# GRODE-RTL: SOURCE-MAPPED DEPENDENCY VIEWS FOR LLM-BASED RTL REPAIR

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Large Language Models show great potential for automated Register-Transfer Level (RTL) repair, yet they struggle with non-local faults where root causes and failing outputs reside far apart in source code. Treating RTL as a flat token sequence obscures hardware dataflows, whereas exposing complete synthesized graphs overwhelms the model with fine-grained metadata. To resolve this representation dilemma, we propose GRODE-RTL, an anchor-centric representation that extracts the single-clock-stage fan-in cone of a suspect register from the netlist, maps netlist nodes back to editable source lines, and organizes them into a compact hop-partitioned dependency view. By prioritizing direct drivers and folding distant hops, GRODE-RTL guides standard tool-using agents without exposing ground-truth hints or oracle feedback. We evaluate GRODE-RTL across 30 formally validated non-local faults using three LLMs (DeepSeek-V4-Flash, GLM-5.2, and MiniMax-M3) alongside a controlled five-configuration ablation study. GRODE-RTL improves pass@1 over source-level exploration by 16.7% on DeepSeek, by 26.7% on GLM-5.2, and by 20.0% on MiniMax-M3, with consistent gains on the hard-to-retrieve far-distance and indirect-anchor subsets of all three models. Our controls isolate structural relevance from generic context expansion, proving that selective dependency organization rather than full graph exposure is key to effective RTL repair.

## 1 INTRODUCTION

LLM-based agents debug software with human-level competence, yet this success only partially transfers to Register-Transfer Level (RTL) designs due to *non-local faults*. In RTL, a divergent output and its offending statement often lie hundreds of lines apart, separated across module boundaries, deeply nested conditional chains, or long dependency paths. While local bugs confined to narrow code windows are easily resolved by LLMs, hardware concurrency and pipelining mean the vast majority of real-world RTL bugs are inherently non-local. Resolving this dominant fault class is thus essential for viable automated hardware repair.

Repairing a non-local RTL fault requires bridging a long causal gap from an observable failure to the source statement that must be edited. This process involves three distinct stages: localizing a suspicious signal from the failing behavior (Sepidband et al., 2026a;b), reconstructing the static dependencies from that signal to its potential source-level causes, and generating and validating a patch. Existing repair methods primarily strengthen the two ends of this process. Feedback-driven approaches use compiler, simulation, or sequential-behavior feedback to generate and refine candidate patches (Santiesteban et al., 2023; Du et al., 2026; Thakur et al., 2024), while trace-based approaches use execution behavior to narrow failures to suspicious signals or registers (Wang et al., 2026).

However, neither line of work explicitly constructs the intermediate repair context required by non-local faults. Sequential feedback reveals whether and when candidate behavior diverges, but does not expose which distant source statements statically influence the suspicious signal. Signal-level localization identifies an anchor, but still leaves the agent to search across modules, conditional chains, and dependency paths for an editable root cause. The agent must therefore bridge behavioral evidence and source-level repair locations using either flat RTL text or a raw netlist, leading to a representation dilemma (Figure 1): (1) **Layout–Behavior Mismatch in Flat Source.** RTL text is organized by layout rather than dataflow. Expanding contiguous source windows to capture distant



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

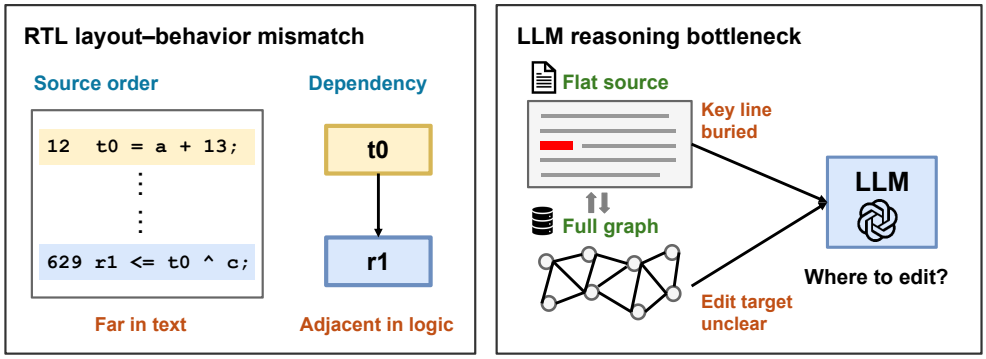



Figure 1: The representation dilemma for LLM-based non-local RTL repair. **Left:** Flat source obscures multi-hop dataflow dependencies across physical lines. **Right:** Passing raw netlists burdens LLMs with metadata, whereas GRODE-RTL extracts, organizes, and compresses fan-in logic into a source-mapped, compact view.

drivers introduces irrelevant code, exacerbating needle-in-a-haystack localization problems. (2) **The Graph Representation Dilemma.** Exposing full netlists captures dependencies but inflates context with low-level metadata, obscuring editable source mappings and repair locations.

To bridge this gap, we propose GRODE-RTL, an anchor-centric, repair-oriented representation constructed via three key operations: (1) *Relevant code filtering* extracts the single-clock-stage fan-in cone of a suspicious anchor and maps netlist nodes back to editable RTL statements; (2) *Explicit dependency organization* orders statements by reverse-dependency depth and tags `DATA`, `CONTROL`, and `STATE` relations; and (3) *Selective graph compression* caps per-hop nodes and folds distant layers into unfoldable summaries.

Our main contributions are:

- We expose the representation dilemma in non-local RTL repair, where flat source code lacks distant context while complete graphs overload LLM attention.
- We present GRODE-RTL, a representation pipeline that combines **source-mapped cone extraction** to eliminate design noise with **compact graph reorganization** to focus LLM reasoning.
- We demonstrate Pass@1 gains of up to **+26.7 pp** across three LLMs (reaching a **63.3%** success rate), with ablations confirming that cone extraction drives the primary performance leap while structural compression optimizes efficiency and edit precision.

## 2 Related Work

**Automated Repair for Hardware Designs.** Automated hardware description language (HDL) repair originated with heuristic Search-Based Program Repair techniques such as CirFix (Santiesteban et al., 2023), which applies genetic programming over line-level coverage. With the emergence of Large Language Models, LLM-based RTL repair frameworks have advanced from single-prompt repair and conversation-driven refinement like AutoChip (Thakur et al., 2024) to autonomous, tool-augmented debugging agents such as VeriPilot (Wang et al., 2026) and SeqFeed (Du et al., 2026). However, SeqFeed's sequential feedback (SeQuery and SeGraph) and VeriPilot's golden-guided control-data-flow tracing expose temporal simulation behavior rather than reorganizing the static dependencies of the current buggy design. GRODE-RTL instead introduces an anchor-centric structural representation between the SAT witness and the editing tools, transforming formal netlist dependencies into an interactive, source-mapped view for the agent.

**Program and Graph Representations for LLMs.** Graph, dataflow, and control-flow abstractions are widely used to enrich program context for machine learning models, often via learned

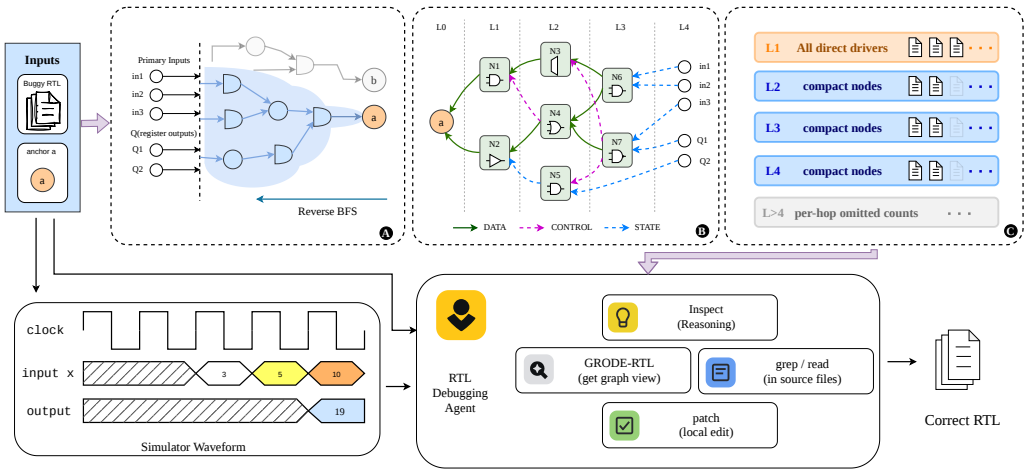



Figure 2: Overview of GRODE-RTL. **A) Subgraph Extraction** Synthesize buggy RTL and run reverse BFS from the anchor to recover the fan-in cone. **B) Dependency Organization** Partition the cone into per-hop levels $L_0, L_1, \ldots$ by reverse-dependency depth. **C) Graph Compression** Select representative statements per hop and cap depth into a compact structural view.

GNN embeddings (Kipf & Welling, 2017; Veličković et al., 2018; Hamilton et al., 2017; Zhang et al., 2020). While effective for general code understanding, generic program graphs fail to capture hardware-specific state boundaries and clock-stage constraints. GRODE-RTL leverages these hardware properties by partitioning netlist fan-in cones into reverse-dependency hops, providing a human-readable and model-editable text view. Furthermore, our controlled ablations explicitly separate structural cone selection from generic context expansion.

**Hardware Diagnosis and Netlist Representation Learning.** Cone-of-influence (COI) reduction is a standard structural pruning technique in formal verification to bound target logic (Clarke et al., 2018; Küchlin & Sinz, 2000; Goldberg et al., 2001; Velev & Bryant, 2001). Concurrently, circuit representation learning (CRI) methods such as DeepGate variants (Li et al., 2022; Shi et al., 2023; Zheng et al., 2025; Wang et al., 2025) embed netlist structures to predict testability or equivalence. However, classical COI operates strictly within formal solver engines, while neural netlist embeddings lack direct source-level mapping for code modification. GRODE-RTL bridges this gap by mapping netlist-level COI logic back to Verilog source statements, transforming formal structural reductions into an interactive context for LLM repair agents.

## 3 Method

Figure 2 outlines the architecture of GRODE-RTL. To address unstructured source-level debugging, GRODE-RTL extracts the anchor's single-clock-stage fan-in cone from the synthesized netlist, organizes dependencies by reverse-depth, and compresses them into a compact, source-mapped structural view for the agent.

### 3.1 Problem Formulation

We define the problem of repairing a buggy RTL design. The agent receives the buggy Verilog source $P_b$, a counterexample waveform $\mathbf{x} \in \{0,1\}^{L \times D}$ over $L$ clock cycles and $D$-bit bus width where $P_b$ diverges, and a suspicious anchor register $a$. The anchor $a = f_{\text{sim}}(P_b, \mathbf{x})$ is deterministically extracted as the first upstream divergent sequential register during counterexample simulation, providing the entry point for structural analysis. The counterexample and anchor serve as behavioral evidence, allowing the agent to observe where the divergence manifests without revealing the underlying root cause. Within a budget of $T$ turns, the agent modifies a range of at most 20 lines in $P_b$ to generate candidate $\hat{P}$ via `patch()`, submitting via `done()` or terminating via `abstain()`. The agent edits raw RTL source directly, never the netlist, and complete file rewrites are prohibited.

### 3.2 ANCHOR-CENTRIC FAN-IN EXTRACTION

We synthesize $P_b$ via Yosys (Wolf et al., 2013) into a netlist $N = (\Gamma, W, \text{src})$ of cells $\Gamma$, wires $W$, and source locations $\text{src} : \Gamma \cup W \to \text{Locations}(P_b)$. From $N$, we extract the statement cone governing the anchor's next-state update.

**State Boundary.** Let $B \subseteq W$ be the set of Q-outputs of all sequential cells in $N$. To enforce a standard single-clock-stage abstraction, the fan-in cone is defined as the closure of $a$ under combinational reverse dependencies without crossing $B$. By bounding extraction at $B$, GRODE-RTL targets intra-stage combinational non-locality within a single clock cycle, deferring multi-cycle temporal unrolling to future work.

**Cone Traversal.** Defining one-step dependency $u \rightsquigarrow v \iff \exists c \in \Gamma, v \in \text{fanin}(c), u \in \text{fanout}(c)$, the anchor's fan-in cone $C(a)$ is:

$$C(a) = \{w \in W \mid a \rightsquigarrow^* w, \text{ no element of } B \text{ on the path}\}, \tag{1}$$

where $\rightsquigarrow^*$ denotes the reflexive transitive closure. We compute $C(a)$ via a reverse BFS seeded at $a$, terminating at reachable Q-outputs and primary inputs.

**Source Mapping.** Using Yosys `src` annotations, we project the netlist cone $C(a)$ back to the unique RTL statements in $P_b$ that generated it:

$$S(a) = \{\text{src}(w) \mid w \in C(a)\} \subseteq \text{Lines}(P_b). \tag{2}$$

Deduplication handles the many-to-one mapping from netlist elements to source lines (Appendix A.3).

### 3.3 CONE PARTITION BY REVERSE HOP

We partition $C(a)$ by reverse-dependency depth to construct a layered view of the computation cone, represented as a typed dependency graph $G = (V, E)$ with $V = V_N \cup V_S$: $V_N$ are the mapped source statements $S(a)$, and $V_S$ are the boundary signals where the reverse traversal terminates (sequential Q-outputs in $B$ and primary inputs).

**Hop Partition.** Let $h(v)$ denote the shortest directed path length from $a$ to $v$ in the reverse dependency graph. We partition the cone into per-hop layers $L_k = \{v \mid h(v) = k\}$ for $k = 0, 1, 2, \ldots$, where $L_0 = \{a\}$, $L_1$ contains immediate drivers of $a$, $L_2$ drives $L_1$, and so forth. This unique partition is computed directly during the reverse BFS traversal.

**Edge Typing.** Edges in the dependency graph are assigned mutually exclusive labels $\tau \in \{\texttt{DATA}, \texttt{CONTROL}, \texttt{STATE}\}$ based on their operational roles: `DATA` for arithmetic and logic operands, `CONTROL` for multiplexer and conditional branching conditions, and `STATE` for dependencies anchored at sequential register boundaries ($B$). Appendix A.4 details the cell-to-graph translation algorithm.

### 3.4 COMPACT GRAPH VIEW

The compact graph view, serialized from $V$ and $E$, is submitted to the agent under four policies to bound token footprint while prioritizing structural attention (Figure 3). By default, per-hop node cap $K = 4$ and visible depth cap $H = 4$.

**Restructuring Policies.** (1) **Direct-driver priority** $L_1$ is emitted in full as the primary search space. (2) **Per-hop representative cap** For $2 \le k \le H$, at most $K$ representative nodes are emitted from $L_k$ in reverse-BFS order. (3) **Hop cap and summary folding** Layers beyond $L_H$ are folded into summaries such as `"L5: 8 omitted"`, acting as cues for dynamic unfolding via `read`. (4) **Path retention** $L_1$ retains at least one shortest path to the state boundary $V_S$. Formally, the view emits vertex set

$$V^{\text{compact}} = L_0 \cup L_1 \cup \bigcup_{k=2}^{H} \text{top}_K(L_k), \tag{3}$$

fold summaries

$$F = \{(h, |L_h|) \mid h > H, L_h \neq \varnothing\}, \tag{4}$$

and induced typed edges $E^{\text{compact}}$.

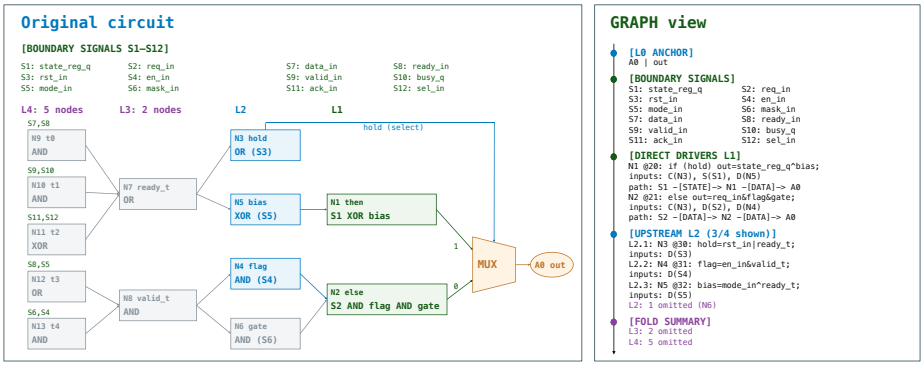



Figure 3: Anchor's single-clock-stage fan-in cone of a far-distance `const_tweak` sample and the corresponding compact graph view. **Left** The cone recovered from Yosys. **Right** The compact graph view submitted to the agent.

**Textual Serialization.** $V^{\text{compact}}$, $E^{\text{compact}}$, and $F$ are serialized into an anchor-centric hierarchical text block. Each statement is annotated with line numbers, dependency types (`DATA`, `CONTROL`, `STATE`), and hop distance $h$, serving as a structured roadmap for precise edits.

### 3.5 Agent Interaction and Information Boundary

GRODE-RTL operates as an interactive representation layer enabling multi-turn, hop-aware localization and repair.

**Tool-Augmented Repair Loop.** The agent accesses standard tools (`read`, `grep`, `patch`, `done`, `abstain`) alongside GRODE-RTL's `get_graph_view()`, which returns $(V^{\text{compact}}, E^{\text{compact}}, F)$ on demand. Patch modifications are restricted to incremental, line-bounded edits to enforce surgical repairs.

**Information Boundary.** To prevent shortcut learning, the agent observes only the buggy source $P_b$, counterexample $\mathbf{x}$, anchor $a$, and structural graph views. Golden reference RTL, ground-truth fault locations, and internal oracle diagnostics are strictly concealed. Submissions are evaluated externally via sequential equivalence checking.

## 4 Experiments

### 4.1 Research Questions

We organize the evaluation around three research questions.

**RQ1: Does GRODE-RTL improve LLM-based RTL repair?** We evaluate whether the proposed structural augmentation improves repair effectiveness across different LLMs and remains useful for bugs that are difficult to retrieve through source-level exploration.

**RQ2: Why does the compact dependency representation help?** We isolate structural relevance, dependency organization, and compactness to determine whether the benefit comes from simply exposing more context or from selectively presenting useful dependency information.

**RQ3: What are the behavioral, cost, and robustness trade-offs?** We examine how structural guidance changes the agent's editing behavior, what reasoning cost it introduces, and how timeout, statistical, and fault-pattern effects bound the strength and scope of the conclusions.

### 4.2 Experimental Setup

**Benchmark.** Our evaluation builds on 85 single-site faults across 55 DeepCircuitX (Li et al., 2025) RTL designs, where each fault is verified non-equivalent under a 20-step zero-initialized SAT

check (Clarke et al., 2001) and lies within its anchor's fan-in cone. We evaluate 30 medium- and long-distance faults (23 designs) with source distance $d$ (anchor declaration to fault site) ranging from 50 to 782 lines (median: 104.5): 18 *mid*-distance ($50 \leq d \leq 150$) and 12 *far*-distance ($d > 150$). Designs range from 63 to 1,137 lines (median: 305). By anchor visibility, 16 are *direct* (anchor name on faulty line) and 14 are *indirect* (anchor name omitted). Fault types comprise 17 `signal_swap`, 8 `const_tweak`, and 5 `op_swap`.

**Common Protocol.** All configurations share identical task prompts, buggy files, and SAT reproduction inputs, revealing neither golden RTL nor fault sites. Episodes run with temperature 1.0, maximum output 8,192 tokens, master seed 0 ($k = 1$), a 12-turn cap, 300 s episode timeout, and 600 s oracle timeout. Each episode permits up to 5 combined `grep`/`read` calls and 1 `patch` call ($\leq 20$ lines; no full rewrites). An isolated budget mode ensures structural tools do not consume text-exploration quotas.

**Evaluation Metrics & Oracle.** Candidates are evaluated via Yosys 20-step bounded SAT equivalence checking against the golden RTL oracle. We report Pass@1 (percentage of single-submission successful repairs) and absolute differences in percentage points (pp).

**Evaluation Scope & Controlled Anchor.** To isolate repair-context quality from localization accuracy, we fix one anchor register $a$ for each case and provide the same anchor to every configuration. The anchor is defined offline as the first upstream divergent sequential register identified through counterexample simulation on the faulty design. Agents receive only the anchor identity; neither the golden RTL nor the fault location is revealed. The evaluation is therefore conditioned on a shared localization starting point and measures how effectively each representation connects that anchor to its editable source-level causes, rather than evaluating end-to-end fault localization.

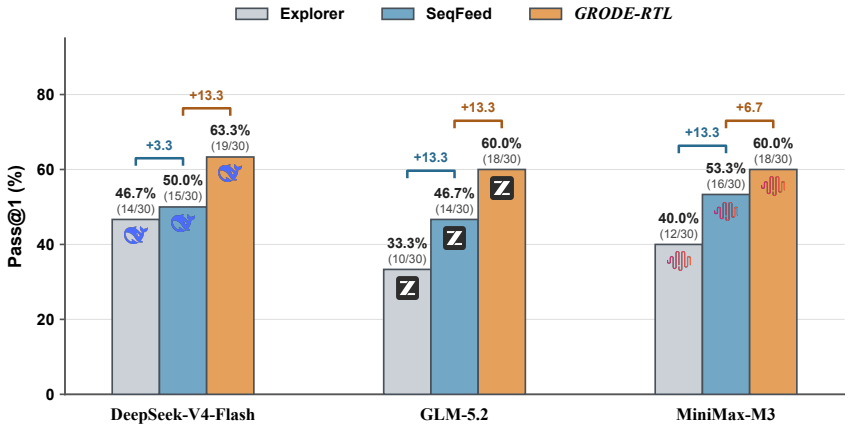



Figure 4: End-to-end pass@1 of Explorer, SeqFeed, and *GRODE-RTL* on the same 30 mid- and far-distance bugs for each model. Success is measured using the repair oracle and semantic audit described in the text.

### 4.3 RQ1: Does GRODE-RTL Improve LLM-based RTL Repair?

We evaluate three methods across 30 mid- and far-distance bugs ($3 \times 30$ evaluations) under the common protocol: source-only exploration (Explorer), the sequential-behavior-feedback baseline SeqFeed (Du et al., 2026), and *GRODE-RTL*.

**Overall Effectiveness.** *GRODE-RTL* consistently boosts repair success across all evaluated LLMs (Figure 4, Table 1). Across the three models, average pass@1 rises from 40.0% (Explorer) to 50.0% (SeqFeed), and reaches **61.1%** with GRODE-RTL (+21.1 pp overall gain). The maximum improvement occurs on GLM-5.2 (33.3% $\rightarrow$ 60.0%, +26.7 pp), where GRODE-RTL nearly doubles the successfully repaired bugs (10 $\rightarrow$ 18). The consistent within-model gains validate that structural augmentation offers model-agnostic benefits to RTL repair workflows.

Table 1: Cross-model comparison of Explorer, SeqFeed, and GRODE-RTL on the 30 mid- and far-distance samples.

| Model | Method | Pass@1 by subset | | | |
|---|---|---|---|---|---|
| | | mid (18) | far (12) | direct (16) | indirect (14) |
| DeepSeek-V4-Flash | Explorer | 9/18, 50.0% | 5/12, 41.7% | 10/16, 62.5% | 4/14, 28.6% |
| DeepSeek-V4-Flash | SeqFeed | 11/18, 61.1% | 4/12, 33.3% | 11/16, 68.8% | 4/14, 28.6% |
| DeepSeek-V4-Flash | *GRODE-RTL* | **11/18, 61.1%** | **8/12, 66.7%** | **11/16, 68.8%** | **8/14, 57.1%** |
| GLM-5.2 | Explorer | 8/18, 44.4% | 2/12, 16.7% | 7/16, 43.8% | 3/14, 21.4% |
| GLM-5.2 | SeqFeed | 10/18, 55.6% | 4/12, 33.3% | 9/16, 56.2% | 5/14, 35.7% |
| GLM-5.2 | *GRODE-RTL* | **13/18, 72.2%** | **5/12, 41.7%** | **12/16, 75.0%** | **6/14, 42.9%** |
| MiniMax-M3 | Explorer | 8/18, 44.4% | 4/12, 33.3% | 9/16, 56.2% | 3/14, 21.4% |
| MiniMax-M3 | SeqFeed | 12/18, 66.7% | 4/12, 33.3% | 11/16, 68.8% | 5/14, 35.7% |
| MiniMax-M3 | *GRODE-RTL* | **12/18, 66.7%** | **6/12, 50.0%** | **12/16, 75.0%** | **6/14, 42.9%** |

| Model | Method | Per episode (of 30) | | Per-episode cost |
|---|---|---|---|---|
| | | Pass | No edit | Tokens |
| DeepSeek-V4-Flash | Explorer | 14 | 12 | 55,982 |
| DeepSeek-V4-Flash | SeqFeed | 15 | 9 | 133,384 |
| DeepSeek-V4-Flash | *GRODE-RTL* | **19** | 11 | 72,317 |
| GLM-5.2 | Explorer | 10 | 10 | 43,030 |
| GLM-5.2 | SeqFeed | 14 | 10 | 96,016 |
| GLM-5.2 | *GRODE-RTL* | **18** | 5 | 57,774 |
| MiniMax-M3 | Explorer | 12 | 10 | 63,054 |
| MiniMax-M3 | SeqFeed | 16 | 7 | 147,426 |
| MiniMax-M3 | *GRODE-RTL* | **18** | 7 | 78,706 |

**Dominance on Search-Hard Subsets.** The advantage of *GRODE-RTL* is especially pronounced on hard-to-retrieve bugs. On far-distance bugs ($d > 150$), average pass@1 jumps from 30.6% to **52.8%** (+22.2 pp); on indirect-anchor bugs (where fault lines omit the anchor name), success increases from 23.8% to **47.6%** (+23.8 pp). Notably, SeqFeed yields limited gains on these subsets (33.3% for both): sequential feedback exposes temporal propagation, yet source-level search over long static dependency chains remains the agent's burden. In contrast, *GRODE-RTL* successfully bridges distant computation paths even when source-level text search yields no immediate matches.

**RQ1 Takeaway.** GRODE-RTL provides substantial, model-agnostic repair gains over both raw exploration and execution-feedback baselines, demonstrating unique strength in resolving spatially obscured and distant RTL faults.

### 4.4 RQ2: Why Does the Compact Dependency Representation Help?

To isolate the sources of improvement, we conduct a controlled representation ablation on DeepSeek-V4-Flash over the same 30 bugs (Table 2). The control configurations incrementally incorporate structural features (Figure 5, Table 3):

- `subgraph`: Localized AST/dependency subgraph without source line context.
- `matched_random`: Size-matched random subgraph to test functional relevance versus raw context volume.
- *GRODE-RTL(cone only)*: Unorganized fan-in cone mapped to source statements without dependency hierarchy.
- *GRODE-RTL(w/o. compact)*: Unpruned netlist dependency graph without selective cone partitioning.

Here, **view nodes** refers to the number of AST/dependency statement nodes serialized into the structural context, and **view chars** denotes its raw string length.

Table 2: Structural features captured by representation variants in the ablation.

| Configuration | Anchor Signal | Source Lines | Subgraph Extract | Dep. Org. | Graph Comp. |
|---|---|---|---|---|---|
| `subgraph` | ✓ | ✗ | ✗ | ✗ | ✗ |
| `matched_random` | ✓ | ✓ | ✗ | ✗ | ✗ |
| *GRODE-RTL(cone only)* | ✓ | ✓ | ✓ | ✗ | ✗ |
| *GRODE-RTL(w/o. compact)* | ✓ | ✓ | ✓ | ✓ | ✗ |
| *GRODE-RTL* | ✓ | ✓ | ✓ | ✓ | ✓ |

**Structural Relevance Over Raw Context.** Exposing only the coarse SAT mismatch signal (tool-call `get_subgraph()`) yields 50.0% pass@1, just 3.3 pp above source-only exploration. Providing line-count-matched random lines (`matched_random`) reaches 53.3%, whereas supplying the actual fan-in cone (tool-call `get_cone()`) raises pass@1 to 60.0%. This confirms that structural relevance to the faulty computation—rather than coarse localization or generic context expansion—drives the performance gain.

**Over-exposure Degrades Reasoning.** Adding unpruned graph detail does not yield monotonic gains. While `full_graph` introduces explicit hop partitions and typed edges, it drops overall Pass@1 to 53.3% ($-6.7$ pp), with severe drops on far-distance (58.3% $\rightarrow$ 33.3%) and indirect-anchor (50.0% $\rightarrow$ 35.7%) subsets. Excessive structural noise burdens LLM attention, counteracting the benefits of graph context.

**Compactness and Typing Enable Targeted Exploration.** Combinational fan-in slicing (*GRODE-RTL(cone only)*) accounts for the primary recovery over baseline exploration (+13.3 pp), matching *GRODE-RTL* on simple direct-anchor (68.8%) and mid-distance (61.1%) subsets under a lower footprint. However, unorganized linear listings stall on complex far-distance and indirect-anchor faults. *GRODE-RTL*'s typed hop-partitioning resolves this bottleneck, delivering vital incremental gains (+3.3 pp overall) that reach peak performance on far-distance (66.7%) and indirect (57.1%) cases. Furthermore, compared to *GRODE-RTL(w/o. compact)*, *GRODE-RTL* prunes node count by 62.6% (9.8 vs. 26.2) and view size by 47.0% (2,398 vs. 4,524 chars), proving that selective, typed partitioning outperforms both unorganized linear slicing and unpruned context expansion.

**RQ2 Takeaway.** The ablation demonstrates a clear progression: localization offers minimal gain, raw dependency context helps, unpruned graph structures degrade reasoning, and GRODE-RTL optimizes the trade-off through compact dependency organization.

Table 3: Five ablation configurations on DeepSeek-V4-Flash.

| | **Overall** | | **Pass@1 by subset** | | | |
|---|---|---|---|---|---|---|
| Configuration | Pass | Δ | mid (18) | far (12) | direct (16) | indirect (14) |
| `subgraph` | 15/30, 50.0% | −13.3% | 11/18, 61.1% | 4/12, 33.3% | 11/16, 68.8% | 4/14, 28.6% |
| `matched_random` | 16/30, 53.3% | −10.0% | 11/18, 61.1% | 5/12, 41.7% | 9/16, 56.2% | 7/14, 50.0% |
| `cone` | 18/30, 60.0% | **−3.3%** | 11/18, 61.1% | 7/12, 58.3% | 11/16, 68.8% | 7/14, 50.0% |
| `full_graph` | 16/30, 53.3% | −10.0% | **12/18, 66.7%** | 4/12, 33.3% | 11/16, 68.8% | 5/14, 35.7% |
| *GRODE-RTL* | **19/30, 63.3%** | — | 11/18, 61.1% | **8/12, 66.7%** | **11/16, 68.8%** | **8/14, 57.1%** |

| | **Edit (of 30)** | | **Per-episode cost** | | | |
|---|---|---|---|---|---|---|
| Configuration | Patches | No edit | Tool calls | Tokens | View nodes | View chars |
| `subgraph` | 20 | 9 | 5.73 | 63,235 | — | — |
| `matched_random` | 19 | 8 | 6.40 | 64,987 | 26.2 | 1,326 |
| `cone` | 21 | 9 | 6.57 | 65,987 | 24.7 | 1,761 |
| `full_graph` | 17 | 12 | 6.03 | 70,112 | 26.2 | 4,524 |
| *GRODE-RTL* | 19 | 11 | 6.43 | 72,317 | 9.8 | 2,398 |

† For configurations where Patches and No Edit sum to less than 30 (e.g., `subgraph`, `matched_random`, and `full_graph`), the remaining episodes terminated via timeout prior to issuing a patch call or exit decision.

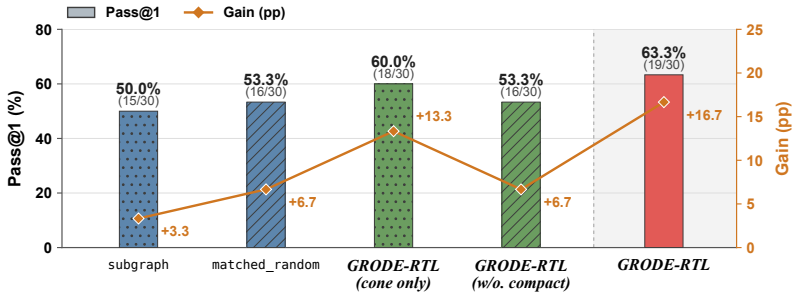



Figure 5: Pass@1 under different dependency representations on the same 30 mid- and far-distance bugs.

### 4.5 RQ3: Behavioral, Cost, and Robustness Trade-offs

RQ3 examines how structural guidance alters agent execution, its associated costs, and the bounds of our findings.

**Selective Guidance Drives Precise Edits.** As shown in Table 1, GRODE-RTL consistently suppresses no-edit fallbacks across models (DeepSeek: 12→11; GLM-5.2: 10→5; MiniMax-M3: 10→7) while increasing repair yield. Conditioned on edited episodes, conditional success rates surge: 77.8%→100.0% (DeepSeek), 50.0%→72.0% (GLM-5.2), and 60.0%→78.3% (MiniMax-M3). This proves the representation enhances repair quality rather than mere edit propensity.

**Reasoning Cost vs. Selective Exposure.** Although GRODE-RTL increases per-episode token consumption compared to the baseline *explorer* (e.g., 72,317 vs. 55,982 on DeepSeek-V4-Flash), it achieves substantial token savings compared to SeqFeed (72,317 vs. 133,384 tokens, a 45.8% reduction; and a 39.8%–46.6% reduction across all backbones, e.g., 57,774 vs. 96,016 on GLM-5.2 and 78,705 vs. 147,426 on MiniMax-M3). This demonstrates that compact view synthesis successfully avoids context overload, enabling selective exposure to channel the token budget into targeted follow-up reads and deeper sequential reasoning.

**Fault Observability and Evaluation Bounds.** The structural gain varies by fault pattern: GRODE-RTL substantially lifts fully observable `const_tweak` bugs (2/8→6/8 on DeepSeek) but provides marginal gains on `signal_swap` (12/17→13/17) and fails on unobservable `op_swap` faults (0/5), which demand semantic rather than structural reasoning. Additionally, while relay timeouts introduce operational noise, the qualitative case (Appendix A.2) confirms that compact views avoid the 27k-char view clutter that causes `full_graph` to forfeit edits.

**RQ3 Takeaway.** GRODE-RTL converts idling into high-precision edits by channeling tokens into deep sequential reasoning rather than raw context display. Its strength centers on structural observability, establishing selective dependency organization as a robust principle within explicit operational boundaries.

## 5 Conclusion

We presented GRODE-RTL, an anchor-centric representation framework that bridges netlist dependencies with editable RTL source for non-local hardware repair. Across three LLMs, GRODE-RTL yields average Pass@1 gains of **+21.1 pp** over exploration baselines and **+11.1 pp** over strong sequential baselines (reaching **63.3%** peak), demonstrating that selective cone partitioning outperforms exhaustive graph serialization. The compact representation enables deeper structural reasoning across extended debugging turns. Future work will extend GRODE-RTL to autonomous upstream localization and operator-level diagnosis.

### REPRODUCIBILITY STATEMENT

The benchmark construction, structural extraction, agent evaluation, and formal-equivalence pipeline are described in Sections 3 and 4. Further implementation details, full prompts, and per-case logs are provided in Appendix A. All evaluation code, raw datasets, and reproduction scripts are included in the supplementary material.

### AI USE STATEMENT

Generative AI tools were employed for manuscript drafting and language refinement. All experimental data, tables, and reported pass@1 metrics were cross-referenced with raw execution logs, which are preserved in the supplementary material for reproducibility. The authors take full responsibility for the final publication.

## REFERENCES

Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. In *International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, 2001.

Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem (eds.). *Handbook of Model Checking*. Springer, 2018. ISBN 978-3-319-10574-1.

Yuxin Du, Juxin Niu, Tao Hu, Xi Wang, Zhe Jiang, and Nan Guan. SeqFeed: Improving agentic RTL code generation with sequential behavior feedback, 2026. URL https://arxiv.org/abs/2608.16934.

Evguenii I Goldberg, Mukul R Prasad, and Robert K Brayton. Using sat for combinational equivalence checking. In *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*, pp. 114–121. IEEE, 2001.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Wolfgang Küchlin and Carsten Sinz. Proving consistency assertions for automotive product data management. *Journal of Automated Reasoning*, 24(1-2):145–163, 2000.

Min Li, Sadaf Khan, Zhengyuan Shi, Naixing Wang, Huang Yu, and Qiang Xu. Deepgate: Learning neural representations of logic gates. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 667–672, 2022.

Zeju Li, Changran Xu, Zhengyuan Shi, Zedong Peng, Yi Liu, Yunhao Zhou, Lingfeng Zhou, Chengyu Ma, Jianyuan Zhong, Xi Wang, Jieru Zhao, Zhufei Chu, Xiaoyan Yang, and Qiang Xu. DeepCircuitX: A comprehensive repository-level dataset for RTL code understanding, generation, and PPA analysis, 2025. URL https://arxiv.org/abs/2502.18297.

Priscila Santiesteban, Yu Huang, Westley Weimer, and Hammad Ahmad. Cirfix: Automated hardware repair and its real-world applications. *IEEE Transactions on Software Engineering*, 49(7):3736–3752, 2023. doi: 10.1109/TSE.2023.3269899.

Melika Sepidband, Hung Viet Pham, and Hadi Hemmati. On the role of fault localization context for LLM-based program repair, 2026a. URL https://arxiv.org/abs/2604.05481.

Melika Sepidband, Hamed Taherkhani, Hung Viet Pham, and Hadi Hemmati. RGFL: Reasoning guided fault localization for automated program repair using large language models, 2026b. URL https://arxiv.org/abs/2601.18044.

Zhengyuan Shi, Hongyang Pan, Sadaf Khan, Min Li, Yi Liu, Junhua Huang, Hui-Ling Zhen, Mingxuan Yuan, Zhufei Chu, and Qiang Xu. Deepgate2: Functionality-aware circuit representation learning. In *Proceedings of the 2023 IEEE/ACM international conference on Computer-aided design*, 2023.

Shailja Thakur, Jason Blocklove, Hammond Pearce, Benjamin Tan, Siddharth Garg, and Ramesh Karri. Autochip: Automating hdl generation using llm feedback, 2024. URL https://arxiv.org/abs/2311.04887.

Miroslav N Velev and Randal E Bryant. Effective use of boolean satisfiability procedures in the formal verification of superscalar and vliw. In *Proceedings of the 38th ACM/IEEE Design Automation Conference*, pp. 226–231, 2001.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Mingjun Wang, Bin Sun, Jianan Mu, Feng Gu, Boyu Han, Tianmeng Yang, Xinyu Zhang, Silin Liu, Yihan Wen, Hui Wang, et al. Moss: Multi-modal representation learning on sequential circuits. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7. IEEE, 2025.

Yihan Wang, Cheng Liu, Jiazheng Zhang, Lei Zhang, Long Cheng, Xiaowei Li, and Huawei Li. Veripilot: An LLM-powered Verilog debugging framework, 2026. URL https://arxiv.org/abs/2606.23759.

Clifford Wolf, Johann Glaser, and Johannes Kepler. Yosys-a free verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, pp. 97, 2013.

Yanqing Zhang, Haoxing Ren, and Brucek Khailany. Grannite: Graph neural network inference for transferable power estimation. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020.

Ziyang Zheng, Shan Huang, Jianyuan Zhong, Zhengyuan Shi, Guohao Dai, Ningyi Xu, and Qiang Xu. Deepgate4: Efficient and effective representation learning for circuit design at scale. *arXiv preprint arXiv:2502.01681*, 2025.

# A Supplementary Material

This appendix first reports the per-case outcomes across all evaluated models and repair methods. It then walks through *why* compact dependency views work on a concrete sample, makes the information boundary between configurations explicit by listing the full prompt and tool descriptions, and shows one representative episode under both the compact and full-view configurations. We refer to the case-study sample as *Sample S* throughout; only its filename is anonymized to `<design>.v`.

## A.1 Per-case outcomes across models

Table 4 lists all 30 evaluation cases using the original circuit name and injected fault type. For each model, it compares `explorer`, SeqFeed, and GRODE-RTL (`explorer_graph_view_compact`). **PASS** denotes a correct repair. For failures, `NE`, `TO`, `ECF`, and `ENE` denote `no_edit`, `timeout`, `equivalence_check_failed`, and `edited_not_equivalent`.

## A.2 Case study: a far-distance `const_tweak` under five configurations

**Design.** Sample S is a small synchronous Verilog module `glyph_addr` that maps a 6-bit character input `glyph` together with a 4-bit vertical counter `vcnt` to a 10-bit output `address`. The body of the module is a single synchronous `always` block containing a 24-branch `case (glyph)` statement; each case branch holds an `if/else if` chain over `vcnt` that updates the internal `cur_add` register by a constant base expression indexed by `vcnt`. The `cur_add` register is then assigned to the `address` output combinationally on the next line. The full source is 784 lines; much of the body consists of near-identical `else if` branches differing only in their `vcnt` literal and, in the buggy version, in one constant term.

Table 4: Per-case outcomes across models and repair methods. All methods are evaluated on the same 30 cases for each model.

| # | Circuit | Fault type | DeepSeek-V4-Flash | | | GLM-5.2 | | | MiniMax-M3 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Explorer | SeqFeed | GRODE | Explorer | SeqFeed | GRODE | Explorer | SeqFeed | GRODE |
| 01 | 14520_2 | signal_swap | NE | NE | NE | NE | NE | NE | TO | NE | NE |
| 02 | 14520_2 | op_swap | NE | NE | NE | NE | TO | TO | ENE | ENE | NE |
| 03 | 1542 | const_tweak | ENE | ENE | **PASS** | NE | NE | NE | ENE | NE | **PASS** |
| 04 | 16017 | signal_swap | **PASS** | **PASS** | **PASS** | **PASS** | **PASS** | **PASS** | **PASS** | **PASS** | **PASS** |
| 05 | 16331_3 | signal_swap | NE | **PASS** | **PASS** | NE | NE | **PASS** | NE | **PASS** | **PASS** |
| 06 | 16331_3 | const_tweak | ENE | NE | **PASS** | NE | **PASS** | **PASS** | NE | **PASS** | **PASS** |
| 07 | 1762_14 | const_tweak | NE | NE | NE | TO | TO | TO | NE | **PASS** | **PASS** |
| 08 | 21502 | const_tweak | **PASS** | **PASS** | **PASS** | TO | NE | **PASS** | **PASS** | **PASS** | **PASS** |
| 09 | 24583_12 | const_tweak | **PASS** | **PASS** | **PASS** | TO | **PASS** | **PASS** | NE | **PASS** | **PASS** |
| 10 | 24583_12 | signal_swap | **PASS** | **PASS** | **PASS** | **PASS** | **PASS** | NE | NE | **PASS** | **PASS** |
| 11 | 24940_10 | signal_swap | **PASS** | **PASS** | **PASS** | NE | NE | **PASS** | **PASS** | ECF | **PASS** |
| 12 | 24940_10 | signal_swap | **PASS** | ENE | **PASS** | TO | NE | NE | **PASS** | NE | **PASS** |
| 13 | 25631 | signal_swap | **PASS** | TO | **PASS** | **PASS** | NE | **PASS** | **PASS** | **PASS** | **PASS** |
| 14 | 25631 | signal_swap | **PASS** | **PASS** | NE | **PASS** | **PASS** | **PASS** | **PASS** | **PASS** | ECF |
| 15 | 29019_6 | signal_swap | ENE | ENE | NE | TO | NE | TO | ENE | ENE | ECF |
| 16 | 300_2 | signal_swap | NE | **PASS** | **PASS** | NE | **PASS** | **PASS** | ECF | **PASS** | NE |
| 17 | 30095_15 | signal_swap | **PASS** | **PASS** | **PASS** | **PASS** | **PASS** | **PASS** | **PASS** | **PASS** | **PASS** |
| 18 | 30895 | signal_swap | **PASS** | NE | **PASS** | TO | **PASS** | **PASS** | ECF | ENE | TO |
| 19 | 3125_6 | signal_swap | **PASS** | **PASS** | **PASS** | TO | **PASS** | **PASS** | **PASS** | NE | ECF |
| 20 | 33437_14 | signal_swap | ENE | **PASS** | **PASS** | **PASS** | **PASS** | TO | **PASS** | NE | ENE |
| 21 | 33437_14 | signal_swap | **PASS** | **PASS** | **PASS** | **PASS** | **PASS** | **PASS** | TO | ECF | **PASS** |
| 22 | 33831_8 | signal_swap | **PASS** | **PASS** | **PASS** | **PASS** | **PASS** | **PASS** | NE | ECF | **PASS** |
| 23 | 35009 | op_swap | NE | NE | NE | TO | TO | TO | **PASS** | **PASS** | **PASS** |
| 24 | 3629_1 | const_tweak | NE | ENE | NE | **PASS** | NE | **PASS** | **PASS** | **PASS** | **PASS** |
| 25 | 3629_1 | op_swap | NE | NE | NE | NE | TO | **PASS** | NE | NE | TO |
| 26 | 6109_13 | const_tweak | NE | **PASS** | **PASS** | TO | **PASS** | **PASS** | ENE | **PASS** | **PASS** |
| 27 | 6716_2 | op_swap | NE | NE | NE | NE | TO | TO | ECF | NE | TO |
| 28 | 6726_6 | const_tweak | NE | **PASS** | **PASS** | TO | **PASS** | **PASS** | NE | **PASS** | **PASS** |
| 29 | 6777_6 | signal_swap | **PASS** | NE | NE | **PASS** | NE | TO | **PASS** | **PASS** | NE |
| 30 | 8610 | op_swap | NE | ENE | NE | NE | ENE | NE | ENE | ENE | NE |
| | | **PASS total** | **14/30** | **15/30** | **19/30** | **10/30** | **14/30** | **18/30** | **12/30** | **16/30** | **18/30** |

**Anchor.** The anchor is the 10-bit output register `address`, declared in the module port list near the top of the file. It is an *indirect* anchor in our taxonomy. The anchor name `address` does not appear on any `cur_add` update line, so a naive `grep("address")` only returns the port declaration and the combinational assignment, neither of which is the bug site. The `get_suspect_subgraph` tool's coarse-localizer signal (simulation mismatch at step 20 on the `address` register, identifying the first upstream divergent sequential register) is the only reason the agent has to focus on this register.

**Fault.** The fault on Sample S is a single-literal `const_tweak` on the `else if (vcnt == 'd10)` branch within one `case (glyph)` branch: the constant `10'd151` should read `10'd150`, the same base used by every other `else if` branch and by the `rst` initialization. The fault line lies $d = 620$ source lines below the anchor declaration, putting Sample S in the `far` bucket of the benchmark (distance measured from the anchor declaration to the fault site). The right edit is therefore a one-character change in a 24-branch switch where every other branch is correct, and where the same constant appears many times across the file.

**Configuration outcomes on this sample.** The five available configuration outcomes on this sample are listed in Table 5. Failure labels are `no_edit` (the agent exhausts its turn budget without calling `patch`) and `edited_not_equivalent` (the agent emits a patch whose post-edit design is not functionally equivalent to the golden RTL).

The full-graph configuration fails by *giving up*: it consumes its exploration budget on reads and greps without ever calling `patch`. The compact configuration patches line 629 with a one-character change and is graded PASS. Matched random context also passes on this sample, but the contrast of interest for the representational claim is full $\rightarrow$ compact: the two configurations see the same cone source set serialized in two ways, and only the compact view converts the structure into an edit. The full

Table 5: Per-configuration outcome on Sample S under each of the five structural configurations.

| Configuration | Outcome | Failure type |
|---|---|---|
| matched_random | **PASS** | None |
| subgraph | FAIL | edited_not_equivalent |
| cone | **PASS** | None |
| full_graph | FAIL | no_edit |
| *GRODE-RTL* | **PASS** | None |

graph view is 27,871 characters of node/edge metadata; the compact view is 2,968 characters of which roughly 1,300 are the navigation header and fold summary, leaving about 1,700 characters of structured source. On this sample, more graph is worse.

**The compact view the agent saw.** The following is the exact `get_graph_view("address")` response delivered to the agent when it called the tool:

```
GRODE-RTL Compact Graph View for anchor [address]
(784 total source lines in P_b; showing 11 representative statements):
Primary structural search space: inspect L1 direct-driver cards first;
expand beyond the view only if they cannot explain the failure.
Compact graph guide: showing 11/161 mapped source statements.
This is navigation, not a complete cone (bounded by K=4, H=4).
Folded statements by hop (F):
  L5: 15 omitted; L6: 6 omitted; L7: 10 omitted; ... L18: 9 omitted.
Use read() around a listed source location when full branch syntax is needed.

GRODE-RTL dependency view (current RTL only; no fault-line scores)
Top module: glyph_addr
How to use: inspect [L1 Direct driver statements] first; then read
a local range around a listed line if more syntax is needed.
Do not grep the anchor unless no mapped source node is shown.

[L0 Anchor]
A0 | address

[L1 Direct-driver statements: inspect and compare first]
N1 | <design>.v:9  | assign address = cur_add;
N2 | <design>.v:13 | if (rst) cur_add <= 10'd150;
     inputs: CONTROL(S2); DATA(N3)
     retained path: S2 -[CONTROL]-> N2 -[DATA]-> A0

[L2 Upstream statements (capped at K=4)]
N3 | <design>.v:16 | inputs: CONTROL(S1); CONTROL(N5); CONTROL(N7); DATA(N4)
                     case (glyph)

[L3 Upstream statements (capped at K=4)]
N4 | <design>.v:19 | inputs: CONTROL(S3); DATA(S1); DATA(S3); DATA(N8)
                     if (vcnt == 'd0) cur_add <= (10'd150+{glyph,4'b0000})+...
N5 | <design>.v:36 | inputs: CONTROL(S1)          end
N6 | <design>.v:39 | inputs: CONTROL(S3); DATA(S1); DATA(S3); DATA(N9)
                     if (vcnt == 'd0) cur_add <= (10'd150+{glyph,4'b0000})+...
N7 | <design>.v:56 | inputs: CONTROL(S1)          end

[L4 Upstream statements (capped at K=4)]
N8 | <design>.v:20 | inputs: CONTROL(S3); DATA(S1); DATA(S3)
                     else if (vcnt == 'd1) cur_add <= (10'd150+{glyph,4'b0000})+...
N9  | <design>.v:40 | inputs: CONTROL(S3); DATA(S1)
                     else if (vcnt == 'd1) cur_add <= (10'd150+...)
N10 | <design>.v:60 | inputs: CONTROL(S3); DATA(S1)
                     else if (vcnt == 'd1) cur_add <= (10'd150+...)
N11 | <design>.v:80 | inputs: CONTROL(S3); DATA(S1)
                     else if (vcnt == 'd1) cur_add <= (10'd150+...)

[State/Input boundary V_S]
S1 | glyph   S2 | rst   S3 | vcnt
```

The compact graph view makes three structural properties visible that flat source code buries: (1) the anchor $a$ has a clear direct driver in $L_1$ (`N2 = <design>.v:13`); (2) intermediate hops $L_2$–$L_4$ are strictly capped at $K = 4$ representative nodes, while the remaining 150 upstream statements across hops $L_5$–$L_{18}$ are cleanly folded into $F = \{(5, 15), (6, 6), \dots\}$ summaries; (3) the sequential

state boundary $V_S$ isolates `glyph, rst, vcnt` as the sole clock-stage inputs. Given this compact representation, the agent identifies the entry point, executes a targeted `read()` around line 629 in the folded logic, and repairs the bug in its first patch submission.

**The full view the agent saw, in contrast.** The same anchor serialized in the `full_graph` configuration contains 161 mapped source nodes and 612 typed edges, organized by reverse hop from L1 to L18. The first 2,000 characters and the last 2,000 characters of that view are:

```
Graph view for address [full graph view]
(784 source lines; showing 161):
... [legend and L0 anchor as in the compact view] ...

[Direct driver statements: inspect and compare first]
N1 | <design>.v:9      assign address = cur_add;
N2 | <design>.v:13     if (rst) cur_add <= 10'd150;
    inputs: CONTROL(S2); DATA(N3)
    path: S2 -[CONTROL]-> N2 ; N2 -[DATA]-> A0

[L2 Remaining upstream source statements]
N3 | <design>.v:16 | inputs: CONTROL(S1); CONTROL(N11);
                     CONTROL(N13); CONTROL(N15); +16 more
               case (glyph)

[L3 Remaining upstream source statements]
N4  | <design>.v:19 | ... if (vcnt == 'd0) cur_add <= (10'd150+...) ...
N5  | <design>.v:36 | end
N6  | <design>.v:39 | ... if (vcnt == 'd0) cur_add <= (10'd150+...) ...
...
[continues through L18 with 9-10 near-identical statements per hop]

[L18 Remaining upstream source statements]
N153 | <design>.v:34  | ... else if (vcnt == 'd15) cur_add <= (10'd150+...) ...
N154 | <design>.v:54  | ... else if (vcnt == 'd15) cur_add <= (10'd150+...) ...
...
N161 | <design>.v:194 | ... else if (vcnt == 'd15) cur_add <= (10'd150+...) ...

[State/input boundary signals]
S1 | glyph   S2 | rst   S3 | vcnt
```

The full view faithfully lists every hop, every node, every edge label. On this sample the result is a 27,871-character payload with 161 nodes, 612 typed edges, and 18 reverse-dependency hops whose per-hop bodies are near-identical `else if` lines differentiated only by `vcnt` literals. The agent that received it performed four more exploration calls (three greps and a targeted read of lines 760–784) without ever producing a patch, and exhausted its 5-call exploration budget.

**Why this matters.** The compact and full views draw on exactly the same upstream `cur_add` nodes. The structural *information* they expose differs only by containment (compact folds by hop, full enumerates by hop). The difference the agent actually uses is the one the view imposes on that information: direct drivers up front, folded summaries behind, and a hard cap on the number of statements the model must read to identify the fault line. The contrast above is a single instance of the population pattern reported in Section 4: the full graph configuration matches the matched random context on pass rate, while the compact configuration matches the source-order cone.

### A.3 Recovering src-less statements

The src-mapping step in Section 3 is many-to-one after deduplication, but a small number of source statements are not reachable through any Yosys cell annotation at all. We recover them by walking the cone and the anchor's declaration sites directly:

- *Anchor bare assignments.* An `assign` $a$ `= expr;` in the RTL is folded into the netlist as a direct connection and produces no per-cell `src`; we recover it by scanning $P_b$ for assignments whose LHS is the anchor and adding them as hop-1 nodes tagged `DIRECT`.
- *Direct signal assignments.* Initial-value assignments of $a$ (synchronous reset defaults inside the `always` block, module-port defaults) carry the same `src`-less status; we recover them by collecting every statement whose target is the anchor.

- *Boundary declarations.* The anchor's declaration site in the module port or the `always`-block local list is not part of any cell's `src` and is recovered by parsing $P_b$'s port list and `always` headers.

All three classes are added as hop-1 nodes tagged `DIRECT` so that the compact view's direct-driver priority (§4.4) treats them on equal footing with cells that do carry `src`. The extractor operates only on $P_b$ and never admits a statement on the basis of a difference against the golden code; this boundary is what keeps the compact view's representational gain attributable to organization rather than to oracle leakage.

### A.4 Recovering the dependency graph

The typed dependency graph of Section 3 is recovered cell-by-cell from the Yosys netlist and then lifted to source-statement and signal nodes.

**Vertex construction.** The vertex set is $V = V_N \cup V_S$ with $V_N = S(a) \setminus \{\text{anchor declaration}\}$ representing the mapped source statements and $V_S$ representing the upstream signal alphabet reached by the reverse BFS that does not cross the sequential boundary $B \subseteq W$. The anchor itself sits as a singleton in $L_0$. Every other vertex carries the deduplicated `src` annotation of the cells that produced it.

**Edge construction.** Edges point from a statement to the statements it depends on, matching the $u \rightsquigarrow v$ convention of Section 3 and the reverse-BFS hop levels; for readability, the serialized view prints retained paths in dataflow direction (e.g. `S2 -[CONTROL]-> N2 -[DATA]-> a`). For each combinational cell $c$ that survives the cone closure, we record an edge from the statement node of $v = \text{fanout}(c)$ to the statement node of each $u \in \text{fanin}(c)$, then label the edge according to $c$'s cell type:

- `DATA`: data operand of a logic or arithmetic cell (e.g. `$and`, `$add`).
- `CONTROL`: selector input of a multiplexor or branch condition of a `$mux`/`$bmux`; these drive which operand is observed rather than carrying data.
- `STATE`: an edge whose source is a Q-output wire in $B$, i.e. the only carrier of state across clock stages.

A single Yosys cell may therefore contribute one or more edges to $E$, each with its own label. The labels are mutually exclusive and exhaustive: every edge in $E$ receives exactly one of the three types, and the union of the three covers $E$. This is the implementation of the edge-type claim in Section 3.

### A.5 System prompt and tool descriptions

All configurations in the ablation share the same opening system prompt, which is implemented as the `FIX_SYSTEM_PROMPT_BASE` constant below. Tool descriptions are JSON-Schema specifications sent verbatim to the model; we list the text they carry.

```
You are fixing a bug in an RTL (Verilog) design. The design currently
fails a functional-equivalence check against its correct reference.
There is no source location or preselected suspect signal. You must
locate the root cause yourself.

Tools
- grep(pattern) regex-search the CURRENT design source; returns
  "file:line: text".
- read(file, start, end) return a numbered range of the current source.
- patch(file, start, end, content) replace a small 1-based inclusive
  line range in the CURRENT file. Use this instead of rewriting the
  complete file.
- done() submit the current files for grading.
- abstain(reason) explicitly stop when you cannot produce a defensible
  fix.
```

```
Method
Use the available evidence to inspect suspicious local operators,
constants, and signal sources before proposing broader redesigns. Trace
back from a divergent signal through the logic that drives it, patch only
the smallest line range justified by the evidence, then submit it. A fix
that changes the suspected logic but breaks other outputs will fail. Never
resend or reconstruct a complete source file. You MUST finish by calling
either done() or abstain(reason); never stop with an ordinary text response.
```

When the configuration registers a structural tool, we appends a short tool-specific block *only for the tools that configuration has*. The conditional append for the compact-graph configuration is:

```
Additional tools available in this episode:
- get_suspect_subgraph(): coarsely locate a suspect dependency subgraph
  and return its root signal together with SAT reproduction evidence.
- get_graph_view(anchor): inspect an anchor-centric CURRENT-RTL
  dependency view grouped by reverse hop with DATA, CONTROL, and STATE
  edges.

Tool-specific method:
Call get_suspect_subgraph() early, then use its root signal with the
structural tool available in this episode.
Inspect get_graph_view() direct-driver cards first, then read a small
local range only when syntax context is needed; do not reflexively grep
the anchor.
```

The same template, with `get_graph_view` replaced by `get_cone(anchor): map the anchor's single-stage fan-in cone back to CURRENT RTL source lines`, is used for the cone configuration. The `subgraph` configuration receives a reduced append containing only `get_suspect_subgraph` (no view tool is registered). The `matched_random` configuration receives the cone block unchanged. The matched-random condition is realized inside the tool implementation, which returns token-count-matched random source lines randomly; the agent's prompt and tool specification are byte-identical to the source-order cone configuration. Two configurations with identical tool sets therefore produce byte-identical system prompts; this is verified by the test suite.

The per-tool description that the model sees as a JSON-Schema tool spec is also fixed. For `get_graph_view`, the description string is:

```
Return an anchor-centric reverse-dependency view of the current RTL.
Call it early. It lists direct driver statements of the anchor first,
with their immediate DATA, CONTROL, and STATE inputs and one
deterministic shortest upstream path. It then lists the remaining
same source statements by reverse hop: L1 directly drives the anchor
and larger L values are farther upstream. A0 denotes the anchor, N#
a source-statement node, and S# an upstream state/input boundary.
Node numbers are stable identifiers, not suspiciousness rankings.
Use read() around a listed line for more local syntax; avoid
grep(anchor) unless this view has no mapped source nodes. Treat the
displayed direct drivers as the primary structural search space before
broad global search. The view uses only current RTL and never uses
golden RTL, the fault location, or a suspiciousness score.
```

This is the literal information boundary: no golden code, no fault line, no fault pattern, no suspiciousness score, no patch hint appears anywhere in the prompt, the tool spec, or the tool output.

### A.6 One sample, two configurations

We follow Sample S under the compact-graph configuration (PASS) and the full-graph configuration (FAIL with `no_edit`). Both configurations consume the same per-episode budget: at most 5 `grep`/`read` calls, at most 1 `patch`, and at most 1 call to each structural tool.

**Compact-graph configuration (PASS).**

1. **turn 0** `get_suspect_subgraph()` → anchor = `address`, mismatch at step 20.
2. **turn 0** `read(<design>.v, 1, 80)` (5,649 chars).
3. **turn 1** `read(<design>.v, 81, 260)` (15,211 chars).
4. **turn 2** `read(<design>.v, 261, 560)` (25,394 chars). Three reads cover lines 1–560 of the 784-line file.
5. **turn 3** `grep("10'd[0-9]+\+|glyph,4'b[01]+|4'b[01]+,vcnt")` → 8,530 chars of repeated `cur_add` lines.
6. **turn 3** `grep("d15|d16|...|d6[0-3]")` → 8,470 chars. The 5-call exploration budget is now full.
7. **turn 4** `get_graph_view("address")` → 2,968 chars (the compact view in Section A.2, fold summary + 11 nodes).
8. **turn 4** `grep("10'd1[0-9][0-9]|glyph,4'b[^0]|4'b[^0],vcnt")` → budget exhausted message.
9. **turns 5–11** reasoning (model writes and reflects without tool calls; the system has issued its user-nudge `use a terminal tool`).
10. **turn 12** `patch(<design>.v, 629, 629, "else if (vcnt == 'd10) cur_add <= (10'd150+{glyph,4'b0000})+{4'b0000,vcnt};")` → patched line 629, replacing `10'd151` with `10'd150`.
11. **turn 13** `done()`. Oracle returns PASS.

The agent uses three reads to map the file, two greps to confirm the repetition pattern, the one permitted `get_graph_view` call to collapse the cone, and the one permitted `patch` call to fix the anomalous literal.

**Full-graph configuration (FAIL, `no_edit`).**

1. **turn 0** `get_suspect_subgraph()` → anchor = `address`.
2. **turn 0** `grep("module|always|assign|glyph|vcnt")` → 8,130 chars of mixed hits.
3. **turn 1** `get_graph_view("address")` → 27,871 chars (the full view).
4. **turn 1** `read(<design>.v, 1, 60)` (3,977 chars).
5. **turn 2** `grep("10'd|'d6[0-9]|default|endcase|endmodule")` → 8,470 chars.
6. **turn 2** `read(<design>.v, 760, 784)` (2,192 chars). The agent is now reading the tail of the file, far from the folded L3 block that contains the fault.
7. **turn 3** `grep("^[ ]+'d[0-9]+:")` → 1,732 chars listing all 24 `case` branch headers.
8. **turn 3** `grep("vcnt")` → budget exhausted message.
9. The agent never calls `patch`; the episode terminates with no edit. Oracle returns FAIL with `no_edit`.

The full-graph configuration has the same upstream information, the same anchor, and the same exploration budget. It converts the larger view into more wandering greps rather than into a focused patch. The asymmetry between the two runs is a single observation, but it is the structural claim of the paper at a sample-sized resolution: compact organization turns the same signal-dependency data into fewer dead-end exploration turns and more first-attempt correct edits.