Constructing Coherent Spatial Memory in LLM Agents Through Graph Rectification

Puzhen Zhang and Xuyang Chen and Yu Feng and Yuhan Jiang and Liqiu Meng
Chair of Cartography and Visual Analytics
Technical University of Munich
{puzhen.zhang, xuyang.chen, y.feng,
yuhan.jiang, liqiu.meng}@tum.de

Abstract

Given a map description through global traversal navigation instructions, an LLM can often infer the implicit spatial layout and answer user queries by providing shortest paths. However, such context-dependent querying becomes incapable as environments grow larger, motivating the need for incremental map construction that builds a complete topological graph from stepwise observations. We propose a framework for LLM-driven construction and map repair, designed to detect, localize, and correct structural inconsistencies in incrementally constructed navigation graphs. Our contributions include a version control mechanism for graph construction, an Edge Impact Score for repair prioritization, and a cleaned variant of the MANGO benchmark tailored for LLM-driven map construction and repair. Compared with direct LLM-based incremental mapping, MapRepair raises node recall by 8.6 percentage points to 94.3% and edge recall by 55.8 percentage points to 88.2%, evaluated on Chapters 16 and 17 of *Dream of the Red Chamber* with GPT-4.1 as the underlying LLM.

1 Introduction

LLMs have shown strong abilities in open-domain reasoning, sequential planning, and text-based navigation. However, in text-processing environments, spatial cognition with LLMs still primarily depends on direct reasoning within the context window (Ding et al., 2024). This approach presents several potential challenges: it may exceed context capacity limitations when processing extensive texts, encounter context forgetting issues when addressing complex problems, and introduce inconsistencies in iterative reasoning processes. Consequently, when tackling complex large-scale spatial problems, adopting a human-like cognitive approach—progressively assembling local spatial cognition to achieve understanding of complex spaces (Xia et al., 2025)—may constitute a superior solution. For LLMs, this methodology alleviates contextual pressure by incrementally storing local spatial cognition in graph structures, requiring the context to process only current local information. Furthermore, the structured storage through graph representation ensures consistency in search structures and provides error correction capabilities when cognitive biases occur. As demonstrated in Figure 1, our MapRepair framework significantly improves map construction quality, achieving 94.3% node recall and 88.2% edge recall.

Despite this potential, maintaining complex spatial layouts in graph structures remains challenging. Small errors made early can silently propagate, manifesting as conflicts only when sufficient context accumulates. This temporal gap between error introduction and detection is exacerbated by *coupled dependencies*, where one error triggers cascading mistakes. Since most LLMs lack persistent memory or version control, they cannot trace error provenance or reason about when faulty edges were introduced.

To address this challenge, we propose **LLM-MapRepair**, a modular framework for detecting and repairing topological inconsistencies in navigation graphs constructed by LLM agents. At the core of our method is the **Version Control**, a versioned graph history that records every modification to the graph, along with its originating observation and time indexed head. Version Control enables time-aware tracing, rollback, and structural comparison, allowing the system to pinpoint the specific actions that introduced inconsistencies, even if they occurred many steps earlier.

To improve the efficiency of graph repair, we introduce an **Edge Impact Score** to prioritize repair actions by estimating the potential downstream effects of each edge based on reachability, usage frequency, and conflict propagation. This enables the system to identify low-impact edges that can be safely edited or removed, thereby reducing the

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 48126–48146

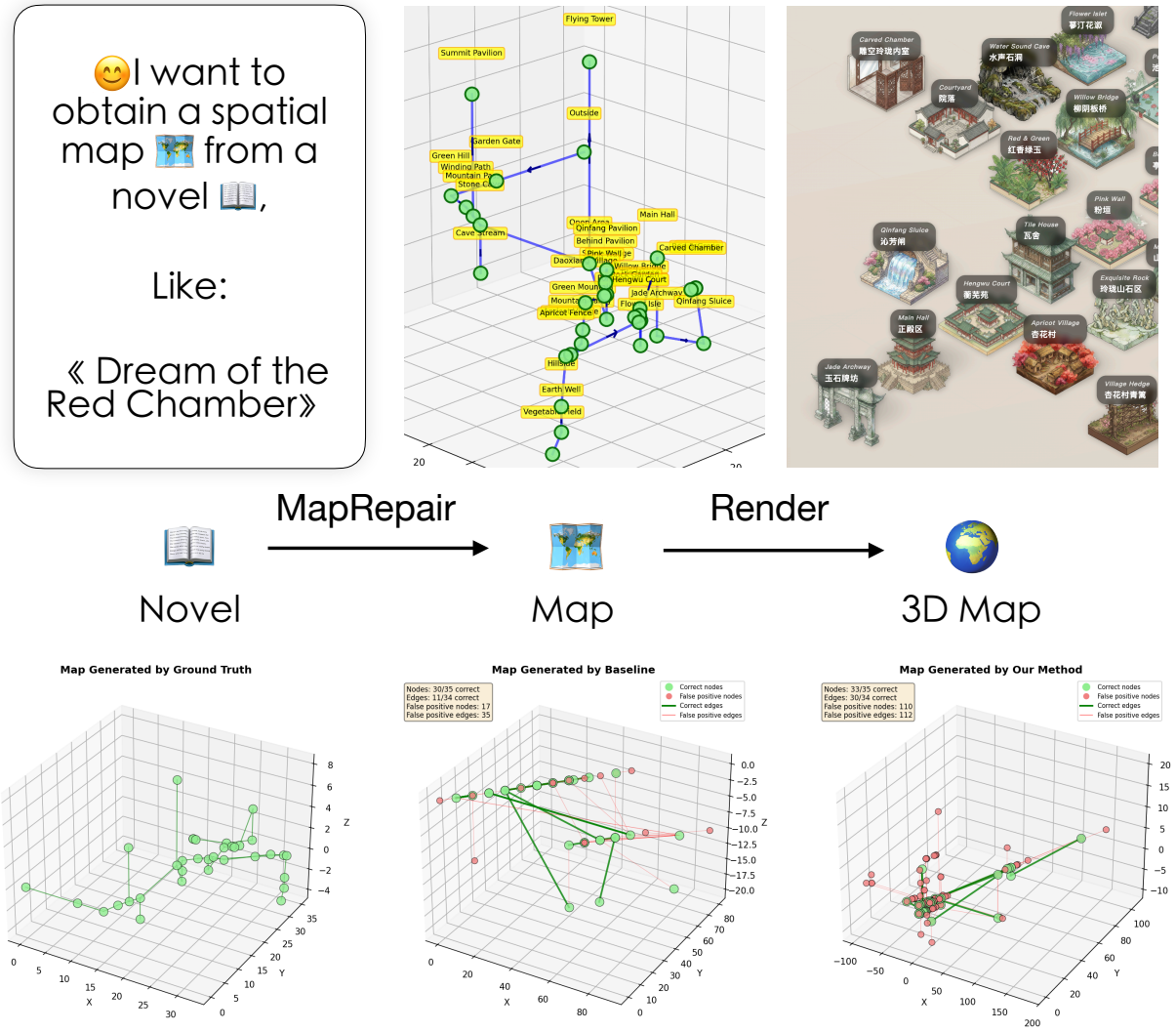


Figure 1: Our framework, MapRepair, can extract spatial relationships between entities from long-form text and produce structured maps that can subsequently be rendered into high-quality visualizations.

risk of introducing further inconsistencies during the repair process.

We evaluate our approach on environments from the MANGO benchmarks (Ding et al., 2024), where LLM agents construct navigation graphs from raw textual observations. Experiments show that our method significantly improves structural integrity and overall task performance, especially in cases involving long-range error propagation. Our contributions are as follows:

- We identify a critical limitation of LLM-based agents in long-horizon exploration: their inability to detect and correct accumulated structural errors that emerge from temporally distant actions.
- We propose a history-aware graph repair framework, integrating Version Control-based error tracing, Edge Impact scoring.
- We refine the MANGO (Ding et al., 2024) benchmark dataset by systematically removing all non-topological actions and inherent structural conflicts, creating a topologically consistent dataset better suited for evaluating LLM-based spatial mapping and navigation.

1.1 Related Work

Enhancing the Spatial Reasoning Ability of LLMs. In recent years, the spatial reasoning ability of large language models (LLMs) has been improved through specialized training and prompting strategies. AlphaMaze (Dao and Vu, 2025) combines supervised learning with reinforcement learning (GRPO) for maze navigation, while Mind's Eye (Wu et al., 2024) introduces a "visualization-of-thought" prompting technique to simulate internal

spatial representations. Although these approaches achieve progress in reasoning, they remain constrained by the context window and lack mechanisms to maintain long-term consistency during extended spatial reasoning tasks.

Mapping Evaluation in Language Agents. LLMs demonstrate a certain degree of spatial understanding when reasoning within short contexts, but they encounter significant limitations in long-horizon or complex textual reasoning. The MANGO benchmark (Ding et al., 2024) shows that even GPT-4, due to its context length limitation, can only process the first 70 steps of a text-based environment. To mitigate this, modular navigation frameworks (Zhang and Ji, 2025) introduce planning–execution modules. However, they lack graph-level consistency tracking mechanisms, which causes structural errors to accumulate unnoticed during extended exploration. These findings highlight that *localized or incremental mapping* is a necessary pathway for extending LLM reasoning in complex environments. To further address these limitations, it is instructive to look at Simultaneous Localization and Mapping (SLAM), where incremental mapping and long-term coherence have been extensively studied.

SLAM as Inspiration. SLAM methods provide theoretical foundations for incremental mapping and consistency maintenance. GraphSLAM (Lu and Milios, 1997) formulates mapping as graph optimization, while loop closure techniques (Gálvez-López and Tardos, 2012; Cummins and Newman, 2008) detect revisited places to trigger global corrections. ORB-SLAM (Mur-Artal et al., 2015) and Cartographer (Hess et al., 2016) demonstrate effective consistency maintenance by separating local and global optimization.

Error Management. While SLAM methods rely on robust kernels and outlier filtering, knowledge graph systems (Zhang et al., 2024; Lu and Wang, 2025; Gil et al., 2024) have developed explicit conflict resolution and version control mechanisms. These approaches demonstrate that maintaining coherence during incremental updates requires systematic error detection beyond geometric constraints.

Incremental Scene Graph Construction. Scene graph methods (Wu et al., 2021; Gu et al., 2024; Yin et al., 2024) explore incremental semantic mapping through multi-view fusion and LLM-based relation inference. However, most lack systematic mechanisms for conflict detection and consistency maintenance during construction.

Building on SLAM's structural principles, we address logical conflicts from LLM reasoning through *version control* that records complete reasoning history and *Edge Impact Scoring* that enables targeted error localization and rollback.

Neuro-Symbolic Positioning. A reasonable question is why we do not benchmark MapRepair against traditional symbolic navigation systems. The reason is that classical symbolic systems require structured, unambiguous input (e.g., sensor readings with known noise models or formal action specifications) and cannot directly process natural-language observations. The core challenge MapRepair addresses—perception-level ambiguity in textual descriptions, including identical room names, narrative-embedded directions, and implicit transitions—is outside their scope. A fair comparison would require either pre-extracting structured spatial relations (which eliminates the perception challenge) or attaching an NLP frontend to the symbolic system (which reintroduces the same LLM errors our framework handles). MapRepair itself already occupies a **neuro-symbolic middle ground**: conflict detection, LCA computation, and Edge Impact Scoring are fully symbolic, deterministic algorithms with provable behavior, while only the perception (text→spatial relations) and repair execution leverage LLM capabilities. This design is directly inspired by SLAM principles—particularly loop closure and graph optimization—adapted to the LLM-generated graph domain.

2 Approach

2.1 Overview of the Framework

As mentioned in Sec 1, directly using LLM context to process complex textual scenarios is impractical due to context length limitations (Ding et al., 2024). We introduce an incremental graph construction approach where the LLM incrementally updates a graph by recording spatial relationships from each new observation. To ensure structural consistency in partially observable environments, we propose a modular repair framework that detects and corrects errors as they emerge during exploration.

Figure 2 illustrates the complete workflow of the LLM-MapRepair framework for spatial graph construction and repair. The framework operates cyclically to transform conflict-laden graphs into

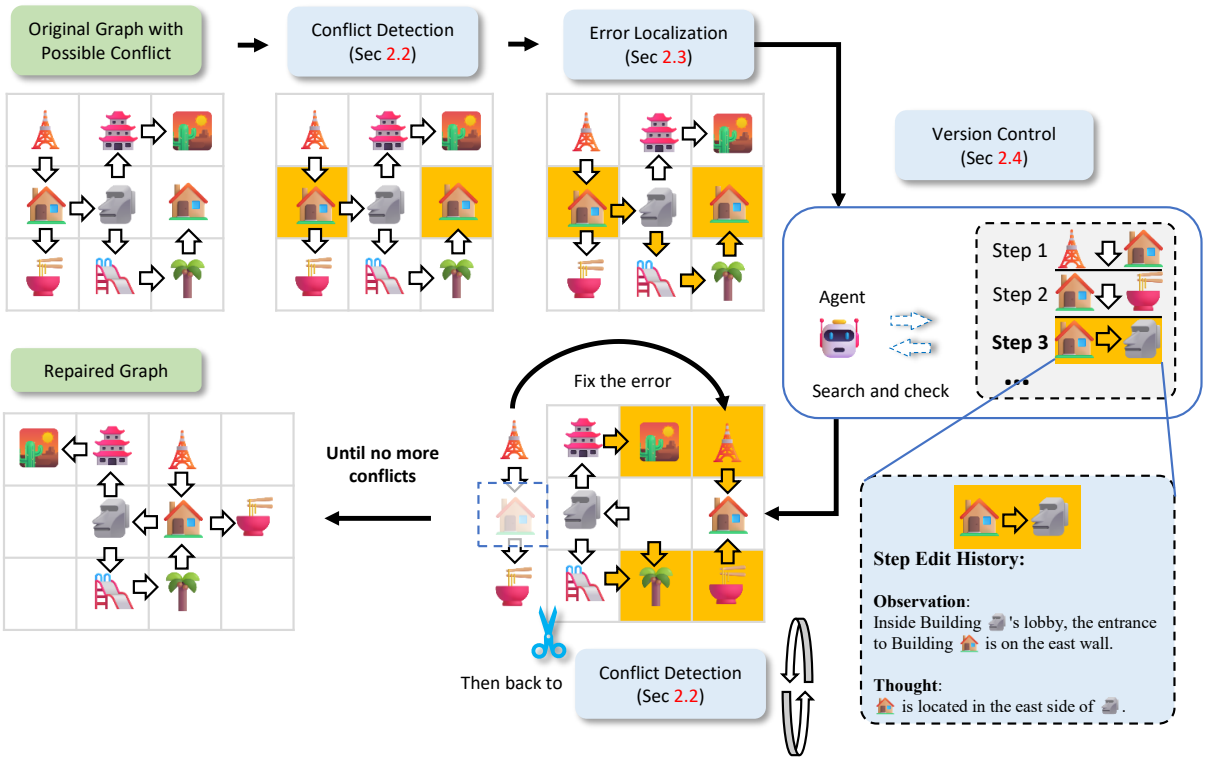


Figure 2: Overview of the LLM-MapRepair framework.

structurally consistent representations through systematic error detection and targeted corrections.

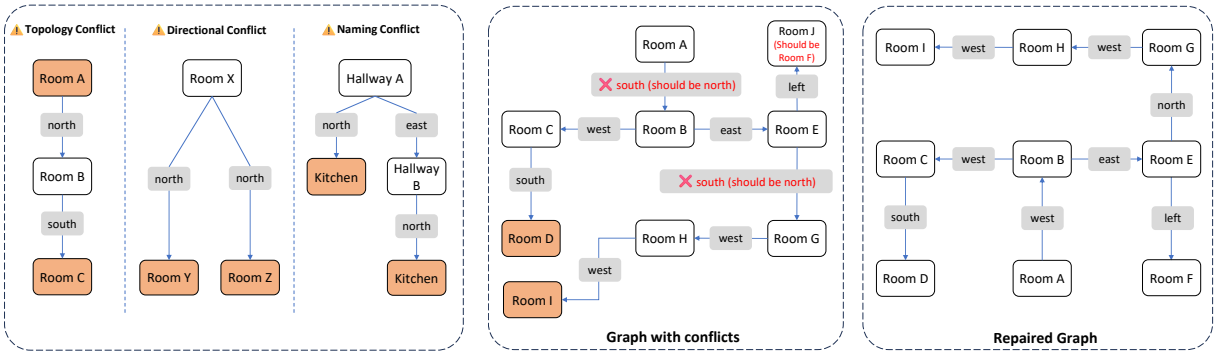
The repair process consists of three stages: (1) Conflict Detection (Sec 2.2) identifies structural inconsistencies; (2) Error Localization (Sec 2.3) employs Edge Impact Scoring to prioritize erroneous edges and trace their origins; (3) Version Control (Sec 2.4) maintains historical context, preserving observations and reasoning processes. When examining commit history, the agent accesses complete contextual information to identify flaws and apply targeted repairs. The system iterates until achieving a conflict-free graph, proving especially effective when early errors manifest much later in exploration.

2.2 Conflict Detection

As LLM agents build navigation graphs from text, inconsistencies may gradually accumulate, resulting in structural conflicts. We identify three major types—*naming*, *directional*, and *topological*—as illustrated in Figure 3(a). These three types are not arbitrary design choices but **inherent structural invariants** of any spatial navigation graph: physical exclusivity (two distinct rooms cannot occupy the same physical location), directional uniqueness (a single doorway cannot lead in two directions simultaneously), and identity uniqueness (each location has a unique name). They are domain-invariant properties of physical environments, analogous to how primary key uniqueness is a structural invariant of relational databases rather than a hand-defined rule. Our conflict detection module is modular by design: new detectors (including semantic ones) can be added as plug-in components without modifying the downstream localization or scoring pipeline, so long as they can be expressed as constraints over graph properties.

- **Topological Conflict:** Two distinct nodes are inferred to occupy the same physical position—a violation of physical exclusivity. Equivalent symptoms include cycles in tree-like spaces, unreachable nodes, or over-connected components.
- **Directional Conflict:** Multiple outgoing edges with the same direction label violating spatial constraints.
- **Naming Conflict:** Identical names assigned to different locations, causing ambiguity in reasoning and localization.

Spatial Assumption. For topological detection, we follow the MANGO benchmark's grid-based



(a) Conflict Types (b) Challenge Scenario

Figure 3: (a) Three types of structural conflicts — naming conflict, directional conflict, and topological conflict. Highlighted nodes indicate the conflicting pairs. (b) A challenging conflict scenario: a misdirected edge from Room E to Room G introduces a latent spatial misalignment. A visible topology conflict emerges later between Room D and Room I (highlighted), while correcting the edge E→G may trigger a new conflict between Room H and Room A. The edge E→J is incorrect, though it is unrelated to the current conflict.

design and adopt a **unit-distance assumption**: each cardinal action displaces the agent by exactly one spatial unit along its labeled direction. Vertical and portal-style actions (`up`, `down`, `in`, `out`, `enter`, `exit`) preserve the planar coordinate. Under this assumption, the position of every node is fully determined by its action sequence from the starting location, and two nodes mapped to the same coordinate constitute a topological conflict. In Figure 3(b), Path_1 B→C→D (south, south) places D at relative position $(0, -2)$ from B, while Path_2 B→E→G→H→I (east, south, west, south) places I at $(1-1, -1-1) = (0, -2)$. The collision at $(0, -2)$ is precisely the topological conflict our detector flags.

2.3 Error Localization

Resolving conflicts in LLM-generated navigation graphs is often more difficult than detecting them, due to several intertwined challenges illustrated in Figure 3(b). The primary complexity stems from **delayed conflicts**, where errors introduced early may not be noticed until much later in the exploration process. As demonstrated in the challenge scenario, a wrong direction from *Room E to G* leads to a cascade of misplacements, but the actual conflict only becomes apparent when the loop reaches overlapping *Rooms D and I*. This temporal gap between error introduction and detection is further complicated by **entangled conflicts**, where attempting to fix one edge can inadvertently create new conflicts elsewhere in the graph. For instance, adjusting the *Room E→G* connection resolves the initial overlap but simultaneously causes a new conflict between *Rooms H and A*. Perhaps most problematically, **silent errors** can persist undetected due to the absence of contradictory evidence. The incorrect direction from *Room E to J* exemplifies this issue, causing no immediate conflict while silently corrupting the underlying map structure.

The asynchrony between graph construction errors and structural conflicts motivate our framework's separation of **conflict detection** and **error localization**, enabling robust identification of true error sources through temporal and structural reasoning.

Once a conflict is detected, the system must identify not just the conflicting edges, but the actual root cause that introduced the inconsistency. This is non-trivial, as the erroneous edge may lie far from the observed conflict and may even appear structurally correct in isolation. Our localization pipeline proceeds in four stages: (1) identifying the minimal conflicting path pair, (2) computing their lowest common ancestor (LCA), (3) extracting divergent edges as error candidates, and (4) scoring and ranking these candidates by impact.

Minimal Conflicting Path Pair Given a structural conflict (e.g., naming or topology), we first locate two distinct paths that lead to the conflicting nodes. For instance, in Figure 3 challenge scenario, the topology conflict between Room D and Room I can be traced to two paths:

$$\text{Path}_1 : \text{Room B} \rightarrow \text{C} \rightarrow \text{D}$$
$$\text{Path}_2 : \text{Room B} \rightarrow \text{E} \rightarrow \text{G} \rightarrow \text{H} \rightarrow \text{I}$$

Both paths result in overlapping node positions, violating spatial exclusivity constraints.

Lowest Common Ancestor (LCA) To identify where the error first diverged, we compute the lowest common ancestor (LCA) of the two conflicting paths. This is the *last* node shared between them before the divergence that leads to inconsistency.

Domain Clarification. Critically, LCA is computed not over the spatial graph G (which may contain cycles and loops), but over the *Reasoning History Tree* $\mathcal{T}$—a directed acyclic graph (DAG) encoding the temporal dependency structure of edge additions during graph construction. Each node $v \in \mathcal{T}$ corresponds to a commit in the version control system and is associated with a unique timestamp $\tau(v)$ reflecting when the corresponding edge was introduced. This temporal ordering ensures that the LCA is well-defined even when the spatial graph contains cycles.

Formally, for two conflicting paths p_1 and p_2 in the spatial graph G, we trace their corresponding reasoning paths π_1, π_2 in $\mathcal{T}$ and compute:

$$\mathrm{LCA}(\pi_1, \pi_2) = \arg\max_{v \in \pi_1 \cap \pi_2} \tau(v) \tag{1}$$

where $\tau(v)$ denotes the timestamp or topological depth of node v in $\mathcal{T}$. The use of $\arg\max$ ensures we identify the *most recent* (temporally latest) common ancestor—the true divergence point where paths began to differ.

Unified Logic for All Conflict Types. This formulation provides a unified framework for both topological and directional conflicts:

- **Long-range conflicts** (topology violations): The LCA is strictly earlier than the source node of the current observation (LCA $\neq$ Source), indicating a *latent error* introduced in a previous step. In Figure 3, Room B serves as the LCA for the conflict between D and I, with the error lying along edges downstream from B.
- **Local conflicts** (directional violations): The LCA coincides with the source node itself (LCA $=$ Source), representing a degenerate case or *zero-length divergence*. This indicates an *immediate error* where the current edge directly contradicts local geometric constraints (e.g., compass directions).

This distinction proves that our LCA-based localization algorithm is a mathematically unified solution applicable to all conflict categories. Edges beyond the LCA node are considered candidate error sources for subsequent ranking.

Candidate Edge Extraction We extract the divergent subpaths from the LCA to each conflict node and collect all edges along these subpaths as potential causes of the inconsistency. In the example:

$$\text{Candidate edges} = \{\text{B} \to \text{C},\ \text{C} \to \text{D},\ \text{E} \to \text{G},\ \text{G} \to \text{H},\ \text{H} \to \text{I}\}$$

Additionally, silent errors (e.g., E→J) not yet resulting in conflicts can also be included as fallback candidates for global ranking.

Edge Scoring and Ranking To determine which candidate edge to prioritize for inspection and repair, we assign each edge a composite score based on three factors: *reachability*, *conflict count*, and *usage*. These reflect the potential structural impact, the degree of inconsistency evidence, and the reliance of observed paths on the edge, respectively.

Inspired by PageRank (Brin and Page, 1998), we model edge importance through three factors:

- **Reachability**: downstream nodes reachable from edge e, reflecting propagation potential.
- **Conflict Count**: distinct conflicts involving e, indicating contribution to inconsistencies.
- **Usage**: conflict-related paths including e, capturing empirical relevance.

We adopt an unweighted scoring function after min-max normalization:

$$\mathrm{score}(e) = \widehat{\mathrm{Reach}}(e) + \widehat{\mathrm{Conflict}}(e) + \widehat{\mathrm{Usage}}(e) \tag{2}$$

Repair Prioritization We prioritize edges by descending score to maximize conflict revelation—repairing high-impact edges first to either resolve existing conflicts or expose hidden errors. To trace an edge's origin or reverse mistaken fixes, we maintain temporal structure through Version Control.

On the Use of Heuristics. Edge Impact Score (EIS) is an empirical heuristic; it does not claim a global optimality guarantee. The absence of such a guarantee does not diminish its utility—this is a well-established principle across AI and computer science. Weighted A* (Pohl, 1970) deliberately

uses inadmissible heuristics, sacrificing optimality for speed. PageRank (Brin and Page, 1998)—which directly inspired our scoring function—has convergence guarantees for its iterative computation, but the claim that the resulting eigenvector represents "page importance" is itself a heuristic assumption with no theoretical proof of relevance (Borodin et al., 2005). Beam search (Meister et al., 2020), the dominant decoding algorithm in neural machine translation and LLM inference, has no optimality guarantee, and larger beams can paradoxically degrade quality. EIS follows this tradition: it is a principled, empirically validated heuristic designed for the target domain of LLM-generated navigation graphs, which are overwhelmingly tree-like with heterogeneous branching. On such structures, EIS achieves perfect rank correlation with cascade potential ($\rho = 1.0$, Appendix B). Even when ranking is imperfect, two safety properties hold: (i) *completeness*—LCA filtering guarantees the true error is always in the candidate set, so scoring only affects inspection order, not coverage; and (ii) *graceful degradation*—in the worst case (uniform scoring), EIS reduces to random search within the already-reduced LCA candidate set, never performing worse than random search within a smaller space.

2.4 Version Control

Version Control maintains a directed chain of versioned graph snapshots, enabling targeted rollback, difference analysis, and complete reasoning history retrieval.

Version Control Structure. Version Control is a directed chain of version records $[G_0, G_1, \ldots, G_t]$, where each commit history G_i represents a stepwise change to the graph. Rather than storing full graph snapshots, each version logs only the incremental updates. Each commit G_i represents a commit to the graph (e.g., edge additions or conflict-triggered replacements), with metadata including the step identifier, specific edge changes (+ for additions, - for removals during replacements), trigger event type, and associated observation.

$$G_i = \{\texttt{Step\_id}, \texttt{Commit}, \texttt{Trigger\_event}, \texttt{Observation\_id}, \texttt{Analysis}\}$$

This structure minimizes memory cost while enabling exact reconstructions.

Supported Operations. Version Control supports three key operations:

- `rollback_to(version)`: Restores the graph to a prior state by undoing subsequent steps.
- `recall_step(version)`: Obtain the thinking history corresponding to the step.
- `diff(`G_i`, `G_j`)`: Computes edge-level differences between two versions.

These operations support both runtime repair decisions and post hoc analysis.

Incremental Evolution. Every LLM-initiated interaction—whether through new observations or repair actions—triggers a graph update and logs a new version in Version Control. This guarantees that even failed or partially correct decisions are preserved for future analysis. Version Control records whether an update was conflict-triggered (e.g., `Trigger_event = conflict_repair`) to provide interpretability in version history. Version Control enables following key capabilities:

- **Graph alignment**: Compare versions before and after a repair to assess changes.
- **Structural diffing**: Detect which steps introduced regressions or inconsistencies.
- **Error propagation tracing**: Model how errors spread over time.

Unlike flat logs or event lists, the versioned graph design supports non-destructive rollbacks and dependency-aware repair strategies.

This aligns with well-established principles in database systems, where *write-ahead logging (WAL)* (Gray and Reuter, 1993) ensures recoverability and traceability by recording all state changes. Similarly, Version Control gives LLM-based systems a foundation for Version Control, self-debugging, and structured repair—all essential for interactive, persistent reasoning tasks.

3 Experiment

3.1 Dataset

We conduct our experiments on the MANGO benchmark (Ding et al., 2024), a curated collection of 53 interactive fiction (IF) environments originally derived from the Jericho benchmark (Hausknecht et al., 2019). Unlike Jericho,

Table 1: Comparison of Repair Method Performance

Method	Avg. Loops	Repair Rate (%)	Acc. (%)
Edge-Impact Ranking Only	**6.39**	**75.21**	44.69
Version Control Only	7.44	63.03	54.00
VC+Edge-Impact Ranking	8.20	68.91	**54.88**
Baseline (GPT-4o)	9.52	21.85	5.77

Table 2: Performance across Different Models

Model	Loops	Repair (%)	Acc. (%)
Version Control+Edge-Impact Ranking			
GPT-4o	8.20	68.91	54.88
GPT-4.1	8.28	64.71	56.49
GPT-4o-mini	9.08	58.40	56.12
Claude-Haiku	6.98	44.31	61.76
Baseline			
GPT-4o	9.52	21.85	5.77
GPT-4.1	8.98	23.05	7.32
GPT-4o-mini	9.52	15.55	5.60
Claude-Haiku	9.33	17.15	6.67

MANGO excludes non-spatial actions (e.g., "take" or "examine"), ensuring that every action directly corresponds to a location change. This design choice makes the environment ideal for evaluating the construction and repair of navigational graphs.

Each episode in MANGO consists of an agent performing a series of movement commands (e.g., "go north", "go down") based on textual observations. The resulting action-observation trajectory is used to incrementally build a topological map. However, we found that the original dataset itself contains numerous structural conflicts and non-topological actions, which requires us to first correct the graphs. The detailed steps for removing structural conflicts can be found in the appendix.

3.2 Graph Construction Process

We employ an LLM to incrementally construct navigation graphs by processing each step of the walkthrough sequentially. For each game, the LLM reads the step-by-step walkthrough and builds the navigation graph incrementally, where each action becomes an edge in the graph and each location becomes a node. The LLM only creates new edges or nodes when it determines that the current location has changed based on the textual observations.

3.3 Ablation Study

To evaluate the contribution of different components in our repair framework, we conduct an ablation study based on the conflict-prone graphs identified in Table 1. Table 2 shows the test results of baseline and our method across different models. We introduce the LLM-based graph repair loop with different tool configurations to measure their individual impact on repair performance. During the repair process, each conflict is given a maximum of 10 repair attempts. If secondary conflicts arise during the repair process, they do not consume additional repair opportunities. Throughout the entire conflict resolution process, the LLM maintains context containing historical repair information to inform subsequent decisions. In all experiments, the LLM is GPT-4o.

We compare four settings in Table 1: Edge-Impact Ranking Only, which prioritizes repair candidates by their scores; Version Control Only, which relies solely on the Version Control for history and rollback; Version Control + Edge-Impact Ranking, our full method combining both; and a Baseline without filtering or prioritization.

The table shows three key metrics for each method: *Avg. Loops* (average number of repair iterations required), *repaired* (number of conflicts successfully addressed), and *correct* (number of conflicts resolved with correct solutions).

The results in Table 1 reveal distinct performance characteristics for each repair strategy. The **Edge-Impact Ranking Only** method achieves the lowest average repair iterations (6.39 loops), requiring approximately 14% fewer iterations compared to Version Control-only and 33% fewer than the baseline approach. This efficiency stems from its focus on edge dependencies, preferring to quickly modify edges that are likely to trigger secondary conflicts, thereby reducing the total number of modifications needed. With a repair rate of 75.21%, the method successfully addresses 19% more conflicts than Version Control-only and over 240% more than the baseline. However, while Edge Impact Ranking excels at conflict resolution, its accuracy is only 44.69%—fixing conflicts does not necessarily mean the underlying graph errors are properly corrected, as conflicts and actual errors are not strongly correlated.

The **Version Control-only** approach exhibits higher average loop counts (7.44, approximately 16% more than Edge Impact Ranking) due to the

additional operations it performs, such as rollback actions and edge information queries, each consuming iteration cycles. However, Version Control's access to historical context and reasoning information recorded during edge insertion enables more accurate identification of root causes. With an accuracy of 54.00%, it achieves 21% higher accuracy than Edge Impact Ranking despite a lower repair rate of 63.03%. This is particularly effective when the candidate edge set is small, leading to more reliable corrections.

When combining both approaches (**Version Control + Edge-Impact Ranking**), we observe a synergistic effect. While the average loop count increases to 8.20 (28% higher than Edge Impact Ranking alone), the accuracy improves to 54.88%. This represents a 22.8% relative improvement in accuracy over Edge Impact Ranking alone, while maintaining a reasonable repair rate of 68.91%. The combined method leverages both structural impact analysis for efficient candidate identification and temporal context for accurate root cause analysis.

The **Baseline(GPT-4o)** without candidate filtering or prioritization mechanisms performs poorly across all metrics. With an average of 9.52 loops per repair attempt, it achieves only a 21.85% repair rate and a mere 5.77% accuracy. Most conflicts reach the maximum iteration limit without successful resolution, highlighting the critical importance of structured error localization and prioritization in graph repair tasks.

We also embed **Version Control + Edge-Impact Ranking** into other LLM modules (e.g., GPT-4o-mini, GPT-4.1) to verify the generalization of our method, please refer to supplementary material for more details.

3.4 Algorithmic Validation without LLM

To isolate and validate the core algorithmic contributions, namely LCA based candidate filtering and edge impact scoring, independently of LLM performance variability, we design a controlled experimental suite on synthetic graphs with known ground truth errors, as detailed in Appendix B. Unlike the MANGO experiments where errors arise from LLM inference, this setup directly injects topological, directional, and naming conflicts into constructed graphs, enabling precise evaluation without confounding effects from language understanding or generation quality.

Key Findings. Across six synthetic scenarios, LCA based filtering reduces the candidate edge search space by an average of 22.7%, with reductions of up to 75% in directional conflicts where the LCA coincides with the source node, corresponding to the degenerate case discussed in §2.3. Edge impact scoring achieves perfect rank correlation with true cascade potential, with Spearman $\rho = 1.0$, correctly prioritizing edges that induce secondary conflicts. Priority based inspection further accelerates high impact error discovery by a factor of 2.3 compared to random traversal, requiring 56.5% fewer edge examinations to identify critical errors. These results demonstrate that the proposed mechanisms operate as intended at the algorithmic level, independent of LLM capabilities.

4 Conclusion

Although LLMs can incrementally construct topological maps from language observations, their outputs are often noisy, accumulating misaligned edges, duplicates, and latent inconsistencies that degrade downstream reasoning. We address this with a three-stage repair pipeline comprising conflict detection, error localization, and impact-aware correction. A version-controlled map history enables rollback, difference analysis, and causal tracing of errors, while an Edge Impact Score prioritizes low-risk edits by quantifying each edge's structural and usage-based influence.

5 Limitations

Our approach has several limitations. **Continuous spatial reasoning.** The refined MANGO benchmark uses discrete cardinal directions and unit distances, which enables precise symbolic conflict detection. However, real-world novels contain continuous spatial descriptions with arbitrary angles and imprecise distances, where conflicts may be obscured by accumulated errors. While our discretization approach (binning angles into directional categories and distances into coarse ranges with elastic optimization) shows promise on literary texts such as *Dream of the Red Chamber* (Figure 1), it significantly increases false positive rates due to approximate granularity.

Semantic conflicts. Our current conflict detection relies on observable structural violations and cannot identify latent semantic inconsistencies that do not manifest as topological errors (e.g., a location described as "indoor" connecting via "across

the river" to an "outdoor" location). Recent work has shown that natural-language statements can be systematically symbolized into formal logic for consistency verification at scale—LINC (Olausson et al., 2023), Logic-LM (Pan et al., 2023), SatLM (Ye et al., 2023), and the FOLIO benchmark (Han et al., 2024)—providing a roadmap for plug-in semantic conflict detectors. Because our Version Control and LCA-based localization are conflict-type-agnostic, such a detector could be integrated without modifying the downstream repair pipeline.

Upstream perception vulnerability. Systematic, consistent perception errors (e.g., always renaming "Kitchen" as "Cooking Room") do not produce structural conflicts and therefore evade our detection. Inconsistent naming across visits is partially mitigated through our naming conflict detector and Version Control–aided merge, which lets the LLM cross-reference observation histories of similar nodes. For consistently incorrect entity identification, preprocessing with dedicated entity linking or coreference resolution modules would be the appropriate complementary mitigation.

Heuristic ranking. The Edge Impact Score, while effective empirically, lacks theoretical guarantees for optimal repair ordering in all graph configurations. Highly symmetric graphs (e.g., regular grids) and hub-and-spoke topologies with peripheral errors form structural boundary cases where reachability provides limited discriminative power. These topologies are uncommon in LLM-generated maps from sequential text exploration, but characterizing them more precisely is important future work, as is investigating provably optimal repair strategies that combine structural and semantic signals.

6 Ethical Considerations

In this paper, LLMs were employed solely as a tool for text refinement and did not contribute to the conceptualization of the work.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback. This work was supported by the Chair of Cartography and Visual Analytics at the Technical University of Munich.

References

Allan Borodin, Gareth O. Roberts, Jeffrey S. Rosenthal, and Panayiotis Tsaparas. 2005. Link analysis ranking: Algorithms, theory, and experiments. *ACM Transactions on Internet Technology (TOIT)*, 5(1):231–297.

Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117.

Mark Cummins and Paul Newman. 2008. FAB-MAP: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 27(6):647–665.

Gia Tuan Dao and Dinh Bach Vu. 2025. AlphaMaze: Enhancing large language models' spatial intelligence via GRPO. *arXiv preprint arXiv:2502.14669*.

Peng Ding, Jiading Fang, Peng Li, Kangrui Wang, Xiaochen Zhou, Mo Yu, Jing Li, Matthew R. Walter, and Hongyuan Mei. 2024. MANGO: A benchmark for evaluating mapping and navigation abilities of large language models. *Preprint*, arXiv:2403.19913.

Dorian Gálvez-López and Juan D Tardos. 2012. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197.

Jey Puget Gil, Emmanuel Coquery, John Samuel, and Gilles Gesquiere. 2024. ConVer-G: Concurrent versioning of knowledge graphs. *Preprint*, arXiv:2409.04499.

Jim Gray and Andreas Reuter. 1993. Transaction processing: Concepts and techniques. In *Morgan Kaufmann*.

Qiao Gu, Ali Kuwajerwala, Sacha Morin, Krishna Murthy Jatavallabhula, Bipasha Sen, Aditya Agarwal, Corban Rivera, William Paul, Kirsty Ellis, Rama Chellappa, and 1 others. 2024. ConceptGraphs: Open-vocabulary 3D scene graphs for perception and planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5021–5028. IEEE.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, Lucy Sun, Alexander Wardle-Solano, Hannah Szabo, Ekaterina Zubova, Matthew Burtell, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, and 11 others. 2024. FOLIO: Natural language reasoning with first-order logic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Matthew Hausknecht, Prithviraj Ammanabrolu, Côté Marc-Alexandre, and Yuan Xingdi. 2019. Interactive fiction games: A colossal adventure. *CoRR*, abs/1909.05398.

Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. 2016. Real-time loop closure in 2D LIDAR SLAM. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278.

Feng Lu and Evangelos Milios. 1997. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4:333–349.

Yuxing Lu and Jinzhuo Wang. 2025. KARMA: Leveraging multi-agent LLMs for automated knowledge graph enrichment. *Preprint*, arXiv:2502.06472.

Clara Meister, Ryan Cotterell, and Tim Vieira. 2020. If beam search is the answer, what was the question? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. 2015. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163.

Theo X. Olausson, Alex Gu, Benjamin Lipkin, Cedegao E. Zhang, Armando Solar-Lezama, Joshua B. Tenenbaum, and Roger Levy. 2023. LINC: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*.

Ira Pohl. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3-4):193–204.

Shun-Cheng Wu, Johanna Wald, Keisuke Tateno, Nassir Navab, and Federico Tombari. 2021. SceneGraphFusion: Incremental 3D scene graph prediction from RGB-D sequences. *Preprint*, arXiv:2103.14898.

Wenshan Wu, Shaoguang Mao, Yadong Zhang, Yan Xia, Li Dong, Lei Cui, and Furu Wei. 2024. Mind's eye of LLMs: Visualization-of-thought elicits spatial reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Sirui Xia, Aili Chen, Xintao Wang, Tinghui Zhu, Yikai Zhang, Jiangjie Chen, and Yanghua Xiao. 2025. Can LLMs learn to map the world from local descriptions? *Preprint*, arXiv:2505.20874.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2023. SatLM: Satisfiability-aided language models using declarative prompting. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Hang Yin, Xiuwei Xu, Zhenyu Wu, Jie Zhou, and Jiwen Lu. 2024. SG-Nav: Online 3D scene graph prompting for LLM-based zero-shot object navigation. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Anlong Zhang and Jianmin Ji. 2025. Research on navigation methods based on LLMs. *arXiv preprint arXiv:2504.15600*.

Ningyu Zhang, Zekun Xi, Yujie Luo, Peng Wang, Bozhong Tian, Yunzhi Yao, Jintian Zhang, Shumin Deng, Mengshu Sun, Lei Liang, Zhiqiang Zhang, Xiaowei Zhu, Jun Zhou, and Huajun Chen. 2024. OneEdit: A neural-symbolic collaboratively knowledge editing system. *Preprint*, arXiv:2409.07497.

A Dataset Refinement Details

A.1 Dataset Refinement

We analyze these trajectories and find that the MANGO dataset itself contains structural inconsistencies, which we categorize into *directional conflicts*, and *topological conflicts*.

Original Dataset Conflicts. We visualize the distribution of detected conflicts across all environments in Appendix Figure A1. All 18 games in the original MANGO dataset (Ding et al., 2024) contain various types of conflicts, including many non-topological actions such as "pray" that do not correspond to spatial movement.

Conflict-Free Dataset Creation. To better evaluate the performance of our MapRepair system, we create a refined dataset that eliminates pre-existing conflicts and retains only topologically meaningful navigation actions. Our dataset refinement process follows a systematic 5-step pipeline:

1. **Action Filtering**: Filter the original dataset to retain only 14 topological movement actions (north, south, east, west, up, down, northeast, northwest, southeast, southwest, in, out, enter, exit), removing non-spatial actions like "pray".

2. **Directional Conflict Resolution**: Remove duplicate edges when the same source node has multiple outgoing edges with identical direction labels. We preserve the edge with the minimum step count and remove all other duplicates.

3. **Topological Conflict Resolution**: Eliminate inconsistent reverse edges when bidirectional connectivity violates spatial symmetry. Edges that create directional mismatches in their reverse direction are removed.

4. **Reverse Edge Conflict Resolution**: Remove original edges that would cause directional

conflicts when their corresponding reverse edges are added to maintain graph symmetry.

5. **Naming Conflict Resolution**: Eliminate edges that cause indirect conflicts through transitive spatial relationships. When different paths lead to inconsistent positional inferences for the same location, edges in the indirect paths are removed.

6. **Self-Loop Removal**: Delete all self-referential edges where a node points to itself, as these do not represent meaningful spatial transitions.

This refinement process removes a total of 160 edges, reducing the edge count from 1,673 to 1,513. The resulting conflict-free dataset (`data_fixed`) contains edges that represent valid topological relationships with clear spatial semantics.

From Grid-Based Games to Real-World Novels. Our refined MANGO dataset uses standardized cardinal directions (north, south, east, west, etc.) and unit distances, which simplifies conflict detection through discrete symbolic matching. However, real-world novels present continuous spatial descriptions with arbitrary angles and imprecise distances. In such settings, conflicts may be obscured by accumulated continuous errors, and novels often lack precise angular or distance information altogether.

To bridge this gap, we employ a discretization approach: angles are binned into finite directional categories, and distances are partitioned into coarse ranges (e.g., near, medium, far). We further apply elastic optimization based on conflict severity, allowing tolerance thresholds to adapt dynamically. Under appropriate configurations, this approach improves mapping success for real novels. For instance, Figure 1 demonstrates our system's performance on chapters 16–17 of *Dream of the Red Chamber*, achieving node recall of 94.3% (an 8.6 percentage point improvement) and edge recall of 88.2% (a 55.8 percentage point improvement). However, the discretization strategy also increases false positive rates, as approximate binning may introduce spurious conflicts or miss genuine errors masked by coarse granularity.

Topological Conflicts. These occur when an edge violates expected spatial symmetry or consistency. For example, in dataset `zork2`, a topological conflict is triggered when the system observes `ledge in ravine - down -> deep ford`, whereas the expected reverse transition from earlier was `deep ford - north -> ledge in ravine`. Based on this prior observation, the forward edge should have been labeled `south`, not `down`.

Directional Conflicts. These arise when a node has multiple outgoing edges with the same direction label. For example, in dataset `zork2`, the node `carousel room` has two edges labeled "north", pointing to both `marble hall` and `topiary`. This violates the uniqueness constraint of directional navigation in deterministic environments.

Appendix Table A1 presents the conflicts generated during the graph construction process across all games without the repair loop activated. The table shows the distribution of directional, topological, and naming conflicts for each game environment.

B Algorithmic Validation Experiments

To validate the core algorithmic mechanisms—LCA-based candidate filtering (§2.3) and edge impact scoring (§2.3)—independent of LLM performance, we design a controlled experimental suite using synthetic graphs with injected errors. This approach enables precise measurement of algorithmic effectiveness with known ground truth, eliminating confounding factors from language model variability.

B.1 Experimental Design

Motivation. To isolate algorithmic contributions independent of LLM variability, we test on synthetic graphs with known error injections.

Methodology. We construct spatial graphs programmatically and inject specific errors at predetermined locations, creating scenarios that mirror the three conflict types identified in §3: *topological conflicts* (overlapping node positions), *directional conflicts* (duplicate direction labels), and *naming conflicts* (identical names at different positions). Each test case includes:

- A spatial graph $G = (V, E)$ with known correct structure
- Injected error edges $E_{err} \subset E$ with documented positions and types
- Expected conflicts C that should be detected

- Ground-truth candidate sets for error localization

Evaluation Metrics. We measure:

- **Candidate reduction rate**: $r = 1 - |E_{LCA}|/|E|$, where E_{LCA} are candidates after LCA filtering
- **Error ranking**: Position of true error $e \in E_{err}$ in score-ranked candidate list
- **Inspection speedup**: Ratio of edges examined (random vs. priority-based traversal) to find high-impact errors
- **Cascade prediction accuracy**: Spearman correlation between edge scores and actual downstream impact

B.2 Test Cases

We design six synthetic scenarios covering diverse error patterns:

TC1: Topological Conflict (Paper Figure Scenario). Reconstructs the challenge scenario from Figure 3: A misdirected edge $E \to G$ causes rooms D and I to overlap. The graph contains 9 edges across 11 nodes. *Expected behavior*: LCA correctly identifies room B as the divergence point; candidate set reduces from 9 to 8 edges (11.1%); error edge $E \to G$ ranks 3rd by impact score.

TC2: Directional Conflict (Degenerate Case). Node C has duplicate "north" edges to both D and E, violating spatial uniqueness. This tests the degenerate case where LCA = Source (§2.3). Graph: 7 edges, 8 nodes. *Expected behavior*: LCA coincides with node C; 14.3% candidate reduction; error edge $C \to E$ ranks 3rd.

TC3: Cascading Secondary Conflicts. A single error edge $R_1 \to R_2$ (incorrect position) triggers two downstream conflicts: R_2 overlaps with L_2, and R_3 overlaps with L_3. Graph: 9 edges, 13 nodes. *Expected behavior*: Both conflicts identify $R_1 \to R_2$ as root cause; 22.2% reduction; error ranks 3rd in both conflict analyses.

TC4: Mixed Conflict Types. Combines topological and directional conflicts in a single graph with 12 edges and 12 nodes. Tests whether LCA-based localization maintains effectiveness across heterogeneous conflict types. *Expected behavior*: Directional conflicts show higher reduction (75%) due to local nature (LCA = Source); topological conflicts achieve 25% reduction.

TC5: Long-Range Conflict. Error introduced at depth 2 (edge $N_1 \to N_2$) causes conflict at depth 10 between M_7 and N_9. Tests LCA's ability to trace back through long paths. Graph: 18 edges, 19 nodes. *Expected behavior*: LCA correctly backtracks to N_0; all edges on divergent path included (0% reduction due to long-range propagation); error still ranks 3rd despite large candidate set.

TC6: Cascade Potential Prediction. Purpose-built graph with 36 edges and 5 injected errors of varying cascade severity: $E \to F$ (10 downstream nodes), $J \to K$ (5 nodes), $L \to M$ (3 nodes), $N \to O$ (2 nodes), $P \to Q$ (1 node). Tests edge scoring's ability to predict secondary conflict potential. *Expected behavior*: Perfect rank correlation between reachability scores and actual cascade sizes.

B.3 Results

LCA-Based Candidate Filtering. Table 3 summarizes reduction rates across all test cases. The average candidate reduction is **22.7%**, with a standard deviation of 24.1% reflecting the diversity of conflict types and graph structures.

Test Case	Total Edges	LCA Cands.	Reduct. Rate	Error Rank
TC1: Topo	9	8	11.1%	3/8
TC2: Dir	7	6	14.3%	3/6
TC3: Cascade	9	7	22.2%	3/7
TC4: Mix(T)	12	9	25.0%	5/9
TC4: Mix(D)	12	3	**75.0%**	3/3
TC5: Long	18	18	0.0%	3/18
Avg.	10.9	8.4	**22.7%**	—

Table 3: LCA-based candidate filtering. "Cands.": edges after filtering. "Rank": error position in ranked candidates.

Key observations: (1) *Directional conflicts* exhibit the highest reduction (75%) because LCA = Source creates zero-length divergence, limiting candidates to immediately adjacent edges. (2) *Long-range conflicts* show 0% reduction when the entire graph lies on divergent paths, but LCA still provides correct localization scope. (3) *True errors consistently rank in top 3* candidates across all scenarios, demonstrating that edge impact scoring effectively prioritizes ground-truth errors even with imperfect filtering.

Edge Scoring and Cascade Prediction. For TC6, we compute edge scores using the composite formula from Eq. 2: $\text{score}(e) = \widehat{\text{Reach}}(e) + \widehat{\text{Conflict}}(e) + \widehat{\text{Usage}}(e)$, where each component is

min-max normalized. Table 4 compares predicted rankings (by score) with actual cascade impact.

Error Edge	Reach Score	Actual Cascade	Rank Match
$E \to F$	10	10 nodes	1 / 1
$J \to K$	5	5 nodes	2 / 2
$L \to M$	3	3 nodes	3 / 3
$N \to O$	2	2 nodes	4 / 4
$P \to Q$	1	1 node	5 / 5
Spearman correlation: $\rho = 1.0$ ($p < 0.001$)			

Table 4: Edge scoring correlation with cascade potential. "Reach Score" = reachability count; "Actual Cascade" = downstream nodes affected by error; "Rank Match" = predicted rank / true rank.

The perfect Spearman correlation ($\rho = 1.0$) confirms that reachability-based scoring accurately predicts which errors will trigger larger cascades. This validates the PageRank-inspired heuristic (§2.3) that structural influence correlates with error propagation.

Inspection Speedup. We simulate two error discovery strategies on TC6:

- *Random traversal*: Examine edges in random order until all high-impact errors (cascade $\geq$ 5) are found.
- *Priority-based traversal*: Examine edges in descending score order.

Results: Random strategy requires examining **23 edges** (63.9% of graph) before discovering both high-impact errors ($E \to F$ and $J \to K$). Priority-based strategy finds them in **10 edges** (27.8%), achieving a **2.3× speedup** and reducing inspections by 56.5%. Furthermore, priority-based inspection reaches 80% of total error impact after examining only 17 edges (47.2%), compared to 31 edges (86.1%) for random traversal—a **1.82× acceleration** in high-impact error detection.

B.4 Discussion

These controlled experiments validate three key algorithmic properties:

1. LCA-Based Filtering Reduces Search Space. The 22.7% average reduction demonstrates tangible efficiency gains, particularly pronounced (75%) for directional conflicts where the degenerate case (LCA = Source) applies. Even in worst-case scenarios (long-range conflicts with 0% reduction), LCA correctly delimits the error scope to divergent subpaths rather than the entire graph.

2. Edge Scoring Predicts Cascade Potential. Perfect rank correlation ($\rho = 1.0$) between scores and actual downstream impact confirms that the composite scoring function (reachability + conflict count + usage) effectively captures structural influence. This validates the claim (§2.3) that edges with high reachability are more likely to trigger secondary conflicts when erroneous.

3. Prioritization Accelerates High-Impact Error Discovery. The 2.3× inspection speedup and 56.5% examination reduction demonstrate practical benefits: by examining high-scoring edges first, the repair system can identify critical errors earlier, reducing wasted effort on low-impact candidates. This is especially valuable in interactive debugging scenarios where human-in-the-loop validation is expensive.

Generalization to Real LLM Scenarios. While these synthetic experiments use injected errors, the structural patterns mirror real LLM-generated conflicts observed in MANGO (§4): delayed error manifestation (TC5), cascading propagation (TC3), and mixed conflict types (TC4). The algorithmic validation confirms that LCA and edge scoring function as designed at a fundamental level, independent of whether errors originate from LLM inference or manual injection. The MANGO experiments then demonstrate that these mechanisms successfully transfer to realistic, LLM-generated graphs with all the additional complexity of natural language grounding.

C Computational Complexity Analysis

We provide a per-component complexity analysis of the MapRepair pipeline. Let $|V|$ denote the number of nodes, $|E|$ the number of edges, ΔG_i the set of edge changes recorded in commit i, and P_1, P_2 the two root-to-conflict-node paths used in LCA computation.

Memory. Version Control uses incremental edge-level diffs (additions / removals) rather than full graph snapshots. Total memory is therefore $\mathcal{O}(\sum_i |\Delta G_i|)$ across all commits, rather than $\mathcal{O}(|V| \times T)$ for storing T complete snapshots of a $|V|$-node graph. For a typical MANGO game (mean 28.5 edges, $\sim$ 10 repair iterations), this amounts to fewer than 500 lightweight version records.

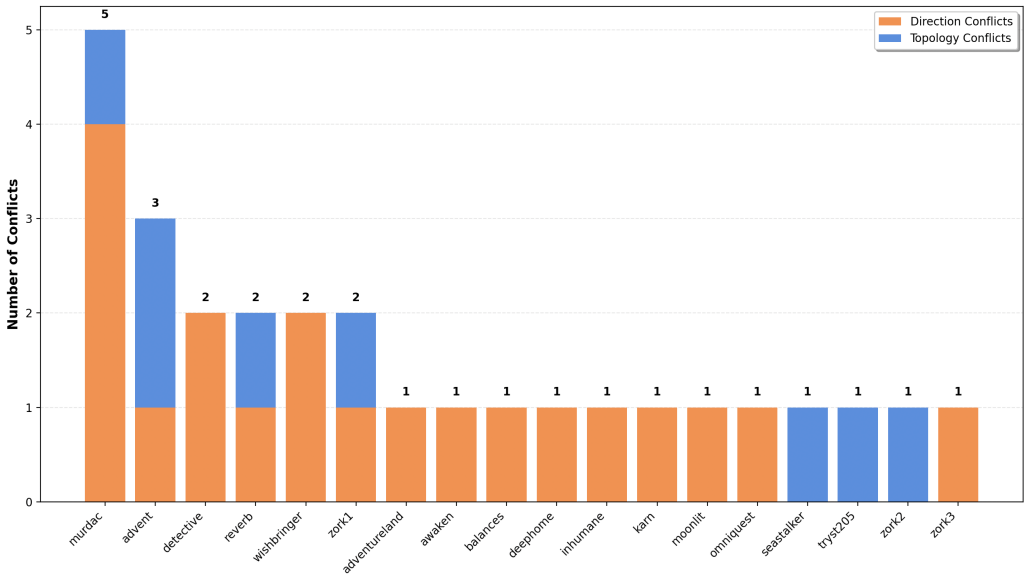


Figure A1: Structural conflict distribution in MANGO environments. Blue segments represent topological conflicts; orange segments represent directional conflicts. Naming conflicts are not observed.

Per-component time. Asymptotic complexity at MANGO scale (mean 17.5 nodes, 28.5 edges per game):

Scaling to larger environments. For environments with hundreds of nodes, the BFS-based reachability computation in EIS is the primary bottleneck. Two practical mitigations: (i) *Incremental reachability updates*—only recompute reachability for subgraphs affected by the modified edge, rather than recomputing globally. (ii) *Approximate scoring with lazy cache invalidation*—cache reachability counts and invalidate only on conflict-triggered modifications. Either mitigation reduces the amortized EIS cost to $\mathcal{O}(|\Delta E| \times |V|)$, where $|\Delta E|$ is the number of edges modified per cycle.

D Refined Dataset Statistics

We provide a comprehensive statistical characterization of our refined MANGO dataset to clarify its scale, structural complexity, and textual challenges.

Graph structure. Across all 53 games, the refined dataset contains 930 unique locations and 1,511 directed edges, traversed over 13,096 walkthrough steps. Per-game statistics are summarized in Table A5. Graphs are sparse and overwhelmingly tree-like (mean density 0.127, mean degree 1.59), consistent with the sequential exploration pattern of interactive fiction.

Edge actions. The 1,511 edges are distributed across 14 distinct movement actions: 75.4% cardinal (north/south/east/west, roughly evenly split at ∼18–20% each), 11.4% vertical (up/down), 12.0% diagonal (NE/NW/SE/SW), and 1.2% non-cardinal (in/out/enter/exit). 71.5% of edges (1,081/1,511) are observed traversed in both directions during the walkthrough; the remaining 28.5% (430/1,511) are one-way. 52 of 53 games contain at least one one-way edge.

Edge attributes. Each edge stores: source node, destination node, action label, first forward traversal step, first reverse traversal step, and minimum step count. No additional attributes (distance weights, semantic tags) are maintained—edges represent purely topological connections under the unit-distance assumption.

Textual complexity. Observation strings range from structured format (location header, description, exit list) to highly narrative prose. Specific challenges that complicate LLM-based extraction include: (i) *identical room names*—5 distinct "Wharf Road" nodes in `cutthroat`, 5 "Outside" and 4 "Hallway" in `detective`; (ii) *dark rooms*, where the destination identity is only revealed after a subsequent illumination action (e.g., `light lamp`); (iii) *implicit transitions* embedded in non-movement actions (e.g., `open window` silently relocates the agent in `zork1`); and (iv) *blocked passages* described in natural language (e.g., "passages in all directions, several blocked by cave-ins") that require the LLM to dis-

Table A1: Game Graph Analysis Results

Game Name	Nodes	Edges	Direction Conflicts	Topology Conflicts	Naming Conflicts	Total Conflicts
905	6	10	0	0	0	0
advent	33	69	1	0	0	1
adventureland	21	52	9	0	0	9
afflicted	12	26	3	0	0	3
anchor	23	46	0	0	0	0
awaken	15	34	3	0	0	3
balances	10	22	4	0	0	4
ballyhoo	19	42	3	0	0	3
curses	14	32	2	0	0	2
cutthroat	26	68	15	1	72	88
deephome	28	54	1	0	0	1
detective	27	54	3	0	8	11
dragon	26	52	0	0	0	0
enchanter	24	52	3	0	0	3
enter	14	28	1	0	0	1
gold	14	27	3	0	0	3
hhgg	8	12	0	0	0	0
hollywood	14	28	2	0	0	2
huntdark	8	12	0	0	0	0
infidel	25	50	0	0	0	0
inhumane	30	65	4	1	8	13
jewel	19	39	4	0	0	4
karn	17	38	3	0	0	3
library	8	16	2	0	0	2
loose	12	22	0	0	0	0
lostpig	6	10	0	0	0	0
ludicorp	23	44	0	0	0	0
lurking	14	26	0	0	0	0
moonlit	6	10	0	0	0	0
murdac	33	71	12	1	24	37
night	21	50	6	0	0	6
omniquest	29	60	0	0	0	0
partyfoul	6	14	2	0	0	2
pentari	16	34	3	0	0	3
planetfall	24	48	1	0	8	9
plundered	21	40	1	0	0	1
reverb	16	37	7	1	0	8
seastalker	18	34	1	0	0	1
sherlock	19	45	5	0	0	5
snacktime	4	8	1	0	0	1
sorcerer	26	51	3	0	0	3
spellbrkr	18	33	0	0	0	0
spirit	21	44	2	0	0	2
temple	20	40	3	0	0	3
trinity	14	26	1	0	0	1
tryst205	8	14	2	0	0	2
wishbringer	26	54	2	0	0	2
yomomma	9	22	0	0	0	0
zenon	14	26	1	0	0	1
zork1	19	44	6	0	0	6
zork2	26	59	7	0	0	7
zork3	24	53	4	0	0	4
ztuu	18	36	2	0	0	2

criminate traversable from non-traversable exits. All statistics are computed programmatically from the refined dataset and are fully reproducible.

E Error Source Taxonomy

Errors observed in LLM-based incremental graph construction arise from two intertwined sources. In practice the two are often inseparable, since distinguishing them requires access to the ground truth—which is precisely the information unavailable during construction. MapRepair handles both uniformly: regardless of origin, any structural conflict triggers the same detect→localize→repair pipeline.

Source 1: LLM hallucination (parsing errors). The LLM generates spatial relationships not supported by the source text. The text contains sufficient information to avoid the error, but the model fails to extract it correctly. *Example* (Zork I, step 30): the observation states "*This is a circular stone room with passages in all directions. Several of them have unfortunately been blocked by cave-ins.*" Ground truth has only 2 traversable exits (southeast and west), but the LLM may create edges in all cardinal directions, ignoring the "blocked by cave-ins" qualifier.

Source 2: Textual abstraction (genuine information gaps). The source text genuinely lacks the information needed for correct graph construction.

Table A2: Game Graph Analysis Results with Repair Methods

Game Name	Total Conflicts	Edge-Impact Ranking Only			Version Control Only			Version Control + Edge-Impact Ranking (Ours)			Baseline		
		Avg Loops	Repaired (#)	Correct (#)	Avg Loops	Repaired (#)	Correct (#)	Avg Loops	Repaired (#)	Correct (#)	Avg Loops	Repaired (#)	Correct (#)
905	0	-	-	-	-	-	-	-	-	-	-	-	-
advent	1	6.0	1	0	7.0	1	1	8.0	1	1	2.0	1	0
adventureland	9	6.67	8	4	7.78	6	3	8.44	7	4	9.56	2	0
afflicted	3	6.0	3	1	7.33	2	1	8.0	2	1	9.67	1	0
anchor	0	-	-	-	-	-	-	-	-	-	-	-	-
awaken	3	6.33	2	1	7.0	2	1	8.33	2	1	9.33	2	0
balances	4	6.25	3	1	7.25	3	2	8.0	3	2	9.5	1	0
ballyhoo	3	6.67	2	1	7.67	2	1	8.33	2	1	9.67	1	0
curses	2	6.0	2	1	7.0	2	1	8.5	2	1	9.5	1	0
cutthroat	88	6.36	58	26	7.64	48	22	8.45	52	29	9.55	3	0
deephome	1	6.0	1	0	7.0	1	1	8.0	1	1	10.0	1	0
detective	11	6.55	7	3	7.64	6	3	8.36	6	3	9.45	2	1
dragon	0	-	-	-	-	-	-	-	-	-	-	-	-
enchanter	3	6.33	2	1	7.33	2	1	8.0	2	1	9.67	1	0
enter	1	6.0	1	0	7.0	1	1	8.0	1	1	10.0	1	0
gold	3	6.67	2	1	7.67	2	1	8.33	2	1	9.33	1	0
hhgg	0	-	-	-	-	-	-	-	-	-	-	-	-
hollywood	2	6.5	2	1	7.5	2	1	8.0	2	1	9.5	1	0
huntdark	0	-	-	-	-	-	-	-	-	-	-	-	-
infidel	0	-	-	-	-	-	-	-	-	-	-	-	-
inhumane	13	6.46	8	4	7.69	7	3	8.31	7	4	9.54	2	1
jewel	4	6.5	3	1	7.5	3	2	8.25	3	2	9.5	1	0
karn	3	6.33	2	1	7.33	2	1	8.0	2	1	9.67	1	0
library	2	6.0	2	1	7.0	2	1	8.5	2	1	9.5	1	0
loose	0	-	-	-	-	-	-	-	-	-	-	-	-
lostpig	0	-	-	-	-	-	-	-	-	-	-	-	-
ludicorp	0	-	-	-	-	-	-	-	-	-	-	-	-
lurking	0	-	-	-	-	-	-	-	-	-	-	-	-
moonlit	0	-	-	-	-	-	-	-	-	-	-	-	-
murdac	37	6.46	24	11	7.57	19	9	8.32	21	12	9.51	3	0
night	6	6.5	4	2	7.67	3	2	8.33	4	2	9.5	2	0
omniquest	0	-	-	-	-	-	-	-	-	-	-	-	-
partyfoul	2	6.5	2	1	7.5	2	1	8.0	2	1	9.5	1	0
pentari	3	6.67	2	1	7.67	2	1	8.33	2	1	9.33	1	0
planetfall	9	6.44	6	3	7.56	5	2	8.33	5	3	9.44	2	0
plundered	1	6.0	1	0	7.0	1	1	8.0	1	1	10.0	1	0
reverb	8	6.5	5	2	7.75	4	2	8.38	4	2	9.5	2	0
seastalker	1	6.0	1	0	7.0	1	1	8.0	1	1	10.0	1	0
sherlock	5	6.4	3	1	7.6	3	2	8.4	3	2	9.6	2	1
snacktime	1	6.0	1	0	7.0	1	1	8.0	1	1	10.0	1	0
sorcerer	3	6.33	2	1	7.33	2	1	8.0	2	1	9.67	1	0
spellbrkr	0	-	-	-	-	-	-	-	-	-	-	-	-
spirit	2	6.5	2	1	7.5	2	1	8.0	2	1	9.5	1	0
temple	3	6.67	2	1	7.67	2	1	8.33	2	1	9.33	1	0
trinity	1	6.0	1	0	7.0	1	1	8.0	1	1	10.0	1	0
tryst205	2	6.5	2	1	7.5	2	1	8.0	2	1	9.5	1	0
wishbringer	2	6.5	2	1	7.5	2	1	8.0	2	1	9.5	1	0
yomomma	0	-	-	-	-	-	-	-	-	-	-	-	-
zenon	1	6.0	1	0	7.0	1	1	8.0	1	1	10.0	1	0
zork1	6	6.5	4	2	7.67	3	2	8.33	3	2	9.5	2	0
zork2	7	6.43	4	2	7.71	4	2	8.29	4	2	9.43	2	0
zork3	4	6.5	3	1	7.5	3	2	8.25	3	2	9.5	1	0
ztuu	2	6.5	2	1	7.5	2	1	8.0	2	1	9.5	1	0

Even a perfect reasoner cannot extract the right edge from the local context alone.

(2a) Dark rooms (Zork I, steps 14–15). The observation at step 14 says only "*You have moved into a dark place. It is pitch black.*" The destination (Attic) is revealed only at step 15 after the action `light lamp`. The MANGO ground truth itself acknowledges this by recording `seen_in_forward_answerable: 15` (not 14) for this edge.

(2b) Identical room names (Cutthroat, steps 15–19). Five physically distinct locations all display as "Wharf Road," differentiated only by subtle contextual cues (e.g., "with the McGinty Salvage office to the south" vs. "with Outfitters International to the south"). Disambiguating these requires fine-grained contextual parsing beyond standard spatial extraction.

(2c) Implicit transitions (Zork I, step 7). The action `open window` (not a movement command) produces "*Standing at the 'behind house' location...*" The player has implicitly moved from "North of House" to "Behind House" via a non-movement action embedded in narrative prose; the corresponding step has no edge in the gold graph.

F Repair Prompt Templates

Both Baseline and our method use identical conflict detection and an identical maximum of 10 repair attempts per conflict. The two systems differ only in the information passed to the LLM at repair time: the Baseline sees the current graph and the conflict, whereas our method additionally sees LCA-filtered candidate edges, their EIS ranks, and per-candidate Version Control records.

Baseline prompt.

> *You are repairing an incrementally constructed navigation graph. The current graph contains the following nodes and edges: `<graph>`. We have detected the following structural conflict: `<conflict_description>`. Identify the erroneous edge and propose a fix (delete, change direction, or merge nodes). Respond*

in JSON: {"action": ..., "edge": ..., "rationale": ...}.

Our prompt (VC + EIS).

You are repairing an incrementally constructed navigation graph. The current graph contains the following nodes and edges: <graph>. A structural conflict has been detected: <conflict_description>. LCA-based localization narrowed the candidate erroneous edges to: <candidate_list>, ranked by Edge Impact Score:

1. <edge_1> EIS = <s_1>
2. <edge_2> EIS = <s_2>
...

For each candidate, you may invoke recall_step(version) to retrieve the original observation and your prior reasoning at the time the edge was added:

recall_step(<v_1>) → <obs_1>, analysis: <a_1>
recall_step(<v_2>) → <obs_2>, analysis: <a_2>
...

Inspect candidates in EIS-rank order. For each, decide CORRECT (skip) or ERROR (apply fix). When you propose a fix, respond in JSON: {"action": ..., "edge": ..., "new_direction": ..., "rationale": ...}.

G Case Studies

We provide two complementary case studies that together exercise the full MapRepair pipeline. Case A illustrates the value of Version Control when EIS scores are tied (a local, "immediate" conflict). Case B exhibits the long-range delayed-manifestation pattern that motivates the entire framework, and shows how EIS and VC act in concert.

G.1 Case Study A: Local Conflict — VC-Driven Repair (Zork I, Steps 14–15)

Setup. During incremental construction of the Zork I navigation graph, the LLM builds 7 nodes and 9 edges through step 13 without conflict. At step 14, the agent issues `up` from *Kitchen*; the observation reads "*You have moved into a dark place. It is pitch black. You are likely to be eaten by a grue.*" Unable to identify the destination, the LLM creates a placeholder node "Dark Place" and the edge `Kitchen`–*up*–`Dark Place`. Version Control records:

```
C9: +[Kitchen -up-> Dark Place]
Trigger: observation
Observation: "pitch black..."
Analysis: "location name
unavailable due to darkness"
```

Conflict Trigger. At step 15 the agent issues `light lamp`; the observation reads "*Attic. This is the attic. The only exit is a stairway leading down.*" The LLM creates `Kitchen`–*up*–`Attic`, recorded as commit C10. *Kitchen* now has two outgoing `up` edges to different destinations—a directional conflict.

Localization & Scoring. LCA = Kitchen (the source itself; degenerate case from §2.3). The candidate set is {`Kitchen`→`Dark Place`, `Kitchen`→`Attic`}. Both candidates are leaf edges with identical reachability, conflict count, and usage; their EIS scores are tied.

Repair via Version Control. Because EIS cannot break the tie, the LLM falls back to Version Control. `recall_step(C9)` returns the "pitch black" observation, indicating that "Dark Place" was a provisional name assigned under darkness; `recall_step(C10)` returns the explicit "Attic" label observed under illumination. The LLM concludes that the two edges denote the same physical destination and merges "Dark Place" into "Attic." Post-repair, *Kitchen* has a single `up` edge to *Attic*; no remaining conflicts.

Takeaway. When the structural signal is uninformative, observation-level evidence preserved by Version Control provides the decisive cue. Without construction history, both candidate edges appear equally valid from the current graph state alone.

G.2 Case Study B: Long-Range Conflict — EIS+VC Synergy

Setup. We construct a representative scenario mirroring the structural patterns observed in MANGO games (sparse, tree-like, heterogeneous branching). A 9-node environment is explored from *Entrance Hall* along two divergent paths:

```
Path 1: Entrance Hall -south->
Corridor -east-> Lab
Path 2: Entrance Hall -east->
Office -east-> Meeting Room
        -south-> Break Room
-south-> Storage
        -east-> Archive -south->
Vault
```

Step 5 — Silent Error. The observation reads "*Leaving the office, you head down the hallway to the meeting room.*" The LLM maps "down the hallway" to direction *south*; the ground-truth label is *east*. *Meeting Room* is placed at coordinate $(1, 1)$ instead of $(2, 0)$. No conflict is detected at this

step—the position $(1, 1)$ does not yet collide with any existing node.

Steps 7–13 — Inherited Misplacement. *Break Room*, *Storage*, *Archive*, and *Vault* are added downstream of *Meeting Room* and inherit its positional shift. The error remains structurally invisible.

Step 20 — Conflict Manifests. The edge `Corridor`–*east*–`Lab` is added; *Lab* is placed at $(1, 1)$. *Meeting Room* is already there. A topological conflict between *Lab* and *Meeting Room* is detected. The error was introduced at step 5; the conflict surfaces at step 20—a **15-step temporal gap**, exemplifying delayed manifestation.

Localization. Path to *Lab*: `Entrance Hall`→`Corridor`→`Lab`. Path to *Meeting Room*: `Entrance Hall`→`Office`→`Meeting Room`. LCA = *Entrance Hall*. The divergent subpaths yield 4 candidate edges out of 8 total (50% search-space reduction).

Scoring. Conflict count $\widehat{C}$ and usage $\widehat{U}$ are uniform across the LCA-filtered candidates, so ranking is determined entirely by reachability $\widehat{R}$. Path 2 has a long downstream chain (5 descendants of *Meeting Room*), giving the true error edge a high reachability score. EIS ranks the true error at **position 2 of 4** (Table A6).

Repair.

Attempt 1 (rank #1): `Entrance Hall` → `Office` (step 3). Version Control returns the original observation "*You walk east from the entrance hall into a spacious office.*" The directional cue "walk east" matches the labeled direction. **Verdict: CORRECT → SKIP.**

Attempt 2 (rank #2): `Office` → `Meeting Room` (step 5). Version Control returns "*Leaving the office, you head down the hallway to the meeting room.*" The phrase "down the hallway" is spatially ambiguous; given that the office faces east, "down the hallway" is more consistent with east than south. **Verdict: ERROR → apply fix (south → east).**

Outcome. A single repair on the high-reachability edge restores all 5 downstream nodes to their ground-truth positions, eliminating the topological conflict. The system locates the error in 2 inspections out of 4 LCA-filtered candidates—versus 4.5 expected under random ordering without LCA filtering (a 56% reduction in inspection count).

Takeaway. Neither component alone suffices. EIS alone would surface the top-ranked edge (innocent `Entrance Hall`→`Office`) for repair without historical context, risking erroneous modification of a structurally important but correct edge. Version Control alone, lacking a prioritization signal, would examine all 4 candidates in arbitrary order (2.5 attempts in expectation). EIS+VC together: EIS narrows the search space and orders candidates by structural impact; VC supplies the observation-level evidence needed to confirm or refute each candidate.

Table A3: Game Graph Analysis Results Comparison - Version Control + Edge-Impact Ranking

Game Name	Total Conflicts	GPT-4o			GPT-4.1			GPT-4o-mini		
		Avg Loops	Repaired (#)	Correct (#)	Avg Loops	Repaired (#)	Correct (#)	Avg Loops	Repaired (#)	Correct (#)
905	0	-	-	-	-	-	-	-	-	-
advent	1	8.00	1	1	8.00	1	1	9.00	1	1
adventureland	9	8.44	7	4	8.22	7	5	9.11	6	3
afflicted	3	8.00	2	1	7.67	2	1	9.00	2	1
anchor	0	-	-	-	-	-	-	-	-	-
awaken	3	8.33	2	1	8.00	2	1	9.00	2	1
balances	4	8.00	3	2	7.75	3	2	9.00	3	2
ballyhoo	3	8.33	2	1	8.67	2	1	9.00	2	1
curses	2	8.50	2	1	8.00	2	1	9.00	2	1
cutthroat	88	8.45	52	29	8.52	54	28	9.18	47	26
deephome	1	8.00	1	1	7.00	1	1	9.00	1	1
detective	11	8.36	6	3	8.18	6	4	9.09	5	2
dragon	0	-	-	-	-	-	-	-	-	-
enchanter	3	8.00	2	1	7.67	2	1	9.00	2	1
enter	1	8.00	1	1	7.00	1	1	9.00	1	1
gold	3	8.33	2	1	8.67	2	1	9.00	2	1
hhgg	0	-	-	-	-	-	-	-	-	-
hollywood	2	8.00	2	1	8.00	2	1	9.00	2	1
huntdark	0	-	-	-	-	-	-	-	-	-
infidel	0	-	-	-	-	-	-	-	-	-
inhumane	13	8.31	7	4	8.15	7	4	9.00	6	3
jewel	4	8.25	3	2	8.00	3	2	9.00	3	2
karn	3	8.00	2	1	8.33	2	1	9.00	2	1
library	2	8.50	2	1	8.00	2	1	9.00	2	1
loose	0	-	-	-	-	-	-	-	-	-
lostpig	0	-	-	-	-	-	-	-	-	-
ludicorp	0	-	-	-	-	-	-	-	-	-
lurking	0	-	-	-	-	-	-	-	-	-
moonlit	0	-	-	-	-	-	-	-	-	-
murdac	37	8.32	21	12	8.46	22	11	9.03	19	10
night	6	8.33	4	2	8.17	4	2	9.00	3	2
omniquest	0	-	-	-	-	-	-	-	-	-
partyfoul	2	8.00	2	1	7.50	2	1	9.00	2	1
pentari	3	8.33	2	1	8.00	2	1	9.00	2	1
planetfall	9	8.33	5	3	8.56	5	2	9.00	4	2
plundered	1	8.00	1	1	7.00	1	1	9.00	1	1
reverb	8	8.38	4	2	8.25	4	3	9.13	3	2
seastalker	1	8.00	1	1	7.00	1	1	9.00	1	1
sherlock	5	8.40	3	2	8.20	3	2	9.20	2	1
snacktime	1	8.00	1	1	7.00	1	1	9.00	1	1
sorcerer	3	8.00	2	1	7.67	2	1	9.00	2	1
spellbrkr	0	-	-	-	-	-	-	-	-	-
spirit	2	8.00	2	1	8.00	2	1	9.00	2	1
temple	3	8.33	2	1	8.00	2	1	9.00	2	1
trinity	1	8.00	1	1	7.00	1	1	9.00	1	1
tryst205	2	8.00	2	1	8.00	2	1	9.00	2	1
wishbringer	2	8.00	2	1	7.50	2	1	9.00	2	1
yomomma	0	-	-	-	-	-	-	-	-	-
zenon	1	8.00	1	1	7.00	1	1	9.00	1	1
zork1	6	8.33	3	2	8.17	3	2	9.00	3	2
zork2	7	8.29	4	2	8.43	4	2	9.00	4	2
zork3	4	8.25	3	2	8.00	3	2	9.00	3	2
ztuu	2	8.00	2	1	7.50	2	1	9.00	2	1

Component	Complexity	MANGO scale
Conflict detection	$\mathcal{O}(\|E\|)$	< 1 ms
LCA computation	$\mathcal{O}(\|P_1\| + \|P_2\|)$	< 1 ms
Edge Impact Scoring	$\mathcal{O}(\|E\| \times \|V\|)$	~ 5 ms
VC operations	$\mathcal{O}(1)$ rollback / recall; $\mathcal{O}(\|E\|)$ diff	< 1 ms
Total algorithmic		< 10 ms / cycle
LLM inference	per API call	$\sim$ 2–5 s / call

Table A4: Per-component complexity. The dominant cost is LLM API latency, not the algorithmic components.

Metric	Min	Max	Mean	Median
Nodes per game	4	32	17.5	18.0
Edges per game	4	57	28.5	29.0
Walkthrough steps	21	1,263	247.1	201.0
Graph density	0.039	0.583	0.127	0.094
Average node degree	0.44	2.11	1.59	1.65

Table A5: Refined MANGO statistics across 53 games.

Rank	Edge	Reach	$\widehat{R}$	EIS
1	Entrance Hall → Office	6	1.00	3.00
2	**Office → Meeting Room**	**5**	**0.80**	**2.80**
3	Entrance Hall → Corridor	2	0.20	2.20
4	Corridor → Lab	1	0.00	2.00

Table A6: Case B EIS ranking. The true error (bold) is the edge with the second-highest reachability among LCA candidates.