Narrative Consistency Verification: A Reasoning-First Approach

Technical Report for National Data Science Hackathon

---

1. Introduction

The task is to verify whether a given backstory about a character is consistent with the full narrative of a novel. This is harder than it sounds. Novels can be hundreds of thousands of words long, and backstories can make claims that require understanding events scattered across different chapters, or even implicit character traits that are never directly stated.

The challenge isn't just about finding matching text. It's about reasoning whether a claim like "Character X fears abandonment due to childhood trauma" is supported, contradicted, or simply not addressed by the narrative. Surface-level keyword matching fails here because consistency depends on causal logic, character psychology, and narrative constraints.

This is a verification task, not generation. The system doesn't create backstories—it judges whether provided ones hold up against the source material.

2. Problem Understanding

Consistency in this context means more than "the text doesn't explicitly contradict the claim." It requires checking three types of alignment:

Direct contradiction: The novel explicitly states something opposite to the backstory claim. For example, if the backstory says "John was an only child" but the novel mentions his sister, that's a clear failure.

Causal implausibility: The claim might not be directly contradicted, but it doesn't make sense given what happens in the story. If a backstory claims a character "always acts rationally under pressure" but the novel shows repeated instances of panic and poor decisions, that's inconsistent even without a direct negation.

Narrative constraint violations: Some claims impose constraints on what should or shouldn't happen. If a backstory says a character "vowed never to return to Paris," but the novel has them casually visiting Paris later, that's a violation.

The key insight is that surface-level text matching is insufficient. You can't just search for keywords and call it done. The system needs to understand what the claim actually implies and whether the narrative supports or undermines that implication.

3. System Overview

The system processes each backstory through a pipeline:

1. **Backstory decomposition**: The input backstory (often a compound statement) is broken down into atomic, testable claims. Each claim is categorized by type (e.g., early life event, fear, motivation) and assigned metadata like confidence and whether it's a core trait.
2. **Claim validation and repair**: Claims are validated to ensure they're atomic (not compound sentences). If validation fails, the system attempts automatic repair by splitting compound claims. This prevents silent failures.
3. **Novel ingestion**: The full novel is ingested and chunked into overlapping paragraphs. This preserves local context while making retrieval tractable.
4. **Evidence retrieval**: For each claim, the system retrieves the most relevant chunks from the novel using TF-IDF similarity. This is a lightweight approach that works without embeddings or external APIs.
5. **Claim evaluation**: Each claim is evaluated against its retrieved evidence. The system looks for supporting patterns (keyword overlap without negation) and contradicting patterns (keyword overlap with negation or adversarial keywords).
6. **Final decision**: The system aggregates evaluated claims. If any core trait is contradicted, or if multiple non-core claims fail, the backstory is marked inconsistent. Otherwise, it's marked consistent.

The flow is linear but each stage has specific reasoning built in.

4. Handling Long Context

This is one of the most important design decisions.

The system ingests the **entire novel**—no truncation, no summarization. For a novel like *The Count of Monte Cristo* (over 2.7 MB of text), this means processing the full narrative. Why? Because consistency violations can appear anywhere. A claim about a character's childhood might only be contradicted in a flashback that appears late in the book. Truncating to the first N tokens would miss this.

Chunking strategy: The novel is split into paragraphs, and each chunk includes the current paragraph plus one overlapping paragraph from before. This overlap ensures that context isn't lost at chunk boundaries. For example, if a character's motivation is explained across two consecutive paragraphs, the overlap ensures both are captured together in at least one chunk.

Retrieval instead of full-text reasoning: Rather than trying to reason over the entire novel at once (which would be computationally expensive and

error-prone), the system uses TF-IDF retrieval to find the top-k most relevant chunks for each claim. This is different from truncation-based methods because: - Truncation discards information permanently. - Retrieval dynamically selects relevant sections based on the claim being tested.

The retrieval approach means the system can handle novels of arbitrary length without running into context window limits. It also means the system focuses its reasoning on the parts of the novel that actually matter for each claim.

5. Consistency & Causal Reasoning

The system doesn't just check if words match. It tries to reason about what the claim means and whether the evidence supports or contradicts it.

Claims as constraints: Each claim is treated as a constraint on the narrative. For example, "Character X is motivated by revenge" implies that X's actions should align with revenge-seeking behavior. If the novel shows X repeatedly forgiving enemies, that's evidence against the claim.

Core traits dominate: Not all claims are equally important. A claim marked as a "core trait" (e.g., "defining childhood trauma") is weighted more heavily. If a core trait is contradicted, the entire backstory is marked inconsistent, even if other claims pass. This reflects the intuition that some aspects of a character's backstory are foundational.

Contradictions override weak evidence: The evaluation logic is conservative. If any evidence chunk shows a contradiction (keyword overlap + negation patterns), the claim is marked as FAIL, regardless of how much supporting evidence exists. This is intentional—one strong contradiction is more meaningful than several weak supports.

Accumulation of weak signals: For claims without contradictions, the system requires at least two supporting evidence chunks to mark the claim as PASS. A single weak match is treated as UNKNOWN. This prevents false positives from coincidental keyword overlap.

The reasoning here is pattern-based, not model-based. There's no LLM doing inference. Instead, the system uses heuristics like negation detection, keyword overlap, and adversarial keyword matching (e.g., "however," "but," "contrary"). These heuristics are simple but effective for catching common contradiction patterns.

6. Robustness to Real-World Backstories

Real backstories aren't always clean. They often contain compound statements, ambiguous phrasing, or multiple claims packed into one sentence. The system is designed to handle this.

Compound backstory statements: If a backstory says "John was born in

Paris and later moved to London, where he became a sailor," that's actually three claims: 1. John was born in Paris. 2. John moved to London. 3. John became a sailor.

The decomposition module automatically splits these using sentence structure and conjunctions. Each atomic claim is then validated and tested independently.

Automatic claim decomposition: The `BackstoryDecomposer` uses regex patterns to detect compound structures (e.g., ", and", ";", ".") and splits them. It also categorizes each claim by type (early life event, motivation, fear, etc.) and assigns confidence based on linguistic markers (e.g., "always" → high confidence, "maybe" → low confidence).

Strict validation and repair: After decomposition, claims are validated to ensure they're truly atomic. If validation fails (e.g., a claim still contains multiple independent clauses), the system attempts automatic repair by further splitting. If repair fails, the system raises an error rather than proceeding with bad data. This prevents silent failures where a compound claim is treated as atomic and produces misleading results.

Why this matters: Without robust decomposition and validation, the system would either fail on real-world inputs or produce unreliable results. The validation-repair loop ensures that every claim entering the evaluation pipeline is well-formed.

7. Results and Observations

The system produces a binary label (0 = inconsistent, 1 = consistent) and a rationale for each backstory.

The predictions are conservative. The system is more likely to output 1 (consistent) than 0 (inconsistent) because it only marks a backstory as inconsistent if there's clear evidence of contradiction. Absence of evidence is not treated as evidence of absence—if the novel doesn't mention something, that's not a contradiction.

This conservative behavior is intentional. In the absence of strong contradictory evidence, the system defaults to "no decisive contradictions found." This reflects the reality that novels often leave character details implicit or unaddressed.

The rationales are simple and rule-based: - "Core backstory claim contradicted by narrative evidence" → A core trait failed. - "Multiple backstory claims contradicted" → Two or more non-core claims failed. - "No decisive contradictions found" → Default case.

There are no tables of metrics here because the focus is on the reasoning process, not raw accuracy. The system is designed to be interpretable and debuggable, not to maximize a leaderboard score.

8. Limitations

The system has several known limitations:

Implicit contradictions are hard: If a backstory claims "Character X is deeply religious" but the novel shows X mocking religious practices without explicitly stating "X is not religious," the system might miss this. The pattern-matching approach catches explicit negations but struggles with implicit contradictions that require deeper inference.

Character aliases are not fully resolved: If a character is referred to by different names (e.g., "John," "Mr. Smith," "the sailor"), the system treats these as separate entities. This can lead to missed evidence or false negatives. A proper coreference resolution system would help, but that's beyond the scope of this implementation.

Literary ambiguity exists: Novels sometimes intentionally leave character details ambiguous or contradictory (e.g., unreliable narrators, conflicting accounts). The system doesn't handle this well—it treats all narrative text as equally authoritative.

Retrieval noise can mislead: TF-IDF retrieval is simple and fast, but it can surface irrelevant chunks if they happen to share keywords with the claim. For example, a claim about "fear of water" might retrieve chunks mentioning "water" in unrelated contexts (e.g., "drinking water"). The evaluation logic tries to filter this with pattern matching, but it's not perfect.

These limitations are real and not easily fixed without significantly more complexity (e.g., using LLMs for inference, adding coreference resolution, building domain-specific knowledge graphs). The current system is a pragmatic trade-off between simplicity and effectiveness.

9. Conclusion

The system takes a reasoning-first approach to narrative consistency verification. It decomposes backstories into atomic claims, retrieves relevant evidence from the full novel, and evaluates each claim using pattern-based reasoning. The design prioritizes handling long context, robust claim decomposition, and interpretable decision logic over raw performance metrics.

The key strengths are the ability to process full-length novels without truncation, automatic handling of compound backstory statements, and conservative decision-making that avoids false positives. The main weaknesses are limited handling of implicit contradictions and reliance on keyword-based retrieval.

This is a rule-based system, not a learned model. There's no training, no fine-tuning, just explicit reasoning encoded in code.