# DO RTL REPRESENTATIONS UNDERSTAND CIRCUITS? A DIAGNOSTIC BENCHMARK FOR FUNCTIONAL UNDERSTANDING AND IMPLEMENTATION AWARENESS

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

## ABSTRACT

Learning-based methods increasingly assist register-transfer level (RTL) circuit design and optimization, making effective RTL representations an important foundation for downstream applications. Such representations should capture shared functional semantics across equivalent implementations while remaining sensitive to implementation differences relevant to design quality. However, existing evaluations of RTL representations assess functional behavior and implementation quality separately and on different designs, leaving it unclear how well representations capture each capability and how the two interact. In this paper, we introduce **UnderRTL**, a unified diagnostic benchmark for evaluating these two capabilities, termed *functional understanding* and *implementation awareness*. Built on 7,075 curated RTL designs, UnderRTL provides verified functional variants, synthesis-based PPA labels across multiple PDKs and constraints, and natural-language queries for six linked tasks. Functionally equivalent implementations bridge the two axes, enabling controlled analysis of *functional invariance* and *implementation sensitivity*. UnderRTL integrates 24 engineered-feature, text, graph, and multimodal methods under standardized evaluation protocols. Our experiments reveal task-dependent limitations in both capabilities, as well as weak transfer to unseen designs, motivating more robust and generalizable RTL representations.

## 1 INTRODUCTION

As chip design grows increasingly complex, learning-based methods are being explored to assist RTL design, reuse, and optimization. In practice, designers repeatedly create, reuse, and refine RTL implementations to satisfy functional specifications while improving power, performance, and area (PPA). Supporting these processes requires representations that can serve diverse applications, from functional retrieval and retrieval-augmented generation to early PPA estimation (Xu et al., 2023; Fang et al., 2025; Zhao et al., 2026; Liu et al., 2025; 2026b;a). These applications impose complementary requirements: RTL representations should capture shared functional semantics across equivalent implementations while remaining sensitive to implementation differences relevant to PPA, since functionally equivalent designs can still exhibit different implementation quality (Micheli, 1994). We call these capabilities *functional understanding*—the function a circuit implements independent of how it is described—and *implementation awareness*—how implementation choices affect synthesized PPA. Both are essential for practical RTL representations.

Existing circuit datasets and benchmarks have provided valuable resources for RTL learning and evaluation (Wang et al., 2026; Li et al., 2025; Liu et al., 2023; Lu et al., 2024). However, existing evaluations are often built around particular datasets, tasks, or model families, making systematic comparison difficult. More importantly, RTL representations are typically evaluated for functional behavior and implementation quality on separate designs. Without linked samples that preserve functionality while varying implementation, it is difficult to determine whether a representation captures functional semantics rather than incidental syntactic or structural cues, or whether it preserves implementation distinctions that matter for PPA. This motivates a unified diagnostic benchmark.

We argue that such a benchmark should provide *linkage*, *comparability*, and *informativeness*. **(1) Linkage** requires evaluating both functional understanding and implementation awareness and link-



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

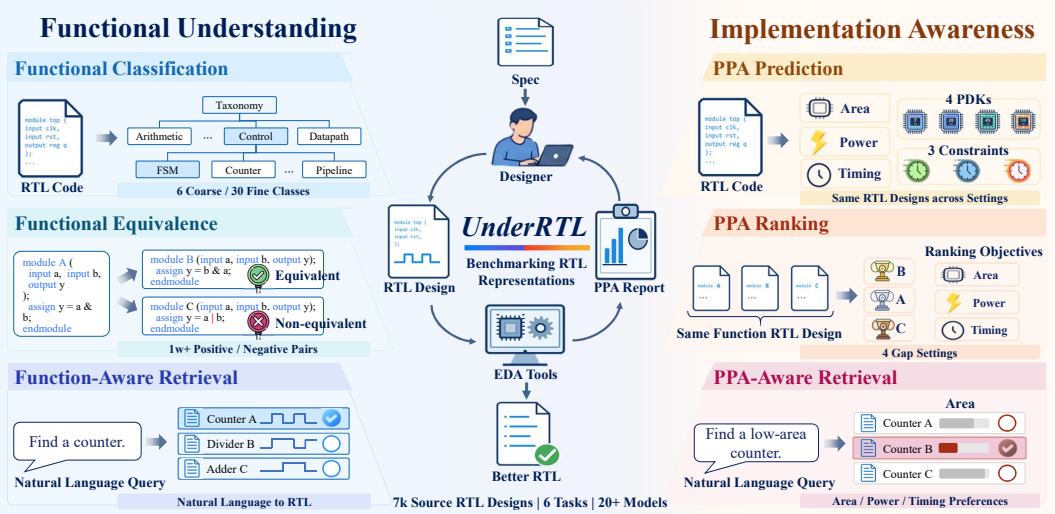


Figure 1: Overview of *UnderRTL*. The center shows the practical RTL design and optimization flow, with tasks for functional understanding and implementation awareness on the left and right. Functionally equivalent implementations link the two axes for joint analysis.

ing them through functionally equivalent implementations. **(2) Comparability** requires consistent data usage, representation extraction, and evaluation protocols across model families. **(3) Informativeness** requires controlled variations in functionality, implementation, and design environment to reveal what representations encode, when they remain invariant, and where they fail to generalize. Together, these properties turn benchmarking from a collection of task scores into a diagnostic tool for understanding RTL representations.

To this end, we introduce **UnderRTL**, illustrated in Figure 1, a unified benchmark for RTL representations. Built on 7,075 curated RTL designs, UnderRTL combines verified functional variants, synthesis-based PPA labels under multiple PDKs and constraints, and natural-language queries to support six linked tasks spanning functional recognition, PPA modeling, and retrieval. Functionally equivalent implementations bridge the two axes, enabling controlled analysis of *functional invariance* and *implementation sensitivity*. UnderRTL further provides a unified library integrating 24 engineered-feature, text, graph, and multimodal methods under standardized data processing, representation extraction, and downstream evaluation protocols.

Using UnderRTL, we investigate what information current RTL representations encode (**RQ1: Encoding**), whether they remain functionally consistent across equivalent implementations (**RQ2: Invariance**), how functional invariance interacts with implementation sensitivity (**RQ3: Invariance-Sensitivity Interplay**), and how these capabilities generalize across designs and synthesis environments (**RQ4: Generalization**). Our experiments show that strong task performance can mask lexical sensitivity and equivalence failures, while the invariance-sensitivity relationship varies across PPA objectives and model families. Moreover, adaptation across synthesis environments does not ensure robust transfer to unseen designs, motivating RTL representations that better balance semantic robustness, implementation awareness, and generalization.

In summary, our key contributions are:

- We introduce a linked diagnostic framework that jointly evaluates functional understanding and implementation awareness, using functionally equivalent implementations to study functional invariance and implementation sensitivity.
- We construct quality-controlled evaluation data from 7,075 curated RTL designs, supporting six interconnected tasks with verified functional relations, synthesis-based PPA labels, and natural-language queries.
- We develop a unified benchmarking library integrating more than 20 methods with standardized data processing, representation extraction, and downstream evaluation.
- We analyze encoding, invariance, sensitivity, and generalization, revealing representation limitations, heterogeneous capability relationships, and generalization challenges.

Table 1: Comparison of RTL datasets and benchmarks by problem scope and evaluation design.

| | Problem Scope | | | | Evaluation Design | | |
|---|---|---|---|---|---|---|---|
| **Work** | **Target** | **Stage** | **Func.** | **Impl.** | **Linked Impl.** | **Shift Eval.** | **Unified Rep.** |
| VerilogEval | Generation | Front-End | ✓ | – | – | – | – |
| RTLLM | Generation | Front-End | ✓ | ✓ | ✗ | – | – |
| CircuitNet 3.0 | Dataset | Front&Back-End | – | ✓ | – | ✓ | – |
| DeepCircuitX | Dataset | Front&Back-End | ✓ | ✓ | – | – | – |
| R2G | Graph Views | Back-End | – | ✓ | – | – | ✗ |
| **UnderRTL** | Representation | Front-End | ✓ | ✓ | ✓ | ✓ | ✓ |

*Linked Impl.*: evaluation using implementations with explicit functional relationships.
*Shift Eval.*: evaluation under distribution or design-condition shifts.
*Unified Rep.*: common representation-extraction and downstream-evaluation protocols across model families.

## 2 Related Works

**RTL Datasets and Benchmarks.** Existing RTL datasets and benchmarks cover complementary aspects of circuit learning and evaluation. Generation benchmarks assess generated RTL through functional correctness and, in the latter case, synthesis-based implementation quality (Liu et al., 2023; Lu et al., 2024). CircuitNet 3.0 (Wang et al., 2026) and DeepCircuitX (Li et al., 2025) provide large-scale multi-stage circuit data for tasks such as code understanding, generation, and PPA prediction. More recently, R2G (Zhou et al., 2026) benchmarks circuit graph views under controlled protocols, showing that representation choice can outweigh model choice in physical-design prediction. As summarized in Table 1, UnderRTL shares this representation-centric view but targets front-end RTL, comparing representations across paradigms under a shared protocol and linking functional and implementation evaluation through verified equivalent implementations.

**RTL Representation Methods.** RTL representation methods differ in how representations are constructed and what circuit information they encode. **Engineered-feature methods** derive explicit structural and synthesis-related descriptors from RTL circuit graphs (Fang et al., 2023; Sengupta et al., 2022). **Text-based methods** encode RTL as token sequences using pretrained or hardware-adapted language models (Zhao et al., 2025; Liu et al., 2025). **Graph-based methods** capture connectivity, data/control dependencies, and execution structure through graph neural architectures (Vasudevan et al., 2021; Ma et al., 2026; Liu et al., 2026b; Xu et al., 2023). **Multimodal methods** combine textual and structural views through feature fusion or cross-modal alignment (Fang et al., 2025; Zhao et al., 2026; Liu et al., 2026a). Because these paradigms expose models to different semantic, structural, and implementation cues, UnderRTL evaluates representative methods from all four categories under a shared representation-extraction and downstream evaluation protocol, following probing practice to measure the task-relevant information recoverable from frozen representations by a shared head (Hewitt & Liang, 2019; Belinkov, 2022).

## 3 Benchmark Design and Data Construction

In this section, we present the evaluation design and data construction of UnderRTL. Section 3.1 defines the two evaluation axes and six linked tasks, while Section 3.2 describes the source corpus and the construction and validation of functional annotations, controlled variants, PPA labels, and natural-language queries. Together, they establish the foundation for the diagnostic analyses.

### 3.1 Evaluation Axes and Tasks

Let $C$ denote an RTL circuit and $S \in \mathcal{S}(C)$ one of its subgraphs, with representations $\mathbf{z}_C$ and $\mathbf{z}_S$, respectively. UnderRTL evaluates RTL representations along two complementary axes through the six tasks summarized in Table 2. *Functional understanding* measures whether a representation captures behaviorally relevant semantics, including a circuit's functional role and the equivalence relationships that should persist across alternative RTL implementations. *Implementation awareness* measures whether circuit- and subgraph-level representations retain implementation-specific infor-

Table 2: Tasks and evaluation data in UnderRTL.

| Task | Key Design | #Subtasks | #Samples | Metric |
|---|---|---|---|---|
| *Functional Understanding* | | | | |
| Functional Classification | Coarse/fine function prediction; expert-audited labels | 2 | 7,075 | Acc. / Macro-F1 |
| Functional Equivalence | Eq./non-eq. prediction with formally verified variants | 1 | 11,693 / 17,363 | AP |
| Function-Aware Retrieval | Functional relevance with hierarchical descriptions | 1 | 7,075 | MRR |
| *Implementation Awareness* | | | | |
| PPA Prediction | Area/power/timing prediction across 6 design settings; IP-level OOD | 5 | 7,075 / 36,082 | $R^2$ |
| PPA Ranking | Area/power/WNS ranking within equivalent groups | 3 | 1,054 | Pairwise Acc. |
| PPA-Aware Retrieval | Area/power/timing preferences over equivalent groups | 3 | 3,162 | NDCG@1 |

*#Subtasks* counts distinct prediction objectives, excluding PDK-constraint settings.

mation predictive of synthesized PPA under a given PDK–constraint setting. Functionally equivalent implementations connect the two axes: they share functional semantics while potentially exhibiting different PPA. This enables two linked diagnostics: *functional invariance*, which examines whether shared functional information is preserved across equivalent implementations, and *implementation sensitivity*, which examines whether their PPA-relevant differences remain distinguishable.

**Functional Understanding Tasks.** We evaluate functional information from individual designs to cross-design matching. At the single-circuit level, *Functional Classification* predicts the functional category of a circuit, $\hat{y}_C = h_{\text{cls}}(\mathbf{z}_C)$, examining whether its representation captures explicit functional semantics. Moving beyond individual circuits, *Functional Equivalence* predicts whether two implementations realize the same functionality, $\hat{p}_{ij}^{\text{eq}} = h_{\text{eq}}(\mathbf{z}_{C_i}, \mathbf{z}_{C_j})$, testing whether functional information remains consistent across implementation differences. Finally, *Function-Aware Retrieval* ranks a candidate set $\mathcal{C}$ for a natural-language functional query $q$ using $r_{\text{func}}(q, C) = h_{\text{func}}(\mathbf{z}_q, \mathbf{z}_C)$, evaluating whether the learned semantics support function-driven design matching and reuse.

**Implementation Awareness Tasks.** Let $e = (p, c)$ denote a fixed PDK-constraint setting. We first use *PPA Prediction* to examine whether circuit- and subcircuit-level representations retain implementation information, e.g., $\hat{y}_{C,t}^{(e)} = h_t^{(e)}(\mathbf{z}_C)$ and $\hat{y}_{S,\text{Slack}}^{(e)} = h_{\text{Slack}}^{(e)}(\mathbf{z}_S)$. We then move from absolute prediction to relative comparison with *PPA Ranking*, where an equivalent-implementation group $\mathcal{G} = \{C_1, \ldots, C_m\}$ is ordered according to a specified area, power, or timing objective. Finally, *PPA-Aware Retrieval* combines a functional description with an implementation preference and ranks candidates within $\mathcal{G}$, evaluating whether representations can support preference-driven selection among functionally equivalent implementations.

### 3.2 Task Data Construction and Validation

**Source Data and Splits.** We collect raw RTL designs from CircuitNet 3.0 (Wang et al., 2026) and, after deduplication and benchmark-specific quality filtering, retain 7,075 circuits as the source corpus. CircuitNet 3.0 serves only as a source of RTL code; all task data described below are newly constructed for UnderRTL. For out-of-distribution evaluation, we additionally collect an independent set of larger IP-level designs from OpenCores (OpenCores). We assign all data from the same source circuit to the same split to prevent cross-variant leakage.

**Functional Labels.** Based on expert experience, we define a hierarchical functional taxonomy with 6 coarse categories and 5 fine-grained categories under each coarse class, supporting evaluation at different levels of functional granularity. We adopt a multi-stage annotation procedure: an initial LLM assigns functional labels with explanations, a stronger LLM reviews the initial annotation under a verification protocol, and uncertain or inconsistent cases are resolved by expert review. A stratified expert audit is further conducted to assess annotation quality.

**Functional Variants.** To construct controlled functional relationships, we generate both functionally equivalent and non-equivalent variants of the source circuits. LLM-based RTL rewriting provides flexible changes in coding style and structure for equivalent implementations, while targeted AST mutations introduce localized semantic changes while preserving much of the original syntax and structure, producing challenging non-equivalent examples. All candidate relationships are formally verified with Yosys (Wolf et al., 2013), and only cases with conclusive results are retained, yielding 11,693 equivalent and 17,363 non-equivalent circuit pairs.

**PPA Labels and Ranking Groups.** We synthesize all 7,075 designs across 4 PDKs spanning a broad technology range from 7 nm to 130 nm (ASAP7, SMIC28, NanGate45, and SkyWater130) under a common 0.5 ns clock constraint. To further study constraint shifts, NanGate45 is additionally evaluated at 0.4 ns and 0.6 ns. We use Synopsys Design Compiler for synthesis and PrimeTime for timing and power analysis, following a consistent flow across all settings. For PPA ranking and PPA-aware retrieval, equivalent implementations are evaluated under the same environment, and only groups with sufficiently separated PPA values are retained to avoid ambiguous orderings.

**Natural-Language Queries.** Following the hierarchical summarization strategy (Zhao et al., 2026), we construct circuit-level functional descriptions by combining module-level summaries with their structural relationships, allowing queries to capture both local and global behavior. Original identifiers are excluded from the final queries to reduce direct lexical cues and encourage descriptions centered on functionality. For PPA-aware retrieval, each functional description is further paired with area-, power-, and timing-oriented preferences, producing three queries per equivalent-implementation group that share the same function but differ in implementation objective.

# 4 Unified Benchmarking Framework

UnderRTL standardizes heterogeneous input modalities (Section 4.1), integrates models and extracts representations while preserving their native inputs and learning objectives (Section 4.2), and applies a shared downstream evaluation protocol (Section 4.3) to enable consistent comparison across representation paradigms. This preserves methodological diversity while improving comparability.

## 4.1 Data Standardization

UnderRTL organizes circuit data into two modalities: the *text modality*, including RTL code and functional summaries, and the *graph modality*, including control–data flow graphs (CDFGs) and register-interaction graphs (RIGs). Circuit and subgraph data are constructed as needed, with functional summaries kept separate from downstream task retrieval queries. Each model uses only the modalities required by its original design. In contrast, we align different modalities and derived data through unique circuit identifiers to maintain consistent correspondence across models and tasks.

## 4.2 Model Integration and Representation Extraction

UnderRTL integrates 24 engineered-feature, text, graph, and multimodal methods. We maintain each method's original input format rather than enforcing a uniform design. Pretrained methods follow their inference settings; methods with native objectives use their original pretraining objectives, and supervised or non-circuit-specific methods are adapted to a common pretraining objective for RTL representation learning. For these models, the checkpoint with the lowest pretraining loss is chosen based on the relevant objective, ensuring a consistent model-selection rule across various methods. The representations are extracted and cached through a common interface as circuit-level and subgraph-level embeddings for downstream evaluation.

## 4.3 Unified Downstream Evaluation

All downstream tasks follow a frozen-representation protocol: encoders remain fixed, and all methods use the same prediction-head architecture and training setting for each task, reducing variation from task-specific adaptation and enabling more consistent representation comparison. Retrieval tasks additionally share a fixed query encoder, ensuring consistent query representations across methods. Source circuits are split into training, validation, and test sets at a 60%:20%:20% ratio,

Table 3: Overall results. Best and second-best results are **bolded** and underlined. Cls./Eq./Ret. denote classification/equivalence/retrieval. Cls. reports coarse Accuracy / fine Macro-F1; PPA results follow Area / Power / WNS, with Timing used for PPA-Aware Retrieval. Leading zeros are omitted.

| Model | Functional Understanding | | | Implementation Awareness | | |
|---|---|---|---|---|---|---|
| | Cls. Acc./F1 ↑ | Eq. AP ↑ | Ret. MRR ↑ | Prediction $R^2$ ↑ | Ranking Pair Acc. ↑ | PPA Ret. NDCG@1 ↑ |
| *Feature Engineering* | | | | | | |
| Full features | .768 / .490 | .767 | .218 | .284 / .634 / .686 | **.632** / **.629** / .601 | .627 / .617 / .614 |
| Structure features | .762 / .482 | .744 | .217 | −.232 / **.684** / .637 | .626 / .600 / .610 | .637 / .619 / .641 |
| *Graph* | | | | | | |
| GCN | .614 / .262 | .677 | .096 | .209 / .110 / .082 | .599 / .585 / .610 | .610 / .585 / .626 |
| GIN | .603 / .274 | .667 | .089 | −.424 / −.040 / .056 | .580 / .577 / .546 | .585 / .558 / .576 |
| Graphormer | .625 / .319 | .693 | .077 | .226 / .172 / .080 | .584 / .568 / .553 | .572 / .559 / .617 |
| SGFormer | .628 / .289 | .723 | .085 | .325 / .199 / .077 | .621 / .603 / .620 | .610 / .584 / .591 |
| GraphGPS | .650 / .284 | .699 | .090 | .335 / .259 / .070 | .583 / .573 / .607 | .577 / .564 / .598 |
| RiemannGFM | .578 / .316 | .670 | .038 | .216 / .124 / .034 | .519 / .506 / .507 | .537 / .523 / .569 |
| DynamicRTL | .677 / .354 | .721 | .111 | .118 / .170 / .305 | .603 / .613 / **.627** | .611 / .609 / **.642** |
| *Text* | | | | | | |
| Qwen2.5-Coder | .852 / .591 | .788 | .347 | .332 / .355 / .376 | .607 / .574 / .552 | .620 / .561 / .585 |
| DeepSeek-Coder | .855 / .480 | .769 | .333 | .530 / .498 / .408 | .630 / .611 / .576 | .645 / .587 / .571 |
| Qwen3-8B | .849 / .538 | **.789** | .316 | .482 / −.032 / **.688** | .612 / .609 / .554 | .626 / .573 / .592 |
| Qwen3-E-8B | **.873** / **.698** | .711 | **.482** | **.552** / .551 / .491 | .614 / .589 / .601 | .633 / .582 / .592 |
| NV-Embed-v1 | .842 / .610 | .740 | .351 | .327 / .268 / .680 | .589 / .581 / .597 | .610 / .571 / .594 |
| NV-Embed-v2 | .852 / .611 | .729 | .369 | .439 / .353 / .544 | .596 / .581 / .597 | .627 / .591 / .566 |
| CodeV-QC-7B | .858 / .599 | .752 | .337 | .302 / .428 / .283 | .626 / .591 / .608 | .621 / .571 / .580 |
| CodeV-All-QC | .854 / .596 | .750 | .338 | .487 / .473 / .190 | .624 / .586 / .608 | .637 / .572 / .590 |
| CodeV-R1-7B | .855 / .591 | .782 | .358 | .520 / .360 / .272 | .630 / .586 / .598 | .629 / .552 / .586 |
| RTLCoder-DS | .848 / .455 | .788 | .313 | **.552** / .518 / .391 | .619 / .586 / .585 | **.648** / .585 / .595 |
| *Multimodal* | | | | | | |
| GraphCLIP | .683 / .361 | .675 | .126 | −499.762 / −2.395 / .043 | .614 / .595 / .590 | .602 / **.620** / .600 |
| UniGraph | .682 / .169 | .647 | .057 | .142 / .107 / .155 | .557 / .500 / .569 | .526 / .505 / .585 |
| UniGraph2 | .610 / .281 | .710 | .085 | .235 / .105 / .084 | .594 / .552 / .555 | .578 / .546 / .618 |
| CircuitFusion | .674 / .226 | .661 | .091 | .279 / .231 / .231 | .597 / .539 / .568 | .547 / .537 / .568 |
| TopoRTL | .703 / .417 | .701 | .105 | **.552** / .205 / .120 | .608 / .583 / .582 | .572 / .562 / .591 |

with all derived variants, subgraphs, queries, and groups inheriting the source split to avoid information leakage. Downstream checkpoints are selected on the validation set using the corresponding primary metric, and test results are averaged over three random seeds.

## 5 Experiments

We evaluate the representation methods integrated in UnderRTL on the six tasks defined in Section 3, following the unified evaluation protocol in Section 4. We first present the overall benchmark results to compare different representation paradigms across functional understanding and implementation awareness. Beyond overall performance, we investigate what information supports downstream predictions (**RQ1**), whether functional semantics remain robust across equivalent implementations (**RQ2**), how functional invariance interacts with implementation sensitivity (**RQ3**), and how these capabilities generalize across unseen designs and EDA settings (**RQ4**).

### 5.1 Overall Results

Table 3 compares all methods across the six UnderRTL tasks. We draw the main observation:

**Obs: Representation strengths are highly task-dependent.** Text models dominate functional classification and function-aware retrieval: even the weakest text model outperforms the strongest non-text baseline in coarse classification (0.842 vs. 0.768) and retrieval (0.313 vs. 0.218), while explicit features lead several implementation-aware subtasks. Related tasks also show performance

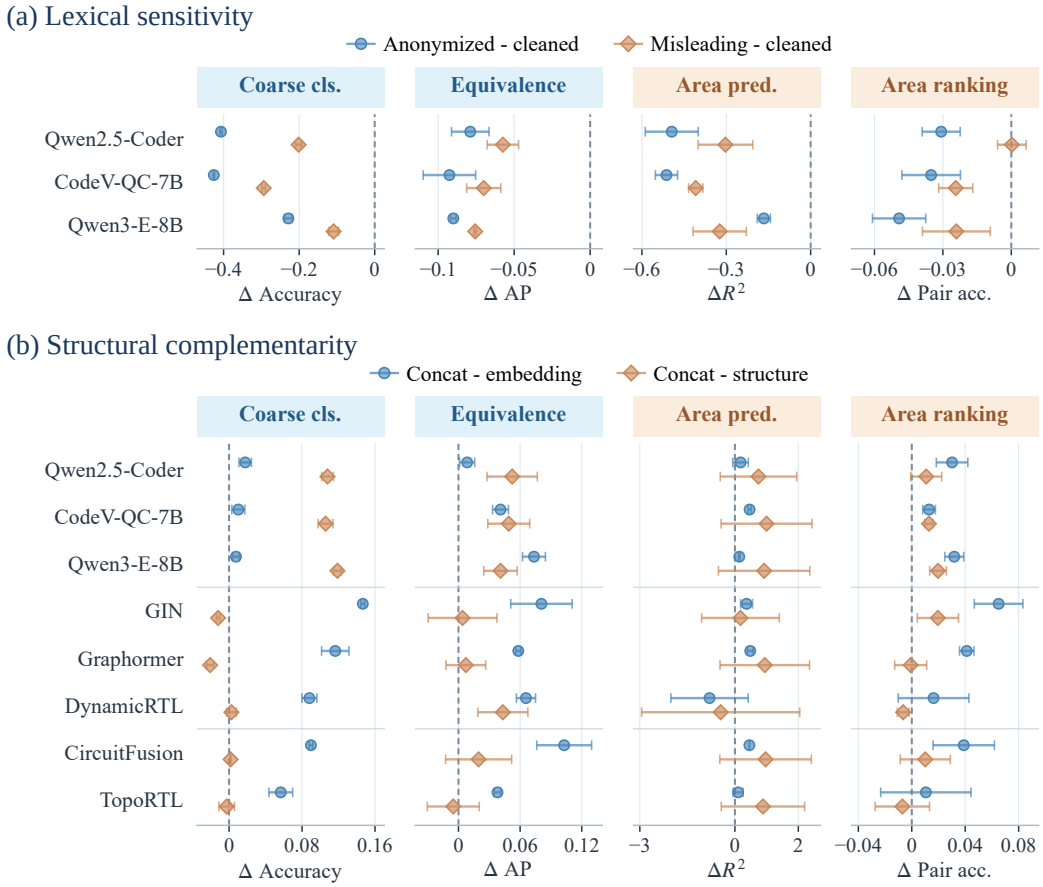


Figure 2: **RQ1.** Lexical sensitivity and structural complementarity of representative RTL representations. (a) reports performance changes of anonymized and misleading RTL relative to the cleaned RTL baseline. (b) compares the concatenated representation $[\mathbf{E}; \mathbf{S}]$ against the learned embedding $\mathbf{E}$ and explicit structural features $\mathbf{S}$. Points and error bars denote the mean and standard deviation.

reversals: Qwen3-Embedding-8B leads fine-grained classification and function-aware retrieval but trails Qwen3-8B on functional equivalence (0.711 vs. 0.789). Thus, no representation paradigm uniformly dominates, and individual tasks are not interchangeable measures of representation quality.

### 5.2 RQ1: Encoding

We first investigate *what information current RTL representations rely on*, focusing on lexical sensitivity and structural complementarity. Predictors trained on cleaned RTL embeddings are evaluated under anonymized and misleading variants, while learned embeddings $\mathbf{E}$ are compared with $[\mathbf{E}; \mathbf{S}]$ using explicit graph-structural features $\mathbf{S}$. Results are reported on representative models and tasks from both evaluation axes in Figure 2.

**Obs: Current RTL representations remain lexically sensitive and incompletely capture structural information.** Lexical perturbations reduce performance in 23 of 24 model–task–perturbation settings across both evaluation axes. Adding explicit structural features also improves several representations; for example, Qwen3-Embedding-8B raises Functional Equivalence AP from 0.711 to 0.785, above the structure-only baseline of 0.744. These results indicate that current embeddings can depend on unstable lexical cues while leaving useful structural information uncaptured.

### 5.3 RQ2: Invariance

We next investigate *where RTL representations fail in functional equivalence prediction*. Test pairs are grouped by canonical-token, AST, and CDFG distances, measured using Levenshtein distance

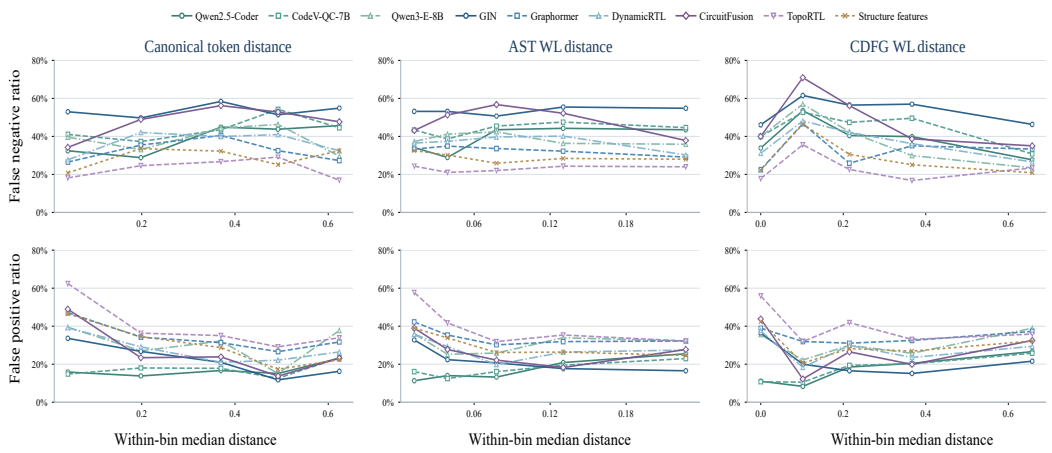


Figure 3: **RQ2.** Failure patterns in functional equivalence across implementation distances. Circuit pairs are grouped into five bins by canonical-token, AST-WL, and CDFG-WL distances, with the within-bin median shown on the x-axis. The top and bottom rows report false negative rate (FNR) on equivalent pairs and false positive rate (FPR) on non-equivalent pairs, respectively; lower is better.

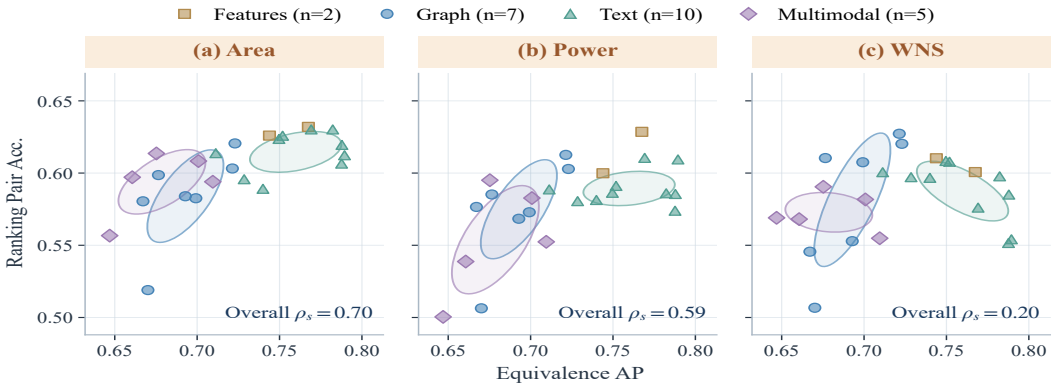


Figure 4: **RQ3.** Relationship between Functional Equivalence and PPA Ranking. Points denote models averaged over three seeds; colors/markers indicate representation families, and ellipses show one-standard-deviation covariance regions. Each panel reports the overall Spearman correlation $\rho_s$.

for tokens and Weisfeiler-Lehman cosine distance for ASTs and CDFGs. This analysis reveals how equivalence failures vary with implementation difference and representation type, shown in Figure 3.

**Obs: Equivalence failures are neither monotonic with implementation distance nor symmetric across positive and negative pairs.** Under CDFG distance, all evaluated representations reach their highest FNR in the second rather than the farthest bin; Qwen2.5-Coder-7B, for example, drops from 53.6% to 27.7%. Lower FNR can also accompany higher FPR: in the closest token-distance bin, TopoRTL obtains 18.2% FNR and 62.4% FPR, versus 32.4% and 15.9% for Qwen2.5-Coder-7B. Thus, implementation distance alone does not determine equivalence difficulty, and aggregate scores can conceal distinct failure modes.

### 5.4 RQ3: Invariance-Sensitivity Interplay

We further investigate *how functional understanding relates to implementation awareness across RTL representations*. For each model, we plot functional equivalence AP against PPA ranking performance for Area, Power, and WNS. This comparison examines whether stronger functional discrimination is associated with better preservation of implementation information, or whether the two capabilities exhibit different behaviors across models. Models are colored by representation family to reveal both overall relationships between the two capabilities and family-specific patterns.

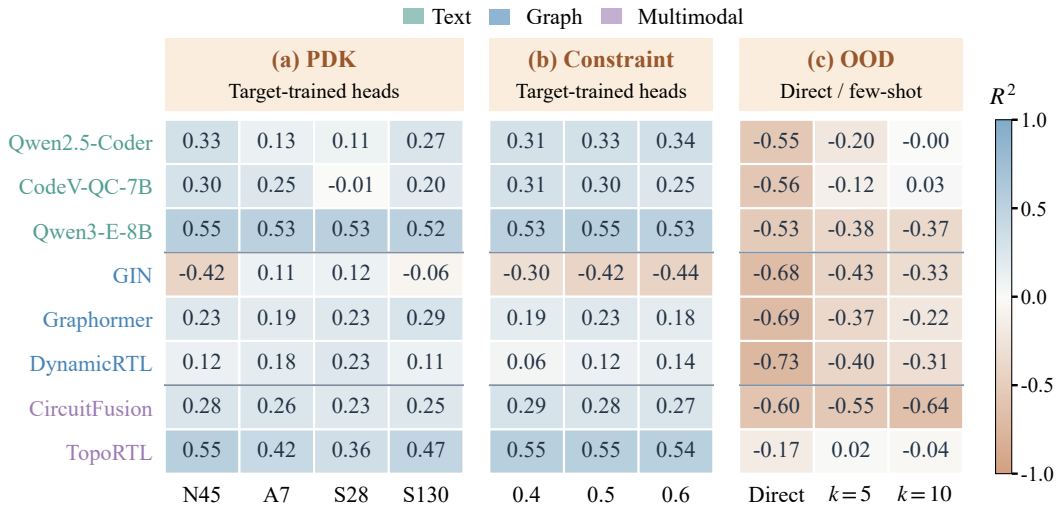


Figure 5: **RQ4.** Generalization results for Area prediction. For PDK and constraint shifts, representations are fixed while prediction heads are trained separately with labels from each target setting. OOD evaluates direct transfer to larger unseen designs and few-shot adaptation with $k$ labeled target circuits. Values report mean $R^2$ over three seeds.

**Obs: The invariance-sensitivity relationship depends strongly on the PPA objective.** Across all models, Functional Equivalence AP correlates positively with Area and Power ranking ($\rho_s = 0.70/0.59$), but only weakly with WNS ranking ($\rho_s = 0.20$). This shows that functional and implementation capabilities do not exhibit a universal trade-off, and their relationship varies with the implementation objective.

**Obs: Overall correlations can conceal distinct relationships across representation families.** For WNS, graph models show a strong positive correlation between Equivalence AP and ranking accuracy ($\rho_s = 0.82$), whereas text models exhibit a negative correlation ($\rho_s = -0.62$), despite the weak overall relationship. The negative trend for text models is also consistent across three seeds, indicating that the invariance-sensitivity interplay is representation-dependent.

### 5.5 RQ4: Generalization

Finally, we investigate *how RTL representations generalize across implementation environments and unseen designs*. We evaluate adaptation across PDKs/constraints and direct or few-shot transfer to OOD circuits, distinguishing environmental adaptability from design-distribution transfer.

**Obs: Environmental adaptability does not imply OOD transfer, and few-shot supervision only partially recovers performance.** Several representations remain effective with target-trained heads across PDKs and constraints; for example, Qwen3-Embedding-8B maintains Area $R^2 > 0.51$ across all PDKs. However, all evaluated models obtain negative Area $R^2$ under direct transfer to larger unseen designs. Although seven of eight models improve with $k = 10$ target samples, most remain below zero, showing that environmental adaptability and OOD transfer are distinct capabilities.

## 6 Conclusion

In this paper, we present **UnderRTL**, a unified diagnostic benchmark for RTL representation learning that jointly evaluates *functional understanding* and *implementation awareness* through six linked tasks and 24 representative methods. Beyond overall performance, UnderRTL reveals that current representations remain sensitive to lexical changes, do not consistently capture useful structural information, and exhibit distinct failure patterns in functional equivalence. We further find that functional and implementation capabilities show no universal trade-off, while strong adaptation across PDKs or constraints does not necessarily translate to robust transfer across unseen designs. These findings highlight the need for RTL representations that better capture functional semantics, implementation-specific information, and generalizable circuit knowledge.

## AI USE STATEMENT

In preparing this manuscript, we used Large Language Models (LLMs) as general-purpose writing assistance tools for language polishing and grammatical correction. Specifically, LLMs were employed to identify syntactic errors, improve sentence clarity, and refine academic phrasing in non-technical portions of the manuscript. All LLM-assisted content was carefully reviewed and verified by the authors to ensure accuracy and preserve scientific integrity. We take responsibility for all content presented in this manuscript and have followed ICLR's policies regarding the use of LLMs.

## ETHICS STATEMENT

This work introduces UnderRTL, a benchmark for RTL representation learning built from existing circuit datasets and open-source RTL designs, without involving personal data or human subjects. LLMs are used for controlled annotation and data construction, with verification and expert review of the resulting data; language-model encoders are also evaluated as benchmark baselines. We follow the ICLR Code of Ethics and encourage transparent reporting, reproducible evaluation, and appropriate human oversight in AI-assisted hardware design.

## REPRODUCIBILITY STATEMENT

We describe the benchmark design, evaluation protocol, and experimental settings in the paper and appendix: task design in Appendix B, data construction and validation in Appendix C, input views, pretraining positives, pooling rules, and downstream heads in Appendix D, and all generation and annotation prompts in Appendix F. Upon acceptance, we will publicly release the UnderRTL dataset, data splits, processing scripts, model configurations, and complete benchmark codebase to enable reproduction of all reported results and further research on RTL representation learning.

## REFERENCES

Yonatan Belinkov. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219, 2022.

Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, and Zhiyao Xie. Masterrtl: A pre-synthesis ppa estimation framework for any rtl design. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9, 2023. doi: 10.1109/ICCAD57390.2023.10323951.

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, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yifan Wu, YK Li, et al. Deepseek-coder: when the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Yufei He, Yuan Sui, Xiaoxin He, and Bryan Hooi. Unigraph: Learning a unified cross-domain foundation model for text-attributed graphs. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 448–459, 2025a.

Yufei He, Yuan Sui, Xiaoxin He, Yue Liu, Yifei Sun, and Bryan Hooi. Unigraph2: Learning a unified embedding space to bind multimodal graphs. In *Proceedings of the ACM on Web Conference 2025*, pp. 1759–1770, 2025b.

John Hewitt and Percy Liang. Designing and interpreting probes with control tasks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp)*, pp. 2733–2743, 2019.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Nv-embed: Improved techniques for training llms as generalist embedding models. In *International Conference on Learning Representations*, volume 2025, pp. 79310–79333, 2025.

Zeju Li, Changran Xu, Zhengyuan Shi, Zedong Peng, Yi Liu, Yunhao Zhou, Lingfeng Zhou, Chengyu Ma, Jianyuan Zhong, Xi Wang, Jieru Zhao, Zhufei Chu, Xiaoyan Yang, and Qiang Xu. Deepcircuitx: A comprehensive repository-level dataset for rtl code understanding, generation, and ppa analysis. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, pp. 204–211, 2025. doi: 10.1109/ICLAD65226.2025.00029.

Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. Verilogeval: Evaluating large language models for verilog code generation. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–8. IEEE, 2023.

Shang Liu, Wenji Fang, Yao Lu, Jing Wang, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. Rtlcoder: Fully open-source and efficient llm-assisted rtl code generation technique. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 44(4):1448–1461, 2024.

Yi Liu, Hongji Zhang, Yunhao Zhou, Zhengyuan Shi, Changran Xu, and Qiang Xu. DeepRTL2: A versatile model for RTL-related tasks. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6485–6500, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.336. URL https://aclanthology.org/2025.findings-acl.336/.

Yi Liu, Hongji Zhang, Lei Chen, Mingxuan Yuan, and Qiang Xu. UniRTL: Unifying code and graph for robust RTL representation learning. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=hzYEWSTOEA.

Yi Liu, Hongji Zhang, Yiwen Wang, Dimitris Tsaras, Lei Chen, Mingxuan Yuan, and Qiang Xu. Beyond tokens: Enhancing RTL quality estimation via structural graph learning. In *Forty-third International Conference on Machine Learning*, 2026b. URL https://openreview.net/forum?id=n86vBxkWIt.

Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. Rtllm: An open-source benchmark for design rtl generation with large language model. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 722–727. IEEE, 2024.

Ruiyang Ma, Yunhao Zhou, Yipeng Wang, Yi Liu, Zhengyuan Shi, Ziyang Zheng, Kexin Chen, Zhiqiang He, Lingwei Yan, Gang Chen, Qiang Xu, and Guojie Luo. Dynamicrtl: Rtl representation learning for dynamic circuit behavior. In *Proceedings of the Fortieth AAAI Conference on Artificial Intelligence and Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence and Sixteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'26/IAAI'26/EAAI'26. AAAI Press, 2026. ISBN 978-1-57735-906-7. doi: 10.1609/aaai.v40i1.37051. URL https://doi.org/10.1609/aaai.v40i1.37051.

Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1st edition, 1994. ISBN 0070163332.

OpenCores. OpenCores. https://opencores.org/.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in neural information processing systems*, 35:14501–14515, 2022.

Prianka Sengupta, Aakash Tyagi, Yiran Chen, and Jiang Hu. How good is your verilog rtl code? a quick answer from machine learning. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2022.

Li Sun, Zhenhao Huang, Suyang Zhou, Qiqi Wan, Hao Peng, and Philip Yu. Riemanngfm: Learning a graph foundation model from riemannian geometry. In *Proceedings of the ACM on Web Conference 2025*, pp. 1154–1165, 2025.

Gemma Team. Gemma 4 technical report, 2026. URL https://arxiv.org/abs/2607.02770.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Shobha Vasudevan, Wenjie Jiang, David Bieber, Rishabh Singh, HAMID SHOJAEI, C. Richard Ho, and Charles Sutton. Learning semantic representations to verify hardware designs. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=oIhzg4GJeOf.

Mingjun Wang, Yihan Wen, Yuntao Lu, Fengrui Liu, Yuxiang Zhao, Boyu Han, Jianan Mu, Yibo Lin, Runsheng Wang, Bei Yu, and Huawei Li. Circuitnet 3.0: A multi-modal dataset with task-oriented augmentation for AI-driven circuit design. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=lEDb4gQ4dB.

Clifford Wolf, Johann Glaser, and Johannes Kepler. Yosys-a free verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, volume 97, pp. 1–6, 2013.

Qitian Wu, Wentao Zhao, Chenxiao Yang, Hengrui Zhang, Fan Nie, Haitian Jiang, Yatao Bian, and Junchi Yan. Sgformer: Simplifying and empowering transformers for large-graph representations. *Advances in neural information processing systems*, 36:64753–64773, 2023.

Ceyu Xu, Pragya Sharma, Tianshu Wang, and Lisa Wu Wills. Fast, robust and transferable prediction for hardware logic synthesis. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 167–179, 2023.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in neural information processing systems*, 34:28877–28888, 2021.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

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

Yang Zhao, Di Huang, Chongxiao Li, et al. Codev: Empowering llms with hdl generation through multi-level summarization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2025. doi: 10.1109/TCAD.2025.3604320.

Zewei Zhou, Jiajun Zou, Jiajia Zhang, Ao Yang, Ruichao He, Haozheng Zhou, Ao Liu, Jiawei Liu, Leilei Jin, Shan Shen, and Daying Sun. R2g: A multi-view circuit graph benchmark suite from rtl to gdsii. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18827–18836, June 2026.

Yaoyu Zhu, Di Huang, Hanqi Lyu, Xiaoyun Zhang, Chongxiao Li, Wenxuan Shi, Yutong Wu, Jianan Mu, Jinghua Wang, Pengwei Jin, et al. Qimeng-codev-r1: Reasoning-enhanced verilog generation. *Advances in Neural Information Processing Systems*, 38:154266–154300, 2026.

Yun Zhu, Haizhou Shi, Xiaotang Wang, Yongchao Liu, Yaoke Wang, Boci Peng, Chuntao Hong, and Siliang Tang. Graphclip: Enhancing transferability in graph foundation models for text-attributed graphs. In *Proceedings of the ACM on Web Conference 2025*, pp. 2183–2197, 2025.

# Appendix Table of Contents

**A Limitations and Future Work** **14**

**B Benchmark and Task Details** **14**

B.1 Evaluation Axes and Task Definitions . . . . . 14

B.2 Evaluation Metrics . . . . . 15

B.3 Benchmark Statistics . . . . . 16

**C Dataset Construction Details** **16**

C.1 Source Corpus and Benchmark Curation . . . . . 16

C.2 Functional Labels . . . . . 17

C.3 Functional Variants . . . . . 18

C.4 PPA Annotation . . . . . 19

C.5 PPA Ranking Groups . . . . . 20

C.6 Natural-Language Query Construction . . . . . 20

**D Unified Benchmarking Framework Details** **21**

D.1 Standardized Input Representations . . . . . 21

D.2 Evaluated Representation Methods . . . . . 21

D.3 Pretraining and Representation Extraction . . . . . 22

D.4 Downstream Evaluation Protocol . . . . . 23

**E Additional Experimental Results** **23**

E.1 Complete Benchmark Results . . . . . 23

E.2 Additional RQ1: Encoding Analysis . . . . . 25

E.3 Additional RQ2: Invariance Analysis . . . . . 27

E.4 Additional RQ3: Invariance–Sensitivity Interplay . . . . . 29

E.5 Additional RQ4: Generalization . . . . . 31

E.6 Further Analysis . . . . . 32

**F Annotation and Prompt Details** **34**

F.1 Functional Annotation Prompts . . . . . 34

F.2 Functional Variant Generation Prompts . . . . . 39

F.3 Functional Description Prompts . . . . . 40

F.4 Retrieval Query Prompts . . . . . 40

# A Limitations and Future Work

While UnderRTL provides a unified diagnostic benchmark for RTL representation learning, several limitations warrant attention. First, our evaluation focuses on frozen representations under a shared downstream protocol, and therefore does not characterize the potential gains from task-specific encoder fine-tuning. Second, the current PPA labels are derived from logic synthesis, and the main benchmark consists predominantly of module-level designs; although the out-of-distribution set partially covers larger IP-level designs, extending the benchmark to post-layout metrics and larger system-level designs would better reflect more complex implementation scenarios. Additionally, the source RTL retains human-written identifiers, which may introduce lexical cues even though equivalence and PPA supervision are obtained independently through formal verification and EDA tools. Future work could further incorporate fine-tuned evaluation, richer representation analyses, larger-scale designs, and identifier-robust settings to broaden the diagnostic scope of UnderRTL. We hope UnderRTL can serve as a useful step toward more unified and systematic evaluation of RTL representations, supporting future research at the intersection of AI and EDA.

# B Benchmark and Task Details

## B.1 Evaluation Axes and Task Definitions

**Evaluation axes.** UnderRTL evaluates RTL representations along two complementary axes. *Functional understanding* measures whether a representation captures the semantics needed to recognize circuit functionality, identify equivalent functions across implementations, and support function-driven retrieval. *Implementation awareness* measures whether it retains implementation-specific information relevant to PPA prediction, comparison, and preference-driven selection. Functionally equivalent implementations link the two axes by sharing functional semantics while exhibiting PPA-relevant implementation differences.

**Notation.** Let $C$ denote an RTL circuit and $S \in \mathcal{S}(C)$ one of its subgraphs. We denote their representations by $\mathbf{z}_C$ and $\mathbf{z}_S$, and the representation of a natural-language query $q$ by $\mathbf{z}_q$. An implementation setting $e = (p, c)$ consists of a PDK $p$ and synthesis constraint $c$. Let $f(C)$ denote the function implemented by $C$, and $\mathcal{G} = \{C_1, \ldots, C_m\}$ an equivalent-implementation group satisfying $f(C_i) = f(C_j)$ for all $C_i, C_j \in \mathcal{G}$.

**Functional Classification.** Given $\mathbf{z}_C$, the task predicts either the coarse or fine-grained functional category:

$$\hat{y}_C^{(l)} = h_{\text{cls}}^{(l)}(\mathbf{z}_C), \qquad l \in \{\text{coarse}, \text{fine}\}. \tag{1}$$

This task evaluates whether circuit representations expose functional semantics at different levels of granularity.

**Functional Equivalence.** Given a circuit pair $(C_i, C_j)$, the task predicts

$$y_{ij}^{\text{eq}} = \mathbf{1}[f(C_i) = f(C_j)], \qquad \hat{p}_{ij}^{\text{eq}} = h_{\text{eq}}(\mathbf{z}_{C_i}, \mathbf{z}_{C_j}). \tag{2}$$

This task probes whether representations preserve functional consistency across implementation differences while distinguishing non-equivalent circuits.

**Function-Aware Retrieval.** Given a functional query $q$ and candidate circuit $C$, the retrieval score is

$$r_{\text{func}}(q, C) = h_{\text{func}}(\mathbf{z}_q, \mathbf{z}_C). \tag{3}$$

Candidates are ranked by this score. The task evaluates alignment between natural-language functional intent and circuit representations.

**PPA Prediction.** UnderRTL provides five prediction targets: Area, Power, Slack, worst negative slack (WNS), and total negative slack (TNS). Area and Power are circuit-level targets predicted from $\mathbf{z}_C$. Slack is a subgraph-level target predicted from $\mathbf{z}_S$. WNS and TNS are circuit-level timing

Table 4: UnderRTL tasks, subtasks, and primary evaluation metrics.

| Task | Subtask or objective | Primary metric |
|---|---|---|
| Functional Classification | Coarse / Fine | Accuracy / Macro-F1 |
| Functional Equivalence | Equivalent vs. non-equivalent | AP |
| Function-Aware Retrieval | Functional description | MRR |
| PPA Prediction | Area / Power / Slack / WNS / TNS | $R^2$ |
| PPA Ranking | Area / Power / WNS | Pairwise Accuracy |
| PPA-Aware Retrieval | Area / Power / Timing | NDCG@1 |

targets predicted from an aggregation of the subgraph representations, denoted $\rho(\{\mathbf{z}_S : S \in \mathcal{S}(C)\})$:

$$\hat{y}^{(e)}_{C,t} = h^{(e)}_t(\mathbf{z}_C),\ t \in \{\text{Area}, \text{Power}\}, \tag{4}$$

$$\hat{y}^{(e)}_{S,\text{Slack}} = h^{(e)}_{\text{Slack}}(\mathbf{z}_S),\ S \in \mathcal{S}(C), \tag{5}$$

$$\hat{y}^{(e)}_{C,t} = h^{(e)}_t(\rho(\{\mathbf{z}_S\}))\,,\ t \in \{\text{WNS}, \text{TNS}\}. \tag{6}$$

These tasks evaluate whether representations retain information predictive of implementation quality at circuit and local levels.

**PPA Ranking.** Given an equivalent group $\mathcal{G}$ and objective $t \in \{\text{Area}, \text{Power}, \text{WNS}\}$, a ranking head assigns each implementation a score

$$s_{i,t} = h_{\text{rank},t}(\mathbf{z}_{C_i}), \qquad C_i \in \mathcal{G}. \tag{7}$$

The induced ordering is compared with the ground-truth PPA ordering. Because functionality is fixed within $\mathcal{G}$, this task directly evaluates sensitivity to implementation-quality differences.

**PPA-Aware Retrieval.** A query $q_{\mathcal{G},t}$ combines the shared functionality of an equivalent group with an implementation preference $t \in \{\text{Area}, \text{Power}, \text{Timing}\}$. Candidates $C_i \in \mathcal{G}$ are ranked by

$$r_{\text{ppa}}(q_{\mathcal{G},t}, C_i) = h_{\text{ppa}}(\mathbf{z}_{q_{\mathcal{G},t}}, \mathbf{z}_{C_i}). \tag{8}$$

This task evaluates whether representations support preference-driven selection among functionally equivalent implementations.

### B.2 Evaluation Metrics

Table 4 summarizes the primary metric for each task. All reported metrics are computed independently for each random seed on the test split; we then report the arithmetic mean and sample standard deviation over seeds $\{0, 42, 615\}$.

**Classification.** For $N$ examples, Accuracy is $N^{-1}\sum_{i=1}^{N}\mathbf{1}[\hat{y}_i = y_i]$. For fine-grained classification, Macro-F1 averages the classwise F1 scores with equal weight, $K^{-1}\sum_{k=1}^{K}\text{F1}_k$, preventing frequent functional classes from dominating the result.

**Functional Equivalence.** Average Precision (AP) summarizes the precision–recall ranking induced by the predicted positive-class scores. If $\pi$ sorts the $N$ pairs by decreasing score and $N_+$ is the number of positive pairs, then

$$\text{AP} = \frac{1}{N_+}\sum_{k=1}^{N}\text{Prec@}k\ \mathbf{1}[y_{\pi_k} = 1]. \tag{9}$$

**Function-Aware Retrieval.** Each query has one relevant circuit. Let $r_q$ be its one-indexed position in the ranked candidate list. Mean Reciprocal Rank is

$$\text{MRR} = \frac{1}{|\mathcal{Q}|}\sum_{q\in\mathcal{Q}}\frac{1}{r_q}. \tag{10}$$

**PPA Prediction.** All prediction metrics are computed after mapping predictions back to the original label space. For test targets $y_i$ and predictions $\hat{y}_i$,

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}. \tag{11}$$

Table 5: Task statistics under the default evaluation settings. "Val." denotes the validation split.

| Task | Sample Type | Train | Val. | Test | Total |
|---|---|---|---|---|---|
| Functional Classification | Circuits | 4,244 | 1,419 | 1,412 | 7,075 |
| Functional Equivalence | Pairs (pos./neg.) | 2,398/2,398 | 576/576 | 593/593 | 3,567/3,567 |
| Function-Aware Retrieval | Queries | 4,244 | 1,419 | 1,412 | 7,075 |
| PPA Prediction | Area / Power / WNS / TNS | 4,244 | 1,419 | 1,412 | 7,075 |
| | Slack | 22,807 | 6,733 | 6,542 | 36,082 |
| PPA Ranking | Equivalent groups | 616 | 224 | 214 | 1,054 |
| PPA-Aware Retrieval | Preference queries | 1,848 | 672 | 642 | 3,162 |

A negative $R^2$ means that the predictor has a larger squared error than the constant predictor $\hat{y}_i = \bar{y}$ on the same test set.

**PPA Ranking.** For target $t$, let $u_i = d_t y_i$ denote the ground-truth utility of implementation $C_i$, where $d_t = 1$ if larger values are preferred and $d_t = -1$ otherwise, and let $\hat{u}_i$ denote its predicted utility score. For an equivalent implementation group $\mathcal{G}$, we define the set of ground-truth non-tied pairs as

$$\mathcal{P}_{\mathcal{G}} = \{(i, j) : i < j,\ u_i \neq u_j\}. \tag{12}$$

Pairwise Accuracy for $\mathcal{G}$ is

$$\mathrm{PA}(\mathcal{G}) = \frac{1}{|\mathcal{P}_{\mathcal{G}}|} \sum_{(i,j)\in\mathcal{P}_{\mathcal{G}}} \left( \mathbf{1}[(u_i - u_j)(\hat{u}_i - \hat{u}_j) > 0] + \frac{1}{2}\mathbf{1}[\hat{u}_i = \hat{u}_j] \right). \tag{13}$$

Thus, correctly ordered pairs receive one point, incorrectly ordered pairs receive zero, and predicted ties receive half a point. Groups without comparable ground-truth pairs are omitted. Letting $\mathbb{G}_+$ denote the remaining groups, the reported metric is the group-level average

$$\mathrm{PA} = \frac{1}{|\mathbb{G}_+|} \sum_{\mathcal{G}\in\mathbb{G}_+} \mathrm{PA}(\mathcal{G}). \tag{14}$$

Hence, each equivalent group contributes equally regardless of its size.

**PPA-Aware Retrieval.** Within each equivalent group, implementations receive graded relevance from their true PPA ordering: rank one denotes the least preferred implementation and the largest rank denotes the most preferred, with average ranks for ties. If $g_i$ is this gain and $\pi_1$ is the top retrieved implementation, then

$$\mathrm{NDCG@1} = \frac{2^{g_{\pi_1}} - 1}{2^{\max_i g_i} - 1}. \tag{15}$$

NDCG@1 is computed per query/group and then averaged, so it rewards retrieving better implementations even when the top item is not the optimum.

### B.3 Benchmark Statistics

Table 5 reports the samples used under the default evaluation settings. These counts may differ from the full constructed data pools because some tasks apply additional matching or filtering.

Function-Aware Retrieval uses one query per circuit and ranks all circuits from the same split. For Functional Equivalence, the complete verified pool contains 11,693 positive and 17,363 negative pairs; the default distance-matched setting ($\tau = 0.3$) produces the balanced subset reported in Table 5. For PPA Ranking, the table uses the default $5/5/2$ separation setting; the three objectives reuse the same equivalent groups. PPA-Aware Retrieval reuses these groups and creates one Area, one Power, and one Timing query per group.

## C Dataset Construction Details

### C.1 Source Corpus and Benchmark Curation

**Source corpus.** The in-distribution corpus is built from the raw RTL designs in CircuitNet 3.0 (Wang et al., 2026). We use CircuitNet 3.0 only as a source of RTL code, without reusing

**Algorithm 1** Source-level canonical-text deduplication

**Require:** Source Verilog files $\mathcal{D}$
**Ensure:** Retained source files $\mathcal{R}$
Initialize hash buckets $\mathcal{H}$
**for all** $C \in \mathcal{D}$ **do**
  $\widetilde{C} \leftarrow \textsc{CanonicalizeText}(C)$
  $h_C \leftarrow \textsc{Hash}(\widetilde{C})$
  Add $C$ to $\mathcal{H}[h_C]$
**end for**
$\mathcal{R} \leftarrow \emptyset$
**for all** bucket $\mathcal{B}$ in $\mathcal{H}$ **do**
  $C^* \leftarrow \textsc{FirstByFilename}(\mathcal{B})$
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{C^*\}$
**end for**
**return** $\mathcal{R}$

Table 6: Scale comparison between the IID and OOD corpora.

| Dataset | Designs | Circuit Size (Min, Avg, Max) | | | |
|---|---|---|---|---|---|
| | | #LoC | #Cells | #Registers | #Logic levels |
| IID | 7,075 | (3, 66.9, 1K) | (1, 645.8, 29K) | (0, 5.0, 301) | (0, 9.3, 340) |
| OOD | 38 | (139, 1K, 15K) | (26, 12K, 238K) | (1, 96.1, 1K) | (3, 10.7, 40) |

its augmented data, other modalities, or labels. To meet the requirements of representation benchmarking, we apply an additional benchmark-specific curation pass that retains standalone usable designs and reduces residual source-level duplication. After this process, 7,075 RTL designs are retained. All task data built on these designs, including functional labels, variants, PPA annotations, and queries, are newly constructed for UnderRTL.

**Source-level RTL deduplication.** Before constructing task-specific data, we perform source-level deduplication over the collected Verilog designs. Each design is converted into a canonical text representation by removing non-semantic annotations, normalizing formatting, and replacing module, signal, parameter, and instance identifiers with role-specific placeholders. This normalization suppresses differences caused by naming and presentation while preserving the RTL structure used for subsequent comparison. Designs with identical canonical representations are grouped by hash, and one source file is retained from each group using a deterministic filename-based rule.

**OOD corpus.** To evaluate transfer beyond the source distribution, we additionally collect 38 complete IP designs from OpenCores (OpenCores), covering processors and controllers, communication interfaces, cryptographic cores, DSP/arithmetic units, memories and error-correction logic, and system interconnects. We retain designs compatible with the Yosys-based processing flow (Wolf et al., 2013). Compared with the in-distribution corpus, these designs differ in both provenance and circuit scale, as summarized in Table 6. They are used exclusively for transfer evaluation and are not included in representation pretraining or in the downstream training and evaluation of the main benchmark.

The 7,075 source designs are partitioned before any variants, subgraphs, pairs, groups, or queries are constructed. All derived samples inherit the split of their source design, preventing related implementations from crossing the training, validation, and test partitions.

### C.2 Functional Labels

**Functional taxonomy.** To evaluate functional understanding at different semantic granularities, we define a two-level taxonomy containing six coarse categories and five fine-grained categories within each coarse category. Labels are assigned according to the primary functional role of the design rather than incidental internal components. For example, a protocol controller containing counters or FIFOs is categorized by its communication function rather than by the presence of these local structures. Table 7 summarizes the resulting class distribution.

Table 7: Functional taxonomy and class distribution over all 7,075 source designs.

| Coarse | Fine category | N | Coarse | Fine category | N |
|---|---|---|---|---|---|
| Arithmetic&Logic (2,409) | Add/Sub/Compare | 874 | Datapath&Transform (1,049) | Mux/Demux | 429 |
| | Mul/Div/MAC | 708 | | Encoder/Decoder | 307 |
| | Shift/Bit Logic | 347 | | Serializer/Deserializer | 107 |
| | General ALU | 438 | | Width/Format Conversion | 190 |
| | Others | 42 | | Others | 16 |
| Storage&Buffer (811) | Register/Latch/FF | 485 | Communication& Interface (1,080) | Serial Protocol | 281 |
| | FIFO/Queue | 129 | | Bus Interface | 484 |
| | RAM/ROM | 191 | | Clock-Domain Sync | 30 |
| | Cache/Scratchpad | 6 | | Protocol Bridge | 194 |
| | Others | 0 | | Others | 91 |
| Control&State (1,255) | FSM Controller | 315 | Signal/Security/ Acceleration (471) | DSP/Filter/Transform | 145 |
| | Counter/Timer | 745 | | Compute Accelerator | 86 |
| | Arbiter/Scheduler | 66 | | Crypto/CRC/Hash | 124 |
| | Pipeline/Hazard | 79 | | RNG/ECC | 112 |
| | Others | 50 | | Others | 4 |

**Annotation and audit.** We use a two-stage LLM-assisted annotation procedure to reduce dependence on a single model. Gemini 3.1 Pro Preview produces an initial hierarchical label together with a confidence estimate and supporting rationale, and Claude Opus 4.7 independently reviews the proposed assignment under the same taxonomy. Cases involving model disagreement, low confidence, coarse-category changes, or other ambiguity are manually adjudicated. This procedure accepts 6,653 labels without intervention and resolves the remaining 422 through manual review.

To independently assess annotation quality, we further perform a stratified audit of 355 designs across the six coarse categories. The audited annotations agree with the manually reviewed labels on 348/355 coarse categories (98.03%) and 343/355 exact fine-grained categories (96.62%). Conditioned on a correct coarse category, fine-grained agreement reaches 343/348 (98.56%). The annotation and review prompts are provided in Appendix F.

### C.3 Functional Variants

**Variant generation.** We construct functionally related RTL variants using two complementary mechanisms designed to create similar-looking examples with different semantic relationships. Equivalent candidates are generated through LLM-based RTL rewriting, which introduces coding-style and structural changes while preserving the intended circuit behavior. Non-equivalent candidates are generated through localized AST mutations, including changes to operators, constants, assignments, and control conditions, while preserving the surrounding program structure and interface. Together, these transformations create implementation diversity for equivalent pairs and localized semantic differences for non-equivalent pairs.

**Formal verification.** Generation provenance does not determine the equivalence label. Every original-variant pair is independently checked using a Yosys-based formal-equivalence flow (Wolf et al., 2013). A successful proof establishes an equivalent pair, whereas a conclusive counterexample establishes a non-equivalent pair; inconclusive cases, including parsing failures and verification timeouts, are excluded. Canonical duplicates are subsequently removed within and across the two classes. This yields 11,693 verified equivalent and 17,363 verified non-equivalent pairs before distance matching.

**Controlling transformation shortcuts.** Because equivalent and non-equivalent candidates are produced by different transformation mechanisms, their superficial edit patterns themselves reveal the class label. To reduce this shortcut, we characterize each original-variant pair using four model-independent measures of implementation change: canonical-token edit distance, AST Weisfeiler–Lehman (WL) distance, CDFG size difference, and CDFG WL distance. Distances are normalized using statistics derived from the training split.

Within each data split, equivalent and non-equivalent pairs are matched one-to-one in this distance space, retaining pairs whose matching distance is below a threshold $\tau$. We use $\tau = 0.3$ for the default

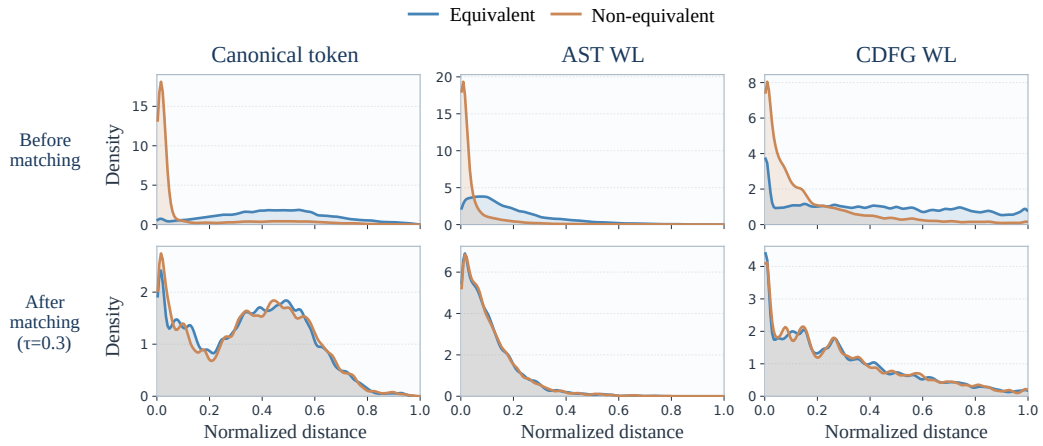


Figure 6: Structural-distance distributions for equivalent and non-equivalent pairs before (top) and after (bottom) one-to-one matching. Three distances are shown for readability; CDFG count distance is also included in matching and in the distance-only probe.

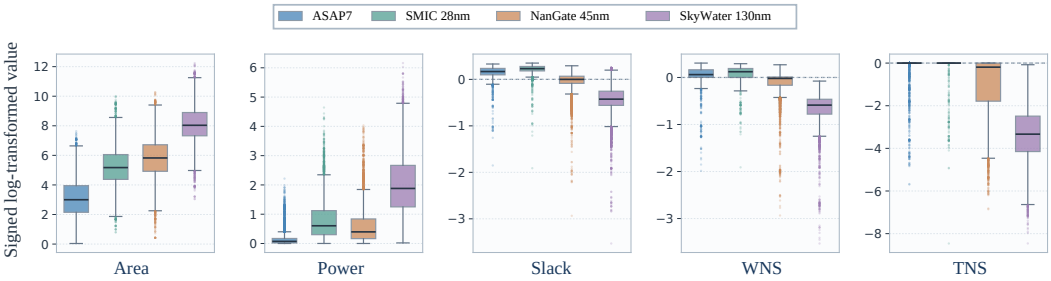


Figure 7: Distribution of the five PPA targets across four PDKs at a 0.5 ns clock constraint. Values are shown on a signed-log scale for visualization.

benchmark, yielding 3,567 pairs per class. Figure 6 shows the resulting alignment of the distance distributions. As an additional diagnostic, a logistic-regression classifier using only the four distance features decreases from 0.884 AUROC before matching to 0.505 after matching, with AP dropping to 0.501. Thus, simple transformation-magnitude cues are approximately non-predictive in the final evaluation subset.

### C.4 PPA Annotation

To construct implementation-aware supervision, we characterize each source design under multiple technology and timing settings using a unified synthesis and analysis flow. Synopsys Design Compiler is used for logic synthesis, with timing and power obtained using PrimeTime. The resulting targets include timing metrics such as Slack, WNS, and TNS, as well as measurements for Area and Power. These quantities are synthesis-derived implementation labels rather than post-layout measurements.

All 7,075 source designs are evaluated with four PDKs under a 0.5 ns clock constraint. To additionally capture constraint-induced variation within a fixed technology, NanGate45 is evaluated at 0.4 ns, 0.5 ns, and 0.6 ns, yielding six PDK–constraint settings in total. Figures 7 and 8 illustrate the resulting variation across technologies and timing constraints. For visualization, signed-log scaling is used to accommodate the wide dynamic range while preserving the sign of timing quantities. The transformations used for downstream regression are described separately in Appendix D.4.

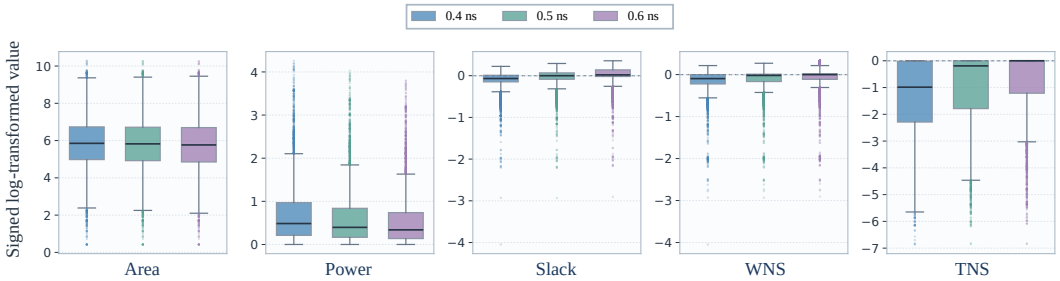


Figure 8: Distribution of the five PPA targets for NanGate45 under 0.4 ns, 0.5 ns, and 0.6 ns clock constraints. Values are shown on the same signed-log scale as Figure 7.

Table 8: Number of equivalent-implementation groups retained under different Area, Power, and WNS separation thresholds.

| Area gap | Power gap | WNS gap | Groups |
|---|---|---|---|
| 5% | 5% | 2% of clock | 1,054 |
| 5% | 5% | 5% of clock | 729 |
| 10% | 10% | 2% of clock | 693 |
| 10% | 10% | 5% of clock | 520 |

### C.5 PPA Ranking Groups

Functionally equivalent implementations can exhibit only marginal PPA differences, making their relative ordering sensitive to synthesis variation and less informative for representation evaluation. We therefore construct ranking groups by retaining equivalent-implementation sets that contain both sufficient implementation diversity and meaningful PPA separation. All implementations within a group are evaluated under the same NanGate45 0.5 ns setting.

For each objective, a group must contain at least three implementations and exhibit sufficient within-group separation: at least half of its pairs must be informative, and the median pairwise gap must meet the corresponding threshold. For Area and Power, pairwise separation is measured by the relative log gap,

$$|\log x_i - \log x_j| \geq \log(1 + r), \tag{16}$$

where $r$ denotes the relative threshold. For WNS, we use the clock-normalized gap

$$d_{ij}^{\mathrm{WNS}} = \frac{|\mathrm{WNS}_i - \mathrm{WNS}_j|}{T_{\mathrm{clk}}}. \tag{17}$$

A WNS pair is considered informative if this gap meets the threshold or if the two implementations differ in timing feasibility. The latter criterion affects only the informative-pair count; the median numerical WNS gap must still satisfy the threshold. The final benchmark retains the intersection of groups satisfying the Area, Power, and WNS criteria.

These criteria reduce ambiguous near-tie comparisons while retaining genuinely groupwise variation. The minimum group size avoids reducing the task to binary comparison, while the pairwise and median-gap requirements prevent isolated outliers from making an otherwise indistinguishable group appear informative. Log-ratio differences provide a scale-independent measure for Area and Power, and clock-normalized WNS differences provide a comparable timing criterion.

We adopt the 5%/5%/2%-of-clock setting as the default because it provides the largest common cohort among the evaluated thresholds while still enforcing non-trivial PPA separation. The remaining settings are used to examine robustness to the group-construction threshold.

### C.6 Natural-Language Query Construction

**Functional queries.** We construct functional retrieval queries through hierarchical summarization rather than directly prompting for a description of the full RTL design. Following the strategy

of TopoRTL (Zhao et al., 2026), Gemini 3.1 Pro Preview first generates module-level functional descriptions and then composes them into a circuit-level query that reflects the design's overall behavior and structural relationships. The resulting queries emphasize observable functionality, dataflow, sequencing, and protocol behavior while excluding module, port, signal, and instance identifiers to reduce direct lexical matching.

Functional queries are generated with a separate prompt (App. F.4) and independently from the Gemma-4-12B cone summaries used as model pretraining inputs (App. D.1), and are stored only on the query side of the retrieval task. This separation prevents retrieval performance from benefiting from shared generated text between the query and candidate representation.

**PPA-aware queries.** PPA-aware retrieval queries are constructed from the same functional descriptions while varying only the desired implementation objective. For each retained equivalent group, we generate three query variants expressing preferences for lower Area, lower Power, or better Timing. Because the functional content is shared across the three variants, differences in the desired ranking are determined by the implementation preference rather than by changes in functional intent. Preference clauses are drawn from a fixed set of manually defined paraphrases to introduce modest linguistic variation without changing the underlying objective.

# D Unified Benchmarking Framework Details

## D.1 Standardized Input Representations

UnderRTL standardizes the correspondence and granularity of circuit data while preserving the native input form required by each method. All circuit-level and subgraph-level views are aligned through shared design and register identifiers, allowing heterogeneous models to operate on different representations of the same underlying circuits.

**Text modality.** The text modality consists of cleaned RTL code and functional summaries. Cleaned RTL removes comments and tool-specific attributes while preserving executable statements and identifiers, and is constructed at both circuit- and register-centered cone levels. Functional summaries describe the logic implemented by individual cones and provide semantic context complementary to raw RTL code for methods that require natural-language inputs. Prior results from TopoRTL (Zhao et al., 2026) show that the quality of such summaries can materially affect downstream representation performance. Balancing summary quality with generation efficiency, we use Gemma-4-12B (Team, 2026) to generate the cone-level summaries used for model pretraining. These summaries are kept separate from downstream retrieval queries so that query information is never included in candidate circuit representations. The anonymized and misleading RTL views used in RQ1 are introduced only for robustness analysis and do not alter the default text input (construction in Appendix E.2).

**Graph modality.** The graph modality contains control–data flow graphs (CDFGs) and register-interaction graphs (RIGs). Following CircuitFusion (Fang et al., 2025), we construct CDFGs to capture operation-level data and control dependencies at circuit and cone levels, providing a common structural view for graph encoders. To characterize interactions beyond individual cones, we further construct RIGs following TopoRTL (Zhao et al., 2026), where registers are connected according to their data and control dependencies. Together, the two graph views capture complementary local logic structure and higher-level circuit organization, and are aligned with the corresponding RTL and summary representations through the same circuit hierarchy.

**Engineered features.** We additionally derive two non-neural references from the standardized circuit views. *Full features* combine source-level and structural statistics, while *Structure features* retain only graph-structural descriptors. They provide explicit baselines for assessing whether learned representations capture information beyond shallow circuit statistics.

## D.2 Evaluated Representation Methods

We select methods to cover complementary sources of variation in RTL representation learning, including input modality, architectural paradigm, and degree of circuit specialization. Within each modality, the benchmark spans conventional baselines, more recent general-purpose representation

architectures, and RTL-specific approaches where available. This design enables comparisons not only across modalities, but also between generic and domain-adapted representations under the same downstream protocol.

**Engineered-feature methods.** *Full features* and *Structure features* serve as non-learned reference points. They quantify how much downstream performance can be explained by explicit source and structural statistics, providing a baseline for measuring the additional value of learned representations.

**Graph methods.** GCN (Kipf & Welling, 2017) and GIN (Xu et al., 2019) provide standard message-passing baselines. Graphormer (Ying et al., 2021), SGFormer (Wu et al., 2023), and GraphGPS (Rampášek et al., 2022) represent architectures with stronger global interaction mechanisms, while RiemannGFM (Sun et al., 2025) introduces a general graph-foundation-model perspective. DynamicRTL (Ma et al., 2026) provides an RTL-specific counterpart. All methods operate on the same CDFGs, allowing architectural and domain-specific differences to be studied without changing the underlying circuit graph.

**Text methods.** We include general code-capable language models (Qwen2.5-Coder-7B (Hui et al., 2024), DeepSeek-Coder-6.7B (Guo et al., 2024), and Qwen3-8B (Team, 2025)), embedding-oriented models (Qwen3-Embedding-8B (Zhang et al., 2025) and NV-Embed-v1/v2 (Lee et al., 2025)), and RTL-specialized models (CodeV-QC/CodeV-All-QC (Zhao et al., 2025), CodeV-R1-RL-Qwen-7B (Zhu et al., 2026), and RTLCoder-Deepseek-v1.1 (Liu et al., 2024)). This selection contrasts generative and embedding-oriented representations as well as general code knowledge and RTL-specific adaptation.

**Multimodal methods.** GraphCLIP (Zhu et al., 2025), UniGraph (He et al., 2025a), and UniGraph2 (He et al., 2025b) represent general graph-text alignment and fusion paradigms, whereas CircuitFusion (Fang et al., 2025) and TopoRTL (Zhao et al., 2026) introduce circuit-specific multimodal designs. Their comparison examines whether combining textual and structural information provides benefits beyond single-modality representations, and whether circuit-aware fusion offers additional value over generic multimodal modeling.

### D.3 Pretraining and Representation Extraction

UnderRTL follows a preservation-first integration policy: released pretrained representations and native representation-learning objectives are retained whenever available, while architectures without circuit-specific pretraining are adapted with a shared function-aware objective. This balances fidelity to the original methods with the need to obtain comparable RTL representations. No downstream functional-class, PPA, ranking, or retrieval labels are used during representation pretraining.

**Pretrained and adapted encoders.** Released text encoders are used from their public checkpoints with their native representation-extraction rules. For graph models requiring RTL-specific adaptation, we use function-aware triplet supervision, optionally combined with masked-node reconstruction. Positive pairs consist of an original register-cone CDFG and an equivalence-preserving Yosys-optimized view of the same cone, while negatives are sampled from other training designs; the same positive construction is used across triplet-based pretraining objectives. These rule-based pretraining variants are distinct from the LLM-generated Functional Equivalence evaluation pairs, and validation/test circuits are excluded from encoder training. Models with native graph or multimodal pretraining objectives retain their original learning signals rather than being replaced by a common loss, preserving the representation mechanism each method was designed to study.

For models pretrained within UnderRTL, checkpoints are selected using their validation pretraining objective before any downstream evaluation. The selected encoder is then fixed across all tasks and downstream seeds.

**Representation extraction.** Each method retains its native readout or pooling rule. The resulting embeddings are exposed through a common interface as a circuit representation $\mathbf{z}_C$ and, where supported, aligned subgraph representations $\{\mathbf{z}_S : S \in \mathcal{S}(C)\}$. Downstream tasks access only these fixed representations.

### D.4 Downstream Evaluation Protocol

All downstream evaluations use frozen circuit representations so that performance primarily reflects representation quality rather than task-specific encoder adaptation. Unless otherwise specified, each task uses the same two-hidden-layer MLP architecture across methods, trained with Adam and selected on the validation split using the task's primary metric. Each head is trained with three random seeds $\{0, 42, 615\}$, and we report the mean and sample standard deviation on the test set.

Data splits are defined at the source-circuit level before constructing variants, subgraphs, pairs, groups, or queries, preventing related samples from crossing splits. For retrieval tasks, all methods share Qwen3-Embedding-8B as a fixed query encoder so that comparisons vary only the candidate-circuit representation.

**Functional Classification.** Separate MLP classifiers are trained for coarse and fine-grained functional labels using multiclass cross-entropy, with Accuracy and Macro-F1 used for validation selection, respectively.

**Functional Equivalence.** For a pair $(C_i, C_j)$, we construct the symmetric feature

$$\mathbf{x}_{ij} = \left[|\mathbf{z}_{C_i} - \mathbf{z}_{C_j}|; \mathbf{z}_{C_i} \odot \mathbf{z}_{C_j}\right], \tag{18}$$

and train a binary classifier selected by validation AP. The symmetric construction makes the prediction invariant to the ordering of the two circuits.

**Function-Aware Retrieval.** A fixed query embedding is concatenated with each candidate circuit representation and scored by an MLP trained with positive and sampled negative pairs. At evaluation time, every query is ranked against all candidates in the corresponding split, with MRR used for model selection.

**PPA Prediction.** Area and Power are predicted from circuit-level representations, Slack from subgraph representations, and WNS/TNS from mean- and max-pooled subgraph representations. Each target uses an independent MLP regressor trained with mean squared error. To stabilize regression, labels are transformed using

$$g(y) = \mathrm{sgn}(y)\log(1 + |y|), \qquad \widetilde{y} = \frac{g(y) - \mu_{\mathrm{train}}}{\sigma_{\mathrm{train}}}, \tag{19}$$

with statistics estimated only from the training split. Predictions are mapped back to the original label space before model selection and evaluation.

**PPA Ranking.** A scalar utility head is trained with RankNet loss over non-tied pairs within each equivalent-implementation group. Area and Power favor smaller values, whereas WNS favors larger values. Validation uses group-averaged Pairwise Accuracy.

**PPA-Aware Retrieval.** For each equivalent group, a fixed query embedding encodes its functional description together with an Area, Power, or Timing preference. An MLP scores each candidate from the concatenated query and circuit representations, and candidates are ranked within the group using the learned preference score. Validation uses NDCG@1. RQ4 reuses the same frozen representations and downstream protocol; only the availability of target-environment supervision changes.

## E Additional Experimental Results

### E.1 Complete Benchmark Results

Tables 9-11 report the complete per-subtask results underlying the compact main-paper table. Each entry is the mean $\pm$ sample standard deviation over downstream-training seeds $\{0, 42, 615\}$. We retain the native metric of each subtask rather than constructing an aggregate score across heterogeneous objectives. The complete PPA Prediction results additionally include Slack and TNS, while PPA Ranking and PPA-Aware Retrieval report all three objectives separately. For $R^2$, negative values indicate performance below the test-set mean predictor and are reported without clipping. For the PPA Prediction tasks, since heads are trained on signed-log, standardized targets but evaluated in the original space, errors on a few heavy-tailed test designs may be exponentially amplified and can dominate the residual sum; other tasks are unaffected by this transform.

Table 9: Complete results for the Functional Understanding tasks. Entries report mean ± sample standard deviation over three downstream-training seeds.

| | Functional Classification | | Functional Equivalence | Function-Aware Retrieval |
|---|---|---|---|---|
| **Model** | **Coarse Acc.** | **Fine Macro-F1** | **AP** | **MRR** |
| *Feature Engineering* | | | | |
| Full features | .768 ± .003 | .490 ± .001 | .767 ± .007 | .218 ± .004 |
| Structure features | .762 ± .003 | .482 ± .011 | .744 ± .021 | .217 ± .009 |
| *Graph* | | | | |
| GCN | .614 ± .006 | .262 ± .013 | .677 ± .013 | .096 ± .003 |
| GIN | .603 ± .003 | .274 ± .018 | .667 ± .009 | .089 ± .006 |
| Graphormer | .625 ± .011 | .319 ± .015 | .693 ± .003 | .077 ± .003 |
| SGFormer | .628 ± .006 | .289 ± .008 | .723 ± .008 | .085 ± .004 |
| GraphGPS | .650 ± .005 | .284 ± .012 | .699 ± .002 | .090 ± .005 |
| RiemannGFM | .578 ± .010 | .316 ± .010 | .670 ± .011 | .038 ± .002 |
| DynamicRTL | .677 ± .008 | .354 ± .009 | .721 ± .013 | .111 ± .003 |
| *Text* | | | | |
| Qwen2.5-Coder | .852 ± .003 | .591 ± .006 | .788 ± .007 | .347 ± .008 |
| DeepSeek-Coder | .855 ± .001 | .480 ± .003 | .769 ± .002 | .333 ± .029 |
| Qwen3-8B | .849 ± .001 | .538 ± .012 | .789 ± .003 | .316 ± .008 |
| Qwen3-E-8B | .873 ± .004 | .698 ± .003 | .711 ± .001 | .482 ± .036 |
| NV-Embed-v1 | .842 ± .001 | .610 ± .008 | .740 ± .004 | .351 ± .009 |
| NV-Embed-v2 | .852 ± .001 | .611 ± .024 | .729 ± .002 | .369 ± .012 |
| CodeV-QC-7B | .858 ± .003 | .599 ± .013 | .752 ± .006 | .337 ± .004 |
| CodeV-All-QC | .854 ± .004 | .596 ± .005 | .750 ± .007 | .338 ± .013 |
| CodeV-R1-7B | .855 ± .002 | .591 ± .005 | .782 ± .003 | .358 ± .005 |
| RTLCoder-DS | .848 ± .002 | .455 ± .008 | .788 ± .003 | .313 ± .022 |
| *Multimodal* | | | | |
| GraphCLIP | .683 ± .006 | .361 ± .009 | .675 ± .011 | .126 ± .002 |
| UniGraph | .682 ± .008 | .169 ± .003 | .647 ± .001 | .057 ± .043 |
| UniGraph2 | .610 ± .006 | .281 ± .015 | .710 ± .009 | .085 ± .009 |
| CircuitFusion | .674 ± .004 | .226 ± .009 | .661 ± .001 | .091 ± .002 |
| TopoRTL | .703 ± .001 | .417 ± .009 | .701 ± .008 | .105 ± .007 |

**The expanded results reinforce the task-specificity of RTL representations.** The additional subtasks do not reveal a representation family that consistently dominates the benchmark. In particular, engineered features remain highly competitive on implementation-aware targets, reaching $R^2 = 0.758$ for Slack and $0.691$ for TNS with Full features, while Structure features achieve $0.684$ on Power. At the same time, text representations retain a clear advantage on several functional tasks. These results strengthen the observation that different representation paradigms capture complementary aspects of RTL rather than exhibiting a single consistent ordering across tasks.

**Absolute PPA prediction and relative implementation discrimination can dissociate.** GraphCLIP provides the clearest example: its Area and Power prediction results are strongly negative ($R^2 = -499.762$ and $-2.395$), while the same frozen representation supports above-chance Area and Power ranking (Pairwise Accuracy $0.614$ and $0.595$). A similar separation appears for timing, where WNS prediction reaches only $R^2 = 0.043$ but WNS ranking reaches $0.590$. The extremely negative Area and TNS values, together with their large seed variance, indicate unstable global regression rather than proportionally greater information loss. These results reveal that absolute PPA estimation and within-group implementation discrimination are distinct capabilities: a representation can preserve useful relative ordering while failing to recover a stable global mapping to physical values.

**Local timing information does not necessarily translate to design-level timing awareness.** Several graph representations achieve moderate endpoint-level Slack prediction but degrade substantially on circuit-level timing targets. GCN and GIN, for example, obtain Slack $R^2$ of $0.483$ and $0.503$, respectively, while their WNS performance falls to $0.082$ and $0.056$, and their TNS predictions become strongly negative. Slack reflects timing behavior at individual register-centered subgraphs, whereas WNS and TNS depend on extrema and accumulated violations across the design. The observed gap therefore indicates that local timing information can remain recoverable even when the representation and its aggregation fail to capture the global timing structure required

Table 10: Complete PPA Prediction results across all five targets. Entries report $R^2$ mean $\pm$ sample standard deviation over three downstream-training seeds, evaluated in the original label space.

| Model | Area | Power | Slack | WNS | TNS |
|---|---|---|---|---|---|
| *Feature Engineering* | | | | | |
| Full features | $.284 \pm .415$ | $.634 \pm .077$ | $.758 \pm .004$ | $.686 \pm .050$ | $.691 \pm .043$ |
| Structure features | $-.232 \pm 1.441$ | $.684 \pm .074$ | $.751 \pm .006$ | $.637 \pm .098$ | $.603 \pm .068$ |
| *Graph* | | | | | |
| GCN | $.209 \pm .071$ | $.110 \pm .023$ | $.483 \pm .012$ | $.082 \pm .053$ | $-13432.736 \pm 22783.591$ |
| GIN | $-.424 \pm .587$ | $-.040 \pm .287$ | $.503 \pm .016$ | $.056 \pm .022$ | $-2696.273 \pm 2289.307$ |
| Graphormer | $.226 \pm .089$ | $.172 \pm .009$ | $.588 \pm .024$ | $.080 \pm .022$ | $.086 \pm .023$ |
| SGFormer | $.325 \pm .035$ | $.199 \pm .023$ | $.557 \pm .005$ | $.077 \pm .026$ | $-.086 \pm .155$ |
| GraphGPS | $.335 \pm .014$ | $.259 \pm .024$ | $.506 \pm .006$ | $.070 \pm .022$ | $.130 \pm .260$ |
| RiemannGFM | $.216 \pm .034$ | $.124 \pm .010$ | $.343 \pm .006$ | $.034 \pm .007$ | $.068 \pm .059$ |
| DynamicRTL | $.118 \pm .038$ | $.170 \pm .031$ | $.498 \pm .009$ | $.305 \pm .010$ | $-.711 \pm .535$ |
| *Text* | | | | | |
| Qwen2.5-Coder | $.332 \pm .060$ | $.355 \pm .028$ | $.666 \pm .008$ | $.376 \pm .022$ | $.092 \pm .034$ |
| DeepSeek-Coder | $.530 \pm .009$ | $.498 \pm .023$ | $.642 \pm .001$ | $.408 \pm .027$ | $.120 \pm .014$ |
| Qwen3-8B | $.482 \pm .056$ | $-.032 \pm .213$ | $.665 \pm .000$ | $.688 \pm .054$ | $.437 \pm .035$ |
| Qwen3-E-8B | $.552 \pm .018$ | $.551 \pm .016$ | $.615 \pm .014$ | $.491 \pm .013$ | $.543 \pm .051$ |
| NV-Embed-v1 | $.327 \pm .085$ | $.268 \pm .036$ | $.616 \pm .011$ | $.680 \pm .019$ | $.297 \pm .056$ |
| NV-Embed-v2 | $.439 \pm .010$ | $.353 \pm .035$ | $.638 \pm .015$ | $.544 \pm .052$ | $.645 \pm .035$ |
| CodeV-QC-7B | $.302 \pm .037$ | $.428 \pm .011$ | $.612 \pm .003$ | $.283 \pm .064$ | $.215 \pm .017$ |
| CodeV-All-QC | $.487 \pm .027$ | $.473 \pm .003$ | $.615 \pm .008$ | $.190 \pm .053$ | $.185 \pm .034$ |
| CodeV-R1-7B | $.520 \pm .011$ | $.360 \pm .046$ | $.654 \pm .008$ | $.272 \pm .031$ | $.101 \pm .032$ |
| RTLCoder-DS | $.552 \pm .002$ | $.518 \pm .011$ | $.639 \pm .007$ | $.391 \pm .020$ | $.150 \pm .023$ |
| *Multimodal* | | | | | |
| GraphCLIP | $-499.762 \pm 742.781$ | $-2.395 \pm 2.318$ | $.226 \pm .452$ | $.043 \pm .004$ | $-154241.293 \pm 156162.167$ |
| UniGraph | $.142 \pm .003$ | $.107 \pm .002$ | $.490 \pm .007$ | $.155 \pm .013$ | $.038 \pm .004$ |
| UniGraph2 | $.235 \pm .009$ | $.105 \pm .029$ | $.451 \pm .017$ | $.084 \pm .018$ | $-.029 \pm .071$ |
| CircuitFusion | $.279 \pm .012$ | $.231 \pm .006$ | $.549 \pm .009$ | $.231 \pm .045$ | $.251 \pm .013$ |
| TopoRTL | $.552 \pm .035$ | $.205 \pm .155$ | $.500 \pm .008$ | $.120 \pm .022$ | $.125 \pm .239$ |

by WNS and TNS. In other words, timing awareness does not emerge uniformly across granularity: endpoint-level predictability can coexist with poor design-level timing characterization.

### E.2 Additional RQ1: Encoding Analysis

**Perturbation construction.** Both lexical views are generated from the cleaned RTL of test circuits and preserve circuit logic. The *anonymized* view replaces module and signal names with consistent hashed names derived from the design ID, the original name, and a fixed seed, so that each identifier maps to the same new name throughout a circuit. The *misleading* view splits identifiers into semantic fragments and replaces each fragment with an antonym or opposite-function word (e.g., read$\rightarrow$write); names without a matching fragment are replaced with meaningless names. Every rewritten circuit is checked for name uniqueness, avoidance of Verilog keywords, parse errors, and reversibility of the mapping.

**Lexical sensitivity.** Table 12 reports the complete results underlying Figure 2(a). The downstream head is trained only on cleaned RTL and evaluated without adaptation on anonymized or misleading-name variants. For pair- and group-based tasks, the perturbation is applied consistently to all circuits in the same input instance, avoiding mixed-view cues. Performance decreases in 23 of the 24 perturbation settings. Anonymization produces the larger degradation in 11 of the 12 model-task comparisons, indicating that removing identifier semantics is generally more disruptive than replacing them with misleading cues. The pattern is not universal, however: for Qwen3-E-8B Area prediction, misleading identifiers reduce $R^2$ by $0.323$, compared with $0.166$ under anonymization. Thus, current text representations rely substantially on lexical information, but the form and magnitude of this dependence vary across both models and tasks.

Relative ranking is also less sensitive to these lexical perturbations than the absolute tasks considered here. Across the three text models, Area Ranking changes by at most $0.049$ Pairwise Accuracy, whereas coarse classification and Area prediction undergo substantially larger degradations. One possible reason is that ranking depends primarily on relative differences among implementations

Table 11: Complete results for PPA Ranking and PPA-Aware Retrieval. Entries report mean ± sample standard deviation over three downstream-training seeds.

| Model | PPA Ranking (Pairwise Acc.) | | | PPA-Aware Retrieval (NDCG@1) | | |
|---|---|---|---|---|---|---|
| | Area | Power | WNS | Area | Power | Timing |
| *Feature Engineering* | | | | | | |
| Full features | .632 ± .011 | .629 ± .015 | .601 ± .006 | .627 ± .020 | .617 ± .016 | .614 ± .004 |
| Structure features | .626 ± .005 | .600 ± .001 | .610 ± .010 | .637 ± .002 | .619 ± .005 | .641 ± .011 |
| *Graph* | | | | | | |
| GCN | .599 ± .008 | .585 ± .005 | .610 ± .017 | .610 ± .012 | .585 ± .012 | .626 ± .022 |
| GIN | .580 ± .006 | .577 ± .013 | .546 ± .024 | .585 ± .028 | .558 ± .029 | .576 ± .011 |
| Graphormer | .584 ± .011 | .568 ± .010 | .553 ± .017 | .572 ± .029 | .559 ± .024 | .617 ± .015 |
| SGFormer | .621 ± .006 | .603 ± .002 | .620 ± .014 | .610 ± .031 | .584 ± .018 | .591 ± .024 |
| GraphGPS | .583 ± .008 | .573 ± .011 | .607 ± .020 | .577 ± .011 | .564 ± .013 | .598 ± .015 |
| RiemannGFM | .519 ± .016 | .506 ± .005 | .507 ± .008 | .537 ± .016 | .523 ± .011 | .569 ± .038 |
| DynamicRTL | .603 ± .022 | .613 ± .007 | .627 ± .016 | .611 ± .023 | .609 ± .010 | .642 ± .012 |
| *Text* | | | | | | |
| Qwen2.5-Coder | .607 ± .002 | .574 ± .004 | .552 ± .011 | .620 ± .024 | .561 ± .004 | .585 ± .007 |
| DeepSeek-Coder | .630 ± .002 | .611 ± .008 | .576 ± .006 | .645 ± .014 | .587 ± .008 | .571 ± .010 |
| Qwen3-8B | .612 ± .004 | .609 ± .019 | .554 ± .010 | .626 ± .013 | .573 ± .019 | .592 ± .012 |
| Qwen3-E-8B | .614 ± .004 | .589 ± .004 | .601 ± .004 | .633 ± .005 | .582 ± .010 | .592 ± .014 |
| NV-Embed-v1 | .589 ± .004 | .581 ± .002 | .597 ± .006 | .610 ± .012 | .571 ± .015 | .594 ± .010 |
| NV-Embed-v2 | .596 ± .005 | .581 ± .004 | .597 ± .010 | .627 ± .009 | .591 ± .006 | .566 ± .015 |
| CodeV-QC-7B | .626 ± .007 | .591 ± .007 | .608 ± .005 | .621 ± .001 | .571 ± .005 | .580 ± .015 |
| CodeV-All-QC | .624 ± .006 | .586 ± .008 | .608 ± .004 | .637 ± .018 | .572 ± .007 | .590 ± .025 |
| CodeV-R1-7B | .630 ± .007 | .586 ± .008 | .598 ± .004 | .629 ± .020 | .552 ± .005 | .586 ± .011 |
| RTLCoder-DS | .619 ± .004 | .586 ± .007 | .585 ± .001 | .648 ± .013 | .585 ± .006 | .595 ± .025 |
| *Multimodal* | | | | | | |
| GraphCLIP | .614 ± .008 | .595 ± .011 | .590 ± .031 | .602 ± .024 | .620 ± .004 | .600 ± .005 |
| UniGraph | .557 ± .003 | .500 ± .004 | .569 ± .014 | .526 ± .005 | .505 ± .013 | .585 ± .011 |
| UniGraph2 | .594 ± .008 | .552 ± .005 | .555 ± .011 | .578 ± .015 | .546 ± .029 | .618 ± .018 |
| CircuitFusion | .597 ± .008 | .539 ± .001 | .568 ± .010 | .547 ± .017 | .537 ± .014 | .568 ± .003 |
| TopoRTL | .608 ± .020 | .583 ± .004 | .582 ± .018 | .572 ± .021 | .562 ± .024 | .591 ± .019 |

within the same functional group and can therefore be less sensitive to lexical shifts shared across those implementations. In contrast, classification and regression must transfer a decision or regression function learned entirely from the cleaned-input distribution. The Qwen3-E-8B Area result further shows that misleading identifiers can be more disruptive than simply removing identifier semantics, suggesting that lexical dependence is not limited to exploiting informative names: incorrect lexical cues can also interfere with the information used by downstream predictors.

**Structural complementarity.** Let $M_t^{(s)}(X)$ denote the downstream performance on task $t$ using representation $X$ under seed $s$. We measure structural complementarity as

$$\Delta_{\text{comp}}^{(t,s)} = M_t^{(s)}([\mathbf{E};\mathbf{S}]) - \max\left\{M_t^{(s)}(\mathbf{E}), M_t^{(s)}(\mathbf{S})\right\}, \tag{20}$$

where $\mathbf{E}$ is the learned embedding and $\mathbf{S}$ denotes explicit structural features. We average $\Delta_{\text{comp}}^{(t,s)}$ across seeds. A positive value indicates that combining the two representations improves over either representation alone, providing evidence of complementary task-relevant information.

The two tasks exhibit qualitatively different complementarity patterns. Functional Equivalence shows broadly consistent gains, with positive $\Delta_{\text{comp}}$ for seven of eight models, indicating that explicit structural descriptors provide information complementary to learned embeddings for cross-implementation matching. Area prediction instead exhibits larger but less consistent effects: substantial gains for CodeV-QC-7B, Graphormer, and CircuitFusion coexist with the pronounced DynamicRTL degradation. This contrast suggests that structural information is more consistently exploitable for relational functional comparison, whereas its contribution to absolute PPA prediction depends more strongly on the learned representation.

The large negative DynamicRTL result also highlights that unsuccessful concatenation does not necessarily imply structural redundancy. Because concatenation changes the feature space presented to the downstream regressor, the added structural features may interact poorly with the learned embed-

Table 12: Complete lexical-sensitivity results underlying Figure 2(a). $\Delta_{\text{anon}}$ and $\Delta_{\text{mis}}$ denote the paired anonymized-minus-cleaned and misleading-minus-cleaned differences, respectively. Values are mean $\pm$ sample standard deviation over three downstream-training seeds.

| Model | Task | Cleaned | Anonymized | Misleading | $\Delta_{\text{anon}}$ | $\Delta_{\text{mis}}$ |
|---|---|---|---|---|---|---|
| Qwen2.5-Coder | Coarse cls. | $.852 \pm .003$ | $.445 \pm .002$ | $.651 \pm .007$ | $-.407 \pm .003$ | $-.201 \pm .008$ |
| Qwen2.5-Coder | Equivalence | $.788 \pm .007$ | $.709 \pm .005$ | $.730 \pm .003$ | $-.079 \pm .012$ | $-.057 \pm .010$ |
| Qwen2.5-Coder | Area pred. | $.332 \pm .060$ | $-.161 \pm .042$ | $.029 \pm .040$ | $-.494 \pm .094$ | $-.303 \pm .097$ |
| Qwen2.5-Coder | Area ranking | $.607 \pm .002$ | $.576 \pm .010$ | $.607 \pm .005$ | $-.031 \pm .008$ | $.000 \pm .006$ |
| CodeV-QC-7B | Coarse cls. | $.858 \pm .003$ | $.432 \pm .000$ | $.564 \pm .004$ | $-.426 \pm .002$ | $-.294 \pm .005$ |
| CodeV-QC-7B | Equivalence | $.752 \pm .006$ | $.659 \pm .012$ | $.682 \pm .006$ | $-.093 \pm .017$ | $-.070 \pm .011$ |
| CodeV-QC-7B | Area pred. | $.302 \pm .037$ | $-.211 \pm .011$ | $-.107 \pm .015$ | $-.513 \pm .039$ | $-.409 \pm .025$ |
| CodeV-QC-7B | Area ranking | $.626 \pm .007$ | $.591 \pm .007$ | $.601 \pm .007$ | $-.035 \pm .013$ | $-.024 \pm .007$ |
| Qwen3-E-8B | Coarse cls. | $.873 \pm .004$ | $.645 \pm .006$ | $.764 \pm .010$ | $-.229 \pm .010$ | $-.109 \pm .013$ |
| Qwen3-E-8B | Equivalence | $.711 \pm .001$ | $.621 \pm .001$ | $.636 \pm .000$ | $-.090 \pm .002$ | $-.076 \pm .001$ |
| Qwen3-E-8B | Area pred. | $.552 \pm .018$ | $.386 \pm .012$ | $.229 \pm .077$ | $-.166 \pm .023$ | $-.323 \pm .095$ |
| Qwen3-E-8B | Area ranking | $.614 \pm .004$ | $.565 \pm .008$ | $.590 \pm .011$ | $-.049 \pm .012$ | $-.024 \pm .015$ |

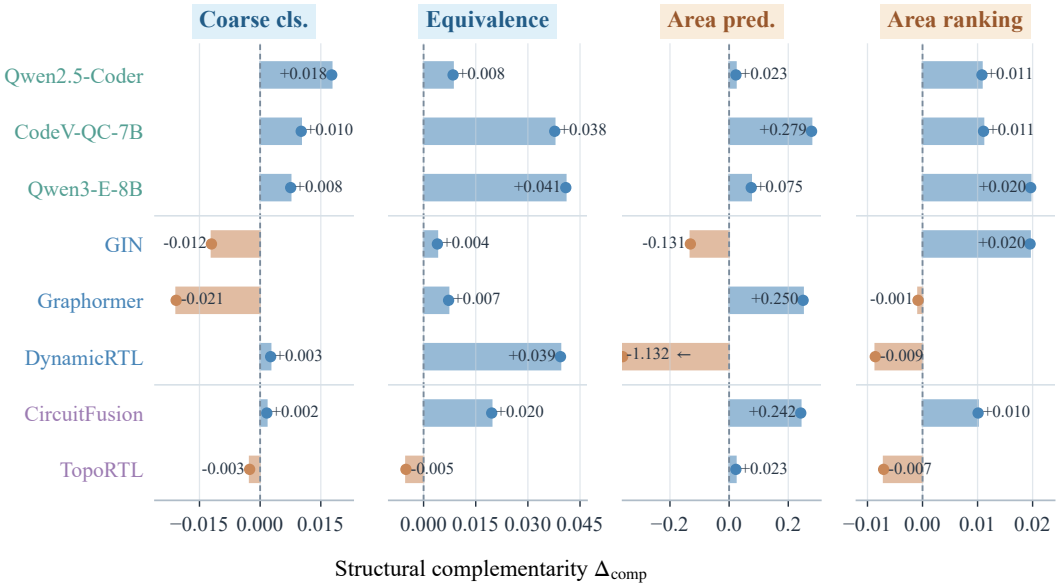


Figure 9: **Additional RQ1 results: structural complementarity.** $\Delta_{\text{comp}}$ measures the downstream-performance gain of $[\mathbf{E}; \mathbf{S}]$ over the stronger of $\mathbf{E}$ and $\mathbf{S}$, computed per seed and then averaged. Positive values indicate gains beyond both individual representations, while negative values indicate degradation after concatenation. Axes are scaled independently across tasks for readability, and the clipped DynamicRTL result is annotated with its exact value.

ding even when they contain useful task-relevant information. Hence, structural complementarity is evident for many model-task settings, but the effectiveness of simple feature-level fusion remains model-dependent.

### E.3 Additional RQ2: Invariance Analysis

**Matching-threshold robustness.** Figure 10 evaluates Functional Equivalence under four distance-matching thresholds, reporting family means together with the within-family model ranges. Relaxing the threshold from $\tau = 0.1$ to $0.4$ does not induce a common performance direction: the feature and text families decrease by $0.033$ and $0.015$ AP on average, whereas the graph and multimodal families increase by $0.014$ and $0.011$, respectively. Despite these shifts, model ordering remains relatively stable, with a Spearman correlation of $\rho_s = 0.877$ between the two extreme thresholds. The opposite family-level responses indicate that admitting more weakly matched im-

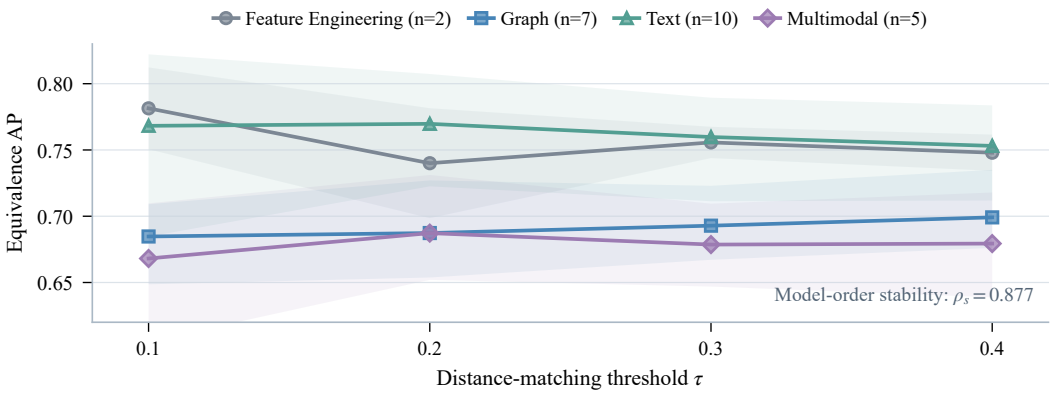


Figure 10: **Additional RQ2 analysis: matching-threshold robustness.** Functional Equivalence AP under four distance-matching thresholds. Curves show representation-family means, and shaded regions span the minimum and maximum model performance within each family. The reported Spearman correlation compares model rankings at $\tau = 0.1$ and $\tau = 0.4$.

Table 13: Distance-bin statistics for the Functional Equivalence test set at the default matching threshold $\tau = 0.3$. Bins are constructed separately for each distance using approximately equal numbers of pairs. $N_+$ and $N_-$ denote equivalent and non-equivalent pairs, respectively.

| Distance | Bin | Observed range | Median | $N_+$ | $N_-$ |
|---|---|---|---|---|---|
| Canonical token | 1 | 0.001–0.115 | 0.041 | 112 | 126 |
| Canonical token | 2 | 0.116–0.313 | 0.198 | 126 | 111 |
| Canonical token | 3 | 0.315–0.428 | 0.370 | 121 | 116 |
| Canonical token | 4 | 0.429–0.555 | 0.492 | 118 | 119 |
| Canonical token | 5 | 0.556–0.926 | 0.624 | 116 | 121 |
| AST-WL | 1 | 0.000–0.023 | 0.013 | 126 | 112 |
| AST-WL | 2 | 0.023–0.057 | 0.039 | 106 | 131 |
| AST-WL | 3 | 0.057–0.102 | 0.078 | 121 | 116 |
| AST-WL | 4 | 0.102–0.167 | 0.130 | 122 | 115 |
| AST-WL | 5 | 0.167–0.728 | 0.227 | 118 | 119 |
| CDFG-WL | 1 | -0.000–0.044 | 0.000 | 120 | 118 |
| CDFG-WL | 2 | 0.045–0.147 | 0.102 | 117 | 120 |
| CDFG-WL | 3 | 0.147–0.286 | 0.215 | 121 | 117 |
| CDFG-WL | 4 | 0.286–0.484 | 0.367 | 117 | 119 |
| CDFG-WL | 5 | 0.486–1.000 | 0.658 | 118 | 119 |

plementation changes affects representation paradigms differently, even though much of the cross-model ordering is preserved.

**Distance-bin construction and functional control.** Table 13 reports the observed ranges and class counts for the five equal-frequency bins used in Figure 3. All error rates use the default decision rule $\hat{p}^{\text{eq}} > 0.5$. Positive and negative examples remain broadly balanced across bins, with no bin exhibiting severe class imbalance. The non-monotonic error curves are therefore unlikely to arise simply from differences in class proportion.

We further control for functional composition by repeating the CDFG-distance analysis using only arithmetic-logic circuits in Table 14. Eight of the nine representative models retain their highest FNR in the second bin. Thus, the intermediate-distance failure peak persists even within a single coarse functional category and cannot be explained solely by category composition. More generally, token-, AST-, and CDFG-based distances quantify the magnitude of change in particular representation spaces, but they need not provide a monotonic ordering of semantic difficulty. The results therefore indicate that equivalence failures depend on properties of the implementation change beyond distance magnitude alone.

**Functional-category asymmetry.** Figure 11 decomposes the equivalence errors at $\tau = 0.3$ across the six coarse functional categories. For all nine representative models, communication/interface circuits exhibit higher FNR than arithmetic/logic circuits, but lower FPR. Thus, a functional category

Table 14: Functional Equivalence FNR across CDFG-distance bins after restricting the test set to arithmetic/logic circuits. Lower is better; values are mean ± sample standard deviation over three downstream-training seeds.

| Model | B1 | B2 | B3 | B4 | B5 |
|---|---|---|---|---|---|
| Structure features | .159 ± .014 | .359 ± .068 | .179 ± .044 | .167 ± .069 | .174 ± .023 |
| Qwen2.5-Coder | .230 ± .014 | .402 ± .015 | .231 ± .019 | .227 ± .023 | .189 ± .017 |
| CodeV-QC-7B | .310 ± .041 | .385 ± .089 | .333 ± .022 | .341 ± .045 | .214 ± .023 |
| Qwen3-E-8B | .381 ± .024 | .470 ± .039 | .333 ± .078 | .136 ± .039 | .219 ± .017 |
| GIN | .397 ± .050 | .419 ± .015 | .378 ± .040 | .485 ± .080 | .383 ± .048 |
| Graphormer | .270 ± .077 | .350 ± .141 | .128 ± .011 | .235 ± .035 | .313 ± .026 |
| DynamicRTL | .183 ± .014 | .350 ± .097 | .199 ± .044 | .189 ± .013 | .209 ± .030 |
| CircuitFusion | .254 ± .096 | .538 ± .092 | .436 ± .113 | .235 ± .107 | .234 ± .046 |
| TopoRTL | .214 ± .024 | .342 ± .090 | .103 ± .062 | .106 ± .129 | .199 ± .090 |

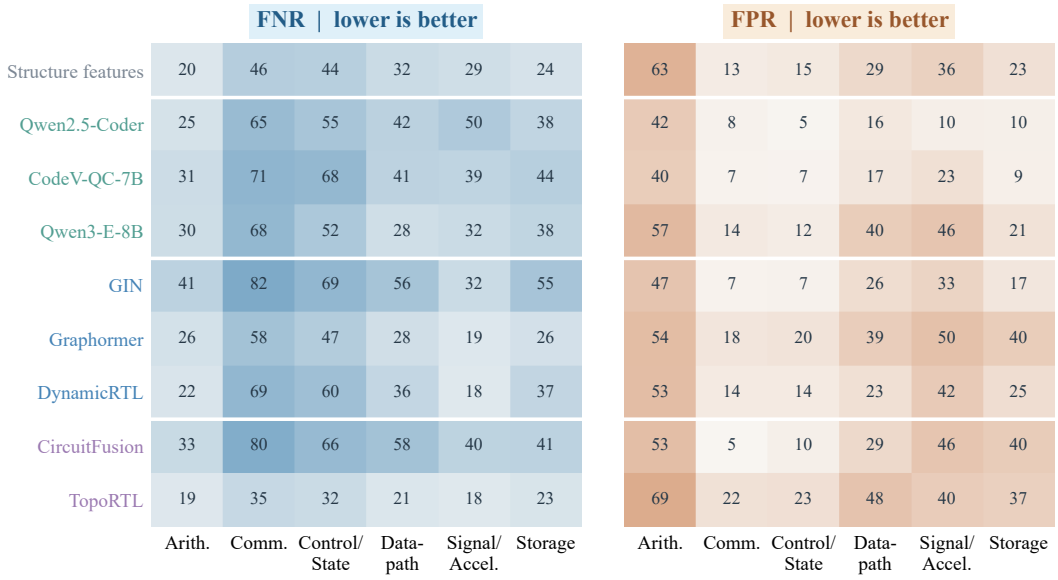


Figure 11: **Additional RQ2 analysis: functional-category error profiles.** Three-seed mean false-negative rate (left) and false-positive rate (right) at $\tau = 0.3$, decomposed by coarse functional category. Values are percentages; lower is better in both panels.

can make equivalent pairs harder to accept while simultaneously making non-equivalent pairs easier to reject. Equivalence difficulty therefore cannot be characterized by a single category-level scalar: different categories induce distinct false-negative and false-positive error profiles that are hidden by aggregate AP.

TopoRTL illustrates this asymmetry particularly clearly. Averaged across the six functional categories, it has the lowest FNR among the nine representative models (0.244) but the highest FPR (0.398). The contrast is especially pronounced for arithmetic/logic circuits, where its FNR is 0.187 while its FPR reaches 0.688. At the evaluated decision threshold, TopoRTL therefore occupies an equivalence-favoring operating regime: equivalent pairs are rarely rejected, but non-equivalent pairs are accepted much more frequently. This asymmetric error profile helps explain why strong recall on equivalent pairs does not translate into the strongest overall Equivalence AP, and shows that models with similar aggregate performance can fail in systematically different directions.

### E.4 Additional RQ3: Invariance–Sensitivity Interplay

**Seed-wise and family-wise correlations.** The main analysis computes correlations using each model's three-seed mean. To examine whether these relationships depend on downstream-training randomness, Table 15 recomputes the Spearman correlation across models independently for each

Table 15: Seed-wise Spearman correlations between Functional Equivalence AP and PPA Ranking Pairwise Accuracy. Correlations are computed across models independently for each downstream-training seed; values report mean ± sample standard deviation across three seeds. $N$ denotes the number of models in each scope. Feature Engineering is omitted from family-level analysis because its two-model sample is insufficient for an informative rank correlation.

| Scope | $N$ | Area | Power | WNS |
|---|---|---|---|---|
| All | 24 | $.661 \pm .080$ | $.519 \pm .082$ | $.164 \pm .064$ |
| Graph | 7 | $.595 \pm .206$ | $.667 \pm .149$ | $.702 \pm .103$ |
| Text | 10 | $.220 \pm .136$ | $.091 \pm .149$ | $-.693 \pm .055$ |
| Multimodal | 5 | $.467 \pm .404$ | $.667 \pm .058$ | $.033 \pm .702$ |

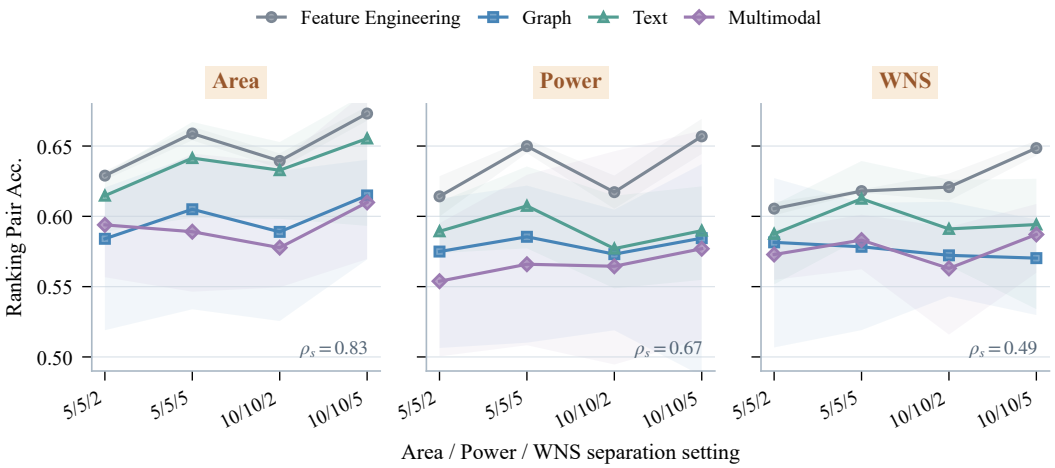


Figure 12: **Additional RQ3 analysis: ranking-group construction sensitivity.** PPA Ranking Pairwise Accuracy under four Area/Power/WNS separation settings. Curves show representation-family means, while shaded regions span the minimum and maximum model performance within each family. Reported Spearman correlations compare model rankings between the first and last separation settings.

seed. The overall relationship remains positive for Area and Power ($0.661 \pm 0.080$ and $0.519 \pm 0.082$) but weak for WNS ($0.164 \pm 0.064$), consistent with the model-mean analysis in the main paper. More notably, the contrasting WNS relationships across representation families persist across seeds: graph models show a consistently positive correlation ($0.702 \pm 0.103$), whereas text models show a consistently negative one ($-0.693 \pm 0.055$). The multimodal WNS correlation is highly variable ($0.033 \pm 0.702$), so no stable family-level direction can be inferred for that group.

The family-wise results further show that WNS is the primary source of heterogeneity in the invariance-sensitivity relationship. For Area and Power, both graph and text families retain non-negative cross-model correlations, although the relationships are much weaker among text models. For WNS, however, the direction reverses sharply: stronger Functional Equivalence performance accompanies stronger WNS ranking among graph models but weaker WNS ranking among text models. The weak overall WNS correlation therefore arises alongside two opposing family-level capability regimes rather than from uniformly weak relationships within all representation families.

**Ranking-group construction sensitivity.** Figure 12 evaluates PPA Ranking under four Area/Power/WNS separation settings. Model ordering between the first and last settings is most stable for Area ($\rho_s = 0.826$), followed by Power ($0.675$), and substantially less stable for WNS ($0.490$). Thus, Area ranking preserves a relatively consistent cross-model ordering across group definitions, whereas WNS ranking is considerably more sensitive to which equivalent implementations are retained.

The family-level curves are also non-monotonic, particularly for Power and WNS. Because the four configurations jointly vary the Area, Power, and WNS separation criteria, each setting retains a different subset of equivalent groups rather than simply increasing task difficulty. The substantially

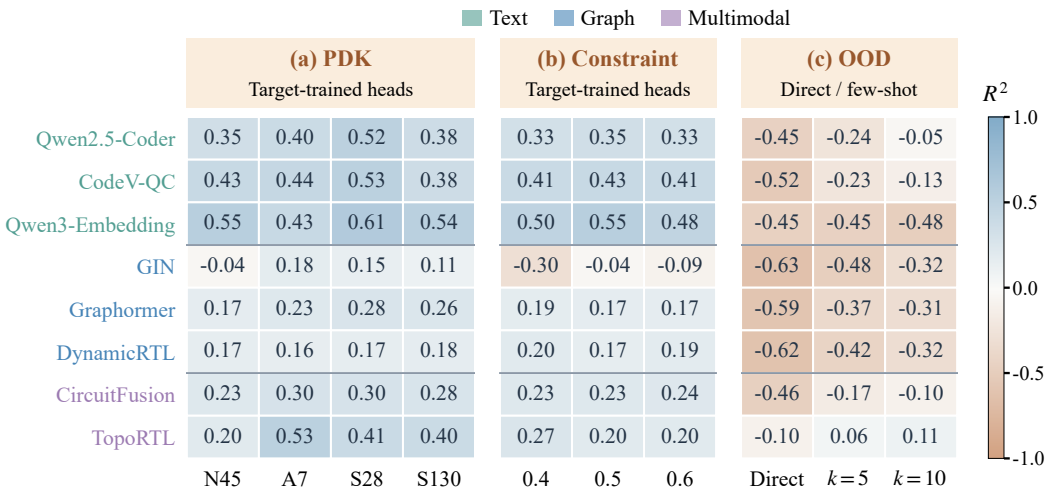


Figure 13: **Additional RQ4 results: Power.** Three-seed mean $R^2$ under target-trained PDK/constraint settings and direct or few-shot OOD transfer.

lower order stability for WNS indicates that relative timing performance depends more strongly on the composition of timing-distinguishable implementation groups. This cohort sensitivity may partly contribute to the weaker and more family-dependent relationship between Functional Equivalence and WNS Ranking observed in the main analysis.

### E.5 Additional RQ4: Generalization

Figures 13 and 14 extend the Area analysis in Figure 5 to Power and WNS. As in the main experiment, the PDK and constraint settings use separately trained target-environment heads, whereas OOD results evaluate direct transfer or $k$-shot head adaptation with the representation encoder frozen.

**Power.** Power exhibits a generalization pattern similar to Area. Seven of the eight representative models maintain positive $R^2$ across every target-trained PDK and constraint setting, with GIN as the only exception. Qwen3-Embedding-8B is particularly stable, ranging from $0.43$ to $0.61$ across PDKs and from $0.48$ to $0.55$ across clock constraints. In contrast, direct OOD transfer is negative for all eight models. With ten target examples, seven models improve, but only TopoRTL crosses zero, increasing from $-0.10$ to $0.11$.

The contrast between broadly positive target-trained results and uniformly negative direct OOD transfer reinforces the distinction between environmental adaptability and transfer across unseen design distributions. In particular, representations that support accurate Power prediction after limited environment-specific supervision can still fail to preserve an absolute predictor when the circuit population itself changes. Few-shot supervision narrows this gap for most models, but generally does not recover in-distribution prediction quality.

**WNS.** WNS exhibits a substantially different OOD behavior. All eight models achieve positive $R^2$ under every target-trained PDK and constraint setting, yet direct OOD transfer varies widely across representations. Graphormer, CircuitFusion, and TopoRTL remain positive under direct transfer ($0.13$, $0.51$, and $0.40$, respectively), whereas the other five models fall below zero. After ten-shot adaptation, six of eight models improve in mean: Graphormer rises to $0.52$ and GIN from $-0.87$ to $0.31$, while CircuitFusion and TopoRTL remain close to their already positive direct-transfer performance.

Few-shot recovery is also strongly target-dependent. As shown in Table 16, Qwen2.5-Coder and CodeV-QC-7B both improve substantially on Area and Power with ten shots, yet their WNS changes are negative ($-0.983$ and $-5.240$, respectively). Conversely, all graph and multimodal representatives improve or remain approximately unchanged on WNS in mean, while two of the three text models degrade. The large WNS variance for several text models further indicates that the direction of few-shot adaptation can be unstable across downstream-training seeds. Thus, transferability is not

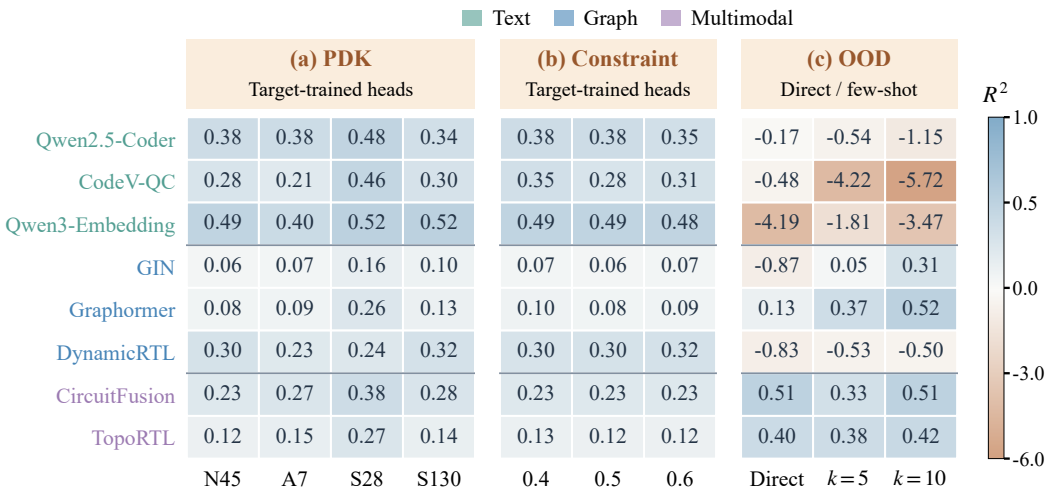


Figure 14: **Additional RQ4 results: WNS.** Three-seed mean $R^2$ under target-trained PDK/constraint settings and direct or few-shot OOD transfer. The color scale is extended to preserve strongly negative OOD values.

Table 16: Change in OOD $R^2$ from direct transfer to $k = 10$ adaptation. Positive values indicate improved OOD performance. Values are mean ± sample standard deviation over three downstream-training seeds.

| Model | Area | Power | WNS |
|---|---|---|---|
| Qwen2.5-Coder | $.550 \pm .187$ | $.405 \pm .028$ | $-.983 \pm 1.270$ |
| CodeV-QC-7B | $.597 \pm .172$ | $.392 \pm .017$ | $-5.240 \pm 2.832$ |
| Qwen3-E-8B | $.167 \pm .034$ | $-.023 \pm .112$ | $.724 \pm 3.657$ |
| GIN | $.347 \pm .145$ | $.307 \pm .223$ | $1.185 \pm .378$ |
| Graphormer | $.479 \pm .098$ | $.274 \pm .060$ | $.389 \pm .337$ |
| DynamicRTL | $.420 \pm .045$ | $.298 \pm .084$ | $.321 \pm .398$ |
| CircuitFusion | $-.048 \pm .576$ | $.365 \pm .053$ | $.002 \pm .071$ |
| TopoRTL | $.128 \pm .114$ | $.205 \pm .062$ | $.020 \pm .034$ |

a single property of an RTL representation: the same frozen embedding can exhibit strong few-shot recovery for physical cost targets while remaining unstable for timing.

### E.6 Further Analysis

We further examine whether PPA predictability varies with circuit structure, focusing on the distinction between combinational and sequential designs and the scale of sequential state. We stratify the existing NanGate45 0.5 ns test predictions without retraining and consider five representative methods spanning engineered features, text, graph, and circuit-specific multimodal representations: Full features, Qwen3-E-8B, DynamicRTL, CircuitFusion, and TopoRTL. Among the 1,412 test circuits, 507 are combinational ($N_{\text{reg}} = 0$) and 905 are sequential ($N_{\text{reg}} > 0$).

Because $R^2$ is normalized by the target variance within each evaluated subset, we additionally report a fixed-scale error for stratified analyses:

$$\text{NRMSE}_b = \frac{\sqrt{|b|^{-1} \sum_{i \in b} (y_i - \hat{y}_i)^2}}{\sigma_{\text{train}}(y)}, \tag{21}$$

where $\sigma_{\text{train}}(y)$ is shared across all test subsets for the same target.

**Combinational versus sequential circuits.** Figure 15 reveals a strong interaction between circuit type, prediction target, and representation. For Area, sequential-circuit performance improves substantially for TopoRTL ($\Delta R^2 = +0.469$) and CircuitFusion ($+0.147$), but decreases for Qwen3-E-8B ($-0.371$), while DynamicRTL changes little. Power shows a more consistent shift, with four

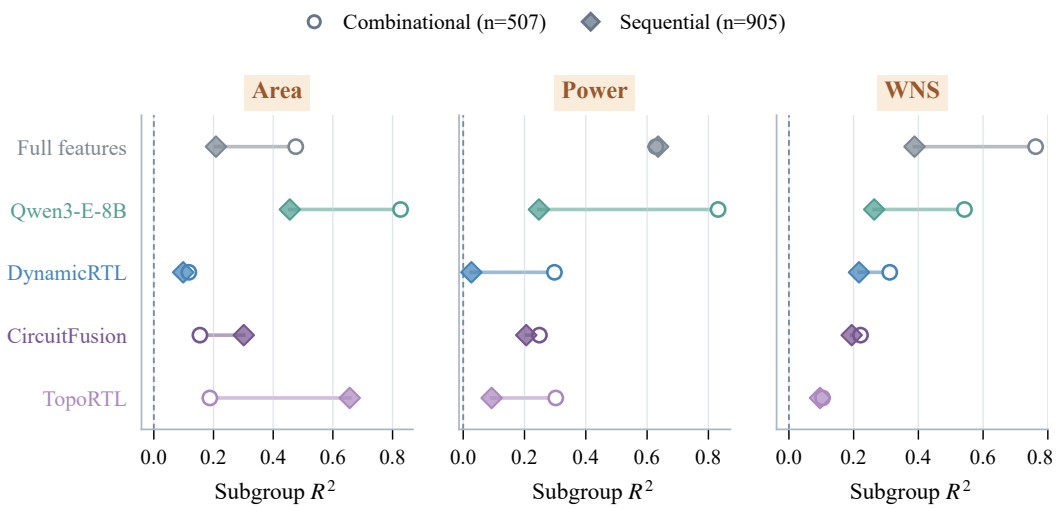


Figure 15: **PPA prediction by circuit type.** Subgroup $R^2$ on 507 combinational and 905 sequential test circuits. Open circles and filled diamonds show three-seed means; connecting segments expose the direction and magnitude of each model's circuit-type shift.

of the five methods obtaining lower $R^2$ on sequential circuits, and all five exhibit lower sequential $R^2$ for WNS.

The WNS error, however, shows a different pattern when measured on a fixed scale. TopoRTL's NRMSE decreases from $0.844$ on combinational circuits to $0.339$ on sequential circuits, while CircuitFusion decreases from $0.787$ to $0.320$. Thus, their lower sequential WNS $R^2$ does not correspond to larger absolute prediction error and instead coincides with a change in the within-subset target variation. Circuit structure therefore affects PPA predictability in a target- and representation-dependent manner, rather than making sequential designs uniformly easier or harder.

**Register-count scaling.** We further partition the 905 sequential circuits by empirical register-count quartiles into 1–2, 3–4, 5–8, and $\geq 9$ registers, containing 297, 204, 191, and 213 circuits, respectively. Figure 16 shows that prediction error evolves differently with state scale across PPA targets.

From the smallest to the largest register-count bin, Area NRMSE increases for all four learned representations by $0.194$–$0.525$, while Full features shows the opposite trend. Power NRMSE likewise increases for all five methods, by $0.060$–$0.285$. WNS follows a qualitatively different pattern: NRMSE reaches its maximum in the 3–4-register bin and subsequently decreases for every evaluated method. Accordingly, increasing sequential scale is associated with growing error for Area and Power prediction, but not for timing.

The bin-wise $R^2$ results further reveal substantial model-specific variation. TopoRTL, for example, recovers from $R^2 = -0.063$ in the 5–8-register bin to $0.384$ for circuits with at least nine registers, close to CircuitFusion's $0.396$ in the same subset. Together, these results show that circuit scale affects physical-cost and timing information differently: larger state spaces challenge Area/Power prediction more consistently, whereas WNS difficulty is non-monotonic with register count.

**Representation strengths depend on circuit regime.** The stratified results further show that representation advantages can depend strongly on the structural regime of the evaluated circuits. Different methods encode different inductive biases, so a model that is only moderately competitive in aggregate may become substantially stronger on the designs that better match its representation structure.

TopoRTL provides a representative example. Its Area $R^2$ increases from $0.188$ on combinational circuits to $0.656$ on sequential circuits, and reaches $0.782/0.601/0.384$ for Area, Power, and WNS on circuits with at least nine registers. In this subset, its Area and Power results are the strongest among the selected learned representations, while its WNS performance approaches CircuitFusion ($0.396$). This behavior is consistent with TopoRTL's use of a register-interaction graph and topological differences between subgraphs within the same circuit, which expose richer structural information when

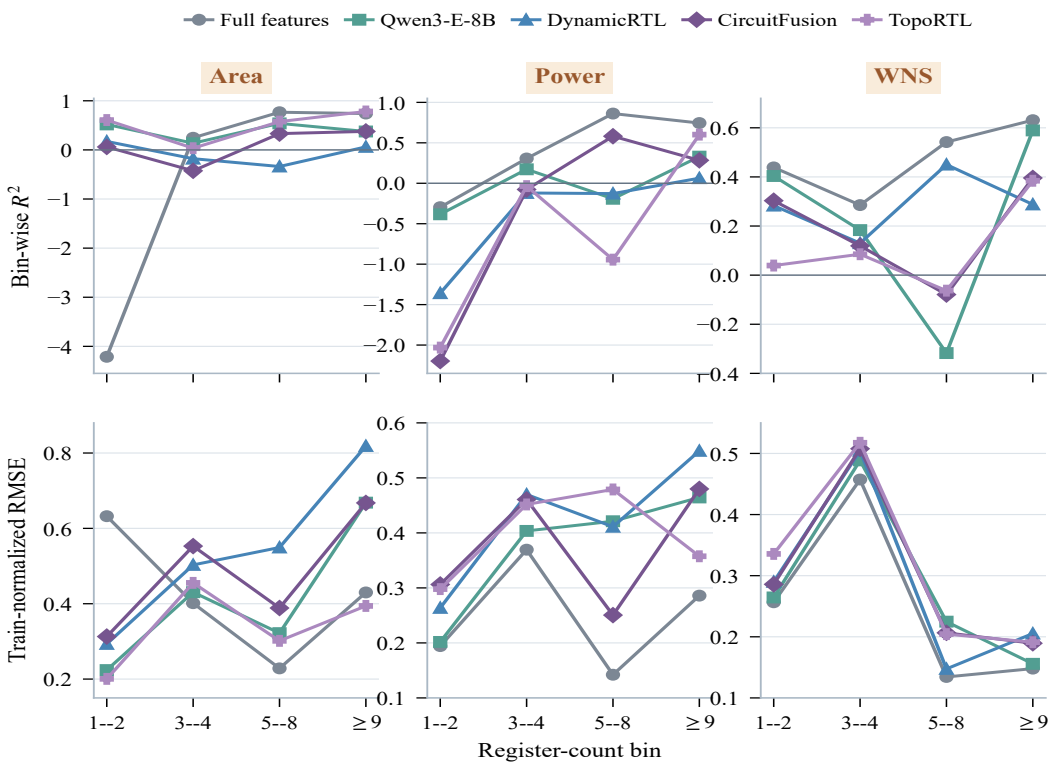


Figure 16: **PPA prediction across sequential-circuit scale.** The top row reports bin-wise $R^2$; the bottom row reports RMSE normalized by the corresponding training-target standard deviation. Curves show three-seed means; error bars are omitted to preserve the cross-bin trends.

inter-register organization is more prominent. The advantage is not monotonic, however: its Power prediction drops to $R^2 = -0.944 \pm 2.039$ in the 5–8-register bin before recovering in the largest subset.

These results suggest that aggregate performance can conceal scenario-specific representation strengths. Future RTL representations should therefore aim for more robust performance across heterogeneous circuit regimes, while scenario-aware model selection may also be beneficial when different design structures favor different representation biases.

# F Annotation and Prompt Details

## F.1 Functional Annotation Prompts

**Prompt for RTL Circuit Functional Classification**

You are a senior RTL design expert with 20+ years of experience in digital circuit design. Your task is to classify an RTL circuit written in Verilog/SystemVerilog into **one coarse-grained category** and **one fine-grained subcategory** from the taxonomy below. Classify by the module's **primary functional purpose**.

**Taxonomy**

**1. `arithmetic_logic` - Arithmetic & Logic**
- `add_sub_compare`: Adders, subtractors, comparators, carry-lookahead/save units.
- `mul_div_mac`: Single multipliers, dividers, square-root units, single MAC units, FPU units.
- `shift_bit_logic`: Barrel shifters, popcount, parity generators, bitwise operations.
- `general_alu_logic`: Multi-function ALUs and combined arithmetic-logic units.
- `others`: Arithmetic/logic modules not fitting any subcategory above.

**2. `storage_buffer` - Storage & Buffer**
- `reg_latch_flipflop`: D/JK/T flip-flops, latches, shift registers, register files.
- `fifo_queue`: Synchronous FIFO, asynchronous FIFO, circular buffer, LIFO stack.
- `ram_rom_memory`: Single/dual-port SRAM, ROM, CAM, content-addressable memory.
- `cache_scratch`: Direct-mapped/set-associative cache, scratchpad, tag arrays.
- `others`: Storage modules not fitting any subcategory above.

**3. `control_state` - Control & State**
- `fsm_controller`: Mealy/Moore FSMs, sequence detectors, state-encoded controllers.
- `counter_timer`: Synchronous/asynchronous counters, BCD/Johnson counters, PWM generators, clock dividers.
- `arbiter_scheduler`: Fixed-priority arbiters, round-robin arbiters, token-ring schedulers.
- `pipeline_hazard`: Hazard detection, stall insertion, forwarding/bypass units, pipeline control.
- `others`: Control/state modules not fitting any subcategory above.

**4. `datapath_transform` - Datapath & Transform**
- `mux_demux_select`: Multiplexers, demultiplexers, crossbar switches, data routing.
- `encoder_decoder`: Binary encoders/decoders, priority encoders, Gray code converters, BCD converters.
- `serializer_deserializer`: Serial-to-parallel, parallel-to-serial, SERDES, bitstream alignment.
- `width_format_convert`: Bit-width extension/truncation, byte-order conversion, alignment padding.
- `others`: Datapath/transform modules not fitting any subcategory above.

**5. `communication_interface` - Communication & Interface**
- `serial_protocol`: UART, SPI, I2C, 1-Wire, CAN controllers.
- `bus_interface`: AXI4/AXI-Lite, APB, AHB, Wishbone, on-chip bus interfaces and bus arbitration.
- `clock_domain_sync`: Dual-FF synchronizers, handshake synchronizers, CDC FIFO, metastability mitigation.
- `protocol_bridge`: Protocol conversion bridges, Ethernet MAC, USB controller, DMA/bus-to-memory data movers.
- `others`: Interface modules not fitting any subcategory above.

**6. `signal_security_accel` - Signal Processing / Security / Acceleration**
- `dsp_filter_transform`: FFT, FIR/IIR filters, CORDIC, interpolators, decimators.
- `compute_accel`: MAC arrays, systolic arrays, activation functions, image filters, matrix multiply.
- `crypto_crc_hash`: AES, DES, ChaCha20, SHA-256/512, HMAC, RSA, ECC crypto, CRC/checksum.
- `rng_ecc_encoding`: LFSR used as RNG, TRNG, PRNG, Hamming ECC, error-correction codes.
- `others`: Signal/security/acceleration modules not fitting any subcategory above.

**Classification Rules**

1. **Classify by primary purpose.** Focus on what the module fundamentally does.
2. **Determine the coarse category first.** Select the best coarse category, then choose the best fine subcategory.
3. **Use names and ports as evidence,** but not as the only evidence.
4. **Wrapper or structural modules.** Classify by the dominant child module's function.
5. **Conflict priority.** Protocol/bus/interface modules are `communication_interface` even if they contain FSMs/FIFOs. FIFO/RAM are `storage_buffer` unless they implement a complete bus protocol.
6. **Overlapping cases.** e.g., MAC array is `signal_security_accel / compute_accel`. CDC FIFO is `communication_interface / clock_domain_sync`.
7. **Use `others` sparingly.**
8. **Always choose the best label.** If ambiguous, choose the most plausible and lower confidence.

**Confidence Guide**
- `high`: 0.85-1.00 - Clearly fits one category.
- `medium`: 0.65-0.84 - Plausible, but reasonable alternative exists.
- `low`: 0.00-0.64 - Ambiguous, incomplete, or mixed-purpose.

**Output Format**
Return a JSON object only. Do not include markdown, comments, or extra explanation. `second_coarse` and `second_fine` are required only when `confidence_level` is `medium` or `low`; set both to `null` only when `confidence_level` is `high`.

```
{
  "coarse": "<category_name>",
  "fine": "<subcategory_name>",
  "confidence_score": 0.0,
  "confidence_level": "high | medium | low",
  "second_coarse": "<category_name>",
  "second_fine": "<subcategory_name>",
  "reason": "<few sentences justification>"
}
```

Prompt for RTL Circuit Functional Classification Verification

**Role**
You are a senior RTL design expert with 20+ years of experience in digital circuit design and hardware architecture. Your job is to verify a first-round RTL functional classification result. You should be conservative: keep the first-round label if it is reasonable, revise it only when the RTL evidence clearly supports a better label, and request human review when the evidence is ambiguous.

**Task**
You are given:
1. A Verilog/SystemVerilog RTL circuit.
2. A first-round classification result for that circuit.
Verify whether the first-round classification is correct under the taxonomy and rules below.
Return exactly one of the following verdicts:
- `accept`: the first-round label is reasonable and should be kept.
- `revise`: the first-round label is clearly wrong or suboptimal, and a better label is strongly supported by the RTL evidence.
- `human_review`: the circuit is ambiguous, mixed-purpose, incomplete, or the correct coarse category cannot be determined confidently.

Do not reclassify from scratch unless the first-round label is clearly inconsistent with the RTL evidence.

**Taxonomy**
**1. `arithmetic_logic` - Arithmetic & Logic**
- `add_sub_compare`: Adders, subtractors, comparators, carry-lookahead/save units.
- `mul_div_mac`: Single multipliers, dividers, square-root units, single MAC units, FPU units.
- `shift_bit_logic`: Barrel shifters, popcount, parity generators, bitwise operations.
- `general_alu_logic`: Multi-function ALUs and combined arithmetic-logic units.
- `others`: Arithmetic/logic modules not fitting any subcategory above.

**2. `storage_buffer` - Storage & Buffer**
- `reg_latch_flipflop`: D/JK/T flip-flops, latches, shift registers, register files.
- `fifo_queue`: Synchronous FIFO, asynchronous FIFO, circular buffer, LIFO stack.
- `ram_rom_memory`: Single/dual-port SRAM, ROM, CAM, content-addressable memory.
- `cache_scratch`: Direct-mapped/set-associative cache, scratchpad, tag arrays.
- `others`: Storage modules not fitting any subcategory above.

**3. `control_state` - Control & State** • `fsm_controller`: Mealy/Moore FSMs, sequence detectors, state-encoded controllers.
- `counter_timer`: Synchronous/asynchronous counters, BCD/Johnson counters, PWM generators, clock dividers.
- `arbiter_scheduler`: Fixed-priority arbiters, round-robin arbiters, token-ring schedulers.
- `pipeline_hazard`: Hazard detection, stall insertion, forwarding/bypass units, pipeline control.
- `others`: Control/state modules not fitting any subcategory above.

**4. `datapath_transform` - Datapath & Transform** • `mux_demux_select`: Multiplexers, demultiplexers, crossbar switches, data routing.
- `encoder_decoder`: Binary encoders/decoders, priority encoders, Gray code converters, BCD converters.
- `serializer_deserializer`: Serial-to-parallel, parallel-to-serial, SERDES, bitstream alignment.
- `width_format_convert`: Bit-width extension/truncation, byte-order conversion, alignment padding.
- `others`: Datapath/transform modules not fitting any subcategory above.

**5. `communication_interface` - Communication & Interface** • `serial_protocol`: UART, SPI, I2C, 1-Wire, CAN controllers.
- `bus_interface`: AXI4/AXI-Lite, APB, AHB, Wishbone, on-chip bus interfaces and bus arbitration.
- `clock_domain_sync`: Dual-FF synchronizers, handshake synchronizers, CDC FIFO, metastability mitigation.
- `protocol_bridge`: Protocol conversion bridges, Ethernet MAC, USB controller, DMA/bus-to-memory data movers.
- `others`: Interface modules not fitting any subcategory above.

**6. `signal_security_accel` - Signal Processing / Security / Acceleration** • `dsp_filter_transform`: FFT, FIR/IIR filters, CORDIC, interpolators, decimators.
- `compute_accel`: MAC arrays, systolic arrays, activation functions, image filters, matrix multiply.
- `crypto_crc_hash`: AES, DES, ChaCha20, SHA-256/512, HMAC, RSA, ECC crypto, CRC/checksum.
- `rng_ecc_encoding`: LFSR used as RNG, TRNG, PRNG, Hamming ECC, error-correction codes.
- `others`: Signal/security/acceleration modules not fitting any subcategory above.

**Verification Rules**

1. **Verify by primary purpose.** Judge what the RTL circuit fundamentally does. Do not let incidental counters, FSMs, registers, adders, or bit operations dominate the label if they only support a higher-level function.
2. **Default to accepting reasonable labels.** If the first-round label is plausible and supported by the RTL evidence, return `accept`, even if another label is also possible.
3. **Revise only with clear evidence.** Return `revise` only when the first-round label is clearly inconsistent with the circuit's primary function, or when the fine subcategory is clearly wrong within the same coarse category.
4. **Request human review for genuinely hard cases.** Return `human_review` when two coarse categories are both plausible, the module is a structural wrapper with no dominant purpose, or the RTL code is incomplete. Do not use `human_review` as a fallback for uncertain fine subcategories.
5. **Use names and ports as evidence,** but not as the only evidence.
6. **Wrapper or structural modules.** Verify the label against the dominant child function. If multiple child modules expose independent functions, return `human_review`.
7. **Be conservative with coarse-category revisions.** A coarse-category revision requires strong, unambiguous RTL evidence.

**Priority Rules**

Apply the first matching rule in the order listed when categories overlap:

- Protocol, bus, or interface modules are `communication_interface` even if they contain FSMs, FIFOs, counters, or CRC logic.
- FIFO, RAM, register-file, cache, and scratchpad modules are `storage_buffer` even if they contain pointer counters, full/empty logic, or read/write control. Exception: if the module primarily implements a complete bus/protocol interface around storage, classify as `communication_interface`.
- AES, SHA, CRC, ECC, FFT, FIR, CORDIC, matrix, image, and ML kernels are `signal_security_accel` when they are the module's main purpose.
- Pure mux, demux, encoder, decoder, serializer, deserializer, or width-conversion modules are `datapath_transform`.
- Pure add, subtract, multiply, divide, compare, shift, bitwise, or ALU modules are `arithmetic_logic`.
- Pure FSM, counter, arbiter, timer, hazard, or handshake modules with no stronger higher-level purpose are `control_state`.

**Overlapping Cases**

- A single multiplier, divider, or MAC is `arithmetic_logic / mul_div_mac`.
- A MAC array, systolic array, matrix unit, image filter, or accelerator datapath is `signal_security_accel / compute_accel`.
- A priority encoder is `datapath_transform / encoder_decoder` unless it is embedded inside an ALU.
- A CDC FIFO is `communication_interface / clock_domain_sync` if its main role is clock-domain crossing; otherwise a normal FIFO is `storage_buffer / fifo_queue`.
- A DMA engine is `communication_interface / protocol_bridge` if it primarily moves data between bus or memory interfaces.
- A CRC/checksum module is `signal_security_accel / crypto_crc_hash` if CRC/checksum is the module's main purpose; if it is only part of a larger protocol module, classify the larger module as `communication_interface`.
- An LFSR used as a standalone random/test-pattern generator is `signal_security_accel / rng_ecc_encoding`; an LFSR embedded inside a CRC or protocol module is classified by the enclosing module's function.

**Confidence Guide**

Use both a numeric confidence score and a confidence level.

- `high`: 0.85-1.00 - The verification decision is clear and well-supported by RTL evidence.
- `medium`: 0.65-0.84 - The decision is plausible, but a reasonable alternative exists.
- `low`: 0.00-0.64 - The circuit is ambiguous, incomplete, or mixed-purpose.

**Confidence by verdict:**

- `accept`: confidence reflects how strongly the RTL evidence supports the first-round label.
- `revise`: confidence reflects how strongly the RTL evidence supports the revised label over the first-round label.
- `human_review`: use `medium` when the circuit is functionally mixed but the RTL is clear; use `low` when the RTL is incomplete or too ambiguous. Do not use `high` with `human_review`.

**Output Format**

Return a JSON object only. Do not include markdown, comments, or extra explanation. {

```
  "verdict":  "accept | revise | human_review",
  "final_coarse":  "<category_name>",
  "final_fine":  "<subcategory_name>",
  "confidence_score":  0.0,
  "confidence_level":  "high | medium | low",
  "reason":  "<few sentences justification based on RTL
evidence>"
}
```

**Field rules:**

- `final_coarse` and `final_fine` must use only labels from the taxonomy above.
- If `verdict` is `accept`: set `final_coarse` and `final_fine` to the first-round label.

- If `verdict` is `revise`: set `final_coarse` and `final_fine` to the revised label.
- If `verdict` is `human_review`: set `final_coarse` to the most plausible coarse category, and set `final_fine` to `others` if no subcategory can be determined confidently.
- `reason` must cite concrete RTL evidence (signal names, port behavior, instantiated modules, etc.). Do not restate the label name as the justification.

### F.2 Functional Variant Generation Prompts

**Prompt for RTL Optimization and PPA Exploration**

**Role**
You are an expert Circuit Design Engineer specializing in RTL optimization and architectural exploration. Your goal is to rewrite Verilog modules to create functional equivalents with different PPA profiles while maintaining strict interface integrity.

**Strict Constraints**
1. **Functional Equivalence:** The logic behavior must remain identical to the original code.
2. **Interface Integrity:** Do NOT modify module names, port names, port directions, or port bit-widths.
3. **Hierarchy Preservation:** If multiple modules exist, do NOT change the instantiation structure. Only modify the internal logic of each module.
4. **No Bloatware:** For PPA degradation, do not insert random dummy registers or redundant wires. Instead, use sub-optimal architectural structures (e.g., serial vs. parallel logic).
5. **Output Format:** Output ONLY the complete Verilog code. No explanations, no markdown notes, and no comments.

**Transformation Techniques Library**

**1. Arithmetic Strength Transformation:**
- **Optimization:** Replace expensive operators with cheaper ones (e.g., `y = x * 9` to `y = (x « 3) + x`). Use bit-level manipulation to reduce adder size.
- **Degradation:** Use generic operators (like `*`) where shift-add logic is more efficient, or serialize parallel operations to increase latency.

**2. Resource Sharing:**
- **Optimization:** Force the sharing of expensive operators (multipliers, adders) across mutually exclusive paths using multiplexers before the operator.
- **Degradation:** Duplicate hardware by implementing the same operation in multiple branches of a conditional statement instead of sharing.

**3. Control Path & Mux Restructuring:**
- **Optimization:** Flatten nested `if-else` (where conditions are mutually exclusive) into `case` statements (parallel logic) to reduce logic depth (Timing). Balance Mux trees into binary structures.
- **Degradation:** Increase logic depth by deeply nesting `if-else` statements, creating long chains of priority logic.

**4. Operator Migration & Common Subexpression:**
- **Optimization:** Move operators across Muxes for integer arithmetic (e.g., `s ? (a+b) : (a+c)` becomes `a + (s ? b : c)`). Extract common sub-expressions into wires.
- **Degradation:** Distribute common operations into multiple redundant branches, increasing Area and Power.

**5. FSM Encoding & Logic Reorganization:**
- **Optimization:** Change FSM encoding (One-hot for speed, Binary for area). Optimize state transition logic to reduce the fan-out of control signals.
- **Degradation:** Use sub-optimal encoding for the given state count or implement highly complex, non-minimized next-state logic.

### F.3 Functional Description Prompts

**Prompt for Combinational Verilog Design Summarization**

You are a professional VLSI designer and an expert at Verilog coding.
I will provide you with a combinational Verilog design with multiple inputs and one or more outputs. Please use less than 300 words to summarize the functionality of this Verilog design (based on the input/output/intermediate signal names, the computational operations, etc.).

Here is the Verilog design:

```
```verilog
{documents}
```
```

### F.4 Retrieval Query Prompts

**Prompt for Module-level Analysis and JSON Description**

You are a professional VLSI designer and an expert at Verilog coding. Your task is to analyze a Verilog module and provide a structured description in JSON format.
Analyze the following Verilog module. Your response MUST be a single, valid JSON object. Do not include any introductory text or explanations outside of the JSON structure.

The JSON object should have the following keys:

1. `"suggested_name"`: A short, descriptive, and functional name for the module (e.g., `"ALU"`, `"FIFO Controller"`).
2. `"inputs"`: A list of strings, where each string is a high-level description of an input's purpose (e.g., `"Clock signal"`, `"Data to be written"`, `"Reset signal"`). Do not use signal names from the code.
3. `"outputs"`: A list of strings, similar to inputs, describing each output's purpose (e.g., `"Result of calculation"`, `"Indicates buffer is full"`).
4. `"functionality"`: A concise paragraph describing what the module does, its main operations, and its purpose. Avoid implementation details.
5. `"sub_modules_called"`: A list of strings containing the names of any other modules instantiated within this module. If none, provide an empty list `[]`.

Here is the Verilog module code: ```verilog
{module_code}
```

**Prompt for Design-level Query and High-level Overview**

You are a professional VLSI designer and an expert technical writer. You synthesize descriptions of individual circuit modules into a cohesive, high-level overview of the entire design.
You are given descriptions for individual hardware modules that make up a larger digital circuit. Your task is to generate a single, high-level natural language description of the **entire circuit's functionality**. Follow these requirements:

1. Focus on the overall purpose and main operations of the complete design. Synthesize, do not just list the parts.
2. Do not include any variable names, signal names, or the suggested module names from the provided context.
3. The description should be concise, clear, and written as if a human user is describing what they want the final circuit to achieve.
4. Keep the final description under 400 words.

Here are the descriptions of the individual modules: `{context_str}`
```