WHERE DOES A HARDWARE CHANGE ACT? INTENT-CONDITIONED STRUCTURAL REASONING FOR RTL MODIFICATION-SUPPORT RECOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Reliable hardware evolution requires identifying the register-transfer-level (RTL) locations affected by a natural-language change request. This is challenging because changes often span multiple locations coupled by hardware dependencies, beyond independent request–to–code matching. We formulate this task as $\Delta\mathrm{Spec}$-to-RTL localization and introduce EvoRTL-Bench, the first benchmark aligning natural-language change requests with fine-grained modification support sets from real RTL revisions. Our analysis shows that modification support sets are sparse yet structurally cohesive, exhibit non-random dependency patterns, and have sizes and topologies predictable from change requests. Motivated by these findings, we propose RTLocating, which fuses textual semantics, functional behavior, and architectural context into selective semantic seeds, then performs intent-conditioned impact propagation over directed, typed RTL dependencies with adaptive channel and scope selection. Counterfactual decision supervision trains these decisions using the support-recovery utility of alternative evidence allocations and propagation actions. On EvoRTL-Bench, RTLocating achieves state-of-the-art performance with a 17.3% relative improvement in mean average precision over the strongest baseline. End-to-end editing experiments further indicate that its localization guidance improves downstream RTL editing.

1 INTRODUCTION

As chip designs grow increasingly complex, LLM-aided hardware design (LAD) has emerged as a promising route to greater design productivity (Pan et al., 2025; Wu et al., 2024; Wang et al., 2024). A primary focus is the register-transfer level (RTL), which links high-level design intent to circuit implementation by describing state transitions, data transfers, and logic at clock-cycle granularity. Despite progress in natural-language-driven RTL generation and understanding, existing work largely studies one-shot tasks on self-contained design units (Pei et al., 2024; Wang et al., 2025; Xing et al., 2026; Liu et al., 2025a). Mature designs instead evolve through incremental feature additions, behavioral changes, and bug fixes. HWE-Bench and Phoenix-Bench show that changes often span modules and hierarchy levels along signal-flow paths, making cross-hierarchy localization and coordinated editing key challenges for repository-level hardware agents (Cui et al., 2026; Zou et al., 2026). Moving LAD beyond one-shot generation toward reliable hardware evolution begins with a prerequisite question: given an RTL design and a natural-language (NL) change request, which implementation locations must be modified?

We formulate this problem as $\Delta\mathrm{Spec}$-to-RTL localization: given an RTL design and a natural-language change request specifying a desired specification change $\Delta\mathrm{Spec}$, the goal is to recover the corresponding RTL modification support set. Unlike NL-to-RTL retrieval (Liu et al., 2026; Zhao et al., 2026), which finds code implementing queried functionality, our task identifies blocks affected by a change even when they neither implement nor mention that functionality directly. Unlike Verilog bug localization (Yao et al., 2025; Xu et al., 2024; Wan et al., 2026; Liu et al., 2025a), it also covers feature extensions, behavioral changes, and microarchitectural optimizations without assuming that the current design is faulty. We define an RTL block as a behaviorally meaningful unit of local combinational or sequential logic. Because a change may span blocks coupled by signal

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.

flow, control or state dependencies, and cross-hierarchy interfaces, localization is a multi-positive task requiring complete support-set recovery rather than independent query–block matching.

To characterize this task empirically, we analyze real-world RTL evolution and identify two regularities. First, modification support sets are sparse yet structurally cohesive, with non-random distribution across dependency types. Second, although their extent varies across changes, change requests carry predictive signal about support-set size, cross-module span, and graph dispersion. These findings motivate selective localization over the full candidate space, dependency-aware recovery beyond direct semantic matches, and structural reasoning conditioned on change intent.

Existing methods address different subsets of these requirements. RTL representation and retrieval methods model the functional semantics, sometimes with design hierarchy or circuit topology. However, their objectives primarily target RTL understanding, generation or retrieval from functional descriptions rather than how modification relevance over the same structure varies across change requests (Liu et al., 2025b; Zhao et al., 2026; Zheng et al., 2026; Wu et al., 2025). NL-driven software issue localization is closer in task formulation, but its structural reasoning relies on software-specific relations such as containment, imports, calls, and inheritance, and thus misses the typed signal dependencies and cross-state relations that define RTL's functional and temporal contracts (Reddy et al., 2025; Chen et al., 2025). Software fault localization and hardware debugging diagnose existing defects from coverage, test outcomes, or fault traces, all of which are typically unavailable in our setting (Zhang et al., 2023; Qian et al., 2023; Malburg et al., 2016).

To address this gap, we introduce **EvoRTL-Bench**. To our knowledge, it is the first benchmark to align natural-language change requests with fine-grained modification support sets mined from real RTL evolution. Building on the empirical regularities above, we propose **RTLocating**, which fuses textual semantics, functional behavior, and architectural context into selective semantic seeds, then performs intent-conditioned typed propagation over a Design-Wide Topology Graph (DTG) that encodes dependencies among RTL blocks. It adapts dependency channels and propagation scope to recover affected blocks missed by direct semantic evidence. To turn unlabeled intermediate decisions into measurable signals, we introduce **counterfactual decision supervision**, which grounds evidence-fusion configurations and propagation actions in their realized support-recovery utility. On EvoRTL-Bench, RTLocating consistently outperforms strong retrieval and hardware-aware representation baselines.

2 PRELIMINARY

2.1 PROBLEM FORMULATION

We formulate change localization in RTL design evolution as $\Delta\mathrm{Spec}$-to-RTL localization: given an existing RTL design D and a natural-language change request q specifying a desired specification change $\Delta\mathrm{Spec}$, the task is to recover the observed modification support set $\mathcal{S}_q \subseteq \mathcal{B}(D)$, consisting of the pre-existing RTL blocks modified in the reference revision. The candidate set $\mathcal{B}(D) = \{b_1, \ldots, b_N\}$ consists of module-scoped behavioral constructs, each defined as either a continuous assignment or a complete SystemVerilog `always_comb` or `always_ff` procedural block. This granularity preserves complete assignment, control, and clocking semantics while providing finer localization than modules. Because a change request may require coordinated edits across multiple blocks, we formulate multi-block localization as multi-positive ranking: for every $b^+ \in \mathcal{S}_q$ and $b^- \in \mathcal{B}(D) \setminus \mathcal{S}_q$, the scorer must satisfy $s_\theta(q, b^+; D) > s_\theta(q, b^-; D)$.

2.2 EVORTL-BENCH: DATASET CONSTRUCTION

We construct EvoRTL-Bench from OpenTitan's Git history (lowRISC, 2025), covering production-scale intellectual properties (IPs) developed under standardized practices. To our knowledge, it is the first benchmark aligning change requests with fine-grained modification support sets from real RTL revisions. Each instance is a triplet $(q, D_q^-, \mathcal{S}_q)$: a natural-language change request, the pre-change design D_q^- before the change, and the modification support set. We retain only commits that modify synthesizable SystemVerilog, excluding testbenches and auto-generated files, and use each revision's parent snapshot as D_q^-. Based on the commit description and change context, an LLM classifies each revision as a behavior-changing functional change, behavior-preserving maintenance,

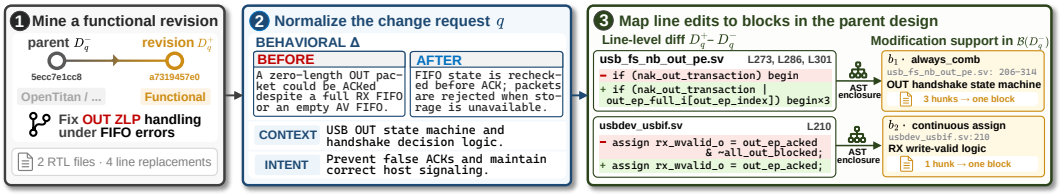


Figure 1: Example construction of an EvoRTL-Bench instance from a functional RTL revision.

Table 1: Empirical structure and scale predictability of RTL modification support sets.

(a) Support-set structure

Endpoint	n	Effect
Harmonic proximity H	440	$\Delta = +0.1051^{**}$ $[0.0934, 0.1186]$
Boundary compactness C	544	$\Delta = +0.0550^{**}$ $[0.0474, 0.0636]$
Role composition omnibus	428	$D = 11.4095^{**}$
Timing composition omnibus	428	$D = 3.3212^{**}$

(b) Scope-scale predictability

Target	n	Metadata	TF–IDF
Harmonic dispersion	76	-0.121 $[-0.318, 0.273]$	0.304^{*} $[0.125, 0.480]$
Cross-module pair ratio	76	-0.011 $[-0.337, 0.319]$	0.323^{*} $[0.123, 0.518]$
Support cardinality	110	0.158 $[-0.110, 0.420]$	0.562^{***} $[0.402, 0.670]$

Note. (a) Δ and D denote scalar mean and composition shifts. (b) partial Spearman ρ controlling for IP fixed effects, candidate-graph size, and query length. Brackets give 95% CIs. Stars use Holm-adjusted p-values in both panels. $^{*}p < 0.05$, $^{**}p < 0.01$, and $^{***}p < 0.001$.

or unclear. Only functional revisions are retained and reformulated into requests q comprising a before/after behavior summary, functional context, and high-level change intent. To derive $\mathcal{S}_q$, we compute the line-level diff between D_q^+ and D_q^- and map edited lines to candidate blocks in D_q^- via AST parsing. After filtering and validation, EvoRTL-Bench contains 1,094 instances. Figure 1 illustrates this construction process with an example. See Appendix A for details.

2.3 Design-Wide Topology Graph

As a shared structural representation, we model each RTL version D as a directed, typed Design-Wide Topology Graph (DTG), $G_D = (\mathcal{B}(D), E_D, \mathrm{dep}_D, \mathrm{time}_D)$, with one node per candidate RTL block. An edge $b_u \to b_v$ exists when a signal driven by b_u is used by b_v; dep_D and time_D assign each edge a dependency class and timing class, respectively. Dependency classes are Control (behavioral conditions and choices), Datapath (data and address computation or transfer), Clock/Reset (clocking, reset, and state retention), and Structural (organization induced by assignment targets and type associations). Timing classes are Combinational for within-cycle dependencies and Sequential for dependencies crossing clocked state boundaries.

3 Empirical Regularities of Real RTL Modifications

This chapter empirically analyzes real RTL evolution data to reveal structural and predictive regularities of modification support sets, providing a principled basis for the design of RTLocating. Metric definitions, model training details, sample inclusion criteria, statistical tests, complete results, and robustness analyses are provided in Appendix B.

3.1 RQ1: What structural shape do real functional modifications take?

We treat the modification support set $\mathcal{S}_q$ as a node subset of the pre-change design's DTG. Sparsity is measured by the proportion of candidate blocks in the support set, while harmonic proximity H and boundary compactness $C = 1 - \mathrm{conductance}$ measure how support blocks are mutually close on the weak DTG and how they form a well-defined structural boundary, respectively. Omnibus tests over dependency and timing classes on shortest paths characterize the hardware composition of co-modification relations. For topology and composition, we compare against conditional null sets matched by support size, module, and node degree.

Obs.1 Real functional modifications concentrate in small, cohesive RTL regions with dependency-class and timing-class bias. Support sets cover a median of 0.57% of candidate

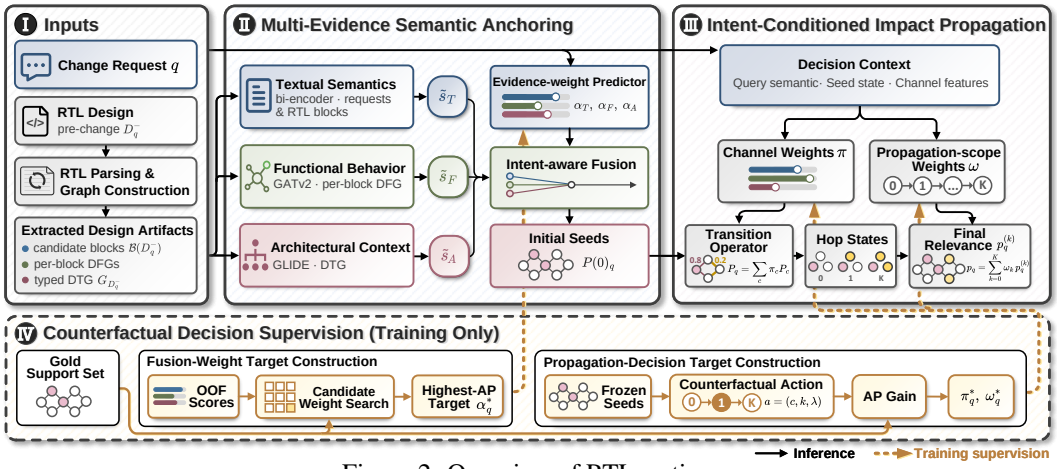


Figure 2: Overview of RTLocating.

blocks. Among queries eligible for matched randomization, observed supports are more cohesive than their null counterparts and exhibit nonrandom dependency-class and timing-class compositions, as shown in Table 1(a).

3.2 RQ2: Does the change request description reveal modification structure?

We quantify structural scale using harmonic dispersion (distance/disconnection on the weak DTG), the cross-module pair ratio (module span), and support cardinality (block count). To test whether change requests predict these properties, we compare a linear TF–IDF Ridge probe with a metadata baseline on the held-out Test split. The metadata baseline uses IP identity, candidate-graph size, and query length. For the two topological targets, it additionally includes support cardinality to isolate spatial organization beyond modification size.

Obs.2 Change requests predict modification size and spatial organization. TF–IDF predictions based on the full change request show significant partial associations with spatial organization, whereas the metadata baseline shows no significant association with any target.

Summary. Real support sets are sparse, cohesive, and typed, while their scale and topology are predictable from change requests. These findings motivate selective semantic anchoring over the full candidate space, followed by propagation along typed hardware dependencies with channels and scope adapted to each change intent.

4 Methodology

We design RTLocating for modification-support recovery in RTL design evolution. RTLocating fuses textual semantics, functional behavior, and architectural context into a selective semantic seed distribution over candidate blocks, and propagates these seeds along the typed hardware dependencies of the Design-Wide Topology Graph (DTG), adapting dependency channels and propagation scope to the given intent. Since these decisions cannot be determined a priori, we introduce counterfactual decision supervision, which treats candidate decisions as counterfactual actions and uses the realized gain on recovery as the direct supervision signal. Figure 2 illustrates the overall architecture.

4.1 Multi-Evidence Semantic Anchoring

The anchoring stage aims to extract a small set of reliable seeds from a large candidate space, providing selective initial evidence for subsequent propagation. We score block relevance using textual semantics, functional behavior, and architectural context, and fuse their scores with intent-aware weights to accommodate potential variation in source informativeness.

Textual semantics. A parameter-shared bi-encoder embeds the request and each block into a common metric space, trained with Multiple Negatives Ranking Loss. Textual similarity then anchors the blocks that the request directly mentions or describes.

Functional behavior. Since change requests describe intended behavior rather than identifiers, textual evidence alone underrepresents blocks defined by internal logic. We encode each block as a typed data-flow graph (DFG) over signals and operators, with edges for computation, control, addressing, assignment, clocking, and type associations. The graph is encoded by an edge-aware GATv2, and the pooled representation is aligned with a frozen request representation via InfoNCE.

Architectural context. Block-local evidence cannot characterize a block's role in the full design. We therefore encode each block in the DTG using a query-independent design-context encoder that alternates typed neighborhood aggregation with design-level global attention, capturing both typed local dependencies and cross-block interactions beyond fixed graph neighborhoods. The resulting representations can be precomputed per design version and are aligned with modification requests during training via listwise InfoNCE.

Intent-aware evidence fusion. For each source v, we z-normalize its query–block cosine similarities across $\mathcal{B}(D)$, yielding $\widetilde{s}_v(q,b)$. A two-layer MLP weight predictor takes only the query representation as input and predicts evidence weights used to compute the seed score $s_q^{\text{seed}}(b)$:

$$s_q^{\text{seed}}(b) = \sum_{v\in\{T,F,G\}} \alpha_v(q)\,\widetilde{s}_v(q,b), \qquad \boldsymbol{\alpha}(q) = \text{softmax}\big(f_{\text{weight}}(\mathbf{q})\big). \tag{1}$$

Under counterfactual decision supervision, we treat the fusion weights as decision actions and evaluate their effect individually. To decouple the score from the performance of the evidence encoders on the training set, we group by design version and cross-fit to obtain out-of-fold (OOF) score vectors, so each query is scored only by encoders untrained on the corresponding design. We then search a finite grid $\mathcal{W}$ on the three-source probability simplex using each query's OOF scores, taking the AP-maximizing configuration as the supervision target $\boldsymbol{\alpha}_q^*$:

$$\boldsymbol{\alpha}_q^* = \arg\max_{\boldsymbol{\alpha}\in\mathcal{W}} \text{AP}\left(\sum_v \alpha_v \widetilde{\mathbf{s}}_v^{\text{OOF}}(q), \mathcal{S}_q\right). \tag{2}$$

Here, $\widetilde{\mathbf{s}}_v^{\text{OOF}}(q)$ is the calibrated OOF score vector of evidence v, and AP denotes standard non-interpolated average precision over the full candidate ranking.

This supervision applies only to queries with multiple support blocks. Let $m_q \in \{0,1\}$ denote the corresponding supervision mask, $\mathcal{Q}_{\text{train}}$ the set of training queries, $\mathcal{L}_{\text{rank}}$ a multi-positive listwise ranking loss over $\mathcal{B}(D)$, and η the strength of counterfactual decision supervision. The fusion module is trained by

$$\mathcal{L}_{\text{anchor}} = \frac{1}{|\mathcal{Q}_{\text{train}}|} \sum_{q\in\mathcal{Q}_{\text{train}}} \left[\mathcal{L}_{\text{rank}}\big(s_q^{\text{seed}}, \mathcal{S}_q, \mathcal{B}(D)\big) + \eta\, m_q\, D_{\text{KL}}\big(\boldsymbol{\alpha}_q^* \,\|\, \boldsymbol{\alpha}(q)\big)\right]. \tag{3}$$

At inference, the three evidence encoders fitted on the full training set remain frozen, while f_{weight} predicts their weights from the query alone. The fused scores are temperature-normalized to obtain the initial seed distribution $p_q^{(0)}$.

4.2 Intent-Conditioned Typed Impact Propagation

Starting from the initial seeds $p_q^{(0)}$, we recover the full support set by propagating along typed dependencies. We reuse the DTG as the structural basis, whose typed edges define a set of channels $\mathcal{C}(G)$, each corresponding to a sparse transition operator P_c that distributes relevance mass uniformly over outgoing edges of type c. Nodes without such neighbors retain their original mass.

Channel weights. To ground the choice of channel and propagation scope, we build a decision context: the query representation provides the semantic prior over which typed channels the modification propagates along, the seed retrieval state summarizes the sharpness, leading-candidate confidence, and evidence consistency, and channel-conditioned features capture each channel's mass-carrying capacity and redistribution magnitude on the current seeds. An MLP head maps this context to a compatibility logit per channel, normalized by softmax into weights $\boldsymbol{\pi}(q,G)$. The channel operators are then combined to form the intent-specific transition operator $P_q = \sum_{c\in\mathcal{C}(G)} \pi_c(q,G)\, P_c$.

Propagation scope. To learn the depth of modification propagation, the model first generates hop-wise relevance distributions $p_q^{(k)} = (P_q^\top)^k\, p_q^{(0)}$, $(k = 0, \ldots, K)$, where K is the maximum propagation depth. The model predicts a softmax-normalized scope distribution $\boldsymbol{\omega}(q, G)$ from the decision context. The final relevance distribution over candidate blocks is

$$p_q = \sum_{k=0}^{K} \omega_k(q, G)\, p_q^{(k)}. \tag{4}$$

The channel weights and propagation scope jointly define an intent-specific impact geometry: the typed transition determines *where* relevance propagates, and the scope determines *how far*.

4.3 Learning Structural Decisions via Counterfactual Decision Supervision

We evaluate candidate decisions as counterfactual actions on frozen seeds constructed from OOF evidence scores, and take the realized gain on support-set recovery as the utility.

Actions. Each action $a = (c, k, \lambda)$ selects a typed channel $c \in \mathcal{C}(G)$, a propagation depth $k \le K$, and an interpolation strength λ from a finite grid $\Lambda \subset (0, 1]$, and produces a candidate distribution $p_{q,a}$ and an action-level hop profile $\boldsymbol{\omega}^{(a)}$. Let $\mathbf{e}_k \in \mathbb{R}^{K+1}$ denote the k-th standard basis vector.

$$p_{q,a} = (1 - \lambda)\, p_q^{(0)} + \lambda\, (P_c^\top)^k\, p_q^{(0)}, \qquad \boldsymbol{\omega}^{(a)} = (1 - \lambda)\, \mathbf{e}_0 + \lambda\, \mathbf{e}_k. \tag{5}$$

The null action a_0 leaves the seeds unchanged: $p_{q,a_0} = p_q^{(0)}, \boldsymbol{\omega}^{(a_0)} = \mathbf{e}_0$. The utility of an action is defined as its localization gain over the seeds:

$$U_q(a) = \mathrm{AP}(p_{q,a}, \mathcal{S}_q) - \mathrm{AP}\left(p_q^{(0)}, \mathcal{S}_q\right). \tag{6}$$

Supervision targets. We derive channel and scope utility targets from a finite set of offline single-channel actions. A temperature-scaled softmax over the action utilities $\{U_q(a)\}_{a \in \mathcal{A}_q}$ yields an action distribution ρ_q. Removing the null action a_0 and renormalizing yields $\bar{\rho}_q$, which captures relative utility over effective actions. Aggregating $\bar{\rho}_q$ over each channel's actions yields the typed-channel target $\boldsymbol{\pi}_q^*$, and utility-weighted averaging over the propagation-scope profiles yields the effective hop target $\boldsymbol{\omega}_q^*$:

$$\boldsymbol{\pi}_q^* = \left[\sum_{a \in \mathcal{A}_{q,c}} \bar{\rho}_q(a)\right]_{c \in \mathcal{C}(G)}, \qquad \boldsymbol{\omega}_q^* = \sum_{a \in \mathcal{A}_q} \rho_q(a)\, \boldsymbol{\omega}^{(a)}. \tag{7}$$

If no action exceeds a practical-gain threshold, we set $\boldsymbol{\omega}_q^* = \mathbf{e}_0$ and the channel-supervision mask $\chi_q^{\mathrm{ch}} = 0$; otherwise, $\chi_q^{\mathrm{ch}} = 1$. This factorization avoids direct optimization over the joint channel–scope space, while the listwise ranking loss constrains the final composed-propagation distribution and captures cross-channel interactions. Appendix D.2 evaluates the surrogate under the deployed propagation operator.

Training objective. The structural decisions and the support-set ranking are trained jointly:

$$\mathcal{L} = \mathcal{L}_{\mathrm{rank}} + \lambda_{\mathrm{CF}} \left[\chi_q^{\mathrm{ch}}\, \mathrm{CE}\big(\boldsymbol{\pi}_q^*, \boldsymbol{\pi}(q, G)\big) + \mathrm{CE}\big(\boldsymbol{\omega}_q^*, \boldsymbol{\omega}(q, G)\big)\right]. \tag{8}$$

All targets are generated offline from frozen seeds, so that every action shares the same propagation starting point and the supervision is decoupled from the model's own intermediate predictions. The complete action set, the target construction, and the loss weights are given in Appendix C.

5 Experiments

5.1 Experimental Setup

We summarize the dataset splits, baselines, and metrics in this section. Implementation details, hyperparameters, and task adaptations for all baselines are provided in Appendix C.

Table 2: Main localization performance of RTLocating and baselines on EvoRTL-Bench.

Category	Method	MAP	MRR	Recall		Accuracy	
				@R	@$2R$	@R	@$2R$
Lexical retrieval	BM25	0.2982	0.4339	23.35%	36.14%	6.36%	13.64%
Semantic retrieval	GraphCodeBERT	0.3181	0.4696	26.45%	37.87%	7.27%	10.91%
	CodeV-DS-6.7B	0.3216	0.5219	25.96%	39.95%	6.36%	11.82%
	SweRank-Retriever	0.4012	0.5814	32.88%	45.81%	8.48%	16.97%
Graph structure	HOGA	0.2570	0.4665	22.10%	32.53%	5.45%	9.09%
	Dir-GNN	0.2483	0.4468	19.80%	28.86%	3.64%	6.67%
Hardware-specific representation	CircuitFusion	0.2991	0.4934	24.76%	36.50%	5.45%	10.91%
	UniRTL	0.3167	0.5472	24.41%	35.89%	6.36%	11.82%
Ours	**RTLocating**	**0.4708**	**0.6556**	**40.45%**	**56.42%**	**15.45%**	**25.76%**

Dataset and splits. We adopt an IP-disjoint split on EvoRTL-Bench to assess cross-design generalization: models are trained on a subset of IPs and evaluated on held-out IPs, preventing reliance on IP-specific identifiers or features. The benchmark contains 1,094 queries, with 875/109/110 for training/validation/test.

Baselines. We compare BM25 (Robertson & Zaragoza, 2009), GraphCodeBERT (Guo et al., 2020), CodeV-DS-6.7B (Zhao et al., 2025), and SweRank-Retriever (Reddy et al., 2025), representing lexical retrieval, software code data-flow pretraining, RTL language modeling, and software change localization. Graph baselines include HOGA (Deng et al., 2024) for hop-wise multiscale circuit aggregation and Dir-GNN (Rossi et al., 2024) for direction-aware message passing. Hardware-specific representation baselines include CircuitFusion (Fang et al., 2025), which fuses HDL, structural graphs, and functional summaries through RTL-netlist alignment, and UniRTL (Liu et al., 2026) for graph-aware CDFG tokenization and mutual masked modeling. Together, they test whether strong retrieval, static circuit topology, or RTL representations alone suffice for intent-conditioned modification-support recovery.

Evaluation metrics. We evaluate localization performance with mean average precision (MAP), mean reciprocal rank (MRR), and scale-adaptive Recall@k and Accuracy@k, where $k \in \{R_q, 2R_q\}$ and $R_q = |\mathcal{S}_q|$ is the number of reference support blocks. Table headers abbreviate R_q as R. MAP characterizes the overall ranking of all support blocks, whereas MRR characterizes the rank of the first support block. Recall@k is the fraction of support blocks covered by the top k candidates, whereas Accuracy@k credits a query as successful only when its entire evaluation support falls within the top k candidates. Results are averaged over three random seeds.

5.2 End-to-End Localization Performance

Table 2 shows that RTLocating achieves the best results on both overall ranking and support recovery. SweRank-Retriever, trained for software change localization, outperforms hardware representation methods that emphasize functional summaries, HDL, and structural graph alignment. This suggests that learning the functional implementation of the current design cannot substitute for task-specific adaptation to change localization. By recovering coordinated blocks through impact propagation, RTLocating improves MAP by 17.3% relative to SweRank-Retriever. This gain demonstrates that hardware-aware modeling further strengthens recovery of the full pre-change block support set.

5.3 Robustness across Modification and Design Scales

Figure 3 examines model robustness along two axes: support-set size and candidate design scale. RTLocating attains the best results across all support-set size and IP-scale bins. Its advantage persists in the larger bins, indicating favorable scalability to long and complex RTL designs.

5.4 Does Evidence Fusion Learn Intent-Specific Allocation?

The following analyses use frozen evidence encoders and impact propagation and evaluate rankings over the full candidate set.

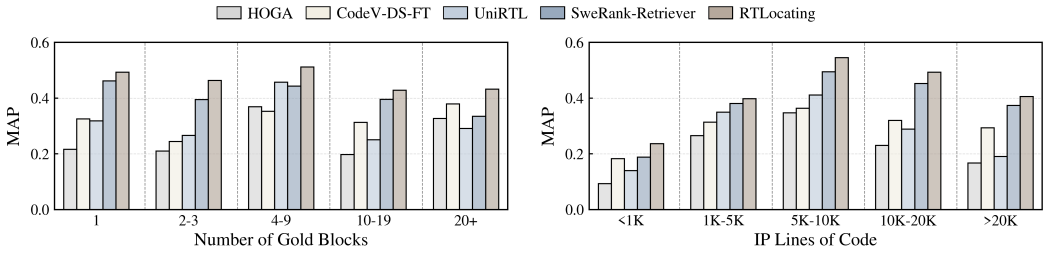


Figure 3: MAP stratified by modification-support size and IP code size.

Evidence complementarity. Intent-aware fusion raises MAP from 0.414 for the strongest individual source, textual semantics, to 0.448. Under the same top-R_q budget, functional and architectural evidence uniquely cover 7.27% and 12.02% of reference support blocks, showing that fusion expands coverage beyond commonly retrieved candidates. Figure 4 shows that the best-source share generally shifts from textual toward functional and architectural evidence as support size increases.

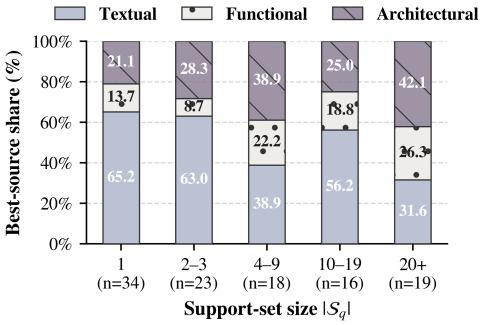


Figure 4: Best-source share by support size.

Fusion weights align with intent-specific utility. We define a source's realized utility as its leave-one-out contribution to AP. For 64% of queries, the source assigned the highest weight also yields the largest AP contribution, compared with 34% under a cross-query permutation of the fusion weights. This permutation preserves the marginal weight distribution and leaves the relevance scores and propagation parameters unchanged, yet reduces seed MAP from 0.448 to 0.432 and final MAP from approximately 0.471 to 0.443. This degradation suggests that fusion adapts source weights to each query's change intent rather than encoding a fixed global preference.

5.5 How Does Intent-Conditioned Impact Propagation Improve Localization?

Table 3: Factorized impact propagation from a shared, frozen multi-evidence semantic seed.

Variant	Channel $\boldsymbol{\pi}$	Scope $\boldsymbol{\omega}$	MAP ↑	MRR ↑	Recall @$2R$ ↑	Accuracy @$2R$ ↑
E0 Initial seed	-	-	0.448 ± 0.002	0.620 ± 0.002	0.526 ± 0.004	0.227 ± 0.000
E1 Homogeneous	homogeneous	$\bar{\boldsymbol{\omega}}$	0.437 ± 0.003	0.602 ± 0.002	0.538 ± 0.003	0.212 ± 0.005
E2 Global typed	$\bar{\boldsymbol{\pi}}$	$\bar{\boldsymbol{\omega}}$	0.445 ± 0.004	0.614 ± 0.003	0.547 ± 0.004	0.233 ± 0.005
E3 Channel-adaptive	$\boldsymbol{\pi}(q, G)$	$\bar{\boldsymbol{\omega}}$	$\underline{0.452 \pm 0.004}$	$\underline{0.628 \pm 0.003}$	$\underline{0.552 \pm 0.003}$	$\underline{0.236 \pm 0.000}$
E4 Scope-adaptive	$\bar{\boldsymbol{\pi}}$	$\boldsymbol{\omega}(q, G)$	0.449 ± 0.003	0.621 ± 0.002	0.550 ± 0.003	0.236 ± 0.009
E5 Full impact propagation	$\boldsymbol{\pi}(q, G)$	$\boldsymbol{\omega}(q, G)$	$\mathbf{0.471 \pm 0.003}$	$\mathbf{0.656 \pm 0.003}$	$\mathbf{0.564 \pm 0.003}$	$\mathbf{0.258 \pm 0.005}$

Note. Values are means ± sample standard deviations over three random seeds. $\bar{\boldsymbol{\pi}}$ and $\bar{\boldsymbol{\omega}}$ are query-independent mixtures shared across queries; their unbarred counterparts are query-conditioned.

Channel and scope adaptation provide complementary gains in impact propagation. We train E1–E5 independently from the same frozen semantic seed under an otherwise identical setup. E1 collapses all dependency channels into a single homogeneous operator. In E2–E5, any disabled adaptation uses a single set of weights shared across all queries with all other components held fixed. Relative to E0, homogeneous propagation (E1) increases Recall@$2R$ but reduces MAP and Accuracy@$2R$. Restoring dependency types (E2) largely reverses the MAP degradation of homogeneous propagation while improving Accuracy@$2R$ beyond the initial seed. Adapting either the channel mixture (E3) or the scope mixture (E4) improves MAP and Accuracy@$2R$ over E2, while adapting both (E5) performs best across all metrics.

Impact Propagation recovers support missed by the initial seed. We define reference support blocks in the seed top-R_q as anchors and those outside it but reachable from an anchor within the

maximum propagation depth as recovery targets. Such targets occur in 48.2% of test queries. Across these queries, propagation moves 48.7% of the targets into the final top-R_q. Because all targets lie outside the seed top-R_q, this measures new support recovery rather than within-set reranking.

5.6 Does Counterfactual Decision Supervision Improve Propagation Decisions?

Counterfactual decision supervision improves localization and structural-decision alignment. With propagation module fixed, we compare ranking-only training, query-matched counterfactual decision supervision, and a shuffled-target control. The control permutes joint channel-and-scope targets across queries, retaining the targets while breaking query–target correspondence. We measure structural alignment by the Jensen–Shannon divergence (JSD) between predicted decisions and utility-derived targets, where lower is better. The query-matched variant improves both localization and alignment over ranking-only training, whereas shuffled targets degrade both, indicating that the gains depend on query-specific utility rather than auxiliary supervision alone.

Table 4: Effect of counterfactual supervision on localization and structural decisions.

Training signal	MAP ↑	Channel JSD ↓	Scope JSD ↓
Ranking only	0.460	0.184	0.123
Shuffled targets	0.397	0.256	0.171
Counterfactual decision supervision	**0.471**	**0.096**	**0.071**

5.7 Does Explicit Localization Guidance Improve Downstream RTL Editing?

To assess downstream editing, we evaluate a single-turn LLM and an agentic coding system Codex on 30 test-set change requests under three localization-guidance conditions: RTLocating, SweRank-Retriever and Full-IP. Both editors use gpt-5.5 as the underlying model. For each evaluated change, we hold the change request q and editing budget fixed across conditions. Under RTLocating and SweRank-Retriever, the top $K = 10$ localized blocks are included in the initial prompt as explicit localization cues. Under Full-IP, the editor must identify the relevant code from the complete pre-change design. Unlike the single-turn LLM, Codex can search and iteratively revise the full codebase under all conditions. For each evaluated change, we compile and simulate the pre-change, post-change reference, and edited RTL with Verilator using the same testbench and stimulus sequence. We classify outcomes as full recovery, partial recovery, or no improvement/harm according to the fraction of the parent-to-golden behavioral gap closed. The formal metric and thresholds are given in Appendix D.6.1.

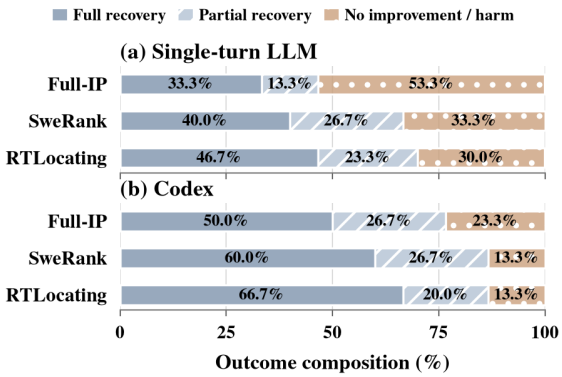


Figure 5: End-to-end editing outcomes.

With localization cues, both editors achieve higher rates of full recovery under SweRank and RTLocating than under Full-IP. RTLocating consistently outperforms SweRank, suggesting that part of the improvement arises from more accurate localization of support blocks. Although agentic search partly mitigates localization errors, explicit localization guidance still provides additional gains.

6 Conclusion

We formalized ΔSpec-to-RTL localization in hardware evolution and empirically revealed two regularities of real RTL modification. We introduced EvoRTL-Bench, the first benchmark for this task, and proposed RTLocating, which combines multi-evidence semantic anchoring with intent-conditioned typed dependency propagation trained via counterfactual decision supervision. Experimental results show that RTLocating improves RTL modification-support recovery and that these gains translate into better downstream RTL editing, thereby supporting reliable repository-level hardware evolution agents.

AI USE STATEMENT

We used GPT-4.1 to classify historical OpenTitan revisions and normalize commit messages into structured change requests for EvoRTL-Bench (Section 2.2). In the downstream editing experiments, we used GPT-5.5 as both a single-turn editor and the underlying model for Codex to generate RTL modifications (Section 5.7). We manually audited a sample of the AI-generated dataset text (Appendix A.5). We also used large language models to assist with language polishing of the manuscript and modifications to the research code. The authors reviewed the language revisions and verified and tested the modified research code for correctness. The authors take responsibility for the final content of this work, including AI-assisted text, data, code, and claims.

ETHICS STATEMENT

This work aims to support reliable hardware development and maintenance through RTL modification-support recovery. EvoRTL-Bench is constructed from the publicly available RTL code and revision history of OpenTitan. We document the data construction, filtering, and validation procedures to support transparency and reproducibility. We recognize the broader implications of AI-assisted hardware development and encourage human oversight and appropriate verification when applying localization results or generated RTL modifications. We are committed to rigorous experimentation, accurate reporting of results, and responsible use of AI in hardware engineering.

REPRODUCIBILITY STATEMENT

The appendices provide detailed descriptions of dataset construction, model implementation, and experimental protocols to support reproducibility. An anonymous implementation of RTLocating is available at `https://anonymous.4open.science/r/rtlocating-anonymous-0C84/`. EvoRTL-Bench will be released upon acceptance.

REFERENCES

Zhaoling Chen, Robert Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor Prasanna, Arman Cohan, and Xingyao Wang. Locagent: Graph-guided llm agents for code localization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8697–8727, 2025.

Fan Cui, Hongyuan Hou, Zizhang Luo, Chenyun Yin, and Yun Liang. Hwe-bench: Benchmarking llm agents on real-world hardware bug repair tasks. *arXiv preprint arXiv:2604.14709*, 2026.

Chenhui Deng, Zichao Yue, Cunxi Yu, Gokce Sarar, Ryan Carey, Rajeev Jain, and Zhiru Zhang. Less is more: Hop-wise graph attention for scalable and generalizable learning on circuits. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pp. 1–6, 2024.

Wenji Fang, Shang Liu, Jing Wang, and Zhiyao Xie. Circuitfusion: multimodal circuit representation learning for agile chip design. In *International Conference on Learning Representations*, volume 2025, pp. 51523–51545, 2025.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366*, 2020.

Mingjie Liu, Yun-Da Tsai, Wenfei Zhou, and Haoxing Ren. Craftrtl: High-quality synthetic data generation for verilog code models with correct-by-construction non-textual representations and targeted code repair. In *International Conference on Learning Representations*, volume 2025, pp. 90377–90422, 2025a.

Yi Liu, Changran Xu, Yunhao Zhou, Zeju Li, and Qiang Xu. Deeprtl: Bridging verilog understanding and generation with a unified representation model. *arXiv preprint arXiv:2502.15832*, 2025b.

Yi Liu, Hongji Zhang, Lei Chen, Mingxuan Yuan, and Qiang Xu. Unirtl: Unifying code and graph for robust rtl representation learning. *arXiv preprint arXiv:2605.31040*, 2026.

lowRISC. OpenTitan. https://github.com/lowRISC/opentitan, 2025.

Jan Malburg, Alexander Finder, and Görschwin Fey. Debugging hardware designs using dynamic dependency graphs. *Microprocessors and Microsystems*, 47:347–359, 2016.

Jingyu Pan, Guanglei Zhou, Chen-Chia Chang, Isaac Jacobson, Jiang Hu, and Yiran Chen. A survey of research in large language models for electronic design automation. *ACM Transactions on Design Automation of Electronic Systems*, 30(3):1–21, 2025.

Zehua Pei, Hui-Ling Zhen, Mingxuan Yuan, Yu Huang, and Bei Yu. Betterv: Controlled verilog generation with discriminative guidance. *arXiv preprint arXiv:2402.03375*, 2024.

Jie Qian, Xiaolin Ju, and Xiang Chen. Gnet4fl: effective fault localization via graph convolutional neural network. *Automated Software Engineering*, 30(2):16, 2023.

Revanth Gangi Reddy, Tarun Suresh, JaeHyeok Doo, Ye Liu, Xuan Phi Nguyen, Yingbo Zhou, Semih Yavuz, Caiming Xiong, Heng Ji, and Shafiq Joty. Swerank: Software issue localization with code ranking. *arXiv preprint arXiv:2505.07849*, 2025.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and trends® in information retrieval*, 4(1-2):1–174, 2009.

Emanuele Rossi, Bertrand Charpentier, Francesco Di Giovanni, Fabrizio Frasca, Stephan Günnemann, and Michael M Bronstein. Edge directionality improves learning on heterophilic graphs. In *Learning on graphs conference*, pp. 25–1. PMLR, 2024.

Gwok-Waa Wan, SamZaak Wong, Shengchu Su, Chenxu Niu, Ning Wang, Xinlai Wan, Qixiang Chen, Mengnv Xing, Jingyi Zhang, Jianmin Ye, et al. Fixme: Towards end-to-end benchmarking of llm-aided design verification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 1087–1095, 2026.

Ning Wang, Bingkun Yao, Jie Zhou, Yuchen Hu, Xi Wang, Zhe Jiang, and Nan Guan. Large language model for verilog generation with code-structure-guided reinforcement learning. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, pp. 164–170. IEEE, 2025.

Xi Wang, Gwok-Waa Wan, Sam-Zaak Wong, Layton Zhang, Tianyang Liu, Qi Tian, and Jianmin Ye. Chatcpu: An agile cpu design and verification platform with llm. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pp. 1–6, 2024.

Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. Chateda: A large language model powered autonomous agent for eda. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(10):3184–3197, 2024.

Haoyuan Wu, Haisheng Zheng, Yuan Pu, and Bei Yu. Circuit representation learning with masked gate modeling and verilog-aig alignment. In *International Conference on Learning Representations*, volume 2025, pp. 86310–86326, 2025.

Changwen Xing, SamZaak Wong, Xinlai Wan, Yanfeng Lu, Mengli Zhang, Zebin Ma, Lei Qi, Zhengxiong Li, Nan Guan, Zhe Jiang, et al. Chipmind: Retrieval-augmented reasoning for long-context circuit design specifications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 1337–1345, 2026.

Ke Xu, Jialin Sun, Yuchen Hu, Xinwei Fang, Weiwei Shan, Xi Wang, and Zhe Jiang. Meic: Re-thinking rtl debug automation using llms. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2024.

Bingkun Yao, Ning Wang, Jie Zhou, Xi Wang, Hong Gao, Zhe Jiang, and Nan Guan. Location is key: Leveraging llm for functional bug localization in verilog design. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7. IEEE, 2025.

Zhuo Zhang, Yan Lei, Xiaoguang Mao, Meng Yan, Xin Xia, and David Lo. Context-aware neural fault localization. *IEEE Transactions on Software Engineering*, 49(7):3939–3954, 2023.

Mingyu Zhao, Xun He, Jiawei Liu, Jianwang Zhai, and Chuan Shi. Topology matters in rtl circuit representation learning. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026.

Yang Zhao, Di Huang, Chongxiao Li, Pengwei Jin, Muxin Song, Yinan Xu, Ziyuan Nan, Mingju Gao, Tianyun Ma, Lei Qi, et al. Codev: Empowering llms with hdl generation through multi-level summarization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.

Pingqing Zheng, Jiayin Qin, Fuqi Zhang, Niraj Chitla, Zishen Wan, Shang Wu, Yu Kevin Cao, Caiwen Ding, and Yang Katie Zhao. Hdlxgraph: Bridging large language models and hdl repositories via hdl graph databases. In *2026 31st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 533–539. IEEE, 2026.

Qingyun Zou, Feng Yu, Hongshi Tan, Bingsheng He, and WengFai Wong. Is agentic ai ready for real-world hardware engineering? a deep dive with phoenix-bench. *arXiv preprint arXiv:2605.15226*, 2026.

A EvoRTL-Bench Construction and Validation

A.1 Data Source and Design Boundaries

EvoRTL-Bench is constructed from the public Git history of OpenTitan through commit `7302728723b80904fb79a909da9b45ce947a60c4` (October 3, 2025). We pair each commit with its unique parent and denote the designs before and after the revision by D_q^- and D_q^+. We retain only revisions that can be assigned unambiguously to a single target IP. Each D_q^- is a complete RTL snapshot of that IP at the parent revision, rather than an individual modified file. Each record contains a normalized request q, the pre-change design D_q^-, the observed support $\mathcal{S}_q$ of the reference revision among pre-change candidates, and addition events $\mathcal{A}_q^+$ without pre-change counterparts.

Source scale. SystemVerilog lines of code (LOC) provide a coarse proxy for design size. The 37 directories range from a few hundred lines of glue logic to 32,155 lines for `spi_device`. The median is 4,482 LOC, with an interquartile range of 1,906 to 8,954 LOC. Table 5 gives the distribution.

Functional coverage. The source spans nine categories of SoC subsystems. Table 6 lists their directory level coverage.

Table 5: Distribution of SystemVerilog code size across OpenTitan IP and RTL library directories.

Size category	LOC range	Directories	Share
Lightweight	< 1k	6	16.2%
Small or utility	$[1\text{k}, 5\text{k})$	15	40.5%
Medium	$[5\text{k}, 10\text{k})$	8	21.6%
Large	$[10\text{k}, 20\text{k})$	6	16.2%
Very large	≥ 20k	2	5.4%
Total		37	100.0%

Table 6: Functional categories and IP coverage in the OpenTitan source.

Subsystem	IP	Function
Cryptography	`aes`, `ascon`, `hmac`, `kmac`, `otbn`	Symmetric and authenticated encryption, hashing, message authentication, and large integer computation
Entropy and keys	`entropy_src`, `csrng`, `edn`, `keymgr`, `keymgr_dpe`, `lc_ctrl`, `racl_ctrl`	Entropy collection and testing, random number distribution, key derivation, lifecycle control, and access control
Compute and DMA	`rv_core_ibex`, `dma`	RISC-V core integration and transfers between memory and peripherals
Storage and boot	`flash_ctrl`, `otp_ctrl`, `otp_macro`, `rom_ctrl`, `sram_ctrl`	Access, integrity protection, scrambling, and boot verification for Flash, OTP, ROM, and SRAM
Peripherals	`adc_ctrl`, `pattgen`, `i2c`, `spi_host`, `spi_device`, `uart`, `usbdev`	ADC control, pattern generation, and I^2C, SPI, UART, and USB interfaces
Timers and reset	`aon_timer`, `rv_timer`, `sysrst_ctrl`	Wakeup, watchdogs, timer interrupts, and board reset control
Secure debug	`rv_dm`, `soc_dbg_ctrl`	RISC-V debugging, system bus access, and debug authorization subject to security policy
Interconnect	`tlul`, `mbx`	TL-UL interconnect and message channels
Primitives	`prim`, `prim_generic`, `prim_xilinx`, `prim_xilinx_ultrascale`	Technology-independent RTL primitives and their generic or Xilinx-specific implementations

A.2 Construction of Observed Modification Support

Candidates and annotations. For a revision pair $c = (D_q^-, D_q^+)$, we extract all behavioral constructs from the AST of D_q^-. These are continuous assignments and complete `always`, `always_comb`, `always_ff`, and `always_latch` procedural blocks. We deduplicate them by source within each module to obtain the candidate set $\mathcal{B}(D_q^-)$. A candidate is uniquely identified by its repository relative path, module, AST type, and source interval in D_q^-.

Table 7: Rules for mapping reference diffs to the candidate space.

Change type	Mapping rule
Block modification	Map to every intersecting complete candidate
Block deletion	Add the deleted block from D_q^- to $\mathcal{S}_q$
In-block insertion	Map only if the insertion point lies within a unique pre-change candidate
Top-level block addition	Identify an addition event using the new-side AST
Module or file addition	Record a module- or file-level addition event
Declaration or connection change	Record provenance without treating the changed construct as a candidate
Comment or formatting change	Do not mark overlapping candidates as modified
Macro expansion	Map to all matching source candidates; discard repeated matches
Unresolvable source mapping	Exclude the revision

Table 8: Fields in the structured ΔSpec.

Field	Definition
`Change type`	`Functional` changes externally observable or software visible RTL behavior. `Non-Functional` covers behavior preserving refactoring, movement, renaming, PPA optimization, and documentation or verification changes. `Unclear` denotes insufficient information for a reliable judgment
`Confidence`	Heuristic quality score comprising description clarity, from 0 to 0.8, and internal consistency, from 0 to 0.2. It is not a calibrated probability
`Rationale`	Basis for the `label` and `confidence` assignments
`S_old` and `S_new`	Summaries of RTL behavior before and after the revision
`Context`	Prior functional context explicitly stated in the commit message, excluding the change itself and domain based inferences
`Intention`	Design motivation or purpose expressed in the commit message

Let $\mathcal{H}_q^{\text{pre}}$ be the behaviorally relevant hunks in the reference diff that involve pre-change candidates, and let $M(h, D_q^-)$ map hunk h to candidate blocks. The observed modification support is

$$\mathcal{S}_q = \bigcup_{h \in \mathcal{H}_q^{\text{pre}}} M(h, D_q^-) \subseteq \mathcal{B}(D_q^-).$$

We separately record newly added top level blocks, modules, or files without pre-change counterparts in the addition event set $\mathcal{A}_q^+$. Each event is uniquely identified by its path on the new side, module, event type, and source interval. A new file has event type `source_file`. The reference-derived annotation for revision pair q is $\mathcal{Y}_q = (\mathcal{S}_q, \mathcal{A}_q^+)$. The set $\mathcal{S}_q$ records pre-change candidates that the reference revision actually modifies or deletes. It is not necessarily the unique or smallest set sufficient to implement the request. Addition events in $\mathcal{A}_q^+$ do not participate in block ranking.

Mapping rules. We normalize the reference diff by paths on the old and new sides, file status, and line intervals. Context lines are used only for alignment and do not determine candidate matches. Every behaviorally relevant change must map to $\mathcal{S}_q$ or be recorded in $\mathcal{A}_q^+$. Otherwise, we exclude the revision. A block matched by multiple hunks is counted once. Table 7 gives the rules.

A.3 Natural Language Request Generation and Quality Control

We use gpt-4.1 with temperature 0, top-p 1, and seed 42 to normalize the subject and body of each commit into a structured ΔSpec. The model does not see the reference diff, the list of modified files, RTL source, the AST, or observed support. It returns a single JSON object with the fields in Table 8.

We retain only outputs with `label=Functional` and `confidence` ≥ 0.85. Both `S_old` and `S_new` must be nonempty and contain no more than two sentences each. The four fields used to form the request must be supported by the original commit message. They must not contain file paths, line numbers, block identifiers, diff syntax, or verbatim RTL fragments, except for hardware identifiers present in the original message. Automated checks enforce field completeness, types, summary length, and these exclusions. An invalid output is regenerated at most once and then discarded if it remains invalid. The final request q concatenates the four fields using fixed markers:

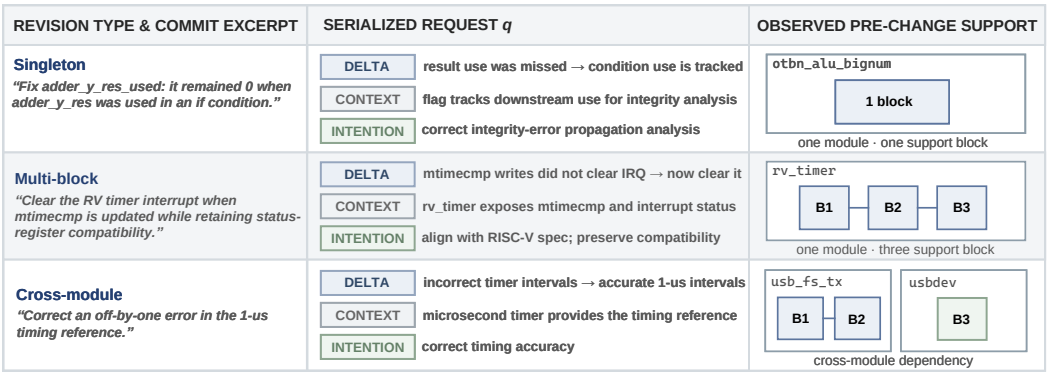


Figure 6: Representative serialized requests and observed pre-change support.

```
[DELTA] S_old [SEP] S_new
[CONTEXT] Context
[INTENTION] Intention
```

Compact examples. Figure 6 gives three actual examples covering a single candidate, multiple blocks in one module, and changes across modules. The original messages are excerpted to show the behaviorally relevant content. The released data retain the complete strings.

Revision filtering. Figure 7 summarizes the revisions retained at each stage. To prevent reference-implementation leakage, q is derived solely from the commit description, while $\mathcal{S}_q$ is independently constructed from the reference diff and D_q^-. Each retained revision yields one localization request.

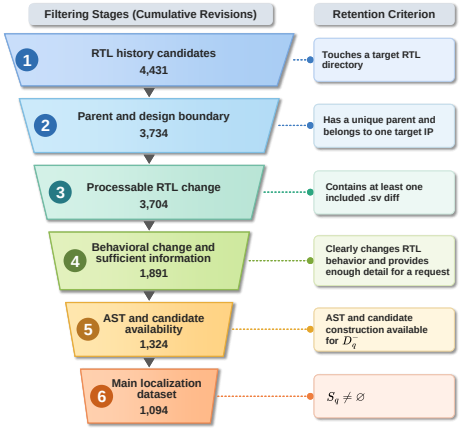


Figure 7: Cumulative revision counts and retention criteria across filtering stages.

A.4 Graph Construction

For each pre-change design, we construct a local dataflow graph (DFG) for every candidate block, resolve design wide signals and hierarchical connections, and lift the resulting relations to a Design-Wide Topology Graph (DTG). The process reads only pre-change design D_q^- and its compilation and elaboration configuration. DTG can be precomputed and cached for each design version.

Local DFG construction. For each candidate block $b \in \mathcal{B}(D)$, we extract a local DFG from the elaborated AST. Nodes encode signals and operations, while fine-grained relations capture evaluation, control, access, assignment, timing, and type dependencies (Table 9). Block ownership and instance context are retained to identify signal definitions, uses, and timing contexts.

DTG construction. We define the directed, typed DTG of D as $G_D = \big(\mathcal{B}(D), E_D, \mathrm{dep}_D, \mathrm{time}_D\big)$, where $\mathcal{B}(D)$ is the set of candidate RTL blocks and E_D is the set of directed dependency edges. For each $e \in E_D$, $\mathrm{dep}_D(e)$ gives its dependency class and $\mathrm{time}_D(e)$ its timing class.

We canonicalize signals across local DFGs, resolve hierarchical connections through elaborated instance ports, and map fine-grained relations to the semantic classes in Table 9. Timing classes are assigned from consumer use contexts, with uses classified as Combinational when they do not cross a state-retaining boundary and as Sequential otherwise, including edge-triggered, level-sensitive, and other state-retaining updates. Here, $\mathrm{dep}(\ell)$ maps relations to dependency classes, while $\mathrm{time}(\xi)$ maps use contexts to timing classes. Producers come from assignment left-hand sides, and consumers from data, control, address/index, and event uses.

For each canonical signal s, we collect its defining blocks $\mathrm{Def}_D(s)$ and the blocks $\mathrm{Use}_D^{\ell,\xi}(s)$ that use it under relation ℓ and timing context ξ. Lifting both endpoints to their owning blocks and applying

Table 9: Mapping from fine grained relations to semantic dependency classes.

Dependency class	Fine grained relations	Criterion	Direction between blocks	RTL example
Control	`cond_expr`, `control`, `event`, `op_ctrl`, `pred_data`, `predicate`	A signal controls a condition, selection, enable, predicate, or event other than a clock	Producer → controlled block	`if (en) q <= d`
Datapath	`addr`, `compute`, `op_in`	A signal is used for data, operands, indices, or address computation	Producer → consumer	`assign y = a + b`
Clock/Reset	`clock`, `reset`, `latch`	A signal determines a clock, reset, or state retention	Source → block updating state	`always_ff @(posedge clk)`
Structural	`left_value`, `typealias`	An assignment target or type association	Direction of the underlying relation	Assignment target or `typedef` association

the two class mappings yields the semantic relations between blocks:

$$\widetilde{\mathcal{E}}_D^{\text{sem}} = \left\{ (b_u, b_v, \text{dep}(\ell), \text{time}(\xi)) : \exists s,\, b_u \in \text{Def}_D(s),\, b_v \in \text{Use}_D^{\ell,\xi}(s),\, b_u \neq b_v \right\}.$$

Assignment target and type association relations are lifted to Structural relations using the same provenance rule. The former inherit the timing of the enclosing assignment or process, while the latter are Combinational. We omit relations that cannot be mapped to two distinct candidate blocks.

We lift hierarchical connections along resolved signal flow: for input ports, the producer of the parent's actual signal points to blocks using the child's formal signal; for output ports, blocks driving the child's formal signal point to blocks using the parent's actual signal. An inout direction is assigned only when both its driver and use are resolved. We add no artificial environment nodes for top-level external ports and omit relations whose endpoints, dependency class, or timing class cannot be reliably determined.

Propagation-channel schema. Propagation uses the Cartesian product of the Control, Datapath, and Clock/Reset dependency classes with the Combinational and Sequential timing classes, yielding six channels. Structural relations retain assignment-target and type-association provenance for representation learning, but are excluded because they do not encode directed runtime signal influence. We use $\mathcal{C}(G)$ to denote the nonempty propagation channels in design G.

Construction cost and DTG size. We measure offline DTG construction cost and graph size for the 1,094 distinct pre-change design versions in the main dataset. Construction time includes elaboration, local DFG extraction, hierarchical connection resolution, lifting to blocks, and normalization. We measure graph size by the number of candidate blocks, deduplicated typed edges, and candidate blocks with no valid dependency connections. Table 10 reports construction cost in panel (a) and DTG size in panel (b).

Table 10: Offline DTG construction cost and size statistics across pre-change design versions.

(a) Construction cost

Statistic	Value
Design versions	1,094
Min / median / mean time (s)	0.7 / 9.4 / 17.6
P90 / P95 time (s)	34.1 / 56.3
Maximum time (s)	114.8
Total time (s)	19,254.4

(b) DTG size

Metric	Min	Median	Mean	Max
Candidate blocks	27	411	484.0	1,245
Deduplicated typed edges	133	16,318	60,325.2	580,073
Unconnected candidate blocks	0	12	12.9	83

A.5 Audit of Functional Filtering and Request Quality

The reliability of functional filtering and request generation is assessed through a manual audit.

Audit population and stratified sampling. The audit population comprises the 1,094 instances in the main dataset, with 875 in Train, 109 in Validation, and 110 in Test. We sample 500 instances using quotas of 400, 50, and 50 for these respective splits. Joint strata are defined by split, IP, and generated confidence score. We first allocate one instance to each nonempty stratum and assign the remaining slots by the largest remainder method in proportion to stratum size. Table 11 reports population and sample counts.

Table 11: Coverage of stratification variables and complex cases in the manual audit.

Coverage dimension	Group	Population	Audited
Generated confidence	$[0.85, 0.90)$	147	81
	$[0.90, 0.95)$	253	125
	$[0.95, 1]$	694	294
Support size	1 block	336	142
	2 to 5 blocks	418	197
	6 to 10 blocks	131	61
	More than 10 blocks	209	100
Support topology	Single candidate	336	142
	Multiple blocks in one module	326	155
	Multiple blocks across modules	432	203
Revision form	pre-change side only	1,026	467
	Mixed	68	33

Table 12: Manual audit criteria, evidence boundaries, and acceptance conditions.

Criterion	Main evidence	Acceptance condition
Correct change type	D_q^-, D_q^+, and complete diff	The revision changes externally observable or software visible RTL behavior, rather than being a behavior preserving maintenance change or a change with insufficient evidence
Fidelity of S_{old} and S_{new}	Original message and designs before and after the revision	The summaries accurately describe pre-change and revised behavior without reversing essential semantics or adding claims inconsistent with the revision
Fidelity of Context and Intention	Original message and full q. The designs are used only to check for contradictions	Both fields are explicitly supported by the original message. `Context` describes only relevant pre-change functionality, and `Intention` describes only the purpose of the change. Their roles remain distinct and neither contradicts the revision
Exclusion of location hints	Complete serialized request q	No file paths, line numbers, block or candidate identifiers, diff syntax, verbatim RTL fragments, or functionally equivalent direct location hints. Hardware identifiers explicitly present in the original message do not by themselves constitute leakage
Exclusion of retrospective implementation details	Original message and full q	Entities, conditions, operations, and purposes in q are supported by the original message. Details available only from the patch or RTL fail even if factually correct

Two-stage independent review. Three experienced RTL design and verification practitioners conduct the audit: two independently review each instance in two ordered stages, and a third adjudicates disagreements. During request review, annotators see only the original commit message, generated fields, and final request q, which they assess for fidelity, location hints, and post-revision implementation details; the diff, RTL source, automated label, and observed support remain hidden. After locking these judgments, they inspect the pre- and post-revision designs and complete reference diff to determine the change type and verify the generated content against the actual revision. The automated change type label is revealed for comparison only after these judgments are locked. The adjudicated conclusions constitute the final audit results.

Audit criteria and results. Each criterion receives a supported pass, fail, or not-applicable judgment, and an instance passes overall only if all applicable criteria pass. Table 12 defines the evidence and acceptance conditions. For the 500 audited instances, Table 13 reports applicable counts and adjudicated pass rates for each criterion and for passing all applicable criteria.

Table 13: Results of the manual audit of functional filtering and request quality.

Criterion	Applicable n	Passed n (rate)
Correct `Functional` label	500	497 (99.4%)
Fidelity of S_{old} and S_{new}	500	493 (98.6%)
Fidelity of `Context` and `Intention`	500	495 (99.0%)
Exclusion of location hints	500	499 (99.8%)
Exclusion of retrospective implementation details	500	494 (98.8%)
All applicable criteria	500	491 (98.2%)

A.6 Data Limitations

These audits improve the traceability of the data, but several limitations remain. First, $\mathcal{S}_q$ records locations modified in the reference revision, not the unique or smallest support sufficient to implement the request. Second, normalized requests derive from historical commit messages, which may omit unstated intentions and favor well described changes. Third, the main task ranks only pre-change candidate blocks. Newly added structures are recorded separately as addition events, and changes that cannot otherwise be represented in the candidate space are excluded. Fourth, a single open source project does not represent private RTL, other coding conventions, or other toolchains. Finally, manual audits use stratified samples of retained instances. They neither validate every instance nor estimate the rate at which functional revisions were missed during filtering.

B Details of Empirical Analyses

This section details the metrics, training protocols, sample selection, statistical tests, results, and robustness analyses for RQ1 and RQ2.

B.1 RQ1 Conditional Null Models

Randomization operates at the query level and resamples the support set while holding the pre-change DTG and support size fixed. The module-only null preserves the number of support blocks within each module. The primary joint null used in the main text additionally preserves that number within each joint stratum. A stratum is determined by module identity and the quartile bins of the candidate node's in-degree and out-degree: $h = (\text{module}, Q_{\text{in}}, Q_{\text{out}})$. The bins Q_{in} and Q_{out} are formed from the quartile cutoffs of $\log(1 + d_{\text{in}})$ and $\log(1 + d_{\text{out}})$ over all candidate nodes in the design. Nodes with tied degrees are not split between bins. Repeated cutoffs and empty bins are removed, and zero degree nodes enter the lowest nonempty bin. Sampling is independent across strata. Let $\mathcal{H}_q$ be the set of nonempty strata for query q. For each $h \in \mathcal{H}_q$, let V_{qh} contain its candidate nodes, $N_{qh} = |V_{qh}|$, and $k_{qh} = |\mathcal{S}_q \cap V_{qh}|$. At repetition b, we sample k_{qh} nodes uniformly without replacement from each V_{qh}, or equivalently sample uniformly among its subsets of that size. The sampled subsets form one control support set $\widetilde{\mathcal{S}}_{q,b}$. Observed support nodes remain eligible for sampling. Neither null model constrains connectivity, distances, boundaries, or type composition.

B.2 RQ1 Metrics and Effect Sizes

B.2.1 Support Density and Topological Endpoints

Let G_q^{w} be the unweighted, weak, simple graph obtained from DTG G_q by retaining all vertices, ignoring edge directions, and collapsing relations with the same endpoints. Thus, node $V(G_q^{\text{w}}) = V_q = V(G_q)$. Both distance and boundary metrics are computed per query on G_q^{w} and then averaged with equal weight over the valid queries.

Support ratio. We define the descriptive support ratio as $\eta_q = \frac{|\mathcal{S}_q|}{|V_q|}$, where $\mathcal{S}_q$ is the observed modification support for query q and V_q comprises all candidate RTL blocks in the DTG of its pre-change design D_q^-.

Harmonic proximity. Let $d_{\mathrm{w}}(u, v)$ denote shortest path distance in G_q^{w}. For $|\mathcal{S}_q| \geq 2$, define

$$H_q = \frac{1}{\binom{|\mathcal{S}_q|}{2}} \sum_{\{u,v\} \subseteq \mathcal{S}_q} \frac{1}{d_{\mathrm{w}}(u, v)}, \qquad \frac{1}{\infty} = 0.$$

Unreachable pairs contribute zero. Larger H_q indicates closer support blocks. H_q is undefined for a singleton because it contains no block pair.

Conductance and boundary compactness. Let $\overline{\mathcal{S}}_q = V_q \setminus \mathcal{S}_q$. We define support set conductance ϕ_q and boundary compactness C_q by

$$\phi_q = \frac{\mathrm{cut}(\mathcal{S}_q, \overline{\mathcal{S}}_q)}{\min\{\mathrm{vol}(\mathcal{S}_q), \mathrm{vol}(\overline{\mathcal{S}}_q)\}}, \qquad C_q = 1 - \phi_q,$$

where $\mathrm{cut}(A, \overline{A})$ counts edges crossing the set boundary and $\mathrm{vol}(A) = \sum_{u \in A} \deg_{G_q^{\mathrm{w}}}(u)$. Larger C_q indicates fewer boundary connections relative to graph volume and is undefined if either side has zero volume. For a nonisolated singleton, $\mathrm{cut} = \mathrm{vol}(\mathcal{S}_q) = \deg(u)$, so $C_q = 0$.

B.2.2 Typed Composition along Dependency Paths

To assess whether functional changes exhibit nonrandom preferences for dependency and timing classes, we measure the edge attribute composition of dependency paths between modified support blocks. As with the scalar endpoints in Section B.2.1, compositions are computed per query before taking an equally weighted mean across valid queries.

Composition space and hop masses. Let the attribute axis be $a \in \{\mathrm{dep}, \mathrm{time}\}$. The dependency axis has the ordered categories Control, Datapath, Clock/Reset, and Structural. The timing axis has Combinational and Sequential. The number of categories and the dimension of Helmert coordinates are $K_{\mathrm{dep}} = 4, K_{\mathrm{time}} = 2, d_a \triangleq K_a - 1$.

For each directed relation record $e \in E_q$, define its attribute by

$$\mathrm{attr}_{q,a}(e) = \begin{cases} \mathrm{dep}_q(e), & a = \mathrm{dep}, \\ \mathrm{time}_q(e), & a = \mathrm{time}. \end{cases}$$

For a weak edge $\ell = \{r, s\}$, collect all directed typed records between its endpoints in both directions: $\mathcal{R}_\ell = \{e \in E_q : \{\mathrm{src}(e), \mathrm{dst}(e)\} = \{r, s\}\}$. The mass of category $k \in \{1, \ldots, K_a\}$ on this hop is $w_{\ell k}^{(a)} = \frac{1}{|\mathcal{R}_\ell|} \sum_{e \in \mathcal{R}_\ell} \mathbf{1}\{\mathrm{attr}_{q,a}(e) = k\}$, and $\mathbf{w}_\ell^{(a)} = (w_{\ell 1}^{(a)}, \ldots, w_{\ell K_a}^{(a)})$.

Observed composition per query. We first aggregate over paths for each support block pair and then over pairs within a query. The weakly connected support pairs are

$$\mathcal{P}_q = \{\{u, v\} \subseteq \mathcal{S}_q : d_{\mathrm{w}}(u, v) < \infty\}, \qquad m_q = |\mathcal{P}_q|.$$

For each $\{u, v\} \in \mathcal{P}_q$, let $\Gamma_q(u, v)$ contain up to four canonical shortest paths generated from both endpoints under ascending and descending stable node orders. The pair composition averages hop masses within each path and then across these paths:

$$\mathbf{h}_{q,uv}^{(a)} = \frac{1}{|\Gamma_q(u, v)|} \sum_{\gamma \in \Gamma_q(u,v)} \frac{1}{|\gamma|} \sum_{\ell \in \gamma} \mathbf{w}_\ell^{(a)},$$

where $|\gamma|$ is the number of hops. Giving each weakly connected support pair equal weight yields the query composition $\mathbf{p}_q^{(a)} = \frac{1}{m_q} \sum_{\{u,v\} \in \mathcal{P}_q} \mathbf{h}_{q,uv}^{(a)}$. Unreachable pairs contribute no type mass. Composition is undefined if $m_q = 0$.

Coordinates for probability vectors. Because $\mathbf{p}_q^{(a)}$ lies on the probability simplex, direct means and covariances are affected by its sum constraint. For $m_q > 0$, we treat $m_q p_{qk}^{(a)}$ as the effective

mass of category k and add 0.5 pseudomass to every category:

$$\widetilde{p}_{qk}^{(a)} = \frac{m_q p_{qk}^{(a)} + 0.5}{m_q + 0.5K_a}.$$

Write $\widetilde{\mathbf{p}}_q^{(a)} = (\widetilde{p}_{q1}^{(a)}, \ldots, \widetilde{p}_{qK_a}^{(a)})$. Its centered log ratio (CLR) coordinates are

$$\mathrm{CLR}_k\left(\widetilde{\mathbf{p}}_q^{(a)}\right) = \log \widetilde{p}_{qk}^{(a)} - \frac{1}{K_a}\sum_{j=1}^{K_a} \log \widetilde{p}_{qj}^{(a)}, \qquad k = 1, \ldots, K_a.$$

Let $V_{K_a} \in \mathbb{R}^{K_a \times (K_a - 1)}$ be a fixed orthonormal Helmert basis. Define

$$\mathbf{c}_q^{(a)} = \mathrm{CLR}\left(\widetilde{\mathbf{p}}_q^{(a)}\right), \qquad \mathbf{z}_q^{(a)} = \mathbf{c}_q^{(a)} V_{K_a} \in \mathbb{R}^{K_a - 1}.$$

The centered log ratio vector $\mathbf{c}_q^{(a)}$ removes the simplex constraint. The vector $\mathbf{z}_q^{(a)}$ represents it in d_a Euclidean dimensions. The overall composition effect is measured in the $\mathbf{z}$ space. Category level directions are interpreted in the $\mathbf{c}$ space.

B.2.3 Effect Sizes and Tests

Per query effects. For $x \in \{H, C\}$, let $\mathcal{Q}_x$ comprise the queries for which both the observed metric and every randomized value are defined, and let $n_x = |\mathcal{Q}_x|$. We measure the difference between the observed and mean randomized metrics within each query, followed by an equally weighted average over valid queries:

$$\delta_q^{(x)} = x(\mathcal{S}_q) - \frac{1}{B}\sum_{b=1}^{B} x(\widetilde{\mathcal{S}}_{q,b}), \qquad \Delta_x = \frac{1}{n_x}\sum_{q \in \mathcal{Q}_x} \delta_q^{(x)}.$$

The one-sided alternatives posit that actual support has larger H and C than conditional random support. Define

$$T_{\mathrm{obs}}^{(x)} = \frac{1}{n_x}\sum_{q \in \mathcal{Q}_x} x(\mathcal{S}_q), \qquad T_b^{(x)} = \frac{1}{n_x}\sum_{q \in \mathcal{Q}_x} x(\widetilde{\mathcal{S}}_{q,b}).$$

We compute their one-sided Monte Carlo p values using the procedure below.

Aggregation of composition across queries. At repetition b, we compute $\mathbf{c}_{q,b}^{(a)}$ and $\mathbf{z}_{q,b}^{(a)}$ from $\widetilde{\mathcal{S}}_{q,b}$ using exactly the same rules as for observed support. Let $\mathcal{Q}_a$ comprise queries for which observed composition and all B randomized compositions are defined, and let $n_a = |\mathcal{Q}_a|$. The observed and randomized centers are

$$\bar{\mathbf{z}}_{\mathrm{obs}}^{(a)} = \frac{1}{n_a}\sum_{q \in \mathcal{Q}_a} \mathbf{z}_q^{(a)}, \qquad \bar{\mathbf{z}}_{0,b}^{(a)} = \frac{1}{n_a}\sum_{q \in \mathcal{Q}_a} \mathbf{z}_{q,b}^{(a)},$$

$$\bar{\mathbf{z}}_0^{(a)} = \frac{1}{B}\sum_{b=1}^{B} \bar{\mathbf{z}}_{0,b}^{(a)}, \qquad \Sigma_0^{(a)} = \mathrm{Cov}_{b=1}^{B}\left(\bar{\mathbf{z}}_{0,b}^{(a)}\right).$$

Here $\bar{\mathbf{z}}_{\mathrm{obs}}^{(a)}$ is the empirical center of observed compositions, $\bar{\mathbf{z}}_{0,b}^{(a)}$ is the center at randomized repetition b, and $\bar{\mathbf{z}}_0^{(a)}$ and $\Sigma_0^{(a)}$ are the mean and covariance of randomized centers in Helmert coordinates.

Overall composition effect. With the null center and regularized covariance held fixed, define $D_a = D^{(a)}(\bar{\mathbf{z}}_{\mathrm{obs}}^{(a)})$, where

$$D^{(a)}(\mathbf{z}) = \sqrt{(\mathbf{z} - \bar{\mathbf{z}}_0^{(a)})^\top (\Sigma_0^{(a)} + \lambda_a I)^{+} (\mathbf{z} - \bar{\mathbf{z}}_0^{(a)})}, \quad \text{where} \lambda_a = 10^{-6} \begin{cases} \mathrm{tr}(\Sigma_0^{(a)})/d_a, & \mathrm{tr}(\Sigma_0^{(a)}) > 0, \\ 1, & \text{otherwise.} \end{cases}$$

Category level decomposition. The observed and null CLR centers are

$$\bar{\mathbf{c}}_{\mathrm{obs}}^{(a)} = \frac{1}{n_a}\sum_{q \in \mathcal{Q}_a} \mathbf{c}_q^{(a)}, \qquad \bar{\mathbf{c}}_0^{(a)} = \frac{1}{Bn_a}\sum_{b=1}^{B}\sum_{q \in \mathcal{Q}_a} \mathbf{c}_{q,b}^{(a)}.$$

Table 14: RQ1 sample funnel.

Endpoint or stage	Initial	Valid n	Eligibility or exclusion rule
Support ratio η_q	1,094	1,094	None; all support sets are nonempty.
Primary joint null pool	1,094	556	Require sufficient r_q and Ω_q.
Harmonic proximity H	556	440	Exclude singletons.
Boundary compactness C	556	544	Exclude queries with undefined conductance for the observed or any null support; retain defined singletons.
Dependency composition	556	428	Exclude singletons and queries without a weakly connected support pair.
Timing composition	556	428	Use the same valid queries as dependency composition.

For category k, the difference is $\Delta^{(a)}_{\mathrm{CLR},k} = \bar{c}^{(a)}_{\mathrm{obs},k} - \bar{c}^{(a)}_{0,k}$.

Randomization tests. The scalar endpoints H and C, the overall effects D_a, and the category components $\Delta^{(a)}_{\mathrm{CLR},k}$ use $B = 600$ conditional randomizations as their null distributions, with a $+1$ Monte Carlo correction. For an upper tail statistic T, with observed value T_{obs} and randomized values T_b, the empirical probability is

$$p = \frac{1 + \#\{b : T_b \geq T_{\mathrm{obs}}\}}{B + 1}.$$

Adding one to both numerator and denominator avoids a zero p value. For scalar endpoints, we substitute $T^{(x)}_{\mathrm{obs}}$ and $T^{(x)}_b$ into this expression for the one-sided alternative. For the overall composition test, we use $T_{\mathrm{obs}} = D_a$ and $T_b = D_{a,b}$, where $D_{a,b} = D^{(a)}\left(\bar{\mathbf{z}}^{(a)}_{0,b}\right)$. To test either enrichment or depletion of category k, we center the randomized CLR components at $\bar{\mathbf{c}}^{(a)}_0$ and use

$$p_{a,k} = \frac{1 + \#\left\{b : \left|\bar{c}^{(a)}_{0,b,k} - \bar{c}^{(a)}_{0,k}\right| \geq \left|\Delta^{(a)}_{\mathrm{CLR},k}\right|\right\}}{B + 1}.$$

Multiplicity and intervals. The four overall endpoints $\{H, C, D_{\mathrm{dep}}, D_{\mathrm{time}}\}$ form one Holm family. We interpret category directions only if the corresponding overall endpoint has a Holm adjusted $p < 0.05$. Within each of the dependency and timing axes, we adjust the K_a category level tests by the Benjamini and Hochberg procedure. Because D_a is nonnegative and has no direction on the original scale, we do not report a bootstrap confidence interval for it. Its randomization p value expresses uncertainty. The 95% confidence intervals for scalar effect sizes H and C use 2,000 percentile bootstrap resamples of queries.

B.3 RQ1 Inclusion Criteria and Valid Samples

RQ1 requires sufficient variation under conditional randomization. Define the replaceable support fraction r_q as the fraction of observed support nodes that lie in unsaturated strata satisfying $0 < k_{qh} < N_{qh}$. Let Ω_q be the number of possible control support sets, namely the product of stratum specific binomial coefficients under the stratification in Section B.1. The primary analysis requires $r_q \geq 0.8$. A query with multiple support blocks additionally requires $\Omega_q \geq 1000$, whereas a singleton requires $\Omega_q \geq 20$. Composition endpoints require at least one weakly connected support pair in the observed set and in every control set. These restrictions depend only on support occupancy and candidate graph structure. Consequently, the conclusions concern the subpopulation of queries for which joint matching is feasible. Table 14 summarizes the filtering stages, endpoint-specific exclusions, and valid query counts.

B.4 Complete RQ1 Results and Sensitivity Analysis

Under the primary joint null, observed supports exhibit higher harmonic proximity and boundary compactness, and both composition endpoints differ significantly from the null, as shown in Table 15. Figure 8 presents the category-level decomposition of dependency and timing composition, both evaluated on the same 428 valid queries.

To assess sensitivity to degree matching, we compare the module-only and joint nulls on Test, where they admit 91 and 56 queries with sufficient randomization freedom, respectively. Table 16 restricts

Table 15: RQ1 endpoints under the primary null matched by module and degree bins.

Endpoint	n	Observed / null mean	Effect	95% CI	Raw p	Holm p
Harmonic proximity H	440	0.5330 / 0.4278	$\Delta = +0.1051$	[0.0934, 0.1186]	0.001664	0.006656
Boundary compactness C	544	0.1524 / 0.0974	$\Delta = +0.0550$	[0.0474, 0.0636]	0.001664	0.006656
Dependency composition	428	Not applicable	$D_{\text{dep}} = 11.4095$	Not applicable	0.001664	0.006656
Timing composition	428	Not applicable	$D_{\text{time}} = 3.3212$	Not applicable	0.001664	0.006656

Note. D denotes a standardized composition distance. We report 95% confidence intervals for scalar effects only; composition effects are assessed using randomization p-values.

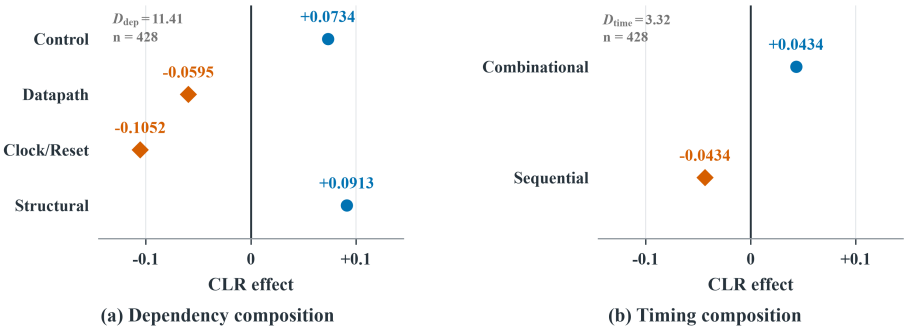


Figure 8: Category-level CLR decomposition under the primary null matched by module and degree bins. (a) Dependency composition. (b) Timing composition.

Note. CLR signs indicate relative enrichment (+) or depletion (-) against the null; D is an unsigned, covariance-standardized overall composition distance. All category tests use 600 conditional randomizations with a $+1$ Monte Carlo correction, yielding raw and BH-adjusted $p = 0.001664$, with BH applied separately within each composition. CLR components sum to zero within each composition, so category effects are dependent.

this comparison to common valid queries for each endpoint. Degree matching reduces both locality effects, but their confidence intervals remain above zero, and both composition effects remain significant. This analysis controls degree through coarse bins; the main-text conclusions remain based on the pooled joint-null results.

B.5 RQ2 Prediction Targets and Valid Samples

Prediction targets. We derive three continuous targets directly from the observed support $\mathcal{S}_q$. Harmonic dispersion D_H uses the weak simple graph G_q^{w} and harmonic proximity H defined in Section B.2.1. Let $m(u)$ be the module containing node u. The targets are

$$D_H(q) = 1 - H(\mathcal{S}_q),$$
$$M_q = \frac{1}{\binom{|\mathcal{S}_q|}{2}} \sum_{\{u,v\} \subseteq \mathcal{S}_q} \mathbf{1}\{m(u) \neq m(v)\},$$
$$Y_K(q) = \log(1 + K_q), \qquad K_q = |\mathcal{S}_q|.$$

D_H measures distance or disconnection in the weak DTG, M_q measures the proportion of support pairs spanning modules, and K_q is the observed support size. The first two targets are defined only for $|\mathcal{S}_q| \geq 2$. A query with multiple support blocks that are all mutually disconnected has $D_H = 1$.

Inclusion criteria and valid samples. All three targets use the same observed support. The two pairwise targets require at least two support blocks, whereas support size retains singletons. Table 17 gives the valid and total instance counts for each split.

B.6 Predictor Construction and Training

Both predictors are Ridge regressors with an intercept. We train a separate model for each of the three targets.

Table 16: Comparison of module-only and joint null models on common Test sets.

Endpoint	n	Module-only effect (95% CI)	Raw / Holm p	Joint effect (95% CI)	Raw / Holm p
Harmonic proximity H	45	$+0.1025$ [0.0687, 0.1362]	0.001664 / 0.006656	$+0.0722$ [0.0498, 0.0965]	0.001664 / 0.006656
Boundary compactness C	48	$+0.0935$ [0.0561, 0.1405]	0.001664 / 0.006656	$+0.0632$ [0.0375, 0.0936]	0.001664 / 0.006656
Dependency composition	44	$D = 3.5590$ CI not applicable	0.008319 / 0.016639	$D = 3.4799$ CI not applicable	0.008319 / 0.016639
Timing composition	44	$D = 2.8808$ CI not applicable	0.008319 / 0.016639	$D = 1.9749$ CI not applicable	0.038270 / 0.038270

Note. Holm adjustment is performed separately across the four endpoints for each null model.

Table 17: RQ2 sample funnel by target and data split.

Target	Train valid / total	Validation valid / total	Test valid / total	Singleton handling
Harmonic dispersion D_H	608 / 875	74 / 109	76 / 110	Excluded when $\lvert\mathcal{S}_q\rvert = 1$
Cross module pair ratio M_q	608 / 875	74 / 109	76 / 110	Uses the same queries with $\lvert\mathcal{S}_q\rvert \geq 2$ as D_H
Support size Y_K	875 / 875	109 / 109	110 / 110	Singletons retained

TF-IDF probe. The request q is the only input. We use lowercased word unigrams and bigrams, sublinear term frequency, smoothed inverse document frequency, and ℓ_2 normalization. The minimum document frequency is $min_df = 2$, and the vocabulary is capped at 20,000 terms. This probe tests whether request language itself carries structural information.

Metadata control. Features include a one-hot encoding of IP, $\log(1+|V_q|)$, and request length. For D_H and M_q, we also include $\log(1 + K_q)$. We omit this last feature when predicting Y_K because it directly encodes the target. For the two pairwise targets, the control thus has access to support size and tests whether spatial organization remains predictable when size is known.

For both predictor families, we search $\alpha \in \{10^{-4}, \ldots, 10^4\}$ in two stages. During discovery, candidate models are fitted only on Train, and α is selected by Validation Spearman correlation. These correlations are used for model selection and exploratory description, not as independent confirmatory evidence. For the final Test evaluation, we rebuild features and fit with the selected α on the combined Train and Validation splits. We then freeze preprocessing and model parameters and predict Test once. All predictions remain continuous without clipping or rounding. Table 18 lists the selected settings.

Table 18: Fitted settings of the RQ2 predictors.

Structural target	TF-IDF α	Metadata α	Final TF-IDF vocabulary
D_H	1,000	10	8,754
M_q	10	10	8,754
Y_K	10,000	0.1	12,019

B.7 Partial Rank Association Test

Let $r_{\widehat{y}}$ and r_y be the average ranks of predictions and target values. Let Z contain an intercept, IP fixed effects, and continuous controls after ranking and standardization. Set $P_Z = ZZ^+$, where Z^+ denotes the pseudoinverse of Z, and let I be the identity matrix of the appropriate sample size. We define partial Spearman correlation by

$$\rho_{\text{partial}} = \text{corr}((I - P_Z)r_{\widehat{y}}, (I - P_Z)r_y)\,.$$

For D_H and M_q, the controls are $\log(1 + K_q)$, $\log(1 + |V_q|)$, and request length. For Y_K, only the latter two controls are included.

We test the one-sided alternative $\rho_{\text{partial}} > 0$. Within each IP, we permute the prediction rank residuals while holding target residuals and the control structure fixed. We perform 15,000 permutations. If ρ_{obs} and ρ_b denote the observed and permuted correlations, the corrected probability is

$$p = \frac{1 + \sum_{b=1}^{15{,}000} \mathbf{1}\{\rho_b \geq \rho_{\text{obs}}\}}{15{,}001}.$$

We obtain 95% confidence intervals by bootstrapping entire IP clusters. Each replicate samples IPs with replacement and includes all queries of each sampled IP. We then recompute ranks, the control matrix, and ρ_{partial} without refitting either predictor. The interval endpoints are the 2.5th and 97.5th percentiles of 5,000 bootstrap correlations.

B.8 RQ2 Predictive Association Results

Figure 9 shows partial Spearman associations on the held-out Test split using predictions from frozen models. Correlations control for IP fixed effects, graph size, and request length, with additional control for support size for the two pairwise targets. The 95% percentile CIs use 5,000 IP-cluster bootstrap resamples; one-sided p-values use 15,000 within-IP permutations of prediction rank residuals with a $+1$ correction. Holm correction is applied separately across the three targets within each predictor family, yielding adjusted $p < 0.05$ for all TF–IDF targets and $p > 0.05$ for all Metadata targets.

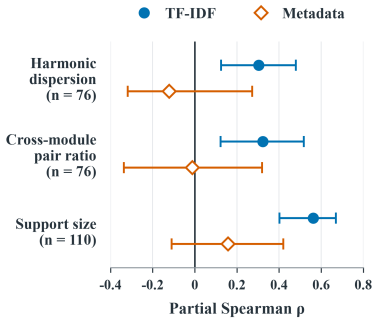


Figure 9: RQ2 predictive associations.

C Implementation and Training Details

Experimental environment. We implement the models with Hugging Face, SentenceTransformers, and PyTorch Geometric. The experimental server has two Intel Xeon Platinum 8462Y+ CPUs, 1 TiB of memory, and eight NVIDIA A800-SXM4-80GB GPUs. Each training run uses one GPU. We use random seeds 36, 42, and 77.

C.1 Model Components and Training Configuration

Textual Encoder. The Textual Encoder produces ℓ_2-normalized representations of dimension 768 for queries and blocks. It is initialized from CodeRankEmbed and adapted to the RTL modification localization task. The maximum input length is 1,024 tokens for both input types. We use Multiple Negatives Ranking Loss in its InfoNCE form with temperature $\tau = 0.05$. Each observed support block is paired with 15 negative blocks from the same design that do not belong to $\mathcal{S}_q$. Other blocks in the batch provide additional negatives. We optimize the model with AdamW using a learning rate of 2×10^{-5}, a microbatch size of 4, and 16 gradient accumulation steps, which yields an effective batch size of 64. The first 5% of optimization steps use linear warmup. The checkpoint is selected by Validation MAP.

Functional Encoder. The Functional Encoder uses three layers of edge-aware GATv2. Each layer has four attention heads of dimension 128, yielding a node state of dimension 512 after concatenation. The outputs of mean pooling and attention pooling are projected into a block representation of dimension 768. Dependency and timing classes use embeddings of dimension 8. The dropout rate is 0.3. We train the model with multi-positive InfoNCE at temperature $\tau = 0.07$ and sample 15 same-design negatives for each query. Training begins with two epochs of random negatives, followed by at most eight epochs of hard negatives. We apply early stopping according to Validation MAP over the full candidate set. We use AdamW with a learning rate of 5×10^{-4}, weight decay of 10^{-5}, and a query batch size of 8.

Architectural Encoder. GLIDE concatenates the frozen Textual and Functional representations of dimension 768 to form a node input of dimension 1,536. It then applies a typed GATv2 layer, design-level global multihead self-attention, and a second typed GATv2 layer. The node hidden dimension is 192, the output dimension is 128, and the number of attention heads is 2. Dependency and timing classes use embeddings of dimensions 16 and 8, respectively. An MLP maps their concatenation to an edge feature of dimension 32. Projection heads of size $768 \to 128$ and $128 \to 128$ align the query and node representations. We use a sampled listwise InfoNCE objective constructed per positive with temperature $\tau = 0.07$. Each observed support block is paired with 15 blocks from the same design that are not in $\mathcal{S}_q$. Training uses AdamW with a learning rate of 5×10^{-4}, weight decay of 10^{-5}, and cosine learning rate decay. We train for a fixed 20 epochs.

Intent-aware evidence fusion. The Fusion Router predicts weights for the three evidence sources. It is an MLP with architecture $\mathrm{LayerNorm}(768) \to 64 \to 3$. The hidden layer uses GELU and dropout of 0.1. A final softmax produces the weights. The output layer is initialized with zero weights and zero biases. Over the full candidate set, the model jointly optimizes a multi-positive listwise ranking loss with temperature $\tau_{\mathrm{rank}} = 1$ and KL supervision masked by m_q with weight $\eta = 1$. Here, $m_q = \mathbf{1}[|\mathcal{S}_q| > 1]$. The initial seed distribution uses $\tau_{\mathrm{seed}} = 0.05$. We use AdamW with a learning rate of 5×10^{-4}, weight decay of 10^{-4}, a query batch size of 32, and a gradient clipping threshold of 1.0. We apply early stopping according to Validation MAP.

Impact Propagation. The propagation module operates on the active subset $\mathcal{C}(G)$ of the six propagation channels defined in Appendix A.4. It predicts a distribution over hop counts from zero to three. The query, seed state, and channel features are projected to dimensions 128, 32, and 64, respectively. Each channel has an embedding of dimension 16. The channel head has architecture $240 \to 64 \to 1$, and the scope head has architecture $240 \to 64 \to 4$. Both use GELU and dropout of 0.1. The offline propagation actions enumerate $k \in \{1, 2, 3\}$ and $\lambda \in \{0.05, 0.10, 0.25, 0.50\}$. The utility temperature is $T_U = 0.02$, and the practical gain threshold is $\delta = 0.01$. The coefficient for counterfactual supervision in the training loss is $\lambda_{\mathrm{CF}} = 0.5$. We use AdamW with a learning rate of 3×10^{-4} and weight decay of 10^{-4}. Training runs for at most 40 epochs and uses early stopping according to Validation MAP.

C.2 Out-of-Fold Fusion Supervision Targets

The fusion supervision targets must be constructed from scores without in-sample bias. Their inputs therefore come from out-of-fold (OOF) predictions generated by cross-fitting performed exclusively within Train. We first describe the generation of OOF predictions and then define the fusion supervision target for each query.

Generation of out-of-fold predictions. We use the pre-change design version, identified by the pair of IP and parent hash, as the grouping unit. We partition the training set into five fixed folds $\{\mathcal{F}_j\}_{j=1}^{5}$. Queries that share a design version always belong to the same fold. The folds are therefore disjoint in design versions. For fold j, we retrain the textual, functional, and architectural encoders in sequence on the other four folds using the configuration above. This fold-specific evidence chain produces the three score vectors only for $\mathcal{F}_j$, together with the textual query representation before ℓ_2 normalization that is required by the fusion and propagation module. After merging the five held-out shards, every training query has exactly one set of predictions from models that did not observe its design version. We apply z-score standardization to the three score vectors separately for each query over $\mathcal{B}(D_q^-)$. If the standard deviation of an evidence source is below 10^{-6}, we set all of its standardized scores to zero.

Construction of fusion supervision targets. The candidate fusion weights form a simplex grid with step size $h_\alpha = 0.05$:

$$\Delta_{h_\alpha}^2 = \{(\alpha_T, \alpha_F, \alpha_G) \colon \alpha_v \in \{0, h_\alpha, \ldots, 1\},\ \textstyle\sum_v \alpha_v = 1\}\,.$$

The grid contains $\binom{1/h_\alpha + 2}{2} = 231$ candidates. Let $\widetilde{\mathbf{s}}_v^{\mathrm{OOF}}(q)$ denote the out-of-fold score vector from evidence source v under a common candidate ordering. The fusion utility of weight configuration $\boldsymbol{\alpha}$ and its supervision target are

$$U_q^{\mathrm{fusion}}(\boldsymbol{\alpha}) = \mathrm{AP}\left(\sum_v \alpha_v \widetilde{\mathbf{s}}_v^{\mathrm{OOF}}(q), \mathcal{S}_q\right), \qquad \boldsymbol{\alpha}_q^* = \arg\max_{\boldsymbol{\alpha} \in \Delta_{h_\alpha}^2} U_q^{\mathrm{fusion}}(\boldsymbol{\alpha}).$$

When several grid points attain the same AP, we first select the configuration with the lower listwise ranking loss and then the one with the more balanced weight distribution. Any remaining ties are resolved by a fixed deterministic rule.

C.3 Propagation Operator and Decision Context

Channel transition matrix. For an active channel $c \in \mathcal{C}(G)$ defined in the main text, let E_c be the set of directed edges assigned to that channel in Appendix A.4, and define $d_c^+(u) = |\{v \colon (u, v) \in$

$E_c\}|$. The channel transition matrix uses uniform outgoing-edge normalization:

$$[P_c]_{uv} = \begin{cases} 1/d_c^+(u), & (u,v) \in E_c, \\ 1, & u = v,\ d_c^+(u) = 0, \\ 0, & \text{otherwise.} \end{cases}$$

Because P_c is row-normalized and candidate distributions are column vectors, one propagation step is written as $P_c^\top p$. Channels with no edges are excluded from the channel softmax.

Decision context and channel weights. The query representation $\mathbf{h}_{q,T}$ provides a semantic prior over modification intent. The remaining decision context consists of sharpness, leading-candidate confidence, evidence consistency, and channel-conditioned features. Let $N_q = |\mathcal{B}(D_q^-)|$, and let $p_{q,(j)}^{(0)}$ denote the seed mass at rank j.

1. **Sharpness.** Sharpness measures the overall concentration of the initial seed distribution. It indicates whether retrieval identifies a focused candidate region or remains diffuse across many candidates. The normalized entropy is $H_q^{\text{seed}} = -\frac{\sum_b p_q^{(0)}(b) \log p_q^{(0)}(b)}{\log N_q}$. H_q^{seed} measures the global dispersion of seed mass. Smaller values indicate greater concentration. We define $0 \log 0 = 0$ and set $H_q^{\text{seed}} = 0$ when $N_q = 1$. The mass of the five highest-ranked candidates is $M_{q,5}^{\text{seed}} = \sum_{j=1}^{\min(5,N_q)} p_{q,(j)}^{(0)}$. $M_{q,5}^{\text{seed}}$ measures the probability mass assigned to the five leading candidates. Larger values indicate greater concentration near the top of the ranking. The sharpness vector is $\mathbf{x}_q^{\text{sharp}} = \left[H_q^{\text{seed}}, M_{q,5}^{\text{seed}}\right]$.
2. **Leading-candidate confidence.** This feature measures both the probability mass of the leading candidate and its margin over the second candidate. It indicates whether the leading candidate provides a reliable anchor for propagation. The absolute mass and relative margin are $\mathbf{x}_q^{\text{lead}} = \left[p_{q,(1)}^{(0)}, p_{q,(1)}^{(0)} - p_{q,(2)}^{(0)}\right]$. We set $p_{q,(2)}^{(0)} = 0$ when $N_q = 1$.
3. **Evidence consistency.** This feature measures agreement among the three evidence sources on the candidate ranking. It distinguishes seeds supported by multiple sources from seeds for which the sources disagree. Let $\widetilde{\mathbf{s}}_T(q)$, $\widetilde{\mathbf{s}}_F(q)$, and $\widetilde{\mathbf{s}}_G(q)$ be the standardized score vectors. We define $\mathbf{x}_q^{\text{cons}} = [\text{corr}(\widetilde{\mathbf{s}}_T(q), \widetilde{\mathbf{s}}_F(q)), \text{corr}(\widetilde{\mathbf{s}}_T(q), \widetilde{\mathbf{s}}_G(q)), \text{corr}(\widetilde{\mathbf{s}}_F(q), \widetilde{\mathbf{s}}_G(q))]$. Here, corr denotes Pearson correlation. If either input vector has zero variance, we set the corresponding entry to zero.
4. **Channel-conditioned features.** These features measure how much of the current seed mass a propagation channel can carry and how one propagation step changes its distribution. They support prediction of the hardware dependency classes along which relevance should propagate. For each active channel c, we first compute the one-step response $\widehat{p}_{q,c} = P_c^\top p_q^{(0)}$. Let $A_c = \{u\colon d_c^+(u) > 0\}$ be the set of nodes with valid outgoing edges in channel c. Let $k_q^{\text{top}} = \min(10, N_q)$ be the top-k cutoff used to compute channel features. The sets T_0 and T_c contain the top k_q^{top} candidates under $p_q^{(0)}$ and $\widehat{p}_{q,c}$, respectively. The six channel features are defined and concatenated in the order in Table 19.

Context encoding. The seed-state vector is the concatenation $\mathbf{x}_q^{\text{seed}} = \left[\mathbf{x}_q^{\text{sharp}} \| \mathbf{x}_q^{\text{lead}} \| \mathbf{x}_q^{\text{cons}}\right]$. For each random seed, we estimate the seed-state normalizers from all Train queries with out-of-fold predictions. For channel c, we estimate the channel-feature normalizers only from Train queries with out-of-fold predictions for which that channel is active. Continuous features are standardized as $(x - \mu)/\max(\sigma, 10^{-6})$. We save the resulting statistics with the propagation module and hold them fixed during Validation and Test. The standardized seed-state and channel features are encoded separately and concatenated with the query representation and channel embedding. The shared channel head f_π maps the resulting joint representation to the compatibility logit for channel c.

Propagation scope. The scope head uses the channel-weighted context

$$\mathbf{u}_{\text{scope}} = \left[\mathbf{u}_q \| \mathbf{u}_{\text{seed}} \| \sum_{c \in \mathcal{C}(G)} \pi_c(q, G)[\mathbf{u}_c \| \mathbf{e}_c]\right].$$

The scope head produces four logits for hop counts from zero to three. Softmax normalization yields the propagation scope weights $\boldsymbol{\omega}(q, G) = [\omega_0(q, G), \ldots, \omega_3(q, G)]$. Here, ω_0 denotes no

Table 19: Definitions and roles of the channel-conditioned features.

Feature	Definition	Interpretation
Coverage	$\text{coverage}_{q,c} = \sum_{u \in A_c} p_q^{(0)}(u)$	Fraction of the current seed mass that can propagate through channel c
Displacement	$\text{displacement}_{q,c} = \|\widehat{p}_{q,c} - p_q^{(0)}\|_1$	Magnitude of the overall distributional change after one propagation step
Entropy	$\text{entropy}_{q,c} = -\frac{\sum_b \widehat{p}_{q,c}(b) \log \widehat{p}_{q,c}(b)}{\log N_q}$	Dispersion of probability mass after propagation
Top-k overlap	$\text{overlap}_{q,c} = \frac{\lvert T_0 \cap T_c \rvert}{k_q^{\text{top}}}$	Stability of the leading candidate set after propagation
Retained mass	$\text{retained}_{q,c} = \sum_{b \in T_0} \widehat{p}_{q,c}(b)$	Probability mass that remains on the original leading candidates
Weighted degree	$\text{degree}_{q,c} = \sum_u p_q^{(0)}(u) \log(1 + d_c^+(u))$	Outgoing degree weighted by seed mass, which characterizes the branching capacity of the current seed

Note. All features are computed only from $p_q^{(0)}$ and the pre-change DTG. They do not use the observed support $\mathcal{S}_q$. We set channel entropy to zero when $N_q = 1$.

propagation. If $\mathcal{C}(G) = \emptyset$, we skip both propagation heads and use the deterministic fallback: $\boldsymbol{\pi}(q, G) = \mathbf{0}, \boldsymbol{\omega}(q, G) = \mathbf{e}_0, p_q = p_q^{(0)}$.

C.4 Utility Supervision and Training Order

Discrete actions for propagation supervision. Every propagation action defined in Section 4.3 starts from the fixed Train out-of-fold seed $p_{q,\text{OOF}}^{(0)}$. We enumerate

$$\mathcal{A}_q = \{a_0\} \cup \{(c, k, \lambda) \colon c \in \mathcal{C}(G),\ k \in \{1, 2, 3\},\ \lambda \in \{0.05, 0.10, 0.25, 0.50\}\}\,.$$

Following Section 4.3, we transform the action utilities using temperature T_U and aggregate them into $\boldsymbol{\pi}_q^*$ and $\boldsymbol{\omega}_q^*$. The validity of channel supervision is determined by

$$\chi_q^{\text{ch}} = \mathbf{1}\left[\mathcal{C}(G) \neq \emptyset \wedge \max_{a \in \mathcal{A}_q \setminus \{a_0\}} U_q(a) > \delta\right].$$

When active channels exist but $\chi_q^{\text{ch}} = 0$, we set $\boldsymbol{\omega}_q^* = \mathbf{e}_0$ and mask channel supervision. The propagation scope supervision and ranking loss remain active. Queries without an active channel use the deterministic fallback in Section C.3 and do not contribute to optimization of the propagation module.

Propagation training objective. We implement $\mathcal{L}_{\text{rank}}$ as multi-positive listwise cross-entropy over the full candidate set:

$$\mathcal{L}_{\text{rank}}(q) = -\sum_{b \in \mathcal{B}(D_q^-)} y_q(b) \log p_q(b), \quad y_q(b) = \begin{cases} 1/|\mathcal{S}_q|, & b \in \mathcal{S}_q, \\ 0, & b \notin \mathcal{S}_q. \end{cases}$$

Here, $p_q(b)$ is the probability mass assigned to block b by the final candidate distribution. The remaining channel and hop supervision terms follow Section 4.3.

C.5 Baseline Adaptation and Implementation

C.5.1 Lexical Retrieval Baseline

BM25. For each pre-change design, we construct a separate `BM25Okapi` index over its raw RTL blocks. We lowercase both requests and code and tokenize them at nonalphanumeric characters. We use $k_1 = 1.5$, $b = 0.75$, and $\epsilon = 0.25$. Inputs are not truncated.

C.5.2 Semantic Retrieval Baselines

Semantic encoders. We apply the same training configuration as the Textual Encoder to all three models. GraphCodeBERT, CodeV-DS-6.7B, and SweRank-Retriever are initialized independently

from their corresponding pretrained weights. GraphCodeBERT and CodeV-DS-6.7B use attention-mask-aware mean pooling over their final layer representations. SweRank-Retriever retains its original query prefix and uses the CLS representation. GraphCodeBERT inputs are truncated to 512 tokens because of its positional encoding limit. CodeV-DS-6.7B and SweRank-Retriever use the same limit of 1,024 tokens as the Textual Encoder. To isolate retriever performance, we do not use SweRank's additional 7B listwise reranker.

C.5.3 Graph Structure Baselines

HOGA and Dir-GNN share a common protocol and differ only in their graph operators. Both use the DTG from Appendix A.4 with relation and timing labels removed. To control for node information, both receive the same frozen Textual and Functional candidate representations used in GLIDE, concatenated before a method-specific input projection; query representations come from the textual encoder and do not participate in propagation, and the frozen encoders are never updated. Queries and candidates are projected into a normalized 128-dimensional space and ranked by cosine similarity. The model is trained with a multi-positive listwise loss (temperature 0.07) that weights positive blocks and queries equally. We use Adam (lr 5×10^{-4}, weight decay 5×10^{-5}, dropout 0.3, query batch size 32) for at most 100 epochs, stopping if Validation MAP does not improve for 10 epochs.

HOGA. We convert the graph to undirected, precompute multi-hop responses, and encode candidates with gated hop-wise attention and attentive readout; hop depth is selected from $\{2, 5, 8\}$.

Dir-GNN. We retain original edge directions without reverse edges; three Dir-GCN layers separately aggregate incoming and outgoing neighbors, with mixing coefficient from $\{0, 0.5, 1\}$.

C.5.4 Hardware-Specific Representation Baselines

CircuitFusion. Following the original implementation, CircuitFusion uses a seven-layer Graphormer for graph inputs, the first six layers of BERT-base for the summary encoder, a frozen NV-Embed-v1 code encoder with 4096-dimensional outputs, and the last six layers of BERT-base for multimodal fusion. We replace subcircuits partitioned by register fan-in logic with RTL blocks as encoding units. We omit the RTL-netlist alignment pretraining task because our setting does not assume a post-synthesis netlist. To adapt the original design-quality prediction task to request-conditioned retrieval, we add residual projections for the request summary and candidate fusion representations and train them with a multi-positive ranking objective over the full candidate set. We update the retrieval head, the final two layers of both the summary and fusion encoders, and the modality projections for summary, graph, and code inputs; all other pretrained parameters remain frozen.

UniRTL. Following the original UniRTL architecture, we use CodeBERT-base-mlm as the 12-layer, 12-head language backbone. Its graph-aware tokenizer consists of an eight-layer GIN followed by an eight-layer Transformer with four attention heads per layer. Node descriptions are embedded with all-mpnet-base-v2 and reduced to 32 dimensions by PCA. We replace the module-level CDFG with a block-level CDFG and jointly encode it with RTL from the pre-change design. This increases the number of block-level CDFGs, so we run graph pretraining on Train for at most 280,000 updates. For downstream training, we replace the original single-positive in-batch contrastive objective with multi-positive retrieval; each example contains all observed support blocks and 64 non-support candidates from the same design.

C.6 Evaluation Metrics

For each evaluation query $q \in \mathcal{Q}$, let $\mathcal{B}(D_q^-)$ and $\mathcal{S}_q \subseteq \mathcal{B}(D_q^-)$ denote the candidate blocks and observed modification support, with $N_q = |\mathcal{B}(D_q^-)|$ and $R_q = |\mathcal{S}_q| > 0$. For candidate $b_{q,(r)}$ at rank r in descending score order, define its support indicator and precision at rank r as

$$y_{q,r} = \mathbf{1}\{b_{q,(r)} \in \mathcal{S}_q\}, \qquad P_q(r) = \frac{1}{r}\sum_{j=1}^{r} y_{q,j}.$$

Average precision and MAP. Average precision (AP) measures how well the full ranking covers the support set. Mean Average Precision (MAP) assigns equal weight to all queries:

$$\mathrm{AP}(q) = \frac{1}{R_q} \sum_{r=1}^{N_q} P_q(r)\, y_{q,r}, \qquad \mathrm{MAP} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \mathrm{AP}(q).$$

Mean Reciprocal Rank (MRR). MRR averages the reciprocal rank of the first support block across queries, measuring how early successful localization occurs:

$$\mathrm{RR}(q) = \frac{1}{\min\{r \colon y_{q,r} = 1\}}, \qquad \mathrm{MRR} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \mathrm{RR}(q).$$

Recall@k. Recall@k measures the fraction of the support set covered within the first k ranks and thus characterizes recovery under a limited inspection budget. Let $k_q = \min(k, N_q)$. Then

$$\mathrm{Recall@}k(q) = \frac{\sum_{r=1}^{k_q} y_{q,r}}{R_q}.$$

Accuracy@k. A query is successful only if every support block appears within the first k_q ranks. It therefore measures recovery of the complete support set in a single localization result:

$$\mathrm{Accuracy@}k(q) = \mathbf{1}\left\{ \sum_{r=1}^{k_q} y_{q,r} = R_q \right\}.$$

Recall@k and Accuracy@k are both averaged arithmetically over $\mathcal{Q}$. We evaluate $k \in \{R_q, 2R_q\}$ and abbreviate R_q as R in table headers.

D Additional Experimental Results

D.1 Model Scale and Online Retrieval Latency

Table 20 reports total model parameters and online latency per query. Parameter counts include all components, including candidate encoders used offline. Latency is measured after candidate representations are precomputed and therefore reflects query encoding and full-candidate ranking.

Table 20: Total model parameters and online retrieval latency of RTLocating and baseline methods.

Method	Total parameters (M)	Latency per query (ms)
BM25	N/A	4.08
GraphCodeBERT	124.65	7.31
CodeV-DS-6.7B	6,774.07	630.88
SweRank-Retriever	136.73	10.45
HOGA	137.15	4.51
Dir-GNN	137.10	8.94
CircuitFusion	7,237.36	5.31
UniRTL	153.45	6.70
RTLocating	142.34	15.29

D.2 Consistency of Factorized Counterfactual Supervision under Composed Propagation

The counterfactual supervision in the main text factorizes the propagation policy used at inference. It evaluates the utility of individual channel actions and aggregates the results into channel and scope targets. This construction avoids a direct search over the continuous joint decision space of channels and scopes. A listwise ranking loss on the final distribution constrains interactions across channels. We examine whether targets derived from individual channel actions retain support recovery utility when applied to the composed propagation operator.

Setting. For each Test query q, we use the frozen initial seed $p_q^{(0)}$. The action set $\mathcal{A}_q$ and validity mask χ_q^{ch} follow Appendix C.4, and the action utility $U_q(a)$ follows Section 4.3. We denote the Test queries satisfying $\chi_q^{\text{ch}} = 1$ by $\mathcal{Q}_{\text{act}}$. This set contains 74 of the 110 Test queries, or 67.3%.

Evaluation under composition. For each $q \in \mathcal{Q}_{\text{act}}$, we substitute the factorized utility targets $\boldsymbol{\pi}_q^*$ and $\boldsymbol{\omega}_q^*$ into the composed propagation operator. Here, $\pi_{q,c}^*$ and $\omega_{q,k}^*$ are the weights for channel c and hop count k, respectively:

$$\widetilde{p}_q = \sum_{k=0}^{K} \omega_{q,k}^* \left(\sum_{c \in \mathcal{C}(G)} \pi_{q,c}^* P_c^\top \right)^k p_q^{(0)}.$$

Empty channels are excluded from $\boldsymbol{\pi}_q^*$. The gain over the frozen seed for query q is $G_q^{\text{comp}} = \text{AP}(\widetilde{p}_q, \mathcal{S}_q) - \text{AP}(p_q^{(0)}, \mathcal{S}_q)$.

We report the mean gain with equal weight assigned to each query, the fraction of degraded queries

$$R_{\text{deg}} = \frac{\left|\left\{q \in \mathcal{Q}_{\text{act}} : G_q^{\text{comp}} < 0\right\}\right|}{|\mathcal{Q}_{\text{act}}|},$$

and the mean signed gap to the best individual channel action:

$$\overline{G}^{\text{comp}} = \frac{1}{|\mathcal{Q}_{\text{act}}|} \sum_{q \in \mathcal{Q}_{\text{act}}} G_q^{\text{comp}}, G_q^{\text{best}} = \max_{a \in \mathcal{A}_q} U_q(a), \Delta_{\text{pure}} = \frac{1}{|\mathcal{Q}_{\text{act}}|} \sum_{q \in \mathcal{Q}_{\text{act}}} \left(G_q^{\text{best}} - G_q^{\text{comp}}\right).$$

For every $q \in \mathcal{Q}_{\text{act}}$, the validity criterion guarantees an individual channel action with positive utility. The maximizer in G_q^{best} is therefore an individual channel action. Because $\mathcal{A}_q$ also contains a_0, we have $G_q^{\text{best}} \geq 0$. The composed distribution $\widetilde{p}_q$ includes paths across channels, so the signed gap can be negative for an individual query.

Table 21: Post hoc utility diagnostics for factorized targets under the composed propagation operator.

Decision	MAP gain ↑	Degraded ↓	Signed gap ↓
Initial Seed	0	N/A	**+0.035**
Factorized targets (composed)	+0.024	**10/74 (13.5%)**	**+0.011**
Best single-channel action	**+0.035**	**0/74 (0.0%)**	0

As shown in Table 21, applying the factorized utility targets to the composed propagation operator increases mean AP by 0.024 over the seed. Degradation occurs for 13.5% of queries, and the mean gap to the best individual channel action is 0.011. These results indicate that the targets retain positive mean support recovery utility after factorization and composed execution. They provide evidence complementary to the jointly trained ablation in Table 3.

D.3 Contribution of Intent Conditioning to Propagation Decisions

This experiment tests whether matching the query representation to the current change instance provides an additional signal for propagation decisions.

Setting. We freeze all model parameters and hold the semantic seed, seed state features, features conditioned on each channel, DTG, and candidate set fixed for every Test query. In the matched condition, the channel and scope heads receive the original query representation. In the shuffled condition, we apply a fixed point free permutation to the query representations within each IP. Each query therefore receives the representation of a different query from the same IP. Restricting the permutation within each IP controls for IP level differences in graph scale and candidate distribution, leaving the query to instance correspondence as the only manipulated factor. All 110 Test queries belong to IPs with at least two queries and are retained. We use permutation instead of a zero vector to avoid inputs outside the training distribution.

Evaluation. For each model seed, we generate 10,000 independent fixed point free permutations within IPs. The primary endpoint is the paired MAP difference between the matched and shuffled

conditions. A randomization test evaluates the directional alternative that matching improves MAP. We obtain the 95% confidence interval using a bootstrap over IP clusters. Ranking metrics assign equal weight to each query. Shuffled results are first averaged across permutations within each seed. The table reports means and standard deviations across three model seeds.

Table 22: Effect of query to instance correspondence on propagation performance.

Propagation decision input	MAP ↑	MRR ↑	Recall@$2R$ ↑	Accuracy@$2R$ ↑
Shuffled query	0.451 ± 0.004	0.629 ± 0.004	0.552 ± 0.004	0.242 ± 0.005
Matched query	$\mathbf{0.471 \pm 0.003}$	$\mathbf{0.656 \pm 0.003}$	$\mathbf{0.564 \pm 0.003}$	$\mathbf{0.258 \pm 0.005}$

Table 22 shows that matching the query representation improves MAP by 0.020 over the shuffled condition, with a 95% confidence interval of [0.012, 0.028] and a randomization test result of $p < 0.001$. Since both conditions share the same semantic seed and all other decision features, this difference isolates the contribution of matching the query representation to the current change instance. The result provides direct evidence that channel and scope selection use request semantics beyond the information already expressed by the retrieval state.

D.4 Performance across Modification and Design Scales

We further analyze each baseline across bins defined by modification scale, measured by support set size, and design complexity, measured by IP code size. This analysis assesses how the inductive bias of each method interacts with task scale. Figure 3 reports the corresponding results.

Observation 1. Semantic retrieval is competitive with RTLocating for small modifications but degrades as the support set grows. For single block modifications, SweRank-Retriever obtains a MAP of 0.462 compared with 0.493 for RTLocating. For modifications involving at least 20 blocks, their absolute MAP difference increases from 0.031 to 0.10. This pattern is consistent with the independent scoring of query and block pairs in SweRank. A single block modification requires no coordination across locations, so semantic matching can suffice. Larger modifications require recovery along dependency paths, which independent semantic retrieval does not represent.

Observation 2. The graph method HOGA reaches a relative peak for modifications spanning 4 to 9 blocks, but degrades for both very small and very large modifications. Its MAP in the 4 to 9 block bin is 0.370, which is 44% above its overall MAP of 0.257. Performance is lower for single block modifications at 0.217 and for modifications involving at least 20 blocks at 0.328. This behavior is consistent with hopwise multiscale aggregation. The topology of medium scale modifications is aligned with its aggregation range. Single block modifications offer little graph structure to exploit, while the global dependencies of very large modifications can exceed its fixed aggregation depth.

Observation 3. The hardware representation method UniRTL outperforms a language model without graph structure for moderately large modifications, but remains behind propagation based methods at the largest scale. In the 4 to 9 block bin, UniRTL reaches a MAP of 0.458 compared with 0.353 for CodeV-DS. This result indicates that its CDFG tokenization informed by graph structure captures dependencies across blocks at moderate complexity. In the bin with at least 20 blocks, however, its MAP of 0.292 remains below the 0.433 of RTLocating. This comparison suggests that static representations of global dependencies do not provide enough information for propagation and recovery at large scale when guided by intent.

Observation 4. All methods degrade on very large IPs with more than 20K lines of code, while the relative advantage of RTLocating remains stable. RTLocating obtains a MAP of 0.406 in this bin, which is 26% below its peak of 0.546 in the 5K to 10K bin. It nevertheless remains ahead of SweRank at 0.374, UniRTL at 0.191, and HOGA at 0.168. A larger candidate space is challenging for every method. Selective propagation guided by intent remains more robust on large designs than independent scoring or fixed graph aggregation.

D.5 Component Ablation

To isolate the contributions of the three evidence views, intent-aware fusion, and impact propagation, we evaluate single-view and component-ablation variants under the same frozen encoders, candidate sets, and evaluation protocol. For a view ablation, we set the corresponding fusion weight to zero at

inference and renormalize the remaining two weights. The encoders and Router stay frozen without retraining, so the variant without Textual evidence still receives the Textual query representation required by the Router. Static fusion uses one set of query-independent weights shared across all queries, selected by Validation MAP without using Test.

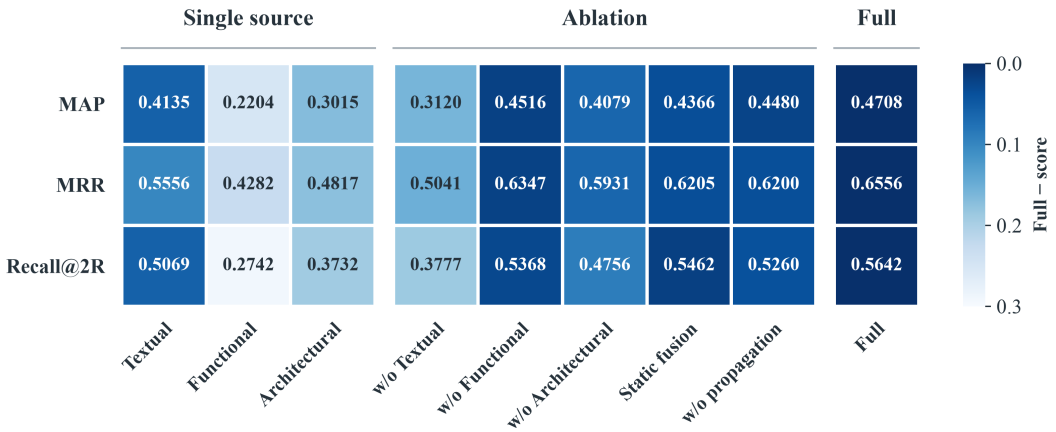


Figure 10: Component ablation of evidence sources, fusion, and propagation.

Textual evidence is the primary source of recall, but multiple sources are needed to exceed the ceiling of any individual view. Removing Textual evidence produces a larger MAP decrease than removing either other sources. No individual view reaches the performance of the full configuration, suggesting that the three sources provide complementary rather than redundant coverage.

Architectural evidence complements textual semantics rather than serving as a strong retriever on its own. Architectural evidence performs worse than Textual evidence when used alone, but removing it incurs a larger penalty than removing Functional evidence. This pattern suggests that its dependencies across modules provide useful localization cues when textual evidence is insufficient. Functional DFG evidence overlaps more with textual semantics and primarily refines the ranking for selected intents.

Fusion conditioned on intent is important for effective propagation. Static fusion obtains a MAP of 0.4366, below the 0.4480 seed without impact propagation. This comparison suggests that propagation can amplify poorly allocated evidence weights and that the initial seed quality constrains the resulting ranking. Together with the shuffled weight control in Section 5.4, the result supports allocating evidence as a function of the query.

Propagation primarily expands coverage rather than only reordering leading candidates. Removing impact propagation causes a larger relative decrease in Recall@$2R$ than in MAP. This pattern consistent with Section 5.5, where propagation recovers support blocks outside the seed top R_q. The gain therefore appears to arise mainly from recovering support missed by semantic retrieval.

D.6 End to End Editing Experiment Details

Both the single-turn LLM and Codex use `gpt-5.5` with a temperature of 0.7 and a maximum of three retries. Each attempt is verified independently. Across the initial attempt and at most three retries, we report the attempt with the largest recovery rate ρ_q; ties are resolved in favor of the earliest attempt. Compilation failures and simulation timeouts are assigned $\rho_q = 0$ and counted as no improvement. The same aggregation rule is applied to all experimental conditions.

D.6.1 Behavioral Recovery Metric

For each generated patch, the verification pipeline constructs the version before the modification D_q^-, denoted by `parent`, the reference version after the modification D_q^+, denoted by `current`, and the generated version $\widehat{D}_q$, denoted by `block_changed`. Verilator compiles all three versions under the same module level harness and executes the same stimuli. Each stimulus sequence is designed for the functionality affected by the change and exposes the corresponding behavioral

difference. Let $\mathbf{o}_t(x)$ be the complete vector of observed outputs from version x at sampled cycle t. Over the T_q executed cycles, exact behavioral agreement is

$$m_q(x, y) = \frac{1}{T_q} \sum_{t=1}^{T_q} \mathbf{1}[\mathbf{o}_t(x) = \mathbf{o}_t(y)] .$$

The current stimulus exposes the reference behavior change only when $m_q(D_q^-, D_q^+) < 1$. For these cases, the recovery rate is

$$\rho_q = \frac{m_q(\widehat{D}_q, D_q^+) - m_q(D_q^-, D_q^+)}{1 - m_q(D_q^-, D_q^+)}.$$

A value of $\rho_q = 1$ denotes full recovery over all observed signals and cycles. Values satisfying $0 < \rho_q < 1$ denote partial recovery. A value of $\rho_q = 0$ denotes no improvement, and $\rho_q < 0$ indicates that the generated edit deviates more from the reference than the version before the modification.

D.6.2 Failure Attribution

We conduct a post hoc attribution analysis for the cases in Section 5.7 that do not achieve full behavioral recovery, including partial recovery and no improvement or harm. For each failed case, we manually inspect the editor output and classify the root cause as a localization failure or an editing failure. A localization failure occurs when the editor does not identify the correct support blocks. An editing failure occurs when the correct blocks are identified but the resulting modification is incorrect or incomplete. Figure 11 summarizes the results.

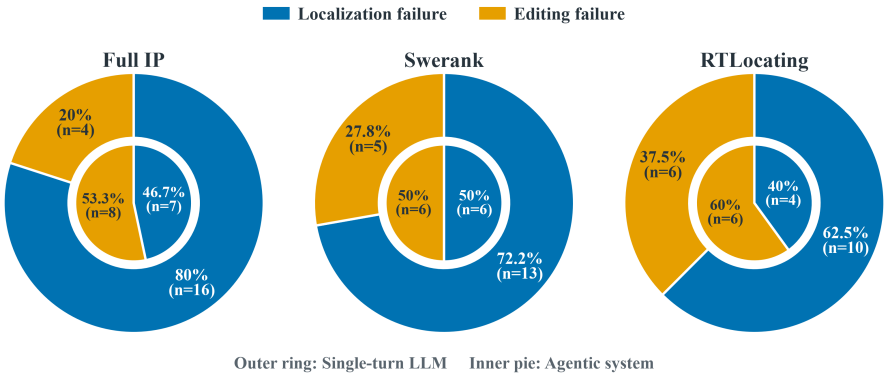


Figure 11: Failure attribution for cases with partial recovery or no improvement or harm.

Localization remains the main bottleneck for single-turn editing. Among unsuccessful cases, the share attributed to localization decreases from 80.0% under Full IP to 72.2% with SweRank-Retriever and 62.5% with RTLocating, yet does not eliminate the need for multistep reasoning.

Agentic search complements explicit localization cues. The agentic system consistently shows lower localization failure rates than the single-turn LLM across all three conditions. Under RTLocating, the share attributed to localization drops to 40.0%, compared to 46.7% under Full IP and 50.0% under SweRank-Retriever, suggesting that explicit guidance remains useful even when the editor can explore the codebase autonomously.

Editing quality becomes the next bottleneck as localization improves. Editing failures account for 60.0% of unsuccessful cases under the agentic system with RTLocating. Once the correct support blocks have been located, producing a correct modification becomes the main limiting factor.

D.6.3 Case Studies

For each index i in the KMAC example, `prim_alert_sender` consumes both alert vectors and `IsFatal(i)` identifies the fatal lane; their [fatal, recoverable] ordering must therefore agree. In the OTBN example, the reference change converts `state_reset_o` from a pulse on leaving halt to a level held while halted, shortening the timing path. The controller-side update remains necessary for full behavioral recovery, so complete localization alone is insufficient.

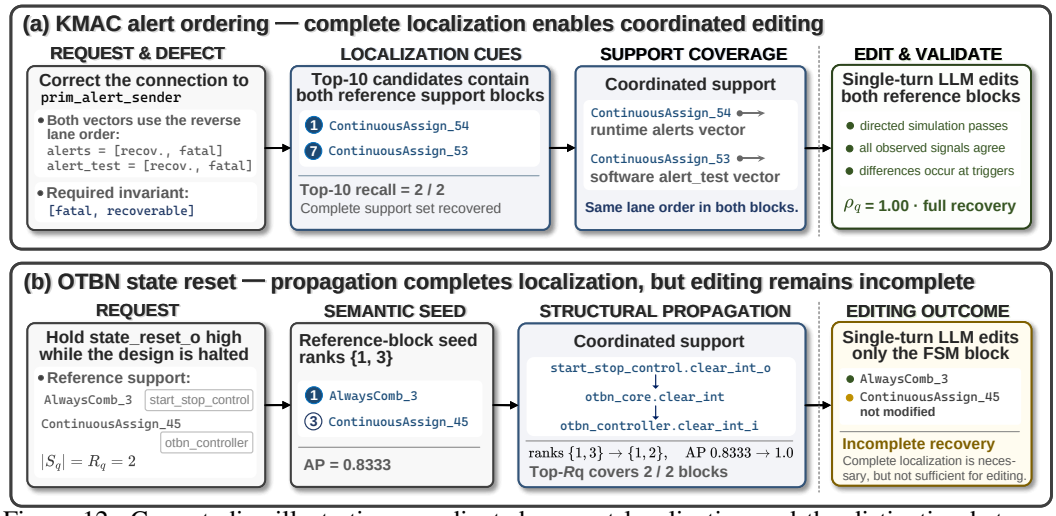


Figure 12: Case studies illustrating coordinated support localization and the distinction between localization and behavioral editing.