QiMeng-VPID: Verification-Grounded Port-Level Iterative Decomposition for Complex Verilog Generation

Hongguang Wang[1,2,3], **Jiaming Guo**[1,3], **Rui Zhang**[1,3], **Zerun Li**[1,3], **Di Huang**[1,3], **Pengwei Jin**[1], **Zidong Du**[1,3], **Xing Hu**[1,3], **Qi Guo**[1,3] and **Yunji Chen**[1,3*]

[1]State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences
[2]Jiangsu Key Laboratory of AI for Industries, Institute of AI for Industries, Chinese Academy of Sciences
[3]University of Chinese Academy of Sciences

Abstract

While Large Language Models (LLMs) have shown promise in translating natural-language specifications to Register-Transfer Level (RTL) designs, they often fail on complex, port-rich IPs. Existing frameworks typically separate generation from debugging, relying on static decomposition and iterative repair, which is hard to verify and yields unstable, inefficient maintenance. In this work, we propose VPID, a multi-agent framework for generating complex Verilog that achieves monotonic functional improvement. Given RTL's inherent concurrency and the alignment between verification and port-level behavior, our key insight is to treat ports as verifiable boundaries for dynamic decomposition, enabling fine-grained analysis to pinpoint root causes and guide targeted debugging. Specifically, we implement a behavior locking mechanism that preserves the behavior of verified ports to ensure monotonicity, preventing regressions in correct functionalities. To accelerate convergence, we introduce an experience-guided refinement strategy that distills historical waveform mismatches into constraints, guiding the targeted debugging for the unverified ports. Together with precise Abstract Syntax Tree (AST)-based code extraction, these mechanisms enable an efficient incremental generation workflow. Experiments on REALBENCH demonstrate that VPID outperforms both one-pass general-purpose LLMs and existing agent-based frameworks in both syntax and functional correctness, presenting a robust approach for automating Verilog generation for complex RTL designs.

1 Introduction

Register-Transfer Level (RTL) code generation is a fundamental yet labor-intensive stage in digital hardware design, requiring engineers to translate complex specifications into large volumes of hardware description language (HDL) code.

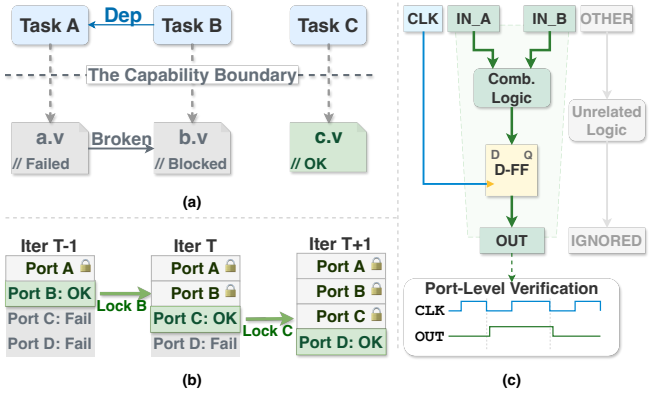

Figure 1: (a) Heuristic decomposition can fail on subtasks. (b) VPID iteratively decomposes the generation task to focus exclusively on remaining unverified ports while preserving previously verified behaviors, ensuring monotonic functional completion. (c) Logical locality and verifiability of ports.

Recent advances in large language models (LLMs) have motivated growing interest in automating this process through general-purpose models [Hui *et al.*, 2024; Team, 2025] or post-training domain-specialized models [Zhao *et al.*, 2025a; Zhu *et al.*, 2025]. Despite promising results on academic benchmarks [Liu *et al.*, 2023; Lu *et al.*, 2024], the LLM-based one-pass generators face significant challenges when applied to complex IP-level designs, which are characterized by long specifications, large multi-port interfaces with strict protocol constraints, and tightly coupled control and datapath logic.

The recent emergent agent-based frameworks are promising to mitigate the brittleness of one-pass generation. These frameworks [Zhao *et al.*, 2024; Ho *et al.*, 2025] typically decompose specifications into simpler subtasks, generate implementations independently, and refine the integrated result using testbench feedback. However, such approaches often rely on heuristic task decompositions, which are not directly verifiable. Verification ultimately evaluates only the functional behavior of the final RTL rather than the correctness of these decomposed subgoals. When a static decomposition is inaccurate, subsequent refinement lacks a principled mechanism to localize, invalidate, or revise it, which can lead to inefficient backtracking or systematic failure modes. Moreover, even plausible decompositions may misalign with the genera-

*Yunji Chen (cyj@ict.ac.cn) is the corresponding author.

tor's practical implementation capacity, over-allocating effort to trivial or unrealizable subtasks while leaving verification-critical behaviors insufficiently addressed. These observations raise a fundamental question: can the decomposition process be grounded in verifiable evidence, rather than relying on heuristic or rigid priors?

In fact, RTL design exhibits a natural port-level locality and verifiability, providing a principled answer to this question. Each output port is driven by a specific fan-in logic cone, and verification environments already assess correctness independently at the port level. As a result, port-level verification status constitutes a semantically faithful unit of progress rather than a heuristic signal. This property allows specification-to-RTL (spec-to-RTL) generation to be reformulated from a binary success-or-failure outcome into a stateful process of partial correctness, in which behaviors of some ports are verified while others remain unresolved. By explicitly representing progress at the port granularity, generation can advance in fine-grained, verifiable steps, offering a concrete alternative to static heuristic decomposition.

Motivated by this port-centric perspective, we introduce VPID, an agent-based spec-to-RTL framework that formulates RTL generation as a port-behavior-driven, iterative decomposition process. Rather than relying on a static decomposition, VPID dynamically induces decomposition boundaries according to port-level verification outcomes. At each iteration, verified behaviors are preserved while incremental generation focuses exclusively on the remaining unverified ports. Specifically, VPID couples two primary mechanisms that convert verification feedback into cumulative functional gains to operationalize this progression. First, *behavior locking* incrementally merges newly verified behaviors into a globally evolving skeleton, while keeping the behaviors of previously verified ports unchanged across iterations. Second, *experience-guided refinement* distills past failures into targeted hypotheses and constraints for subsequent steps, focusing debugging and regeneration on the remaining ports to accelerate convergence toward a valid design. Consequently, decomposition is aligned with verification, and debugging becomes a non-regressive refinement process constrained by previously verified behaviors and guided by observed failures. By exploiting the generator's effective capability, namely the subset of the remaining unverified ports it can correctly implement within each iteration, VPID facilitates monotonic functional convergence. Collectively, these mechanisms yield a non-regressive workflow that monotonically expands the verified port set, dynamically shaping the decomposition through port-level verification feedback, as shown in Figure 1(b).

Experiments on complex spec-to-RTL generation tasks demonstrate that VPID achieves new state-of-the-art results on REALBENCH and a selected subset of CVDP [Jin *et al.*, 2025; Pinckney *et al.*, 2025]. Specifically, with Claude-3.7 as the backbone, VPID achieves a 48.3% functional Pass@1, outperforming the representative multi-agent baseline MAGE [Zhao *et al.*, 2024] and general-purpose LLMs by over 2×, which achieve 21.7% and 20.0% respectively. This advantage is robust across different backbone models, such as Qwen-Coder, and extends to designs of varying complexity. On the CVDP `cid003` task, VPID attains a 62.8% pass rate compared to 44.9% for MAGE and 48.0% for Claude-3.7-thinking. We attribute these gains to our shift from static decomposition to verification-grounded dynamic decomposition. Through port-wise constraint enforcement using behavior locking and waveform analysis, VPID ensures that verified functionality accumulates monotonically, thereby improving functional correctness in complex RTL generation.

2 Related Work

2.1 Training-based RTL Code Generation

Training-based spec-to-RTL research typically treats executable verification as the primary target. Models are trained and evaluated by whether the generated RTL is compilable, synthesizable, and functionally correct under simulation. With increasingly standardized benchmarks and automated compile and simulate pipelines, *pass@k* has become a widely used metric for quantifying generation capability [Chen *et al.*, 2021; Liu *et al.*, 2023; Pinckney *et al.*, 2024; Lu *et al.*, 2024; Liu *et al.*, 2024]. Under this setting, supervised fine-tuning stands as the dominant approach for transferring RTL-specific knowledge. Contemporary improvements focus on refined data and training strategies, including multi-stage, multi-task training and curriculum learning to better align high-level specifications with low-level hardware semantics [Gao *et al.*, 2024; Zhang *et al.*, 2024; Liu *et al.*, 2025]. Recently, reinforcement learning has been explored to further improve reasoning and code generation by using compiler- and simulator-derived feedback as rewards, aiming to reduce dependence on static, dataset-only supervision [Zhu *et al.*, 2025; Qin *et al.*, 2025; Teng *et al.*, 2025]. Nevertheless, empirical results on IP-level benchmarks suggest a persistent gap when scaling from standard tasks to designs with long specifications and high coupling: even leading open-source large language models often struggle to meet stringent verification requirements without additional systematic engineering support, indicating the limits of relying on parameter optimization alone for industry-scale complexity [Jin *et al.*, 2025].

2.2 Agent-based Frameworks

Beyond model fine-tuning, agent-based methods advance RTL generation from one-pass generation to coupling task decomposition and debugging loops. To tackle long specifications and complex designs, decomposition strategies are widely adopted. VERILOGCODER [Ho *et al.*, 2025] proposes a planning mechanism based on a Task and Circuit Relation Graph (TCRG), decomposing designs into subtasks with specific signal details. While effective for local context retrieval, its performance is sensitive to the quality of specification and the precision of relation extraction. SPEC2RTL-AGENT [Yu *et al.*, 2025] processes complex multi-modal specifications via section-by-section summarization and hierarchical decomposition, decomposing the specification into an information dictionary and a list of sub-functions. It also employs an adaptive reflection module for automated error tracing, while allowing for human intervention as a fallback

in terms of unresolved failures. A similar top-down decomposition system [Chao *et al.*, 2025] employs hierarchical planning and decoupled verification. However, the recursive verification and upward backtracking mechanisms can introduce significant latency and token overhead, potentially limiting scalability for complex IP designs. Since decomposition and merging do not guarantee functionally correct RTL, frameworks often utilize verification feedback to guide iterative code refinement. MAGE [Zhao *et al.*, 2024] leverages state checkpoints to drive debugging, refining code globally from expected–actual output mismatches. However, it does not separate trusted logic from editable assignments, so the refinement process does not guarantee monotonic improvement in functional correctness. Furthermore, Abstract Syntax Tree (AST)-based analysis can be used to achieve lightweight fault localization, as demonstrated in VERILOGCODER. However, autonomously selecting logic-cone backtracking depth can require many tool-invocation iterations on complex RTL. Explicit diagnosis and error attribution remain underexplored. Additionally, other reliability enhancement strategies operate at the role level via competitive agents, the tool level through progressive feedback, and the sample level by optimizing search and self-consistency [Mi *et al.*, 2025; Narayanan *et al.*, 2025; Zhao *et al.*, 2025c; Zhao *et al.*, 2025b]. These methods trade high computational cost for performance, and may still fail when the backbone model is fundamentally incapable of solving the task.

Even with these advances, a critical bottleneck is committing too early to the initial decomposition. Such premature lock-in (i) places heavy demands on planning quality and global consistency across subtasks, (ii) often produces non-executable intermediate representations that cannot be directly validated by the testbench at fine granularity, and (iii) frequently induces a misalignment between planned subtasks and the code generator's effective capability. Once the initial plan diverges from the intended circuit semantics, errors can cascade across iterations, especially in tightly coupled, IP-level designs.

3 Problem Formulation

Input. We are given a natural-language specification S and a verification environment V for a target RTL module.

Output. We aim to generate an RTL implementation $R^\star$ that satisfies S under the environment V.

Verification. We use $\mathcal{P}$ to denote the set of output ports implied by S and verified by V. We define a port-aligned verifier $\mathrm{Ver}(\cdot; V)$ that evaluates any candidate RTL R and returns a binary vector $\mathbf{v}(R) = \mathrm{Ver}(R; V) \in \{0,1\}^{|\mathcal{P}|}$. For each $p \in \mathcal{P}$, $\mathbf{v}_p(R) = 1$ indicates that the behavior of port p is verified by V, and $\mathbf{v}_p(R) = 0$ otherwise. If R is not simulatable under V, we set $\mathbf{v}(R) = \mathbf{0}$. We define the verified port set as $P_{\mathrm{ver}}(R) = \{p \in \mathcal{P} \mid \mathbf{v}_p(R) = 1\}$ and the remaining port set as $P_{\mathrm{rem}}(R) = \mathcal{P} \setminus P_{\mathrm{ver}}(R)$.

Objective. Our objective is to obtain $R^\star$ such that $\mathbf{v}(R^\star) = \mathbf{1}$. We maintain an evolving skeleton $\widetilde{R}_0, \widetilde{R}_1, \ldots, \widetilde{R}_T$ and, at iteration t, incrementally complete logic for the remaining ports $P_{\mathrm{rem}}(\widetilde{R}_{t-1})$ without changing the behaviors of ports in $P_{\mathrm{ver}}(\widetilde{R}_{t-1})$. Port-wise non-regression is enforced by requiring $P_{\mathrm{ver}}(\widetilde{R}_{t-1}) \subseteq P_{\mathrm{ver}}(\widetilde{R}_t)$. Therefore, the verified port set expands monotonically. The generation is deemed successful when $P_{\mathrm{ver}}(\widetilde{R}_T) = \mathcal{P}$ and $R^\star = \widetilde{R}_T$ is returned.

4 Methodology

We propose VPID, a port-wise multi-agent framework for complex RTL generation that replaces heuristic decomposition with verification-grounded dynamic decomposition, turning the spec-to-RTL task into an iterative process that monotonically expands the verified port set. At iteration t, VPID focuses on $P_{\mathrm{rem}}(\widetilde{R}_{t-1})$ and performs incremental functional completion. It then generates and merges port-related RTL fragments into an evolving skeleton $\widetilde{R}_t$. Behavior locking is applied during generation and merging to prevent regressions on behaviors of $P_{\mathrm{ver}}(\widetilde{R}_{t-1})$, which guarantees monotonicity. To better leverage candidate RTL samples, VPID uses self-consistency comparison to filter candidates and AST-based analysis to extract reusable fragments, at both the sample level and the port level. In experience-guided refinement, a waveform diagnosis agent distills previous failures into port-specific constraints to guide subsequent generation.

4.1 Port-Verified Dynamic Decomposition

As formalized in Section 3, VPID measures progress in complex RTL generation via a monotonically expanding set of verified ports. In contrast to static heuristic decompositions, VPID performs feedback-guided decomposition, dynamically partitioning ports according to verification results. In other words, each iteration aims to implement the functionality of the remaining unverified ports while preserving the behaviors of $P_{\mathrm{ver}}(R)$, thereby reducing $P_{\mathrm{rem}}(R)$. This converts the original generation objective into an incremental, more tractable procedure of completing local port functionalities, driven by the interaction between LLM's generation capability and verification feedback.

To freeze the behavior of verified ports, we maintain an intermediate skeleton code $\widetilde{R}_t$ that drives $P_{\mathrm{ver}}(\widetilde{R}_t)$, leveraging the high semantic density of the code to represent the attained progress. The full port set $\mathcal{P}$ is produced by the preprocessor, which generates a module header $\mathcal{H}$ from S and uses an LLM-based self-check to verify that the preprocessing directives are satisfied and that the port declarations in $\mathcal{H}$ are complete, direction-correct, and type-consistent. An initial sampling-and-verification pass samples Verilog candidates and computes the verified set $P_{\mathrm{ver}}(\widetilde{R}_0)$ from their verification results, with the remainder forming $P_{\mathrm{rem}}(\widetilde{R}_0)$. The extracted fragments then initialize the skeleton $\widetilde{R}_0$.

Moreover, we leverage dataflow properties of RTL and associate each port with its corresponding fan-in logic cone $\mathrm{Cone}(p)$, making port-wise decomposition feasible. The logic cone is defined as the set of all signals required to determine the driving logic of a specific port, enabling fine-grained tracing across ports, internal variables, and module instances. In principle, iterative generation is confined to edits within $\mathrm{Cone}(P_{\mathrm{rem}})$ to complete the functionality of unverified ports,

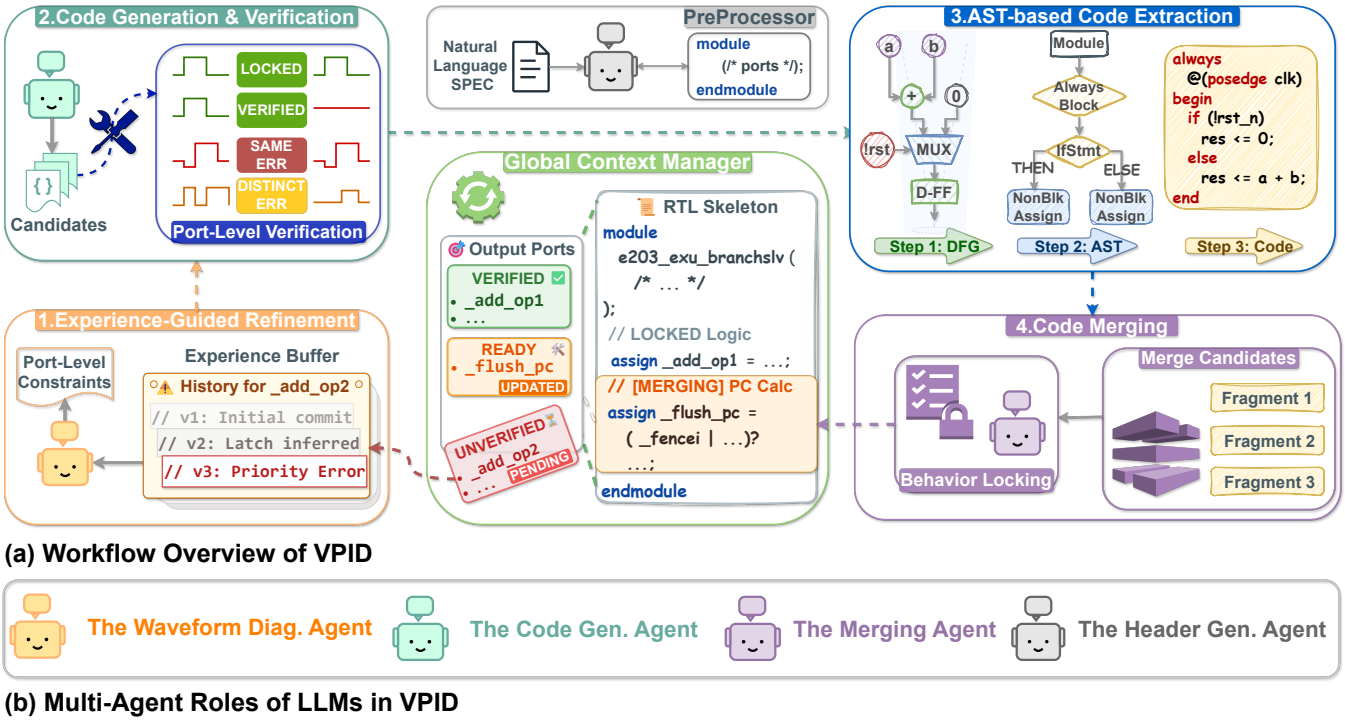

Figure 2: The framework overview of VPID, including (a) the iterative workflow, and (b) multi-agent roles of LLMs.

while avoiding disturbing already verified ports. We align the dataflow graph with its corresponding AST to ground this process at the statement level, supporting the extraction of a minimal, self-contained code fragment $\mathcal{C}_p$ for each port, where $\mathcal{C}_p$ contains all statements necessary to drive port p. This also includes all relevant control paths, signal declarations, and structural blocks. The logic cones of verified ports can guide incremental code integration, whereas those of unverified ports localize failing logic and facilitate the analysis for subsequent iterations.

4.2 Code Generation and Behavior Locking

At the t-th iteration, given the natural-language specification and the current skeleton code $\widetilde{R}_{t-1}$, we prompt the generator agent to incrementally implement $P_{\text{rem}}(\widetilde{R}_{t-1})$ while preserving the behavior of $P_{\text{ver}}(\widetilde{R}_{t-1})$. The generator agent samples diverse candidates and performs port-level verification, greedily identifying the newly verified port set $U_t \subseteq P_{\text{rem}}(\widetilde{R}_{t-1})$. For each $p \in U_t$, we extract the corresponding code fragment $\mathcal{C}_p$, and merge $\{\mathcal{C}_p \mid p \in U_t\}$ into the skeleton to update $\widetilde{R}_t$.

The code generation process enforces compilability and preserves port-level behavioral consistency in a feedback-in-the-loop procedure, implemented via two executable gates: a *compilation gate* and a *behavior gate*. The compilation gate first checks whether a Verilog candidate can be compiled. If compilation fails, we append the compiler logs with the syntax-error code to the next prompt to guide syntax correction, forming a dynamic prompt that improves syntactic correctness. Verilator is employed for this validation, offering configurable tolerance for minor issues such as intermediate logical incompleteness and bit-width mismatches, while effectively pinpointing critical errors like multiple drivers and combinational loops. Regarding $\widetilde{R}_{t-1}$ as reference, the behavior gate performs port-level waveform alignment under identical random stimuli, augmented with occasional reset perturbations. In particular, the behavior gate filters out any candidate failing to match the output of $P_{\text{ver}}(\widetilde{R}_{t-1})$. Behavioral locking requires only I/O equivalence. As a result, the process is insensitive to differences in internal structures or datapath signals, which increases flexibility during incremental generation and helps avoid deadlock. Mismatches trigger the construction of a dynamic prompt to guide the model's self-correction by incorporating the offending code regions and the dynamically constructed corrective instructions. To keep the prompt length from growing across loops, we always retain only the last failing implementation and its associated instructions. This setting is based on the assumption that the iterative loop has successfully resolved syntactic issues and fixed specific behavioral patterns over time. Instead of burdening the context with past failures, this approach utilizes the current refined state to guide the model, effectively reducing the difficulty of future tasks. This mechanism elevates non-regression of verified behaviors from a heuristic preference to an executable and verifiable hard constraint, guaranteeing the monotonicity of the iterative process.

Merging is also implemented with a coder agent that takes $\widetilde{R}_{t-1}$ and the fragments sliced from the union of $\text{Cone}(U_t)$, producing $\widetilde{R}_t$. During merging, the behavior gate is particularly helpful for resolving issues such as multi-driver con-

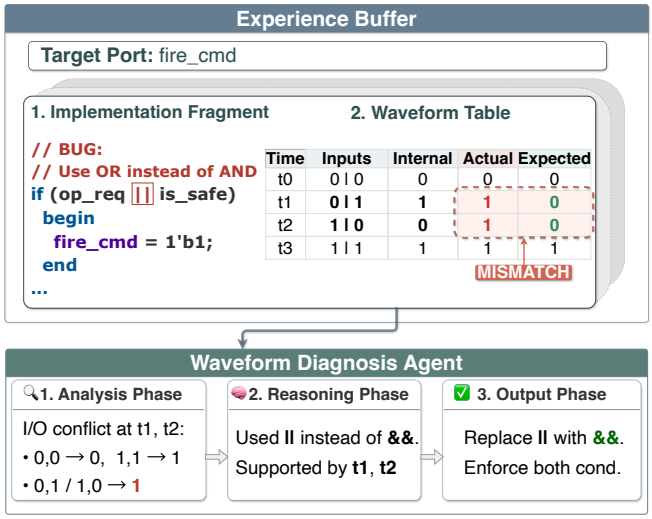

Figure 3: Workflow of the Waveform Diagnosis Agent utilizing Experience Buffer and waveform table evidence to generate port-specific constraints for subsequent Verilog generation.

flicts and naming ambiguities. It works alongside compilation feedback to localize the conflict points and trigger constrained regeneration with rich context. A merged result is accepted only if it satisfies both compilability and behavioral consistency, ensuring $P_{\text{ver}}(\widetilde{R}_{t-1}) \subseteq P_{\text{ver}}(\widetilde{R}_t)$. Finally, AST-based fragment extraction confines the merging context to a minimal closure, which helps control prompt length and prevents irrelevant or erroneous logic from introducing noise.

4.3 Experience-Guided Refinement

While port-level decomposition simplifies the global objective into a local optimization over P_{rem}, a naive generate-verify loop remains inefficient. Port-level failures may stem not only from implementation bugs, but also from incomplete or ambiguous specifications. In such cases, decomposition merely localizes the failing ports but offers no insight into root causes or actionable repairs. Moreover, strict behavior locking narrows the search space, making unguided regeneration repeatedly sample similar errors. To address this, we propose experience-guided refinement that extracts diagnostic features from verification logs and waveforms, turning failures into actionable debugging guidance.

Experience Buffer. Extending our analysis to failing ports, we maintain a global, port-indexed experience buffer $\mathcal{E}$ to store failure experiences. For each failing port $p \in P_{\text{rem}}$, we collect its corresponding code slice and waveform trace, and serialize the trace into a textual table that records the applied stimuli, expected versus observed port values, and relevant internal dataflow signals, over a bounded time window. Here, we further exploit the behavior gate mechanism described in subsection 4.2 to manage these failure experiences efficiently. Operationally, failed candidates are first clustered based on their gated behaviors to identify representative samples. These representatives are then compared against existing entries in $\mathcal{E}$. A new experience is admitted only if the port-level behavior it exhibits differs from all existing experiences in $\mathcal{E}$, and is thus deemed a new error mode. This strategy expands $\mathcal{E}$, covers diverse failure modes while keeping it manageable, and minimizes redundancy.

Waveform Diagnosis Agent. Based on the waveform evidence in $\mathcal{E}$, the diagnosis agent employs abductive reasoning to derive executable correction hypotheses, as shown in Figure 3. It begins with analyzing the temporal and logical structure of mismatches in waveform traces. Guided by a multi-label taxonomy of common failure modes, the agent classifies mismatches into specific error categories such as latency mismatches, control-priority violations, or arbitration flaws. Then it isolates the root cause by comparing the failing trace with the golden reference, deriving a minimal set of logical constraints to explain the observed discrepancies. These inferences are translated into executable constraint prompts for subsequent generations, avoiding rigid template matching and overfitted look-up tables, which serve as deterministic, port-specific behavioral constraints that both (i) fix logical errors and (ii) refine under-specified design requirements. Besides, this mechanism functions as a heuristic pruning operator. Enforcing non-regression constraints in subsequent iterations prevents the regenerating of previously observed errors and drives the process toward verification-grounded functional convergence.

5 Experiments

5.1 Experimental Setup

Benchmarks. We primarily evaluate on REALBENCH [Jin *et al.*, 2025], a realistic IP-level spec-to-RTL benchmark with 60 module tasks from three industrial designs (AES: 6, SDC: 14, E203 CPU: 40), featuring long specifications (averaging ∼10k tokens), multiple ports, and sizable implementations (typically ∼320 LOC per module) under strict simulation-based verification. To test whether VPID transfers beyond a single benchmark, we also report results on the `cid003` subset of CVDP [Pinckney *et al.*, 2025], comprising 78 spec-to-RTL tasks, as an out-of-benchmark generalization check.

Metrics. Syntax and functional Pass@1 serve as the primary evaluation metrics. A solution is considered syntax-correct if it compiles under a standard Verilog toolchain, and functionally correct if it passes all benchmark testbenches. For non-agentic baselines, we average the Pass@1 over 20 independent runs, whereas our agent is evaluated using a single end-to-end run per task.

Baselines In this study, we benchmark against two dominant paradigms for LLM-based RTL generation. *Non-agentic* baselines directly prompt general-purpose models to generate Verilog in a one-pass manner; specifically, we evaluate Claude-3.7, DeepSeek-V3 [DeepSeek-AI *et al.*, 2025b], DeepSeek-R1 [DeepSeek-AI *et al.*, 2025a], and GPT-5. *Agentic* baselines utilize multi-agent spec-to-RTL frameworks, including VERILOGCODER [Ho *et al.*, 2025] and MAGE [Zhao *et al.*, 2024] as representative methods.

5.2 Main Results

Table 1 shows that VPID improves functional Pass@1 on REALBENCH. With Claude-3.7 as the backbone model,

Method	SDC		AES		E203 CPU		ALL	
	Syn.	Func.	Syn.	Func.	Syn.	Func.	Syn.	Func.
Model Baselines								
DeepSeek-V3	44.2%	15.3%	55.8%	23.3%	19.5%	7.5%	28.9%	10.9%
DeepSeek-R1	28.5%	7.1%	66.6%	50.0%	12.5%	10.0%	21.6%	13.3%
Claude-3.7	41.4%	11.7%	46.6%	31.6%	42.7%	20.6%	42.8%	19.6%
GPT-5	35.7%	14.3%	50.0%	33.3%	30.0%	20.0%	33.3%	20.0%
Agent Baselines								
VerilogCoder (Claude)	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
MAGE (Claude)	57.1%	21.4%	66.6%	33.3%	62.5%	20.0%	61.7%	21.7%
VPID (Qwen-Coder)	92.9%	28.6%	100.0%	50.0%	82.5%	45.0%	86.7%	41.7%
VPID (Claude)	**100.0%**	**35.7%**	**100.0%**	**50.0%**	**87.5%**	**52.5%**	**91.7%**	**48.3%**

Table 1: Syntax and functional pass rate comparison on the REALBENCH benchmark.

VPID achieves a 48.3% functional Pass@1, substantially outperforming the representative multi-agent baseline MAGE (21.7%) as well as the strongest non-agentic baseline (20.0%). To ensure a rigorous and fair comparison with agentic baselines, we standardized the computational settings. For MAGE, we aligned the per-batch candidate sample size with VPID to normalize diversity, while granting it a relaxed debugging budget of twice the maximum iterations empirically required by VPID to rule out insufficient exploration. Despite the increased budget, MAGE consistently failed to converge on the complex IP-level tasks. Furthermore, VERILOGCODER was unable to produce valid solutions under limited cost, as its generic ReAct-based workflow appears ill-suited for IP-scale design, where the extensive context and non-deterministic reasoning loops lead to prohibitive computational costs and diminishing success rates.

In line with this, VPID achieves a syntax Pass@1 of 91.7% on REALBENCH, compared to 61.7% for MAGE and 42.8% for the non-agentic baseline. The key lies in the feedback loop of behavior locking, which enforces syntactic correctness and keeps generated RTL compatible with the target EDA tools and simulation setup. Empirically, we observe that the backbone model can generate a compilable Verilog within three iterations, substantially improving the verifiability of generated code. The limitations of current general-purpose backbones further underscore the importance of syntax correctness as a necessary condition to verification. As shown in Table 1, DeepSeek-V3 and DeepSeek-R1 exhibit low syntax Pass@1 on REALBENCH, at 28.9% and 21.6%, respectively. Their outputs frequently fail before simulation, preventing the system from obtaining verification feedback and thus limiting functional performance. Such syntactic failures reduce sample utilization since potentially correct candidates cannot be evaluated. In contrast, the iterative process of VPID works exclusively on compilable code and leverages both correct behaviors and incorrect logic from all samples, yielding a high utilization rate.

For complex specifications with long-range dependencies and implicit reasoning, comparisons with agentic baselines show that workflow design is critical to performance. Existing decomposition methods lack adaptability and may not be suitable for generative models, while global editing is inefficient as it neither preserves verified components nor effectively learns from previous failures. VPID instead takes advantage of iteration history: it freezes verified behaviors and directs the model to focus on the remaining ports. These results suggest that functional correctness depends on leveraging verification feedback, accumulating validation insights, and localizing the search space to align with the model's capabilities.

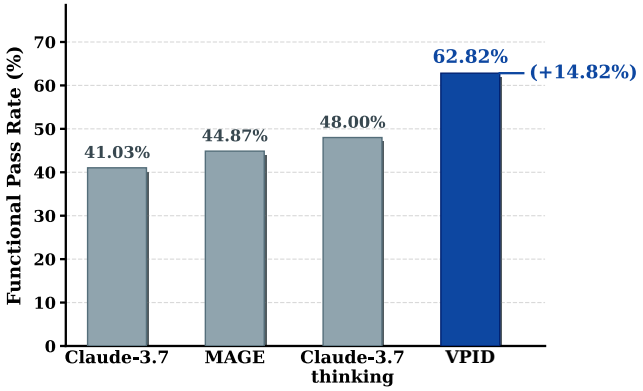

Figure 4: Functional pass rate on the CVDP `cid003` task (78 non-agentic problems).

Finally, VPID exhibits strong robustness across backbone models and varying task complexities. The framework consistently improves functional Pass@1 while maintaining high syntax correctness across multiple backbones; under Qwen-Coder, it achieves a 41.7% functional Pass@1. On the out-of-domain `cid003` task, VPID enables the backbone to significantly outperform the baseline and even surpass the performance reported in the CVDP benchmark using thinking-enabled models, attaining 62.82% functional Pass@1. This suggests that the same port-driven verification-grounded framework is not overfit to REALBENCH, but generalizes to designs with varying complexity.

Method	SDC		AES		E203 CPU		ALL	
	Pass	Iter	Pass	Iter	Pass	Iter	Pass	Iter
VPID	**35.7%**	**3.43**	50.0%	–	**52.5%**	**2.37**	**48.3%**	**2.58**
w/o behavior locking	35.7%	3.50	–	–	47.5%	2.57	45.0%	2.73
w/o behavior locking & naive debugging	21.4%	3.93	–	–	37.5%	2.89	35.0%	3.04
preprocessor output only	21.4%	–	50.0%	–	25.0%	–	26.7%	–

Table 2: Ablation study on the REALBENCH benchmark. We report both the functional pass rate (Pass) and the average number of iterations (Iter), mainly on SDC and E203 CPU tasks.

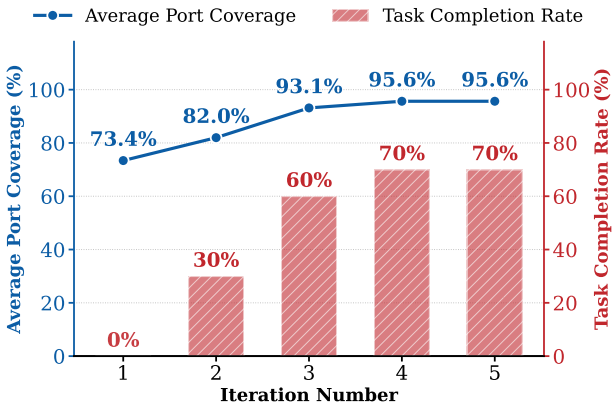

Figure 5: Case study of monotonically increasing port pass rates and task success rates during the iterative process for 10 tasks of REALBENCH where preprocessor generation failed.

5.3 Ablation and Analysis

To disentangle the contributions of VPID's individual components to functional correctness and convergence efficiency, we conduct an ablation study on REALBENCH. We evaluate functional Pass@1 under a fixed iteration budget of 5, and use the preprocessor output as the baseline. Table 2 summarizes the results, comparing the complete framework against variants that remove either the waveform-diagnosis agent or behavior locking. We exclude the AES subset from detailed discussion because it is comparatively simple. Overall, the ablations indicate that syntactic validity, while necessary, is not sufficient. Performance on challenging IP designs hinges on the ability to accumulate functional improvements across iterations in a monotonic manner.

We first evaluate the preprocessor in isolation. As shown in Table 2, using only the preprocessor output achieves a 26.7% pass rate, an increase of 7.1 percentage points over one-pass generation (19.6%). This improvement is likely due to the strict compile-time checks, which eliminate basic syntax errors that can confound the evaluation of functional correctness. However, the substantial gap between this baseline (26.7%) and the full framework (48.3%) suggests that syntactic correctness is necessary but not sufficient for handling complex logic. Iterative behavioral refinement is required to achieve the full framework performance.

Performance degrades considerably when naive debugging is employed without behavior locking, resulting in a pass rate of 35.0%. In this configuration, naive debugging captures only real-time waveform, lacking both guided correction and historical experience replay. The generation process becomes unstable because the agent tends to overfit to instantaneous waveform mismatches, or even modify correct logic, making it difficult to accumulate functional progress during iterations, ultimately leading to failure in complex designs.

The ablation of behavior locking primarily impacts efficiency rather than just functional correctness. On the E203 CPU tasks, the pass rate decreases slightly from 52.5% to 47.5%, accompanied by an increase in average iterations from 2.37 to 2.57. Consistently, we see a slight increase from 3.43 to 3.50 on SDC tasks as well. This consistent increase in iterations across tasks suggests that without locking, the agent struggles to preserve previously verified behaviors. Consequently, it expends additional rounds repairing regressions rather than making monotonic progress toward the solution.

Figure 5 visualizes VPID's monotonicity on a case study of ten challenging tasks for which the preprocessor baseline fails. These failures motivate an iterative, verification-guided refinement loop. Across iterations, the blue verified port coverage and the red overall task completion rate both increase without backtracking. Ablation results suggest that the monotonic trend depends on three components: dynamic decomposition focusing on unsatisfied ports, waveform diagnosis deriving constraints from mismatches, and behavior locking preventing regressions across iterations.

6 Conclusion

In this paper, we present VPID, a port-wise multi-agent framework for complex spec-to-RTL generation. Our key insight lies in the inherent port locality and verifiability of RTL. Ports are verifiable and analyzable, enabling behavioral alignment and can be used to guide dynamic decomposition. Therefore, VPID treats progress as the expansion of a verified port set rather than one-pass generation or uncertain heuristic decomposition, and organizes generation, debugging, and merging around ports. Building on this view, VPID couples behavioral locking and waveform diagnosis, which respectively guarantee iterative monotonicity and accelerate functional convergence. Experimental results demonstrate that VPID achieves 91.7% syntactic correctness and 48.3% functional correctness at Pass@1 on the challenging REALBENCH benchmark, outperforming both general-purpose LLMs and multi-agent frameworks relying on decoupled decomposition and debugging. Our work advances the field toward robust, verification-grounded hardware automation. Future work will extend this port-centric paradigm to system-level integration and larger-scale design spaces.

Ethical Statement

LLMs were used to enhance the readability and language quality of this manuscript. The authors reviewed and revised the output and take full responsibility for the content.

Acknowledgements

This work is supported by National Key R&D Program of China (2024YFE0212000), and the Research Fund of Jiangsu Key Laboratory of AI for Industries (NO. E6420012G8). It is also partially supported by the NSF of China (Grants No.62341411, 62525203, U22A2028, 62222214, 6240073476), Science and Technology Major Special Program of Jiangsu (Grant No. BG2024028), Strategic Priority Research Program of the Chinese Academy of Sciences (Grants No.XDB0660200, XDB0660201, XDB0660202), CAS Project for Young Scientists in Basic Research (YSBR-029) and Youth Innovation Promotion Association CAS.

References

[Chao *et al.*, 2025] Zhiteng Chao, Yonghao Wang, Xinyu Zhang, Jiaxin Zhou, Tenghui Hua, Husheng Han, Tianmeng Yang, Jianan Mu, Bei Yu, Rui Zhang, Jing Ye, and Huawei Li. Think with self-decoupling and self-verification: Automated rtl design with backtrack-tot, 2025.

[Chen *et al.*, 2021] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, et al. Evaluating large language models trained on code, 2021.

[DeepSeek-AI *et al.*, 2025a] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[DeepSeek-AI *et al.*, 2025b] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, et al. Deepseek-v3 technical report, 2025.

[Gao *et al.*, 2024] Mingzhe Gao, Jieru Zhao, Zhe Lin, Wenchao Ding, Xiaofeng Hou, Yu Feng, Chao Li, and Minyi Guo. Autovcoder: A systematic framework for automated verilog code generation using llms. In *2024 IEEE 42nd International Conference on Computer Design (ICCD)*, pages 162–169, 2024.

[Ho *et al.*, 2025] Chia-Tung Ho, Haoxing Ren, and Brucek Khailany. Verilogcoder: autonomous verilog coding agents with graph-based planning and abstract syntax tree (ast)-based waveform tracing tool. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press, 2025.

[Hui *et al.*, 2024] 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.

[Jin *et al.*, 2025] Pengwei Jin, Di Huang, Chongxiao Li, Shuyao Cheng, Yang Zhao, Xinyao Zheng, Jiaguo Zhu, Shuyi Xing, Bohan Dou, Rui Zhang, Zidong Du, Qi Guo, and Xing Hu. Realbench: Benchmarking verilog generation models with real-world ip designs, 2025.

[Liu *et al.*, 2023] 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)*, 2023.

[Liu *et al.*, 2024] Shang Liu, Yao Lu, Wenji Fang, Mengming Li, and Zhiyao Xie. Openllm-rtl: Open dataset and benchmark for llm-aided design rtl generation(invited). In *Proceedings of 2024 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM, 2024.

[Liu *et al.*, 2025] Yi Liu, Changran Xu, Yunhao Zhou, Zeju Li, and Qiang Xu. Deeprtl: Bridging verilog understanding and generation with a unified representation model. In *The Thirteenth International Conference on Learning Representations*, 2025.

[Lu *et al.*, 2024] 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)*, pages 722–727. IEEE, 2024.

[Mi *et al.*, 2025] Zhendong Mi, Renming Zheng, Haowen Zhong, Yue Sun, Seth Kneeland, Sayan Moitra, Ken Kutzer, and Zhaozhuo Xu Shaoyi Huang. Coopetitivev: Leveraging llm-powered coopetitive multi-agent prompting for high-quality verilog generation, 2025.

[Narayanan *et al.*, 2025] Athma Narayanan, Mahesh Subedar, and Omesh Tickoo. Pefa-ai: Advancing open-source llms for rtl generation using progressive error feedback agentic-ai, 2025.

[Pinckney *et al.*, 2024] Nathaniel Pinckney, Christopher Batten, Mingjie Liu, Haoxing Ren, and Brucek Khailany. Revisiting verilogeval: Newer llms, in-context learning, and specification-to-rtl tasks, 2024.

[Pinckney *et al.*, 2025] Nathaniel Pinckney, Chenhui Deng, Chia-Tung Ho, Yun-Da Tsai, Mingjie Liu, Wenfei Zhou, Brucek Khailany, and Haoxing Ren. Comprehensive verilog design problems: A next-generation benchmark dataset for evaluating large language models and agents on rtl design and verification, 2025.

[Qin *et al.*, 2025] Haiyan Qin, Zhiwei Xie, Jingjing Li, LiangChen Li, Xiaotong Feng, Junzhan Liu, and Wang Kang. Reasoningv: Efficient verilog code generation with adaptive hybrid reasoning model. *arXiv preprint arXiv:2504.14560*, 2025.

[Team, 2025] Qwen Team. Qwen3 technical report, 2025.

[Teng *et al.*, 2025] Fu Teng, Miao Pan, Xuhong Zhang, Zhezhi He, Yiyao Yang, Xinyi Chai, Mengnan Qi, Liqiang Lu, and Jianwei Yin. Verirl: Boosting the llm-based verilog codegeneration via reinforcement learning. In *International Conference on Computer-Aided Design(ICCAD)*, 2025.

[Yu *et al.*, 2025] Zhongzhi Yu, Mingjie Liu, Michael Zimmer, Yingyan Celine Lin, Yong Liu, and Haoxing Ren. Spec2rtl-agent: Automated hardware code generation from complex specifications using llm agent systems, 2025.

[Zhang *et al.*, 2024] Yongan Zhang, Zhongzhi Yu, Yonggan Fu, Cheng Wan, and Yingyan Celine Lin. Mg-verilog: Multi-grained dataset towards enhanced llm-assisted verilog generation. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pages 1–5, 2024.

[Zhao *et al.*, 2024] Yujie Zhao, Hejia Zhang, Hanxian Huang, Zhongming Yu, and Jishen Zhao. Mage: A multi-agent engine for automated rtl code generation, 2024.

[Zhao *et al.*, 2025a] Yang Zhao, Di Huang, Chongxiao Li, Pengwei Jin, Muxin Song, Yinan Xu, Ziyuan Nan, Mingju Gao, Tianyun Ma, Lei Qi, Yansong Pan, Zhenxing Zhang, Rui Zhang, Xishan Zhang, Zidong Du, Qi Guo, and Xing Hu. Codev: Empowering llms with hdl generation through multi-level summarization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2025.

[Zhao *et al.*, 2025b] Zhuorui Zhao, Bing Li, Grace Li Zhang, and Ulf Schlichtmann. Vfocus: Better verilog generation from large language model via focused reasoning, 2025.

[Zhao *et al.*, 2025c] Zhuorui Zhao, Ruidi Qiu, Ing-Chao Lin, Grace Li Zhang, Bing Li, and Ulf Schlichtmann. Vrank: Enhancing verilog code generation from large language models via self-consistency. In *2025 26th International Symposium on Quality Electronic Design (ISQED)*, pages 1–7, 2025.

[Zhu *et al.*, 2025] Yaoyu Zhu, Di Huang, Hanqi Lyu, Xiaoyun Zhang, Chongxiao Li, Wenxuan Shi, Yutong Wu, Jianan Mu, Jinghua Wang, Yang zhao, Pengwei Jin, Shuyao Cheng, shengwen Liang, Xishan Zhang, Rui Zhang, Zidong Du, Qi Guo, Xing Hu, and Yunji Chen. Qimeng-codev-r1: Reasoning-enhanced verilog generation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.