

AgenticPD: A Stage-Aware Agentic Framework for Closed-Loop Physical Design Optimization

Shuo Ren
The Chinese University of Hong Kong
Hong Kong, China
sren@cse.cuhk.edu.hk

Zijin Cheng
Southeast University
Nanjing, Jiangsu, China
220246802@seu.edu.cn

Yaohui Han
The Chinese University of Hong Kong
Hong Kong, China
yhhan25@cse.cuhk.edu.hk

Libo Shen
The Chinese University of Hong Kong
Hong Kong, China
lbsen24@cse.cuhk.edu.hk

Leilei Jin
The Chinese University of Hong Kong
Hong Kong, China
lljin@cse.cuhk.edu.hk

Wanting Tian
The Chinese University of Hong Kong
Hong Kong, China
wantingtian@cuhk.edu.hk

Rongliang Fu
The Chinese University of Hong Kong
Hong Kong, China
rongliangfu@outlook.com

Chao Wang
Southeast University
Nanjing, Jiangsu, China
wangc@seu.edu.cn

Bei Yu
The Chinese University of Hong Kong
Hong Kong, China
byu@cse.cuhk.edu.hk

Tsung-Yi Ho
The Chinese University of Hong Kong
Hong Kong, China
tyho@cse.cuhk.edu.hk

Abstract

Physical design quality-of-results (QoR) optimization is hard and expensive. Choices made at one stage can help or hurt later stages. Each evaluation requires a costly EDA run through the full flow. While existing methods still treat optimization as flat parameter tuning or a LLM-based script generation task, we present AgenticPD¹, a stage-aware agentic framework for physical design QoR optimization. Instead of re-running the full flow after every trial, AgenticPD is organized around the stage boundaries of the physical design flow, where a Judge Agent navigates the search and stage-specialized agents make local decisions within their own stage using stage-local tools. Additionally, the agent harness in AgenticPD provides structured observations, execution history, and agent context management. Experimental results show that AgenticPD achieves the best timing performance among the evaluated PD tuners while maintaining competitive power and area.

Keywords

Physical Design, QoR Optimization, Agentic EDA

ACM Reference Format:

Shuo Ren, Zijin Cheng, Yaohui Han, Libo Shen, Leilei Jin, Wanting Tian, Rongliang Fu, Chao Wang, Bei Yu, and Tsung-Yi Ho. 2027. AgenticPD: A Stage-Aware Agentic Framework for Closed-Loop Physical Design Optimization.

¹Our code is open-sourced at <https://github.com/Flans/AgenticPD>.

* Corresponding author: Rongliang Fu (rlfu@cse.cuhk.edu.hk).



This work is licensed under a Creative Commons Attribution 4.0 International License. ASPDAC '27, Tokyo, Japan

© 2027 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2923-2/2027/01

<https://doi.org/10.1145/3840404.3848213>

In *32nd Asia and South Pacific Design Automation Conference (ASPDAC '27)*, January 25–28, 2027, Tokyo, Japan. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3840404.3848213>

1 Introduction

As modern chips grow in design complexity, achieving strong performance, power, and area (PPA) under tight design schedules has become increasingly difficult [13]. Physical design (PD) plays a central role in this process [7], since implementation decisions made during backend optimization directly affect final chip quality. Moreover, physical design quality-of-results (QoR) optimization is far from straightforward because the implementation flow spans multiple connected stages, including floorplan, placement, clock tree synthesis, and routing [15]. Each stage has its own local objectives, optimization methods, and feedback metrics. Yet stage-local improvements do not always translate into better final QoR. A decision that appears highly effective at one stage may leave too little optimization space for later stages and can ultimately degrade final performance, power, or area. As a result, physical-design optimization must be understood not only at the level of individual stages, but also in terms of how decisions propagate across the full flow. Together, these cross-stage dependencies make QoR optimization a challenging full-flow problem in practical chip implementation [15].

Existing methods for physical design QoR optimization can be broadly divided into two categories. The first category includes black-box tuning methods, such as AutoTuner [14], PPATuner [8], REMOTune [29], and FastTuner [12], which improve sampling efficiency by automatically exploring the configuration space. However, these methods typically treat the physical design flow as a flat input-output mapping and do not explicitly exploit stage-wise structure, intermediate feedback, or reusable design states.

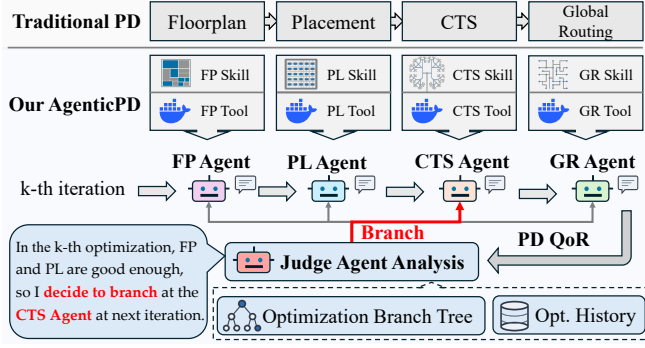


Figure 1: AgenticPD replaces black-box full-flow tuning with closed-loop, stage-aware multi-agent optimization.

The second category includes recent LLM-based systems, such as ORFS-Agent [9], OpenROAD-Assistant [23], OpenROAD Agent [27], and ChatEDA [28], which rely on language models for script generation and document exploration. However, these systems are still largely script-generation-like in how they organize optimization. Such a formulation is not well suited to physical-design QoR optimization, because this task is multi-stage, state-dependent, and requires decisions to be revised according to real execution feedback. Although these methods may incorporate feedback across trials, it remains at the full-flow configuration level; neither category closes the loop over reusable stage states and branch decisions.

To address these challenges, we present AgenticPD, a stage-aware agentic framework for physical design QoR optimization. As shown in Fig. 1, AgenticPD organizes optimization around the structure of the full physical design flow and supports more effective exploration of the QoR search space, rather than treating each trial as an independent attempt. This leads to stronger optimization outcomes than baselines. The main contributions are summarized as follows:

- We formulate physical design QoR optimization as a stage-aware tree search that captures cross-stage dependencies and supports non-sequential exploration across the full physical design flow.
- We propose a hierarchical multi-agent framework that closes the loop between measured backend QoR and history-aware branch navigation, while specialized agents optimize individual physical design stages using stage-local feedback.
- We build an agent-ready physical design environment with stage-scoped actions, structured execution feedback, and stage-aware design-state reuse across explored branches.
- Experiments across six design and technology combinations show improved timing over the evaluated baselines while maintaining competitive power and area across both technologies.

2 Preliminaries and Motivations

2.1 Physical Design

In the physical design flow [15], we focus on four controllable stages: floorplanning (FP), placement (PL), clock tree synthesis (CTS), and global routing (GR). During each optimization run, candidates are evaluated after GR and only the selected candidate proceeds to detailed routing (DR). Headline comparisons use post-DR metrics; in-loop analyses are labeled in Section 4.2. We formulate the physical

design flow as an ordered sequence of stages:

$$\mathcal{S} = (\text{FP}, \text{PL}, \text{CTS}, \text{GR}), \quad (1)$$

where $s \in \mathcal{S}$ denotes a physical design stage. We refer to the output of each stage as an *intermediate design state*, which can be reused when alternative downstream decisions are explored. Each stage s has its own action space Θ_s , representing the tunable parameters at that stage. The full action space factorizes as

$$\Theta_{\text{PD}} = \Theta_{\text{FP}} \times \Theta_{\text{PL}} \times \Theta_{\text{CTS}} \times \Theta_{\text{GR}}. \quad (2)$$

Because the stages execute in a fixed order, any selected stage $b \in \mathcal{S}$ partitions the flow into two parts: the stages before b , denoted $\text{Bef}(b)$, and the stages after b , denoted $\text{Aft}(b)$. For example, $\text{Bef}(\text{CTS}) = \{\text{FP}, \text{PL}\}$ and $\text{Aft}(\text{CTS}) = \{\text{GR}\}$. Moreover, a single complete flow execution is specified by an action tuple

$$\mathbf{a} = (a(\text{FP}), a(\text{PL}), a(\text{CTS}), a(\text{GR})) \in \Theta_{\text{PD}}, \quad (3)$$

where $a(s) \in \Theta_s$ denotes the action applied at stage s . Executing $\mathbf{a}$ runs the four stages in order and produces stage-level feedback $Q(s)$. We consider timing-primary physical design optimization: WNS and TNS are exposed as timing feedback, while post-GR WNS serves as the scalar criterion for incumbent selection; area and power are retained as secondary physical-quality metrics for final evaluation. Within the search budget N , the optimizer seeks the candidate with the highest post-GR WNS:

$$k^* = \arg \max_{0 \leq k < N} \text{WNS}_k(\text{GR}), \quad \mathbf{a}^* = \mathbf{a}_{k^*}. \quad (4)$$

During a standard optimization run, detailed routing is applied once to $\mathbf{a}^*$ for final reporting.

2.2 Agentic AI and Agentic EDA

Recent progress in Agentic AI shows that language models can solve complex tasks more effectively when they are allowed to reason, call tools, inspect execution results, and revise decisions over multiple steps rather than produce one static output [2, 5, 6, 11, 17, 18, 21]. This interaction pattern is especially useful when task state evolves during execution and when later actions depend on earlier observations.

The EDA community has also started to adopt this paradigm and apply agentic AI to EDA problems. Cadence introduced ChipStack for AI-assisted chip design and verification [3]. Synopsys introduced AgentEngineer for multi-agent automation of RTL and verification tasks [25]. Siemens EDA launched Fuse EDA AI Agent for multi-tool automation from design to physical implementation [24]. However, these agentic EDA tools still mainly focus on design, verification, and workflow automation, while the use of agentic AI for physical design optimization remains underexplored.

2.3 Motivations

Physical-design QoR optimization is a natural fit for an agentic framework for three reasons. (1) It is not a one-shot prediction but a sequential process across FP, PL, CTS, and routing, where each stage has its own objectives, action space, and feedback yet depends on the state produced by earlier stages—closer to multi-step decision making than to flat parameter generation. (2) It requires both prior knowledge and execution feedback: prior knowledge guides stage-local choices at the start, but the true effect of a decision is revealed only after running the tools, so the optimizer must often revise or

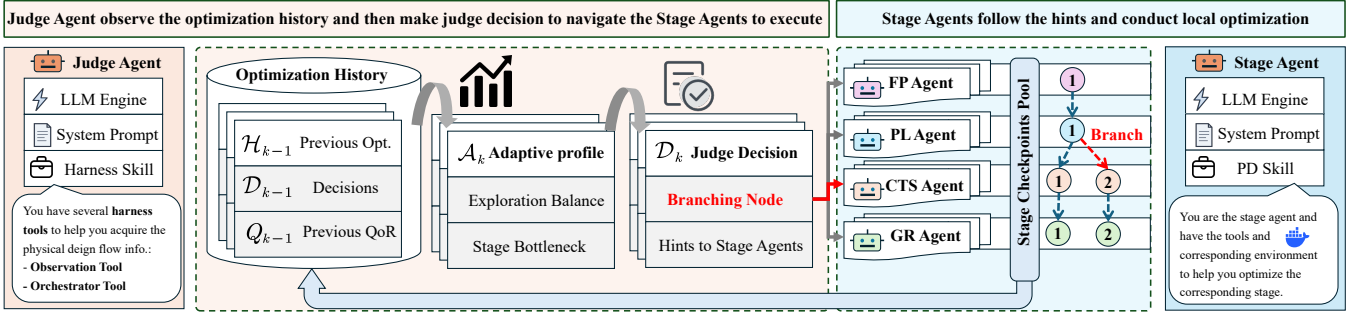


Figure 2: AgenticPD framework overview: measured QoR updates the optimization history, the Judge selects a branch node and stage, and downstream Stage Agents conditionally expand the selected subtree and return new feedback.

discard its initial judgment. (3) It requires repeated decision making and adjustment driven by execution feedback, which a one-shot script cannot naturally support. **Taken together, these properties make physical-design QoR optimization a natural fit for an agentic, stage-aware framework.**

3 Methodology

3.1 Stage-Aware Optimization Tree

Unlike trial-based tuners that generate an independent full-flow configuration at each iteration, AgenticPD can revisit a retained state, select a physical design stage to revise, and expand only the corresponding downstream subtree.

We represent this search as a rooted tree $\mathcal{T}$. The root n_0 represents the post-synthesis design D before any PD stage runs. Each time a stage s is executed in iteration k , a node is created:

$$n_k^s = (x_k(s^+), a_k(s), Q_k(s)), \quad (5)$$

where $x_k(s^+)$ is the resulting intermediate design state, $a_k(s) \in \Theta_s$ is the applied action, and $Q_k(s)$ is the observed stage-level QoR. Every root-to-leaf path traverses all four stages in order and corresponds to one complete PD flow with action tuple $\mathbf{a}_k$ (Equation (3)).

Branching. Not every iteration needs to start from scratch. If a previous iteration already produced a good result at some early stage, re-running that stage with a different action is wasteful; the optimizer should focus its budget on the stages that still have room to improve. Branching makes this possible. At iteration k , the optimizer selects a retained node $\hat{n}$ whose stored state is the input to a branching stage $b \in \mathcal{S}$ and starts a new branch from there. All stages before b are inherited from the ancestor path leading to $\hat{n}$ —the $\text{Bef}(b)$ results are reused at zero cost—and only stage b itself and the stages in $\text{Aft}(b)$ are re-executed with new actions. Let $q = |\mathcal{S}|$ and index the ordered stages by $1, \dots, q$. Their causal order is encoded by $C \in \{0, 1\}^{q \times q}$, and the branch choice activates a downstream suffix through

$$C_{ij} = \mathbb{I}[i \leq j], \quad \mathbf{m}_k = C^\top \mathbf{e}(b_k), \quad (6a)$$

$$\mathcal{T}_k = \mathcal{T}_{k-1} \cup \{n_k^{s_j} : m_{k,j} = 1\}. \quad (6b)$$

where $\mathbf{e}(b_k)$ is the one-hot vector of the branching stage. The first relation maps the Judge’s branch choice to stage activation: $m_{k,j} = 0$ marks a reusable upstream stage and $m_{k,j} = 1$ activates a downstream stage. The second relation attaches exactly those activated nodes to the tree. Together, they define the branch-conditioned subtree expansion while preserving the upstream design state stored

at $\hat{n}$. When $b_k = \text{FP}$ (the first stage), branching from the root reduces to running a completely new flow from scratch.

This tree structure turns flat independent trials into a structured search. It retains every explored design state, not only the current best candidate, so the optimizer can return to another promising node when refinement along one branch reaches a local limit. At each iteration, the Judge observes the search history, selects a branch node and branching stage, and generates stage-specific hints. The downstream Stage Agents then execute the remaining stages from the selected design state. The resulting nodes are added to the tree, while the best-WNS incumbent is updated independently.

3.2 History-Aware Branch Navigation

As shown in Fig. 2, the Judge serves as a navigation policy over the optimization tree rather than a generator of full-flow parameters. We instantiate this policy using a general-purpose LLM $\mathcal{L}$, a role-specific system prompt $\mathcal{P}_J$, and the Agent Harness $\mathcal{U}_J$. The prompt defines its branching output and search guidance, while the harness supplies the structured observation and executes the resulting branch decision (Section 3.3). At iteration k , the optimization history $\mathcal{H}_k$ records the selected node, branching stage, and downstream QoR from every previous iteration. Rather than directly generating a full-flow parameter vector, the Judge uses this history to decide where to resume the search and which stages to revise. The Observation Tool first computes an *adaptive profile*

$$\mathcal{A}_k = (\{E(n)\}_{n \in \mathcal{T}_k}, \{B(s)\}_{s \in \mathcal{S}}), \quad (7)$$

where the exploration balance $E(n)$ measures how often node n has been revisited and the stage bottleneck $B(s)$ measures how far stage s falls behind its best-known result. Together with a compressed tree snapshot, $\mathcal{A}_k$ forms the Judge’s search state. The Judge then produces

$$\mathcal{D}_k = (\hat{n}_k, b_k, h_k) = \text{Judge}(\mathcal{H}_k, \mathcal{A}_k), \quad (8)$$

where $\hat{n}_k$ is the branch node, $b_k \in \mathcal{S}$ is the branching stage, and h_k contains one hint for each downstream Stage Agent. This structured decision jointly determines where to resume the search, which stage to revise, and how to guide downstream expansion. Guided by $B(s)$ and $E(n)$, the Judge balances bottleneck repair with exploration of less-visited nodes. The selected node need not be the current incumbent; any retained node can seed a promising downstream branch. After execution, the new branch and its stage-level results are appended to $\mathcal{T}_k$ and $\mathcal{H}_k$, while the best-WNS incumbent is updated independently according to Equation (4).

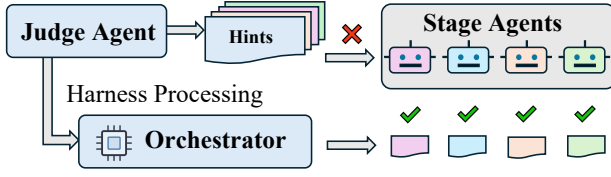


Figure 3: Agent Harness: the Judge’s hints are dispatched to downstream Stage Agents, each of which executes through its own PD Skill.

3.3 Branch-Conditioned Agent Execution

Given the Judge decision $\mathcal{D}_k = (\hat{n}_k, b_k, h_k)$, AgenticPD reuses the design state at $\hat{n}_k$ and activates only the Stage Agents selected by $\mathbf{m}_k$ in Equation (6a). For each activated stage s , the harness assembles

$$ctx_k(s) = (\{Q_k(i)\}_{i \in \text{Bef}(s)}, e_s, K_s, \text{hint}_s), \quad (9)$$

where the four terms are the upstream QoR on the current branch, a cross-iteration experience summary, a retrieved knowledge block, and the Judge’s stage-specific guidance, respectively. GraphRAG retrieves $K_s = \text{Retrieve}(s, ctx_k(s))$ from a documentation-grounded graph. The graph links documented stage-level actions to downstream metrics; its text blocks are embedded offline and matched to the current stage condition.

Let d_j be the number of tunable parameters at stage s_j , stack the stage-local proposals as $\mathbf{a}_k$, and let $\mathbf{a}_{\hat{n}_k}$ denote the configuration on the retained path. The branch-conditioned expansion is

$$\begin{aligned} M_k &= \text{blkdiag}(m_{k,1}I_{d_1}, \dots, m_{k,q}I_{d_q}), \\ \mathbf{a}_k &= (I - M_k)\mathbf{a}_{\hat{n}_k} + M_k\tilde{\mathbf{a}}_k. \end{aligned} \quad (10)$$

Here, $I - M_k$ preserves parameters on the retained path, whereas M_k replaces the activated blocks. Starting from $x(\hat{n}_k)$, each activated agent proposes $\tilde{a}_k(s) = \pi_s(ctx_k(s))$, and the backend advances the design through $x_k(s^+) = F_s(x_k(s), \tilde{a}_k(s))$, passing $Q_k(s)$ downstream. The Agent Harness realizes this execution through three functions. (1) The *Observation Tool* reads $\mathcal{T}_k$ and $\mathcal{H}_k$, computes $\mathcal{A}_k$ (Equation (7)), and constructs a compressed tree snapshot for the Judge. (2) The *Orchestration Tool* converts $\mathcal{D}_k$ into the masked configuration in Equation (10), invokes the activated agents in pipeline order, and dispatches their stage-specific hints (Fig. 3). (3) Each activated agent combines the shared LLM $\mathcal{L}$ with a stage-specific prompt $\mathcal{P}_s$ and *PD Skill* $\mathcal{U}_s$.

3.4 Overall AgenticPD Algorithm

Algorithm 1 integrates history-aware branch navigation, intermediate-state reuse, and conditional downstream expansion.

Lines 1–4 initialize the search by running $\mathbf{a}_0$ through GR and recording the results in the tree $\mathcal{T}$ and history $\mathcal{H}$. Lines 6–14 describe one outer-loop iteration. The Observation Tool first constructs the adaptive profile $\mathcal{A}_k$ from the current tree and history (Line 6). The Judge then produces the branching decision $\mathcal{D}_k$ (Line 7). The system reuses $\text{Bef}(b_k)$ from node $\hat{n}_k$ (Line 8) and re-executes $\{b_k\} \cup \text{Aft}(b_k)$ (Lines 9–12): for each stage s , the harness builds $ctx_k(s)$, and the Stage Agent decides $a_k(s)$ and executes the stage. Lines 13–14 update the tree, history, and best-WNS incumbent. Line 15 applies detailed routing to the final incumbent and reports its post-DR QoR.

Algorithm 1: Stage-Aware Branch Search in AgenticPD

Input: Design D , total evaluation budget N , initial action tuple $\mathbf{a}_0$
Output: Best action tuple $\mathbf{a}^*$ and its post-DR QoR Q^{DR}

- 1 Run $\mathbf{a}_0$ through GR and observe $\{Q_0(s)\}$; // Equation (3)
- 2 Init tree $\mathcal{T}$ with results;
- 3 $(\mathbf{a}^*, Q^*) = (\mathbf{a}_0, Q_0(\text{GR}))$;
- 4 $\mathcal{H} = \emptyset$;
- 5 **for** $k = 1$ to $N - 1$ **do**
- 6 $\mathcal{A}_k = \text{OBSERVATIONTOOL}(\mathcal{T}, \mathcal{H})$; // adaptive profile from Equation (7)
- 7 $(\hat{n}_k, b_k, h_k) = \text{JUDGE}(\mathcal{H}, \mathcal{A}_k)$; // decision from Equation (8)
- 8 Reuse the state after $\text{Bef}(b_k)$ from node $\hat{n}_k$;
- 9 **for each** stage $s \in \{b_k\} \cup \text{Aft}(b_k)$ **do**
- 10 $ctx_k(s) = \text{BUILDCONTEXT}(s, \hat{n}_k, \text{hint}_s)$; // Equation (9)
- 11 $a_k(s) = \text{STAGEAGENT}_s(ctx_k(s))$; // Section 3.3
- 12 $Q_k(s) = \text{EXECUTESTAGE}(s, a_k(s))$; // Section 3.3
- 13 Update $\mathcal{T}$ and $\mathcal{H}$; // Equation (6b)
- 14 Update $(\mathbf{a}^*, Q^*)$ if $Q_k(\text{GR})$ has higher WNS; // Equation (4)
- 15 $Q^{\text{DR}} = \text{DETAILEDROUTE}(\mathbf{a}^*)$; report $(\mathbf{a}^*, Q^{\text{DR}})$;

Table 1: Benchmark designs evaluated on two process nodes.

Function	Design	Cells	Clk	Design	Cells	Clk
AES cipher	AES-128/SKY130HD	12,787	4.50 ns	AES-128/ASAP7	15,621	0.40 ns
RISC-V core	ibex/SKY130HD	15,945	10.00 ns	ibex/ASAP7	18,340	1.26 ns
JPEG encoder	JPEG/SKY130HD	38,216	5.50 ns	JPEG/ASAP7	54,354	0.90 ns

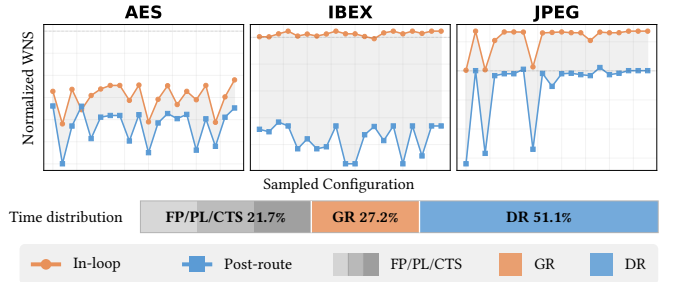


Figure 4: Consistency between in-loop and post-route WNS across sampled configurations, with runtime distribution by stage on ASAP7.

Tree exploration and incumbent selection are decoupled. The tree $\mathcal{T}$ retains every completed node for future branching, whereas the incumbent $\mathbf{a}_k^*$ records the highest-post-GR-WNS candidate observed through iteration k , following Equation (4). The Judge may therefore explore a temporarily worse branch without discarding previously discovered candidates. This workflow brings together the stage-specific expertise of the Stage Agents and the decision-making capabilities of the Judge Agent.

4 Experiments

4.1 Experimental Setup

Experiments run on Ubuntu 22.04 LTS with dual Intel Xeon Gold 6426Y processors and 256 GB RAM. We evaluate AES [19], ibex [22], and JPEG [20] on the SkyWater 130 nm HD library [10] and ASAP

Table 2: AgenticPD vs. Vanilla LLM on ASAP7. Bold = AgenticPD is the best.

Method	Design	Qwen3.5 [21]				Kimi-K2.5 [17]				DeepSeek-V3.2 [5]			
		WNS $\uparrow$	TNS $\uparrow$	Area $\downarrow$	Power $\downarrow$	WNS $\uparrow$	TNS $\uparrow$	Area $\downarrow$	Power $\downarrow$	WNS $\uparrow$	TNS $\uparrow$	Area $\downarrow$	Power $\downarrow$
Vanilla LLM	AES	-50.749	-2958.5	2.1	0.159	-39.768	-2085.8	2.1	0.159	-38.338	-2278.2	2.1	0.158
	ibex	-66.140	-3325.2	2.7	0.048	-66.140	-3325.2	2.7	0.048	-66.140	-3325.2	2.7	0.048
	JPEG	10.646	0	6.8	0.095	-5.006	-7.5	6.7	0.092	-5.006	-7.5	6.7	0.092
AgenticPD	AES	-35.457	-2654.4	2.0	0.157	-40.064	-2561.0	2.0	0.158	-32.271	-1811.3	2.0	0.157
	ibex	-99.135	-11271.5	2.6	0.046	-65.416	-1837.8	2.7	0.048	-53.867	-1098.7	2.7	0.048
	JPEG	55.915	0	6.7	0.092	56.711	0	6.7	0.092	50.569	0	6.7	0.092
Geomean impr.		Geomean gains: Performance 2.4%, Area 2.2%, Power 1.1%											

* WNS/TNS are in ps ($\uparrow$); area is in $10^3 \mu m^2$ and power in W ($\downarrow$). Percentage improvements are computed as one minus the geometric mean of per-case AgenticPD / Vanilla LLM ratios, expressed as percentages, with performance measured by effective clock period (ECP = $T_{clk} - WNS$) for consistent cross-design comparison as in [9].

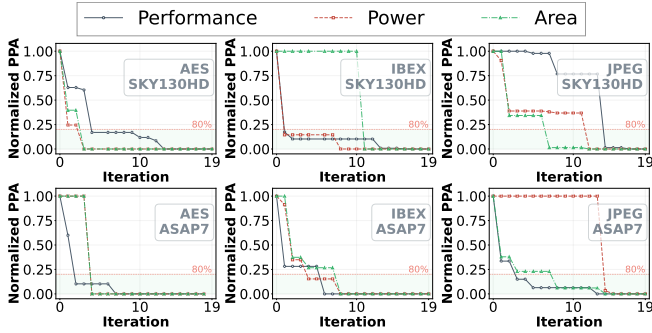


Figure 5: Normalized PPA convergence curves.

7 nm predictive PDK [4], using a 20-evaluation backend budget. TABLE 1 summarizes their post-synthesis sizes and target clock constraints. Post-GR WNS serves as the scalar incumbent-selection criterion, whereas complete post-route QoR is used for final evaluation. TNS reports aggregate timing violations, effective frequency is derived from the target clock period and post-route WNS, and area and power report non-timing physical quality. All optimization methods share the same budget. All methods use the same backend [1, 26], action space, and stage implementations.²

4.2 AgenticPD Mechanisms and Ablations

We examine AgenticPD from five complementary perspectives: in-loop QoR fidelity, robustness across LLM backbones, optimization dynamics, branching mechanisms, and the contribution of its Judge. **QoR fidelity.** AgenticPD ranks candidate configurations by the in-loop WNS signal, while headline comparisons report post-route QoR. To verify its signoff fidelity, we re-run detailed routing and signoff extraction for every configuration explored by AgenticPD on SKY130HD. In-loop and post-route WNS agree closely: Spearman $\rho = 0.86$ (AES), 0.83 (ibex), 0.93 (JPEG), and Pearson $r = 0.96 / 0.96 / 0.91$. On ASAP7, Fig. 4 shows corresponding trends across sampled configurations. Detailed routing takes a median of 38–75 minutes per configuration on identical hardware, so carrying a 20-candidate optimization through DR would add 13–25 hours per session. The in-loop signal removes this dominant cost from every iteration while preserving the ranking that guides optimization.

²Our public repository implements AgenticPD’s history-aware Judge orchestration, stage-specialized agents, tree-structured branch navigation, and reusable intermediate design states, together with experiment configurations, session inspection tools, and mock-mode verification instructions at <https://github.com/Flans/AgenticPD>.

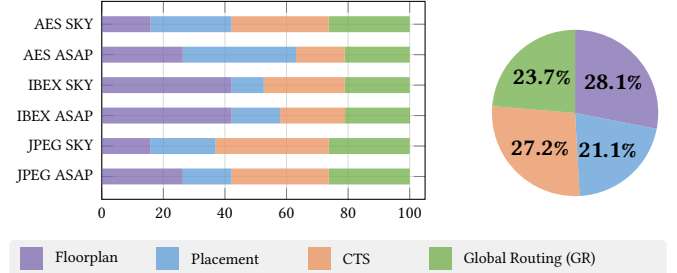


Figure 6: Distribution of branching stages across the six benchmark optimization runs. Left: per-benchmark breakdown. Right: aggregate distribution over all benchmarks.

Architecture robustness across LLM backbones. To evaluate whether the agentic architecture itself drives the improvement, we compare AgenticPD against a Vanilla LLM baseline, a flat single agent that shares the same agent-friendly environment integration as AgenticPD. It uses the same 20-evaluation budget and Docker backend, proposing one complete parameter configuration per iteration while observing the full evaluation history.

We compare the two systems on ASAP7 with DeepSeek-V3.2 [5], Qwen3.5 [21], and Kimi-K2.5 [17]. TABLE 2 reports post-route PPA on three ASAP7 benchmarks. AgenticPD wins on WNS in 7 of 9 pairs. The gains are largest on hard cases: on JPEG, AgenticPD achieves timing closure across all three backbones, while Vanilla LLM fails or barely closes. Winning pairs span all three backbones, so the benefit is not confined to one model. DeepSeek-V3.2 delivers the best overall timing, so we adopt it as the default when an experiment requires an LLM engine.

Optimization dynamics. With DeepSeek-V3.2, we analyze one optimization run on each of the six benchmarks.

Fig. 5 plots best-so-far ECP [9], power, and area normalized between their initial and best observed values. Across all benchmarks, these metrics realize at least 80% of their total improvement by iterations 8–14, indicating that useful configurations emerge well before the budget is exhausted. At $N=20$, the compressed Judge profile and stage-local contexts keep the peak prompt across all agents and benchmarks to 43K tokens, within the context windows of all evaluated LLMs [5, 17, 21].

Branching mechanisms. Fig. 6 shows the distribution of branching stages selected by the Judge across the six optimization runs analyzed above. The selections are balanced across FP (28.1%), CTS (27.2%), GR (23.7%), and PL (21.1%), showing that the Judge does not collapse to a single restart stage. Early-stage branches rebuild the downstream

Table 3: Post-route PPA comparison across six benchmarks. Bold = AgenticPD is the best.

Tech	Design	OpenROAD [1, 26]					AutoTuner [14]					ORFS-Agent [9]					AgenticPD (Ours)				
		WNS $\uparrow$	TNS $\uparrow$	Area $\downarrow$	Power $\downarrow$	Eff. Freq $\uparrow$	WNS $\uparrow$	TNS $\uparrow$	Area $\downarrow$	Power $\downarrow$	Eff. Freq $\uparrow$	WNS $\uparrow$	TNS $\uparrow$	Area $\downarrow$	Power $\downarrow$	Eff. Freq $\uparrow$	WNS $\uparrow$	TNS $\uparrow$	Area $\downarrow$	Power $\downarrow$	Eff. Freq $\uparrow$
SKY130HD	AES	-0.321	-2.2	125.2	0.424	207.4	0.133	0	123.7	0.424	229.0	0.120	0	126.8	0.432	228.3	0.271	0	127.8	0.437	236.5
	ibex	-0.664	-23.8	181.0	0.110	93.8	-0.719	-56.9	183.3	0.112	93.3	-0.581	-22.2	181.6	0.110	94.5	-0.503	-19.3	179.4	0.107	95.2
	JPEG	-0.218	-2.8	516.3	0.542	174.9	-0.207	-4.0	511.1	0.546	175.2	-0.202	-3.6	519.6	0.549	175.4	-0.154	-1.6	517.7	0.531	176.9
ASAP7	AES	-37.614	-2395.2	2.1	0.157	2.285	-43.399	-2392.4	2.0	0.157	2.255	-37.444	-2566.4	2.1	0.158	2.286	-32.271	-1811.3	2.0	0.157	2.313
	ibex	-62.655	-2740.9	2.7	0.047	0.756	-61.852	-3145.1	2.7	0.048	0.757	-63.669	-1057.6	2.7	0.047	0.755	-53.867	-1098.7	2.7	0.048	0.761
	JPEG	-8.229	-37.4	6.7	0.092	1.101	-9.996	-25.9	6.7	0.092	1.099	0.337	0	6.7	0.092	1.112	50.569	0	6.7	0.092	1.177

* WNS and TNS are reported in picoseconds for ASAP7 and nanoseconds for SKY130HD, where higher (less negative) is better ($\uparrow$). Area is in $10^3 \mu m^2$ and power in watts, where lower is better ($\downarrow$). Eff. Freq is the effective frequency $1/(T_{clk} - WNS)$ [16], in MHz for SKY130HD and GHz for ASAP7.

Table 4: Run-to-run stability with DeepSeek-V3.2.

Tech	Design	Default	Run 1	Run 2	Run 3	Mean $\pm$ Std
SKY130HD	AES	-0.526	+0.234	+0.144	+0.232	+0.203 $\pm$ 0.051
	ibex	-0.714	-0.516	-0.470	-0.501	-0.496 $\pm$ 0.024
	JPEG	-0.386	-0.363	-0.386	-0.386	-0.378 $\pm$ 0.013
ASAP7	AES	-35.31	-23.31	-23.58	-26.77	-24.55 $\pm$ 1.92
	ibex	+1.33	+4.61	+4.61	+4.61	+4.61 $\pm$ 0.00
	JPEG	+52.21	+86.35	+88.45	+88.45	+87.75 $\pm$ 1.21

Table 5: Judge ablation on ASAP7.

Design	Judge	WNS (ps) $\uparrow$	TNS (ps) $\uparrow$	Area $\downarrow$	Power $\downarrow$
AES	Heuristic	-35.20 $\pm$ 0.02	-1992 $\pm$ 140	2.031 $\pm$ 0.001	0.154 $\pm$ 0.000
	AgenticPD	-24.55 $\pm$ 1.92	-1675 $\pm$ 212	2.044 $\pm$ 0.013	0.155 $\pm$ 0.001
JPEG	Heuristic	+52.21 $\pm$ 0.00	0 $\pm$ 0	6.729 $\pm$ 0.000	0.092 $\pm$ 0.000
	AgenticPD	+87.75 $\pm$ 1.21	0 $\pm$ 0	6.733 $\pm$ 0.020	0.092 $\pm$ 0.001

flow, whereas later branches preserve proven upstream states and target localized timing or routing bottlenecks.

Combining the branch distribution in Fig. 6 with the stage runtime profile in Fig. 4, about 72% of decisions select $b_k \in \{PL, CTS, GR\}$, reuse the cached stages in $Bef(b_k)$, and execute only $\{b_k\} \cup Aft(b_k)$. These avoided upstream executions indicate backend-time savings over full-flow baselines without additional EDA evaluations.

Stability and Judge ablation. Because the backend produces deterministic QoR for a given configuration (Section 4.1), run-to-run variation arises only from the LLM-guided optimization at temperature 0.6. TABLE 4 reports the best in-loop WNS over three independent runs; metric units follow TABLE 3.

AgenticPD achieves better mean in-loop WNS than the default across all six design–technology pairs, while maintaining low run-to-run variation, with standard deviations of only 0.051 ns and 1.92 ps for AES on SKY130HD and ASAP7, respectively.

To isolate the Judge’s contribution, we replace it with a deterministic max-bottleneck heuristic that branches from the best node and restarts at the stage with the largest upstream-to-downstream WNS degradation. The Stage Agents, prompts, action space, and protocol remain unchanged. As shown in TABLE 5, across three runs, the AgenticPD Judge outperforms the heuristic by 10.6 ps WNS on AES and 35.5 ps on JPEG (where the heuristic never escapes the default configuration), without material tradeoffs in the other PPA metrics: area and power differ by less than 1% between the two judges.

This demonstrates the benefit of history-aware branch selection over greedy local bottleneck repair. Together, the proxy study establishes feedback fidelity, the branch analysis quantifies stage-aware execution, and the Judge ablation isolates the benefit of history-aware navigation. Having established the reliability and signoff fidelity of the in-loop optimization process, all subsequent comparisons report complete post-route QoR.

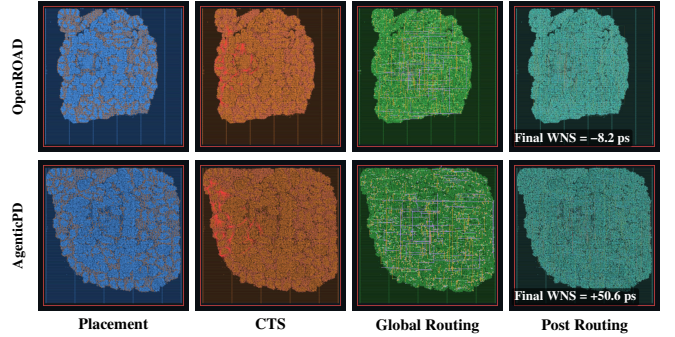


Figure 7: Layout comparison for the JPEG design on ASAP7. AgenticPD improves post-route WNS from -8.2 ps to +50.6 ps.

4.3 AgenticPD vs. Previous Tuners

We next compare AgenticPD against previous state-of-the-art tuners that can be applied to PD optimization problems: AutoTuner [14] as the representative black-box tuner and ORFS-Agent [9] as a representative LLM-based tuner. Both AgenticPD and ORFS-Agent use DeepSeek-V3.2 as the LLM engine. The OpenROAD baseline uses OpenROAD [1] with the optimization flow provided by OpenROAD-flow-scripts (ORFS) [26].

As shown in TABLE 3, AgenticPD achieves the best WNS on all six benchmarks. On JPEG ASAP7, AgenticPD achieves timing closure (WNS = +50.6 ps) where AutoTuner fails (WNS = -10.0 ps) and ORFS-Agent is marginal (WNS = +0.3 ps). On AES ASAP7, the hardest timing case, AgenticPD improves WNS by 11.1 ps over AutoTuner and 5.2 ps over ORFS-Agent. AgenticPD does not trade physical quality for timing: where it wins on WNS, it remains competitive or better in area and power. AutoTuner [14] and ORFS-Agent [9] both treat the parameter space as flat Θ_{PD} and restart the full flow every trial, unable to diagnose which stage causes a regression or preserve upstream progress. AgenticPD instead diagnoses the bottleneck stage and lets each Stage Agent reason within its own Θ_s with stage-level feedback. This is not LLM prompt engineering applied to tuning; it is a purpose-built agentic system whose architecture encodes the structure of the PD optimization problem.

4.4 Case Study: JPEG on ASAP7

Fig. 7 compares the OpenROAD layout with the AgenticPD result on JPEG (ASAP7). To illustrate how the multi-agent system reaches this result, we trace its optimization trajectory.

The Judge first directs exploration toward high core utilization: the FP Agent sets CU = 45 and the CTS Agent selects an aggressive clock-tree configuration (CTS_CLUSTER_SIZE = 60, SLACK_MARGIN = 0.15, TNS% = 10), establishing a strong floorplan-placement foundation. A

later floorplan branch lowers utilization to CU = 44, but this rebuilds the entire downstream pipeline and regresses timing—revealing that even a 1% utilization change can disrupt a well-optimized placement–CTS configuration. The Judge reflects on this regression and shifts strategy: instead of re-exploring floorplan, it branches from the earlier node at the CTS stage, preserving the proven FP and PL stages, and the CTS Agent retunes to a balanced setting (CTS_CLUSTER_SIZE = 45, SLACK_MARGIN = 0.13, TNS% = 35), reaching the selected design whose reported post-route WNS is +50.6 ps without re-running any upstream stage. This trajectory instantiates the stage-aware closed loop: the measured regression changes the next branch point, while the Stage Agents retune parameters within their scopes.

5 Conclusion

In this paper, we presented AgenticPD, a stage-aware agentic framework that closes the loop between measured backend QoR and subsequent branch decisions through a reusable optimization tree. Across vanilla LLM baselines and prior state-of-the-art tuners, AgenticPD demonstrates robust timing performance across all three LLM backbones and achieves the best timing on all six benchmarks against the latter, while remaining competitive in power and area. State reuse avoids redundant execution. Future work will advance AgenticPD toward Agentic EDA systems that accelerate LLM-driven chip design through closed-loop agent–tool collaboration.

Acknowledgment

This research was partially conducted by ACCESS-AI Chip Center for Emerging Smart Systems, supported by the InnoHK initiative of the Innovation and Technology Commission of the Hong Kong Special Administrative Region Government. This work is jointly supported by the Research Grants Council of Hong Kong SAR (Grant No. T46-415/25-R) and the Key Research and Development Program of Jiangsu Province (Grant No. BE2023003).

References

- [1] Tutu Ajayi, Vidya A Chhabria, Mateus Fogaça, Soheil Hashemi, Abdelrahman Hosny, Andrew B Kahng, Minsoo Kim, Jeongsup Lee, Uday Mallappa, Marina Neseem, et al. 2019. Toward an open-source digital flow: First learnings from the OpenROAD project. In *Proceedings of the Design Automation Conference (DAC)*. 1–4.
- [2] Anthropic. 2024. Building Effective Agents. Anthropic Research Blog. <https://www.anthropic.com/research/building-effective-agents>
- [3] Cadence Design Systems. 2025. Cadence Welcomes ChipStack. https://community.cadence.com/cadence_blogs_8/b/corporate-news/posts/cadence-welcomes-chipstack.
- [4] Lawrence T Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandrasekaran Ramamurthy, and Greg Yeric. 2016. ASAP7: A 7-nm finFET predictive process design kit. *Microelectronics Journal* 53 (2016), 105–115.
- [5] DeepSeek. 2025. *DeepSeek-V3.2 Release*. <https://api-docs.deepseek.com/news/news251201>
- [6] Rongliang Fu, Yi Liu, Qiang Xu, and Tsung-Yi Ho. 2026. MappingEvolve: LLM-Driven Code Evolution for Technology Mapping. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*. 1–7. doi:10.1145/3770743.3803988
- [7] Hao Geng, Tinghuan Chen, Yuzhe Ma, Binwu Zhu, and Bei Yu. 2022. PTP: Physical design tool parameter tuning via multi-objective Bayesian optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 1 (2022), 178–189.
- [8] Hao Geng, Qi Xu, Tsung-Yi Ho, and Bei Yu. 2022. PPA-Tuner: Pareto-driven tool parameter auto-tuning in physical design via Gaussian process transfer learning. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*. 1237–1242.
- [9] Amur Ghose, Andrew B Kahng, Sayak Kundu, and Zhiang Wang. 2025. ORFS-Agent: Tool-using agents for chip design optimization. In *Proceedings of the ACM/IEEE Symposium on Machine Learning for CAD (MLCAD)*. IEEE, 1–13.
- [10] Google and SkyWater Technology. 2020. SkyWater SKY130 Open Source PDK. <https://github.com/google/skywater-pdk>. Accessed: 2026-03-20.
- [11] Yaohui Han, Rongliang Fu, Yanming Liu, Shuo Ren, Shuai Dong, Yunpeng Wang, Tinghuan Chen, Bei Yu, and Tsung-Yi Ho. 2026. Expert-level Leaf Cell Layout Generation via Preference-Optimized LLM. In *Proceedings of the International Conference on Machine Learning (ICML)*. <https://openreview.net/forum?id=z834t47Lr4>
- [12] Hao-Hsiang Hsiao, Yi-Chen Lu, Pruek Vanna-Iampikul, and Sung Kyu Lim. 2024. FastTuner: Transferable physical design parameter optimization using fast reinforcement learning. In *Proceedings of the International Symposium on Physical Design (ISPD)*. 93–101.
- [13] Hao-Hsiang Hsiao, Pruek Vanna-Iampikul, Yi-Chen Lu, and Sung Kyu Lim. 2024. ML-based Physical Design Parameter Optimization for 3D ICs: From Parameter Selection to Optimization. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*. Article 252, 6 pages. doi:10.1145/3649329.3656509
- [14] Jinwook Jung, Andrew B Kahng, Seungwon Kim, and Ravi Varadarajan. 2021. METRICS2.1 and flow tuning in the IEEE CEDA robust design flow and OpenROAD ICCAD special session paper. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–9.
- [15] Andrew B Kahng, Jens Lienig, Igor L Markov, and Jin Hu. 2011. *VLSI physical design: from graph partitioning to timing closure*. Vol. 312. Springer.
- [16] Yi-Chen Lu, Sai Surya Kiran Pentapati, Lingjun Zhu, Kambiz Samadi, and Sung Kyu Lim. 2020. TP-GNN: A graph neural network framework for tier partitioning in monolithic 3D ICs. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6.
- [17] Moonshot AI. 2026. *Kimi-K2.5 Model by Moonshotai | NVIDIA NIM*. <https://build.nvidia.com/moonshotai/kimi-k2.5/modelcard>
- [18] OpenAI. 2026. *ChatGPT*. <https://openai.com/research>
- [19] OpenCores. 2002. AES (Rijndael) IP Core. https://opencores.org/projects/aes_core. Accessed: 2026-03-20.
- [20] OpenCores. 2002. JPEG Encoder, Video Compression Systems Project. https://opencores.org/projects/video_systems. Accessed: 2026-03-20.
- [21] Qwen Team. 2026. *Qwen3.5: Towards Native Multimodal Agents*. <https://qwen.ai/blog?id=qwen3.5>
- [22] Pasquale Davide Schiavone, Francesco Conti, Davide Rossi, Michael Gautschi, Antonio Pullini, Eric Flamand, and Luca Benini. 2017. Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications. In *Proceedings of the International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. IEEE, 1–8.
- [23] Utsav Sharma, Bing-Yue Wu, Sai Rahul Dhanvi Kankipati, Vidya A Chhabria, and Austin Rovinski. 2024. OpenROAD-Assistant: An open-source large language model for physical design tasks. In *Proceedings of the ACM/IEEE International Symposium on Machine Learning for CAD (MLCAD)*. 1–7.
- [24] Siemens Digital Industries Software. 2026. Siemens launches Fuse EDA AI Agent. <https://news.siemens.com/en-us/siemens-fuse-eda-ai-agent/>.
- [25] Synopsys. 2026. Agentic AI: Automating Engineering Workflows with AI Agents. <https://www.synopsys.com/ai/agentic-ai.html>.
- [26] The OpenROAD Project. 2024. OpenROAD-flow-scripts: Autonomous Digital Design Flow. <https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts>. Accessed: 2026-03-20.
- [27] Bing-Yue Wu, Utsav Sharma, Austin Rovinski, and Vidya A Chhabria. 2025. OpenROAD Agent: An Intelligent Self-Correcting Script Generator for OpenROAD. In *Proceedings of the IEEE International Conference on LLM-Aided Design (ICLAD)*. IEEE, 16–22.
- [28] Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. 2024. ChatEDA: A large language model powered autonomous agent for EDA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 10 (2024), 3184–3197.
- [29] Su Zheng, Hao Geng, Chen Bai, Bei Yu, and Martin DF Wong. 2023. Boosting VLSI design flow parameter tuning with random embedding and multi-objective trust-region Bayesian optimization. *ACM Transactions on Design Automation of Electronic Systems* 28, 5 (2023), 1–23.