Constraints

When we look at the scalability and accuracy of the Backstory Consistency Verification System, it's clear we've built something robust, but it isn't bulletproof. Every high-performance tool has its "edge cases," and here is a quick reality check on where this system might hit a wall:

1. **The "Epic Novel" Problem:** While the system is designed for heavy lifting, extremely long works—think massive 500,000-word epics—can cause some computational fatigue. To process that much data, we have to break the text into smaller pieces. This "chunking" is necessary, but it can sometimes create artifacts where the system loses the broader narrative thread, leading to a dip in overall performance.
2. **The Gray Areas of Ambiguity:** The system is great at logic, but it's not a literary critic. If a claim is inherently vague or lacks a clear factual foundation within the text, the AI will struggle. It simply doesn't have the human-level intuition required to resolve deep interpretive uncertainty or "read between the lines" when the author is being intentionally cryptic.
3. **Stepping Out of Its Comfort Zone:** Finally, we have to consider Domain Shift. The system is only as good as the data it was raised on. If you try to apply it to a genre or a writing style that is drastically different from its training corpus like shifting from modern thrillers to 18th-century poetry the accuracy is going to take a noticeable hit. It thrives in familiar territory but needs a bit of recalibration when entering a new world.

Conclusion

The **Backstory Consistency Verification System** is essentially our solution to the growing headache of verifying claims at scale without losing the human-readable "why" behind the results. By leaning on semantic embeddings and smart evidence retrieval, we've created a process that is both manageable and transparent. One of its biggest strengths is its ability to wade through massive amounts of text; it's remarkably effective at digging through long-form contexts to find the signals that actually matter while filtering out the unnecessary noise.

However, it's important to remember that this isn't a "set it and forget it" tool. There are some practical hurdles to keep in mind. For instance, the system's performance is heavily tied to the specific embedding models we use, and the "thresholds" it uses to make decisions can be a bit rigid. Furthermore, because the system relies so much on labeled training data, it tends to have a bit of a "home-field advantage" it performs best on topics it has seen before.

If you're planning to deploy this on a completely new dataset or a fresh domain, you'll want to take a cautious approach. It's a powerful engine, but it requires a bit of manual tuning and a critical eye to ensure these underlying constraints don't skew your results in a new environment.