

LATCH: LLM-Assisted Timing Closure Harness for Post-Route FPGA Optimisation

Shu Gu[✉], Matthew Youngman[✉], *Graduate Student Member, IEEE*, Yuqin Zhao[✉],
Cristian Sestito[✉], *Member, IEEE*, Ali Siddique[✉], Ramesh Krishnamurthy[✉], Danial Chitnis[✉],
and Themis Prodromakis[✉], *Senior Member, IEEE*

Abstract—Post-route FPGA timing closure remains expert-dependent and computationally expensive. We present LATCH (LLM-Assisted Timing Closure Harness), a two-layer framework that separates large language model (LLM) orchestration from deterministic FPGA implementation and evaluation. Across sixteen designs and seven LLMs, a single-iteration study compares three agent control granularities – expert-written recipes, direct EDA operations and LLM-constructed procedures – to ask how much control an agent needs per attempt. Functional-pass rates are 98.2%, 71.4% and 17.9%, respectively; yet direct operations give slightly larger 3.4% median $F_{\max}$ gains when passing, versus 3.2% for recipes. This exposes a trade-off between execution reliability and timing improvement. Informed by this trade-off, a multi-iteration study fixes the action tier at expert recipes to ask how repeated attempts are best organised, comparing a fitted deterministic heuristic with sequential LLM reasoning, branching exploration and knowledge base-guided exploration within 12 checkpoints per run. Branching raises mean maximum-frequency improvement from 10.5% (sequential) to 16.0%, and the knowledge base to 20.3%, improving 15 of 16 designs. Characterisation across 2,421 recipe attempts profiled their gain and validation rates, showing hard-block re-placement gives the largest cumulative mean gain at 41.4 MHz.

Index Terms—Electronic design automation (EDA), FPGA, large language model (LLM), LLM agents, timing closure.

I. INTRODUCTION

Timing closure seeks to meet a design’s target frequency by diagnosing failing paths and modifying placement and routing (P&R), physical optimisation, or constraints to improve $F_{\max}$. This remains a critical but expert-dependent stage because the appropriate intervention is determined by design-specific physical bottlenecks [1] and their interactions across many available optimisation options [2]. Each trial can require a full P&R run, with large circuits taking hours or even days [2], [3]. An autonomous timing-optimisation method could reduce this cost by selecting interventions from physical evidence.

Machine learning (ML) has been applied across the EDA flow to improve engineering productivity, either by predicting

physical outcomes earlier in the flow [4] or by selecting tool settings more intelligently [5]. Within FPGA timing optimisation, predictive models estimate post-route net delay during placement [6], calibrate delay inside differentiable timing-driven placement [7], and weight global placement by predicted criticality [8]. By acting inside the placement algorithm, these methods improve the next implementation but cannot choose a local repair for an existing routed checkpoint. Autotuners instead learn which tool settings or optimisation sequences to try: InTime [9] and LAMDA [10] rank tool-parameter combinations, DATuner [2] allocates trials across parameter-space partitions, and FlowTune builds optimisation sequences stage by stage, favouring sub-sequences with better measured results [3]. These autotuners remove manual tuning, but every candidate costs at least a partial implementation rerun and each model is learned within a single design’s campaign. As a result, these methods remain bound to costly iterations and generalise poorly to unseen designs. Where a learned placement policy has reportedly generalised, it relied on pre-training over earlier designs [11], with cross-design generalisation as an open research problem [5].

Large language models (LLMs) target these learning and generalisation limitations because they can reason over textual and structured evidence [12], compare it with tool documentation or prior examples [13], [14], and select actions from tool feedback without fitting a new selector for each design [15]. This is attractive for timing optimisation, where useful evidence appears in timing reports, logs, constraint files and optimisation histories [13]. Physical implementation nevertheless exposes a hardware-specific failure mode termed the *syntax trap*, where LLM outputs can appear plausible as text, scripts or RTL code while remaining physically ineffective, unsafe or disconnected from downstream implementation behaviour [16]. This trap is severe in backend design because public physical-design data is scarce, locked behind licenses, and expensive to generate [17], [18]. Physical-design benchmarks confirm the gap, as agents degrade on geometric reasoning and power, performance and area (PPA) trade-offs [19], and on tool-centric, long-horizon workflows [20].

Existing LLM systems therefore often work around the implemented state rather than improving it directly, generating scripts or templated implementation flows [25], re-running flow parameters or constraints [22], [21], or rewriting RTL through optimisers [26] or end-to-end design agents [27]. Recent ASIC systems move closer to the physical-design problem by editing P&R designs directly through engineering

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) AI Hub for Productive Research and Innovation in eElectronics (APRIL) under Grant No. EP/Y029763/1, and by the Royal Academy of Engineering (RAEng) Chair in Emerging Technologies under Grant No. CiET1819/2/93. (S. Gu and M. Youngman contributed equally to this work. Corresponding author: Matthew Youngman).

S. Gu, M. Youngman, Y. Zhao, C. Sestito, A. Siddique, R. Krishnamurthy and T. Prodromakis are with the Centre for Electronics Frontiers, Institute for Integrated Micro and Nano Systems, School of Engineering, The University of Edinburgh, U.K. (email: m.j.youngman@sms.ed.ac.uk).

D. Chitnis is with the Institute for Integrated Micro and Nano Systems, School of Engineering, The University of Edinburgh, U.K.

TABLE I
SELECTED LEARNING-BASED APPROACHES TO BACKEND TIMING AND PHYSICAL OPTIMISATION. GROUPS, FROM TOP: ML-BASED, LLM-BASED, AND THIS WORK (SHADED). ENTRIES REFER TO THE REPORTED EXPERIMENTS.

Work	Target / Input	Designs	Timing-feedback loop	Physical modification	Knowledge reuse	Search	Control-level study
InTime (2015) [9]	FPGA / RTL	6	✓	✓ ^F	✓	Parallel batches	NR
DATuner (2017) [2]	FPGA / RTL	15 (10+5)	✓	✓ ^F	✓	Adaptive batches	NR
LAMDA (2019) [10]	FPGA / RTL	5	✓ ^E	✓ ^{F,E}	✓	Multi-stage batches	NR
Nexus (2025) [21]	FPGA / RTL + constraints	14	✓	✓ ^F	NR	Sequential refinement	NR
ORFS-agent (2025) [22]	ASIC / RTL + PDK	3 (2 nodes)	✓	✓ ^F	✓	Parallel batches	NR
FluxEDA (2026) [23]	ASIC / routed design	1 (ECO case)	✓	✓ ^D	✓	Sequential + rollback	NR
PACT (2026) [24]	FPGA / routed DCP	35 DCPs	✓	✓ ^D	✓	Ranked sequential	NR
Single-iteration study	FPGA / routed DCP	16	×	✓ ^D	×	Single submission	✓
Multi-iteration study	FPGA / routed DCP	16	✓	✓ ^D	✓ ^K	Sequential / branching	×

✓: explicitly supported; ×: excluded by the stated study; NR: not reported in the reviewed version. Timing feedback: candidate implementation measurements guide further optimisation. Physical modification: F, flow re-execution; D, edits to an existing physical design; E, LAMDA’s search evaluation replays pre-collected tool measurements. Knowledge reuse: supplementary procedural knowledge or experience across trials, not merely an action catalogue; K, the KB-enabled multi-iteration variant. Control-level study: a common-backend comparison of recipe selection, direct operations and constructed procedures; NR does not exclude other ablations. Designs count backend cases, not models or trials: ORFS evaluates three circuits at two nodes, FluxEDA’s entry covers its ECO case and PACT counts DCP instances; both studies here use the same 16 designs.

change orders (ECOs). RSR schedules ECO sizing, threshold-voltage and buffering commands, guided by retrieved expert scheduling strategies and result feedback [14], while FluxEDA keeps commercial tool sessions alive so that an agent can apply such edits and roll back rejected steps [23]. Concurrent with this work, PACT demonstrates LLM-guided ECO tuning of routed FPGA checkpoints by ranking typed edits, applying them sequentially and retaining only valid timing-improving results [24]. PACT thereby establishes routed FPGA checkpoint editing as a strong application of LLMs to backend EDA. Yet how LLMs fit into the backend flow remains an open question: how much action freedom can an agent use reliably, how efficiently can it navigate the timing-optimisation search space, and which aspects of its harness best supply backend design expertise. Table I summarises how prior ML- and LLM-based systems address these dimensions.

This paper therefore presents LATCH (LLM-Assisted Timing Closure Harness), a two-layer framework that improves post-route FPGA $F_{\max}$ to investigate how LLMs behave in backend EDA. The deterministic execution layer characterises each checkpoint, applies physical modifications and validates the result. The LLM orchestration layer interprets backend evidence, requests additional analysis when needed and ranks the next action to try. This separation lets the paper study LLM decision-making whilst keeping implementation, validation and measurement deterministic. Contributions are as follows:

- 1) We introduce LATCH, a reproducible framework for LLM-orchestrated post-route FPGA $F_{\max}$ improvement. Its protocol-mediated interface isolates LLM reasoning from deterministic EDA execution, providing an extensible framework: <https://github.com/aprilaihub/LATCH>
- 2) We contribute nine engineer-written expert optimisation recipes encoding physical-design expertise as executable actions. Each targets a known timing failure mode and is profiled by measured value and validation risk.
- 3) We characterise the trade-off between FPGA backend action freedom, execution reliability and timing improvement in a single-iteration study that compares three

granularities of agent control. We then fix the most reliable recipe tier in a multi-iteration ablation study to explore design search mechanism effectiveness by comparing branching exploration and knowledge base guidance with sequential reasoning and a fitted heuristic.

- 4) We evaluate the framework on sixteen post-route FPGA designs, using seven LLMs in the single-iteration study, three LLMs in the multi-iteration ablation study, structural legality checks on every candidate, and simulation-based functional checking on the published result.

The single-iteration study identifies a trade-off between action reliability and timing gain: recipes are more reliable, while passing direct operations achieve a slightly higher median improvement. The multi-iteration study therefore fixes the recipe interface to examine how search and knowledge reuse affect optimisation. Branching and knowledge base guidance improve mean gains over sequential reasoning, while comparisons with the fitted heuristic reveal differences in generalisation to held-out designs. The remainder of the paper is organised as follows: Section II describes the LATCH framework and expert recipes, Section III presents the benchmarks and both studies, Section IV discusses limitations and how each can be resolved, and Section V concludes.

II. METHODOLOGY

LATCH guides the selection of timing-optimisation procedures for an implemented FPGA design using two vendor tools. AMD Vivado 2025.1 targets FPGA implementation, physical optimisation and timing analysis, and RapidWright 2025.2.1-beta reads and writes the post-P&R netlist directly to reach transformations Vivado does not expose [28]. Both act on a design checkpoint (DCP), which stores the FPGA implemented netlist together with its P&R. The framework is organised in two layers (Fig. 1): the *deterministic execution backend* performs analysis, modification and measurement, and the *orchestration layer* decides which modification to attempt on a selected DCP, using an agent that reasons over the evidence the backend supplies. The two layers run as servers communicating through the Model Context Protocol (MCP).

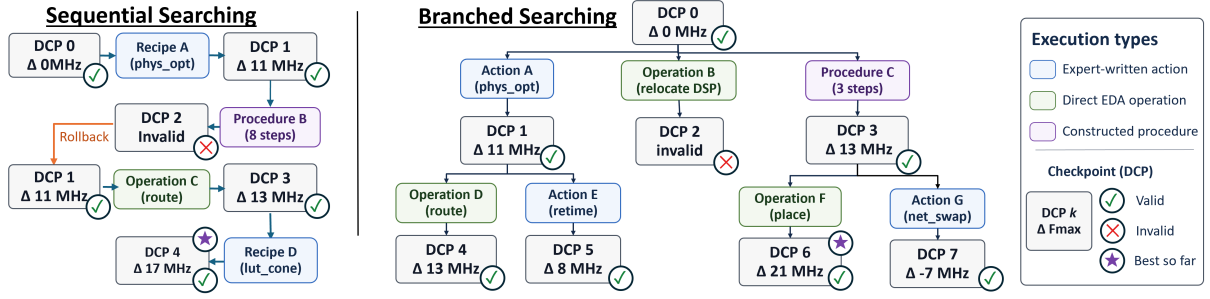


Fig. 2. Illustrated example of sequential and branching optimisation patterns with retained checkpoints. An invalid attempt leaves its parent available, and branching allows alternative changes to start from the same checkpoint. The run history records applied changes and measured outcomes. Numerical values illustrate the workflow rather than experimental results.

TABLE II
EXPERT OPTIMISATION RECIPES. ALL USE VIVADO; † ADDITIONALLY USES RAPIDWRIGHT FOR NETLIST MODIFICATION.

Recipe	Function
fanout†	Splits timing-critical high-fanout nets by a factor of two to eight to reduce loading and routing delay.
auto_pblock	Defines a region from critical-path spread and fabric resources, then unplaces and re-implements the design inside.
net_swap	Targets problematic critical connections to reduce routing-related delay.
phys_opt	Runs Vivado physical optimisation after placement and routing, including replication, cell resizing and register movement.
lut_cone†	Merges LUT-dominated critical logic cones to reduce logic depth and path delay.
retime	Moves registers across combinational logic to reduce delay between sequential stages.
memory_path_opt	Optimises critical paths involving embedded BRAMs using memory-aware physical optimisation.
hardblock_relocate	Improves physical locality between timing-critical hard blocks and the surrounding logic.
congestion_spread	Redistributes logic in congested regions to reduce routing pressure.

the number of checkpoints a run may produce, is fixed in advance, as are the depth, expansion and concurrency of branching. Every proposal, action and measured outcome is written to the run record, so a run can be inspected, resumed or reproduced, and the record itself becomes evidence.

B. The Deterministic Execution Backend

The deterministic backend provides the implementation and evaluation interface between optimisation decisions and the physical FPGA design. The agent may propose backend operations, but analysis probing, execution, checkpoint generation and timing evaluation remain backend-controlled, so optimisation performance is determined by the commands that change the DCP's physical implementation. This section describes how the backend extracts analysis evidence and executes the expert-written recipes, the one action tier the framework authors in advance, each a fixed procedure embedding an engineer's expertise. The finer tiers (Section II-A2) have no such fixed procedure, because the agent generates them at run time from the same EDA tool operations.

1) *Design analysis and diagnostic extraction*: Before optimisation, the backend performs a deterministic analysis of the input implementation: Vivado opens the input checkpoint, parses its timing summary to extract the worst negative slack (WNS), total negative slack (TNS), and number of failing endpoints, and queries the clock period to determine the target clock frequency and the achievable operating frequency.

For a clock period T_{clk} and measured WNS, the backend estimates the achievable frequency as

$$F_{\text{max}} = \frac{1000}{T_{\text{clk}} - \text{WNS}} \text{ MHz}, \quad (1)$$

where timing violations correspond to $\text{WNS} < 0$.

Structural and physical diagnostics complement the timing report. Vivado identifies critical high-fanout nets and extracts the cells and pins lying on the most critical paths, which RapidWright uses to measure the physical spread of critical-path cells (maximum and average cell distances) and the routing detour of critical connections, identifying those whose routed length considerably exceeds their direct physical separation. High spread indicates excessive interconnect distance and motivates region-constrained placement, whereas large detours identify candidates for cell re-placement. The resulting diagnostic set $\mathcal{D}$ comprises WNS, TNS, F_{max} , the number of failing endpoints N_{fail} , critical high-fanout nets $\mathcal{N}_{\text{fanout}}$, critical-path information $\mathcal{P}_{\text{crit}}$ and physical placement characteristics $\mathcal{S}_{\text{place}}$.

2) *Expert optimisation recipes*: The backend exposes a catalogue of expert optimisation recipes, each with a fixed sequence of examination, modification, implementation and evaluation operations directed at one class of timing bottleneck; Table II lists the nine forming the *expert-written recipe* action tier used throughout this study. Because a recipe's internal command sequence is supplied by the framework, selecting it commits the agent to a known implementation procedure rather than an arbitrary tool invocation.

The catalogue is organised around the physical causes of critical-path delay identified by the deterministic and deep analysis. Placement-oriented recipes act on where logic sits: `auto_pblock` re-implements the design within a region derived from critical-path spread, suiting designs whose critical logic is dispersed across the fabric, while `hardblock_relocate` improves locality between timing-

critical hard blocks and the surrounding soft logic. Because embedded block random access memories (BRAMs) and digital signal processing (DSP) columns are fixed in the fabric, `hardblock_relocate` acts on the placement and registering of the logic around them, which bounds its yield on memory-dense designs. Routing-oriented recipes act on how logic is connected (`net_swap`, `congestion_spread` and `fanout`), with the `fanout` split factor bounded between two and eight to avoid excessive replication. Logic and sequential recipes act on the structure of the path itself: `lut_cone` merges lookup table (LUT)-dominated critical cones to reduce logic depth, and `retime` moves registers across combinational logic to balance delay between sequential stages. Finally, `memory_path_opt` targets paths through BRAMs, and `phys_opt` applies Vivado’s general physical optimisation after placement and after routing.

The recipes also differ in where they execute, which matters more for reliability than the ambition of the transformation. Seven perform their modification inside Vivado and inherit its incremental optimisation semantics, under which a transformation is retained only where the tool measures an improvement. The remaining two, `fanout` and `lut_cone`, modify the netlist in RapidWright and write an intermediate checkpoint that Vivado reopens and reroutes, gaining direct access to placement and connectivity that Vivado does not expose but leaving the vendor tool’s gated loop: the returned design is checked for legality when reopened, but no intermediate improvement guarantee applies to the modification itself. Section III-C shows that validation risk concentrates almost entirely in these two recipes.

3) *Timing verification and quantitative feedback*: Every optimisation is evaluated using implementation-level timing data: the backend records the initial WNS and clock period before modification, re-evaluates timing after the candidate has been routed, converts the result to an achievable $F_{\max}$ using (1), and compares it with the baseline. For an initial implementation D_0 and an optimised implementation D_i , the changes $\Delta WNS = WNS(D_i) - WNS(D_0)$ and $\Delta F_{\max} = F_{\max}(D_i) - F_{\max}(D_0)$ are positive for an improvement. The backend tracks the best observed WNS across a run, retains TNS and the failing-endpoint count from the initial report, and inspects route status after each physical modification so that routing errors are identified before a timing result is accepted. The objective continues to reward improvement after timing is met. $F_{\max}$ in (1) rises with positive WNS, and no run terminates on reaching $WNS \geq 0$.

C. The LLM Agent Orchestration Layer

At each step, the agent occupying the pipeline’s decision stage is given the current design state and the actions available at the active control level of Section II-A2. What the agent returns therefore depends on this tool abstraction level: at the recipe interface it returns a ranked choice over the optimisation recipe catalogue, whereas at the direct-operation and constructed-procedure interfaces it submits an executable proposal, which it may revise in response to construction feedback before committing.

Prompt:	
You are a bounded FPGA implementation analysis advisor for timing optimisation search... You may only request one of the provided read-only deep analysis tools, or stop with your selected executable.	
Constraints:	
1. Do not treat generic FPGA knowledge as current-design evidence.	
2. Assign each candidate a distinct <code>priority_score</code> reflecting your actual preference order; do not return equal scores.	
...	
Design State: <code>DCP_node_008_retime</code> , depth 4 of trajectory <code>lut_cone</code> -> <code>phys_opt</code> -> <code>retime</code> . WNS -0.299 ns, fmax 434.97 MHz, ...	
Diagnosis: route_delay_ratio 0.939; dominant_path_type routing_dominated; max_logic_levels 1; 680 DSP; ...	
...	
Available recipes: <code>auto_pblock</code> , <code>congestion_spread</code> , <code>fanout</code> , <code>hardblock_relocate</code> , <code>lut_cone</code> , ...	
Available operations: <code>optimize_fanout</code> (<code>net_name</code> , <code>split_factor</code>); <code>optimize_cell_placement</code> (<code>cell_names</code> , <code>max_candidates</code>); ...	
Available procedures: <code>run_tcl</code> (<code>command</code>); <code>write_checkpoint</code> (<code>dcp_path</code> , <code>force</code>), ...	
...	
LLM Response:	
Stop: Sufficient current-node evidence (<code>node_summary</code> + <code>timing_summary</code>) combined with dense trajectory history ...	
Observation: <code>phys_opt</code> has the strongest observed deltas across all sibling branches (<code>clean</code> =3, <code>best_delta_fmax</code> =7.24MHz) ...	
Prior: Physical optimization after retiming often recovers additional slack by re-legalizing placement around moved registers.	
Uncertainty: No detailed critical-path breakdown retrieved; no congestion map or fanout report pulled for this node...	
Candidate priorities:	
<code>phys_opt</code>	0.82 risk 0.15 conf 0.70 trajectory_observation
<code>lut_cone</code>	0.55 risk 0.40 conf 0.40 trajectory_observation
<code>netswap</code>	0.35 risk 0.10 conf 0.55 trajectory_observation
<code>fanout</code>	0.30 risk 0.50 conf 0.30 heuristic_prior
<code>auto_pblock</code>	0.20 risk 0.50 conf 0.25 heuristic_prior
Selected tool: <code>phys_opt</code>	

Fig. 3. Analysis prompt for a recipe-selection decision. The design state, diagnosis, and available agent control interfaces – which change depending on which control level is available to the agent – are injected (purple), while the underlying template remains (black). A typical agent response is shown (green), ending in the ranked candidate list and the selected action.

Knowledgebase Prompt Injection:	
Flow: <code>auto_pblock</code> — Fabric-aware pblock (spread analysis)	
Mechanism: measures per-path spread, sizes a region for 1.5x the used resources, picks the best contiguous fabric region ...	
Use when: critical-path spread is measurably large, and the incumbent placement is disposable ...	
Avoid when: high utilization, or a resource mix dominated by column-bound hard blocks ...	
Ceiling: The only live flow that re-runs global placement, so it has the largest reachable state change in either direction...	
Flow: <code>hardblock_relocate</code> — Hard-block locality improvement	
...	

Fig. 4. Example knowledge base recipe description, allowing the agent to reason about a recipe’s use case from generalised, non-benchmark-specific past information.

1) *Prompt construction*: The diagnostic evidence extracted before each decision (timing summaries, critical-path composition, and the relevant cell, net, and placement context), together with the recipes or operations available at the active control level, is inserted into a fixed prompt template alongside the task description and the required response format (Fig. 3). At the recipe interface, the agent returns a ranked list of candidate actions each scored for priority and risk, with a stated confidence and evidence basis. A fixed, deterministic rule then selects the candidate to execute: highest priority wins, with ties broken by the stated confidence, then by lower risk, then by checkpoint and catalogue order, after two filters exclude the recipe last applied to a checkpoint and any recipe

sequence already attempted. Stating evidence and uncertainty before committing to a score gives the agent room to reason explicitly and leaves a traceable record of why a recipe was preferred, whether or not that reasoning proves right.

2) *The knowledge base*: This supports action selection at the recipe tier by providing additional information to the agent. Each recipe holds one entry (Fig. 4), distilled from an offline, exhaustive execution of all nine recipes across the known benchmark designs rather than authored from first principles, with the same fields: the mechanism the recipe acts through, the design conditions under which it tends to help or hurt, and the ceiling on what it can achieve. Because checkpoints differ in function, resource mix, and implementation method, numeric thresholds measured on one rarely transfer to another, so findings are generalised into qualitative conditions rather than measured values, for example a large, disposable placement rather than a percentage, so that the same entry describes a recipe’s behaviour on a design the exploration never saw.

Entries are injected as an explicitly labelled prior, and the agent is instructed to cite them as prior evidence rather than as evidence about the design in front of it. Backend timing-closure knowledge resides largely in vendor-specific interfaces and engineering experience unlikely to appear in training data; the knowledge base therefore supplies it as in-context learning.

3) *Deep analysis and the bounded decision loop*: When the cause of a violation remains ambiguous, the agent may request evidence beyond the fixed deterministic analysis through a bounded read-only probe loop, in each turn either issuing probes or returning its selection. Probes expose existing evidence without creating new design states, refining the description of the current DCP only. Ten probes are available in three families: three re-run current-design reports for timing, routing, and resource utilisation; five retrieve established search evidence, including the DCP’s state and parent’s $F_{\max}$ gain, stored diagnosis, ancestor recipe outcomes, outcomes in other branches, and validation errors and warnings; and two summarise broader search history through recurring recipe sequences and comparable prior cases. The available set is decision-dependent: all ten are offered at the root; eight at ordinary checkpoints, omitting routing and utilisation reports already measured; and seven after validation failure, omitting timing.

This deep analysis loop is used only in the multi-iteration study, as the single-iteration study focuses on splitting planning and tool abstraction. To prevent scaling token costs, it is bounded at six commands for the root decision step and four at every other DCP step. Because this bounds evidence-gathering rather than reasoning, a session can reach the bound before returning a ranking; in the reported experiments, such a session terminated without a model ranking and received a fixed ordering over the legal candidate set at neutral priority. The bound applies identically at every level, so comparisons between levels are unaffected, although a substituted choice describes the fallback rather than the agent. A subsequent sweep with a raised bound reduced substitution to below 1% of sessions and did not improve achieved frequency, confirming that the reported results are not an artefact of the bound.

4) *Validation and recovery*: Every candidate DCP must be shown to be both a physically legal design and to agree with the design it replaces on the applied tests before it can be trusted, using two checks that differ in cost and frequency. Structural validation, which establishes *physical validity*, runs on every candidate: the implementation logs and the route-status report are parsed to confirm that the produced checkpoint reopens, that routing completed with no unrouted nets or routing errors, and that timing can be re-measured on the design’s target clock rather than on an all-clock summary. Functional checking, whose success is a *functional pass*, is reserved for the single candidate selected as a run’s published result, which in the single-iteration study is each trial’s committed candidate, since it requires simulation and establishes agreement for the applied tests rather than formal equivalence: a precheck is followed by five deterministic simulation runs of 200 input vectors each, and the candidate’s outputs are compared against those of the starting design; a byte-identical candidate inherits its parent’s certificate rather than being re-simulated. An action that proves infeasible to execute is retried once automatically; a candidate that fails either check, or regresses relative to its parent, is excluded from further search and the step reverts to its parent checkpoint, while its execution record is retained in the run history.

III. RESULTS

A. Benchmark and Experimental Constraints

Evaluation uses sixteen post-route implementation DCPs: fifteen drawn from the FPL 2026 AMD Competition benchmarks [29] and one from RapidWright [30]. The set spans roughly two orders of magnitude in instance count, from approximately 3k to 325k LUTs and FFs, covers soft-logic-, memory- and arithmetic-dominated resource mixes, and was selected to span the range of achievable timing slack, with baseline frequencies from 406 MHz, where WNS is under 1 ns, down to 48.2 MHz, where WNS is -19.2 ns. Several designs therefore yield substantial frequency to almost any intervention, while others are limited by logic depth or by delay internal to a hard block and hence have limited yield to any method examined here.

The corpus is partitioned into three groups: eight *known* designs (the development set), whose characteristics and performance were assessed before full evaluation and were used to develop the knowledge base and fit the deterministic policy described in Section III-C; five *re-implementations* from design families represented in that fitting set; and three *out-of-family* designs from families not represented in it, the latter two groups forming the eight held-out designs. This partition separates designs used to establish and tune the method from those reserved to test generalisation. It applies only to the multi-iteration study, whose knowledge base and fitted heuristic were derived from the known designs; the single-iteration study uses no knowledge base and reports all sixteen designs together.

B. Single-Iteration Agent Control Study

This study examines how agent control affects the reliability and timing gain of a single optimisation step across 16 FPGA

designs and seven LLMs: Claude Sonnet 5, DeepSeek v4 Flash and Pro, Gemini 3.6 Flash, and GPT-5.6 Luna, Sol, and Terra. The dataset contains one selected outcome per design–model pair under Expert optimisation recipes, Direct EDA operations, and Constructed procedures, giving 112 trials per condition and 336 in total. The recipe condition permits only recipe selection; the other conditions retain inherited interfaces but explicitly request direct operations or a constructed procedure. We therefore measure outcomes under capability-use instructions, with actual execution of the requested capability assessed separately. No knowledge base is used in this study. For each design, all conditions share the starting checkpoint, base observations, read-only query definitions, and planning limits: four model turns, eight query items, three construction-feedback rounds, and normally 65,536 input bytes including interface documentation. The LLM may query the design and correct construction errors before committing at most one proposal for execution and scoring, without subsequent revision using measured candidate timing. Each proposal allows up to eight statements, including at most four placement, routing, or physical-optimisation calls.

PicoBlazeArray uses the 2025.1 checkpoint (starting $F_{\max} = 350.8$ MHz). Its 21 selected outcomes come from 38 started engineering attempts, with provider changes and dated DeepSeek endpoints (Flash-0731 and Pro-0813). Continuations follow the approved engineering sequence rather than maximum performance. The final constructed-procedure continuations admitted up to 262,144 input bytes; earlier attempts and direct operations admitted 65,536. These are protocol exceptions rather than independent optimisation repeats or a frozen replay. Three tool-deadline timeouts count as unsuccessful trials in the pass-rate denominator; their unmeasured timing changes are not imputed as zero.

A *physically valid result* passes the execution, output, and routing checks. Functional evaluation follows Section II-C4. A *functional pass* requires all prescribed checks to pass; *output mismatch* denotes differing outputs, and *functional check incomplete* denotes validation-infrastructure errors preventing completion. The end-to-end functional-pass rate is the fraction of each condition’s 112 trials achieving a functional pass.

The signed timing change $\Delta F_{\max}$ of Section II-B3 is taken relative to the starting (root) checkpoint, and Figs. 5 and 7 report it as per-trial percentage improvement, $100 \Delta F_{\max} / F_{\max}^{\text{root}}$; implementation case studies use MHz. Byte-identical checkpoints receive zero change. Distributions and the design–model matrix retain all functional passes. Timing improvement requires a functional pass and $\Delta F_{\max} > 0$; outcome-rate denominators include all assigned trials.

Fig. 5 summarises functional-pass rates and timing improvements across the three control conditions. Recipes achieve a 98.2% pass rate (110/112), direct operations 71.4% (80/112), and constructed procedures 17.9% (20/112). All recorded physically valid candidates pass the functional checks; the remaining trials yield no validated functional-pass output. Among functional passes, direct operations yield a slightly higher median improvement than recipes (+3.2% versus +3.4%), but improve timing in fewer of the 112 assigned trials (67 versus 91). Constructed procedures have a median

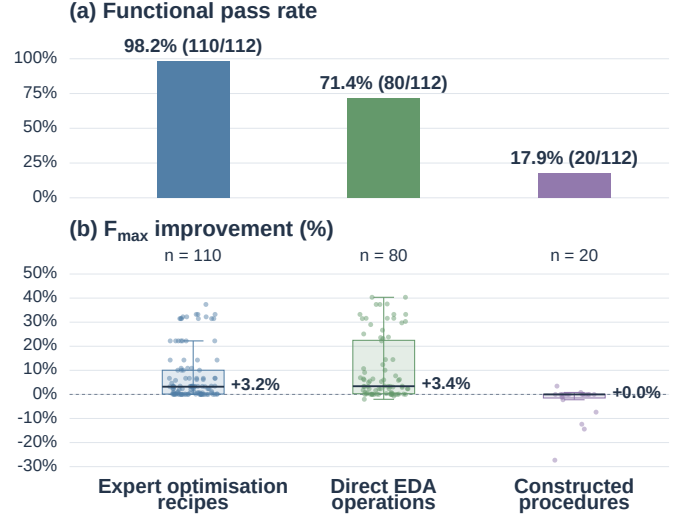


Fig. 5. Reliability and timing improvement across 16 designs and seven LLMs (112 trials per condition). (a) End-to-end functional-pass rates, with passing and total trial counts. (b) Percentage $F_{\max}$ improvement relative to each trial’s starting checkpoint, for functional-pass trials only. Points show individual trials with horizontal jitter for visibility. Boxes span the interquartile range; whiskers extend to the most extreme observations within 1.5 times that range. Labelled lines mark medians. Zero and negative improvements are retained.

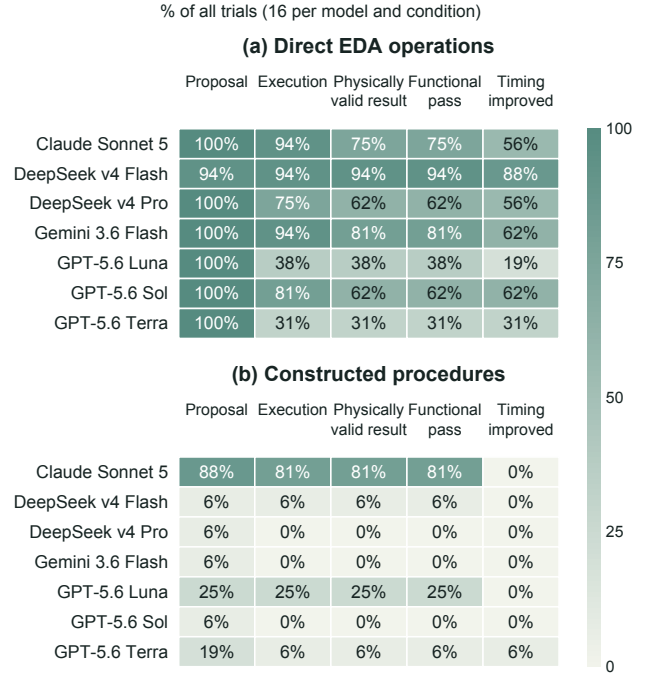


Fig. 6. Progression from proposal generation to timing improvement for (a) Direct EDA operations and (b) Constructed procedures. Each row represents 16 trials for one LLM. Percentages are calculated over all 16 trials, with each stage retaining only trials that satisfy all preceding stages. The final stage requires a functional pass and $\Delta F_{\max} > 0$.

improvement of 0%: 10 of their 20 passing trials show no change, eight regress, and two improve.

Fig. 6 tracks progression from qualifying proposals—executable submissions in the requested interface—to timing

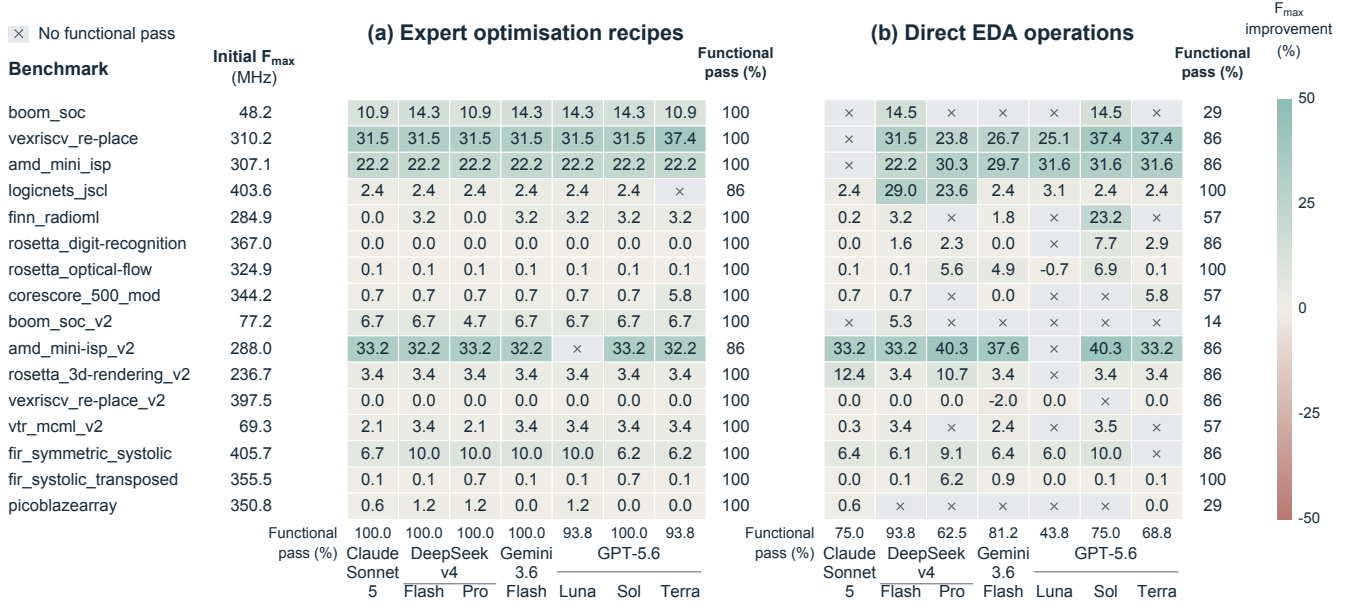


Fig. 7. Design-model timing outcomes under (a) Expert optimisation recipes and (b) Direct EDA operations. Each numeric cell shows percentage $F_{\max}$ improvement relative to the starting checkpoint for one functional-pass trial; × denotes no functional pass. Initial $F_{\max}$ gives the starting frequency in MHz. Designs are ordered by increasing initial $F_{\max}$; both panels use the same design and model ordering and a common colour scale. Row pass percentages are calculated across seven models; column pass percentages are calculated across sixteen designs.

TABLE III

PHYSICAL CASE STUDIES. SELECTED AND BEST DENOTE THE LLM-SELECTED RECIPE AND BEST RECORDED RECIPE FROM THE SAME STARTING CHECKPOINT; DIRECT DENOTES DIRECT OPERATIONS. DELAYS DESCRIBE EACH OUTPUT’S WORST SETUP PATH. RESOURCES DESCRIBE THE INPUT.

Circuit / model	Output	$\Delta F_{\max}$ (MHz)	WNS (ns)	Logic (ns)	Route (ns)	Limiting-path structure
LogicNets / DeepSeek v4 Flash 30.8k LUTs, 1.7k FFs, 0 DSPs, 0 BRAMs	Selected recipe	+9.5	-0.921	0.514	1.887	LUT / multiplexer path
	Best recorded recipe	+98.0	-0.494	0.509	1.467	LUT / multiplexer path
	Direct operations	+117.0	-0.421	0.562	1.340	LUT / multiplexer path
FINN RadioML / GPT-5.6 Sol 73.6k LUTs, 45.8k FFs, 0 DSPs, 24.5 BRAMs	Selected recipe	+9.1	-1.801	1.135	2.198	12-level carry/LUT path
	Best recorded recipe	+54.4	-1.347	1.155	1.566	12-level carry/LUT path
	Direct operations	+66.1	-1.249	2.405	0.196	Six-RAMB36E2 cascade

improvement. Execution requires a direct EDA operation beyond recipe selection in panel (a), or use of a procedure-created intermediate object by a subsequent operation in panel (b). Unlike Fig. 5, later stages count only trials that also execute the requested capability. Of the 112 direct-operation trials, 111 produce qualifying proposals; 81 (72.3%) execute the capability, 71 pass functional checks, and 60 improve timing. For constructed procedures, only 25 of 112 trials produce qualifying proposals; 19 execute and pass, and one improves timing, so their largest loss occurs before a qualifying proposal is produced. Model differences are also substantial: DeepSeek v4 Flash executes direct operations and passes on 15 designs, improving 14, whereas Claude Sonnet 5 executes constructed procedures and passes on 13 designs but improves none, so successful execution does not necessarily yield a timing gain.

Fig. 7 shows that the effect of additional control depends on both design and LLM. On *vexriscv_re-place*, switching from recipes to direct operations increases the timing improvement from +31.5% to +37.4% for GPT-5.6 Sol, but reduces it from +31.5% to +23.8% for DeepSeek v4 Pro. On *boom_soc*, all seven LLMs pass with recipes, whereas

only DeepSeek v4 Flash and GPT-5.6 Sol pass with direct operations, with similar gains to their recipe results. Additional control can therefore increase or reduce timing gains, or fail to produce a functional-pass result.

Table III compares selected recipes, the highest- $F_{\max}$ recorded functional-pass recipes from the same starting checkpoints, and direct-operation results, with execution traces and critical-path reports revealing the corresponding physical changes. On LogicNets, DeepSeek v4 Flash unroutes and unplaces the design, then applies placement, physical optimisation, routing, and final physical optimisation; its +117.0 MHz gain exceeds both its selected *phys_opt* recipe (+9.5 MHz) and the best recorded recipe (+98.0 MHz), whereas DeepSeek v4 Pro’s +95.2 MHz direct-operation gain exceeds only its selected recipe. On FINN RadioML, GPT-5.6 Sol re-replaces the design and applies physical optimisation and routing, gaining +66.1 MHz as the worst setup path shifts from the 12 carry/LUT levels seen in both recipe outputs to six cascaded RAMB36E2 blocks, changing the structure limiting frequency. On *vexriscv_re-place*, DeepSeek v4 Pro’s direct operations yield less gain than its selected recipe: lower

Benchmark	Set	Initial $F_{\max}$ (MHz)	LLM-based												Functional pass (%)	Best $\Delta F_{\max}$ (MHz)	$F_{\max}$ improvement (%)
			(a) Fitted heuristic	(b) Sequential reasoning			(c) Branching exploration			(d) Knowledge base-guided exploration							
boom_soc	K	48.2	91.0	27.0	68.7	29.3	89.0	20.2	85.9	89.9	91.6	102.2	100	+49.3 (102.2)	0 10 20 30 40 50		
vexriscv_re-place	K	310.2	51.4	31.5	31.5	31.5	32.6	32.6	41.9	42.8	32.6	42.8	100	+159.5 (51.4)			
amd_mini_isp	K	307.1	32.7	33.2	22.2	23.6	23.6	29.4	32.6	30.0	33.2	31.6	100	+102.0 (33.2)	Set key K known (development set) R held out, same family O held out, out of family		
logiconets_jscl	K	403.6	23.8	10.3	2.4	2.4	4.3	17.7	23.8	23.8	24.1	3.8	60	+97.2 (24.1)			
finn_radioml	K	284.9	21.1	3.2	3.2	3.2	10.2	20.1	20.1	17.3	20.1	21.2	87	+60.5 (21.2)			
rosetta_digit-recognition	K	367.0	15.9	0.0	0.0	0.0	2.1	14.3	15.0	10.7	11.7	15.7	100	+58.2 (15.9)			
rosetta_optical-flow	K	324.9	4.8	0.1	0.1	0.1	5.6	5.6	5.6	3.9	4.9	4.8	100	+18.3 (5.6)			
corescore_500_mod	K	344.2	0.0	9.3	0.7	0.7	7.4	9.3	9.3	9.3	9.3	9.3	100	+32.1 (9.3)			
boom_soc_v2	R	77.2	14.8	9.4	9.9	9.5	26.4	16.5	16.6	9.0	22.6	25.8	100	+20.4 (26.4)			
amd_mini-isp_v2	R	288.0	41.5	40.9	43.3	40.7	33.2	38.3	39.7	39.7	40.3	40.2	100	+124.7 (43.3)			
rosetta_3d-rendering_v2	R	236.7	21.1	3.4	3.4	3.7	3.7	26.2	26.3	16.4	20.1	26.3	100	+62.2 (26.3)			
vexriscv_re-place_v2	R	397.5	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.0	0.0	0.0	100	+1.7 (0.4)			
vtr_mcml_v2	R	69.3	0.0	3.4	3.4	3.4	3.5	3.4	3.4	3.5	3.4	3.4	100	+2.4 (3.5)			
fir_symmetric_systolic	O	405.7	0.0	11.4	11.4	11.4	10.1	10.0	11.4	10.1	10.6	10.1	100	+46.2 (11.4)			
fir_systolic_transposed	O	355.5	0.0	0.1	0.1	0.1	1.6	1.6	2.3	1.6	3.3	2.6	100	+11.9 (3.3)			
picoblazearray	O	350.8	0.0	1.5	1.9	2.0	1.5	1.5	1.5	1.3	1.5	1.5	100	+7.0 (2.0)			
	Functional pass (%)		100.0	100.0	100.0	95.8	100.0	100.0	93.8	100.0	100.0	93.8					
				Sonnet	Gemini	Luna	Sonnet	Gemini	Luna	Sonnet	Gemini	Luna					

Fig. 8. Best functional-pass $F_{\max}$ improvement per benchmark for (a) the deterministic fitted heuristic and the LLM-based levels (b) sequential reasoning, (c) branching exploration and (d) knowledge base-guided exploration, each with Claude Sonnet 5, Gemini 3.6 Flash and GPT-5.6 Luna (best of three repeats).

TABLE IV
IMPLEMENTATION RESULTS FOR THREE DESIGNS, COMPARING THE INPUT CHECKPOINT WITH THE ABLATION LEVELS.

Circuit / Model / Resources	Output	$\Delta F_{\max}$ (MHz)	WNS (ns)	Logic (ns)	Route (ns)	Limiting-path structure
BOOM SoC	Input checkpoint	–	–19.162	2.001	18.471	27-level LUT path to load queue
GPT-5.6 Luna	Fitted heuristic	+43.9	–9.283	2.072	8.675	20-level LUT path to I-cache BRAM
227k LUTs, 98k FFs, 61 DSPs, 161 BRAMs	Sequential reasoning	+14.1	–14.463	1.220	15.001	16-level LUT path to register file
	Branching exploration	+41.4	–9.583	2.623	8.351	34-level LUT/multiplexer path to register file
	Knowledge base-guided	+49.3	–8.685	1.982	8.314	19-level LUT/carry path to register file
AMD mini-ISP v2	Input checkpoint	–	–1.902	0.394	2.711	5-level LUT path to line-buffer BRAM
Gemini 3.6 Flash	Fitted heuristic	+119.5	–0.884	1.909	0.397	6-level DSP multiply path
3.2k LUTs, 4.5k FFs, 40 DSPs, 12 BRAMs	Sequential reasoning	+124.7	–0.853	1.923	0.350	6-level DSP multiply path
	Branching exploration	+110.2	–0.941	1.911	0.451	6-level DSP multiply path
	Knowledge base-guided	+116.2	–0.904	1.925	0.400	6-level DSP multiply path
Symmetric FIR	Input checkpoint	–	–0.465	0.078	1.999	Delay-line register to DSP input
Claude Sonnet 5	Fitted heuristic	+0.0	–0.465	0.078	1.999	Unchanged; only attempt rejected
0.7k LUTs, 22.6k FFs, 680 DSPs, 0 BRAMs	Sequential reasoning	+46.2	–0.213	0.995	1.048	2-level DSP adder-tree cascade
	Branching exploration	+41.1	–0.238	0.130	1.963	Valid signal to DSP clock enable
	Knowledge base-guided	+41.1	–0.238	0.130	1.963	Valid signal to DSP clock enable

logic delay is outweighed by increased routing delay (2.160 versus 1.840 ns) on the worst setup path.

C. Multi-Iteration, Expert Recipes Ablation Study

Where the single-iteration study asked how much control to give the LLM within one attempt, this study asks how repeated attempts should be organised to efficiently navigate the design space. Given the single-iteration trade-off between reliability and reward, this study fixes the action tier at the nine expert recipes of Table II, which execute identically in both studies, and varies only the selection mechanism over multi-iteration runs to isolate selection from execution. Four mechanisms are compared. The *fitted heuristic* ranks recipes from measured physical features without a language model. *Sequential reasoning* replaces that ranking with LLM-driven selection along a sequential trajectory. *Branching exploration* retains that procedure and adds parallel expansion, evaluating

several candidates from one checkpoint and continuing from the best that validates. *Knowledge base-guided exploration* retains branching and adds the knowledge base described in Section II-C. The last three form a strict inclusion chain, so differences between adjacent levels isolate the added mechanism, while the heuristic is an independent deterministic baseline sharing the same action tier and validation gating. All levels receive the same maximum budget of twelve DCPs per benchmark run. Three of the seven previously characterised LLMs span the cost-capability range: GPT-5.6 Luna and Gemini 3.6 Flash as low-cost options, the latter among the stronger prior performers, and Claude Sonnet 5 as a high-capability reference.

The heuristic is a deterministic, non-learning baseline that establishes the study’s floor condition, fitted onto the DCP and recipe results from the known benchmarks At each step it measures four properties of the current design state:

the proportion of critical-path delay attributable to routing versus logic, the depth of the critical logic chain, the count of instantiated hard-block resources, and the load on the most heavily driven net, each selected for a documented physical link to at least one recipe’s mechanism of action and for statistical prominence in differentiating the best recipe in the prior study. It then applies the single highest-scoring recipe; it never branches, hedges between candidates, or revises its scoring rule within a run. For recipe f and standardised feature vector $\mathbf{z}$, the score is

$$\text{score}_f = \beta_f + \sum_i w_{f,i} z_i, \quad (2)$$

where β_f is the recipe’s mean percentage frequency improvement across the fitting corpus, and the summation is the design-conditioned adjustment; selection is by $\arg\max$ with deterministic tie-breaking. A recipe therefore starts from its average worth and is moved up or down by how far the current design differs from the fitting corpus. Coefficients were fitted by ridge regression against percentage rather than absolute improvement, so that designs with dissimilar baseline clocks contribute comparably, with regularisation strength selected by sweeping several orders of magnitude under leave-one-design-out validation.

Fig. 8 reports, for every benchmark, the best functional-pass $F_{\max}$ improvement of the fitted heuristic and of each LLM at each LLM-based ablation level. The fitted heuristic and knowledge base-guided exploration reach equivalent corpus means, 19.9% and 20.3%, against 10.5% for sequential reasoning and 16.0% for branching exploration, but the per-design results show opposite successes. On the known designs, the fitted heuristic shows the strongest $F_{\max}$ improvement, because it was fitted on those circuits, yet on the held-out designs it returns no improvement on five of eight, and on the known *corescore_500_mod*. On *picoblazearray*, whose 660 memory blocks lie far beyond the fitted range of 250, the policy selected *auto_pblock*, which regressed the design, giving an example where the fitted heuristic cannot reliably extrapolate its selection rule to sufficiently unusual designs. On two further designs, *corescore_500_mod* and *fir_systolic_transposed*, the selected recipe *auto_pblock* could not place the design inside its region, so routing failed on an unplaced netlist, and on the remaining three the selected recipes executed successfully but regressed the design. A fitted policy carries no representation of its own uncertainty, so an out-of-range design produces a confident commitment instead of an abstention, and adding a rejection rule to (2) would drive the heuristic to overfit the corpus, an issue already requiring careful management. In contrast, the three model-driven ablations return a positive gain on 14, 16, and 15 of 16 designs respectively, including every out-of-family design, suggesting an ability to generalise across unseen benchmarks.

Table IV examines this generalisation at the physical level, taking one design from each partition with the LLM that achieved its best result. On BOOM SoC, GPT-5.6 Luna with knowledge base-guided exploration improves $F_{\max}$ by 49.3 MHz; the framework diagnosed the worst path as routing-dominated, with 18.5 of its 20.5 ns spent in wiring, and applied

memory_path_opt, whose re-placement reduced that route delay to 8.3 ns. On the held-out AMD mini-ISP, Gemini 3.6 Flash achieves the largest gain in the study, 124.7 MHz; here every level removes the wire-bound path into a line-buffer BRAM, after which the limit lies inside a DSP multiplier, so the levels converge on similar results. On the out-of-family symmetric FIR filter, Claude Sonnet 5 improves $F_{\max}$ by 46.2 MHz where the fitted heuristic returns nothing. In this case, the reason was the valid signal driving the clock enables of the DSP adder tree: a second *phys_opt* pass re-optimised that path and moved the limit into the adder tree itself, whereas Sonnet’s branching runs stopped after a single pass.

Every ablation nonetheless performs better on designs it has already seen: mean improvement on the development set against the held-out set is 30.1% against 9.7% for the fitted heuristic, 12.7% against 8.4% for sequential reasoning, 20.9% against 11.1% for branching exploration, and 28.6% against 11.9% for knowledge base-guided exploration; since the knowledge base was distilled from the same known designs, part of its advantage remains design-specific.

Sequential reasoning underperforms structurally rather than through weaker inference. Approximately 68% of a design’s best observed result is reachable within a single intervention, so a single-trajectory search commits to one opening and cannot recover from a poor one, nor efficiently from an invalid checkpoint, because expanding one candidate at a time forfeits the remaining budget. Evaluating three openings instead of one gives breadth at the root rather than depth and accounts for approximately +8.6 MHz of gain, close to two-thirds of branching exploration’s 14.2 MHz advantage over sequential reasoning. Breadth pays at the root because depth rewards little beyond it: within branching-capable runs, the share of physically valid DCP improvement attempts achieving no further gain over their own original DCP rises from 9.2% among the opening candidates to 34.4% beyond them, so roughly a third of the checkpoint budget beyond the opening step is spent on attempts that do not advance the search. This stems from the recipes themselves: they act only on currently-failing paths and commit only improvements, so repeated application searches already-harvested ground; for example, *phys_opt* returns approximately +22 MHz on first application and under +5 MHz on repetition.

Fig. 9 relates recipe usage to each recipe’s average cumulative $F_{\max}$ improvement over the starting DCP and validation pass rate. The heuristic spends 77% of its attempts on the two highest-yielding recipes (Fig. 9a), against 0.6% for sequential reasoning and 11% for branching exploration, so selection, not execution, is the binding constraint on the model-driven levels; the knowledge base recovers this to 21% by embedding expert knowledge without altering recipes. Fig. 9b tabulates each recipe’s mean gain beside its validation pass rate, and what each recipe physically changes explains the remaining per-design variation. Hard-block-oriented recipes cannot relocate fixed memory and arithmetic columns, so their effectiveness depends on how much surrounding soft logic can be profitably re-placed. This is evident in the BOOM SoC case of Table IV, where the recipe’s outcome varies with the starting placement. Furthermore, *hardblock_relocate* is

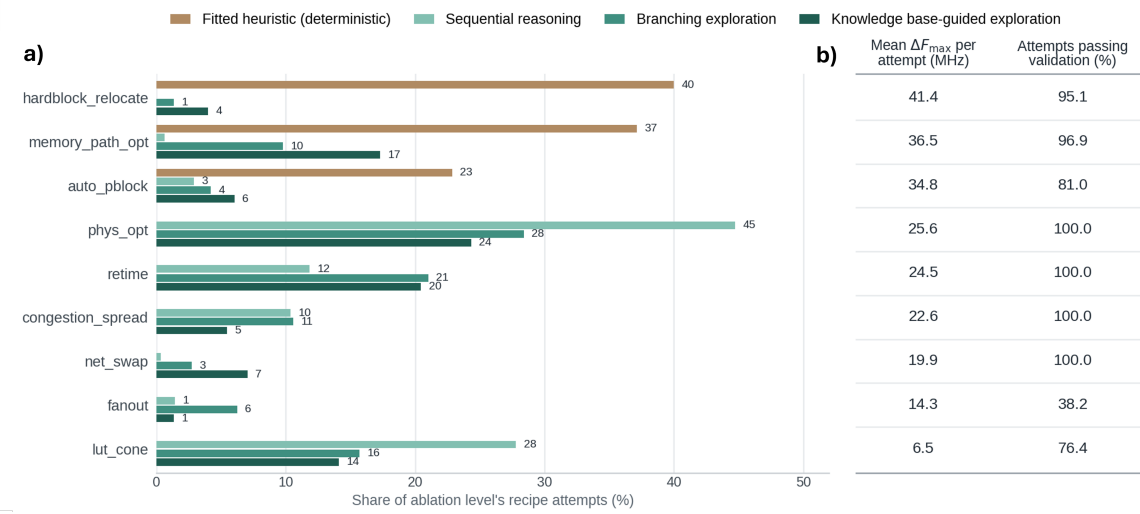


Fig. 9. Effectiveness of the expert recipes across the multi-iteration study. **(a)** Share of each ablation level's recipe attempts spent on each recipe. **(b)** Each recipe's mean $F_{\max}$ improvement per attempt, measured from the starting DCPs, and the share of its attempts passing physical validation.

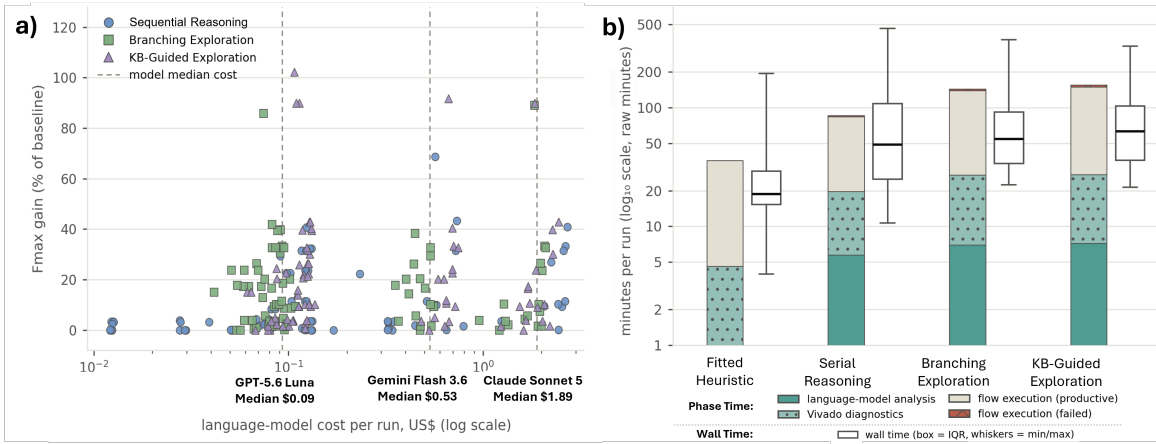


Fig. 10. Cost assessment across multi-iteration runs, **(a)** language-model expense against $F_{\max}$ improvement for every model-driven run, **(b)** phase time decomposed by pipeline stage beside the elapsed wall time.

the best-performing recipe on six of sixteen designs, yet, on *picoblazearray*, its 660 small cores already sit beside their memory blocks, leaving little scope for re-placement. Second, *auto_pblock* unplaces and re-places the whole design within one region, so its risk lies in whether a given design fits that region rather than in the timing limits: it is reliable on twelve designs, but it fails on over 89% of attempts on the other four.

Validation outcomes are likewise structural: four recipes, executing wholly within Vivado, passed validation on all 1,477 of their attempts, whereas *fanout* and *lut_cone*, the two recipes that export the netlist to RapidWright and write it back, forfeiting that guarantee, passed on only 38.2% and 76.4%. Validation critically guarded against results that only appear to succeed: on *fir_systolic_transposed*, a partially placed *auto_pblock* output reported a 105 MHz gain with positive slack and was rejected. In this study, correctness was therefore secured by the execution boundary rather than by the reasoning that selected the action.

Two measures of duration assess the cost of running the

optimisation (Fig. 10). *Phase time* attributes each attempt's duration to the pipeline stage responsible and sums across attempts; *wall time* is the run's own elapsed duration. Their ratio indicates the degree of concurrency achieved: it averages $1.66\times$ for branching exploration and $1.69\times$ for knowledge base-guided exploration (Fig. 10b), below the ideal $3.0\times$ under full overlap of three branch candidates, while remaining below unity for non-branching levels because some elapsed time is not attributed to a labelled stage. This suggests strong utilisation but opportunity for optimising runtime further. Median language-model cost per run varies roughly twentyfold across the three LLMs for a difference in mean improvement of under a percentage point (Fig. 10a), so among the three LLMs tested, model choice mainly affected cost rather than achieved frequency, consistent with the single-iteration study, in which no LLM dominated across designs. Language-model analysis accounts for 4.6–6.7% of phase time and tool execution with diagnostics for 91–92% at every level, so the economic case for orchestration rests on issuing effective tool invocations, not on inexpensive inference. The fitted heuristic

(a) Single iteration														(b) Multi-iteration				(c) Best result per design			
Benchmark	Set	Initial $F_{\max}$ (MHz)	Deterministic				LLM-based				$\Delta F_{\max}$ (MHz)	$\Delta F_{\max}$ (%)	Method, LLM Recipes	Cell key							
			Expert recipe	Direct EDA operations	Fitted heuristic	Sequential reasoning	Branching exploration	KB-guided exploration													
boom_soc	K	48.2	14.3 +6.9	14.5 +7.0	91.0 +43.9	68.7 +33.1	89.0 +42.9	102.2 +49.3	+49.3	102.2	MI KB-guided, Luna PO:RT-MP:NS	33.2 +102.0	← best $\Delta F_{\max}$ (%) ← same result (MHz)								
vexriscv_re-place	K	310.2	37.4 +116.0	37.4 +116.0	51.4 +159.5	31.5 +97.7	41.9 +130.0	42.8 +132.9	+159.5	51.4	MI heuristic, no LLM HB:MP:HB										
amd_mini_isp	K	307.1	22.2 +68.2	31.6 +97.0	32.7 +100.4	33.2 +102.0	32.6 +100.0	33.2 +102.0	+102.0	33.2	MI KB-guided, Gemini LC:PO:MP	F _{max} improvement (%)	0 10 20 30 40 50								
logicnets_jscl	K	403.6	2.4 +9.7	29.0 +117.0	23.8 +95.9	10.3 +41.7	23.8 +95.9	24.1 +97.2	+117.0	29.0	SI direct, DS Flash direct operations										
finn_radioml	K	284.9	3.2 +9.1	23.2 +66.1	21.1 +60.2	3.2 +9.1	20.1 +57.2	21.2 +60.5	+66.1	23.2	SI direct, Sol direct operations	Recipe key PO phys_opt RT retime MP memory_path_opt HB hardblock_relocate AP auto_pblock NS net_swap CS congestion_spread LC lut_cone FO fanout									
rosetta_digit-recognition	K	367.0	0.0 +0.0	7.7 +28.3	15.9 +58.2	0.0 +0.0	15.0 +55.1	15.7 +57.7	+58.2	15.9	MI heuristic, no LLM AP:HB										
rosetta_optical-flow	K	324.9	0.1 +0.3	6.9 +22.4	4.8 +15.7	0.1 +0.4	5.6 +18.3	4.9 +15.9	+22.4	6.9	SI direct, Sol direct operations	Set key K known (development set) R held out, same family O held out, out of family									
corescore_500_mod	K	344.2	5.8 +20.0	5.8 +20.0	0.0 +0.0	9.3 +32.1	9.3 +32.1	9.3 +32.1	+32.1	9.3	MI branching, Gemini PO:MP										
boom_soc_v2	R	77.2	6.7 +5.2	5.3 +4.1	14.8 +11.4	9.9 +7.6	26.4 +20.4	25.8 +19.9	+20.4	26.4	MI branching, Sonnet PO:MP:CS	+124.7	43.3								
amd_mini-isp_v2	R	288.0	33.2 +95.6	40.3 +116.1	41.5 +119.5	43.3 +124.7	39.7 +114.4	40.3 +116.2			MI sequential, Gemini PO:RT-PO:LC:AP										
rosetta_3d-rendering_v2	R	236.7	3.4 +8.0	12.4 +29.4	21.1 +50.0	3.7 +8.8	26.3 +62.2	26.3 +62.2	+62.2	26.3	MI branching, Luna PO:MP:PO	+1.7	0.4								
vexriscv_re-place_v2	R	397.5	0.0 +0.0	0.0 +0.0	0.0 +0.0	0.0 +0.0	0.4 +1.7	0.0 +0.0			MI branching, Luna PO:AP										
vtr_mcml_v2	R	69.3	3.4 +2.4	3.5 +2.4	0.0 +0.0	3.4 +2.4	3.5 +2.4	3.5 +2.4	+2.4	3.5	MI branching, Sonnet LC:RT	+46.2	11.4								
fir_symmetric_systolic	O	405.7	10.0 +40.6	10.0 +40.6	0.0 +0.0	11.4 +46.2	11.4 +46.2	10.6 +43.0			MI sequential, Gemini PO:RT:PO										
fir_systolic_transposed	O	355.5	0.7 +2.5	6.2 +22.0	0.0 +0.0	0.1 +0.5	2.3 +8.3	3.3 +11.9	+22.0	6.2	SI direct, DS Pro direct operations	+7.0	2.0								
picoblazearray	O	350.8	1.2 +4.2	0.6 +2.1	0.0 +0.0	2.0 +7.0	1.5 +5.4	1.5 +5.4			MI sequential, Luna PO:CS:...:CS:PO (9)										
Functional pass (%)			98.2	71.4	100.0	97.5	96.2	96.2													
			best of 7 LLMs, 1 trial each		1 run	best of 3 LLMs, 5 runs (Luna ×3)															

Fig. 11. Summary of both studies: best functional-pass $F_{\max}$ improvement per benchmark. (a) Single-iteration study: best of seven LLMs, one trial each, under expert recipes and direct EDA operations. (b) Multi-iteration study: the deterministic fitted heuristic (one run) and the LLM-based levels (best of three LLMs over five runs, three of them GPT-5.6 Luna repeats). (c) The best result per design, with the method, LLM and recipe sequence that achieved it.

finishes sooner only because its ranking is fixed before the run begins: once its top-ranked recipes are spent, it self-terminates after two to four checkpoints, where the model-driven levels re-diagnose and consume the full allowance. Per attempt it is the slowest level in the study, because its two highest-scoring recipes (hardblock_relocate and memory_path_opt) are also the longest to run.

IV. DISCUSSION

Comparing the studies design by design exposes effects hidden by their corpus means (Fig. 11). In the single-iteration study, direct operations matched or exceeded the best recipe result on fourteen of the sixteen designs, despite passing functional checks in 71.4% of trials against 98.2% for recipes. The multi-iteration study reverses this pattern: iterative recipe selection exceeds the best single recipe attempt on all sixteen designs and exceeds the best direct-operation attempt on eleven, with one tie. The four remaining direct-operation wins are narrow, ranging from 1.3 to 4.9 percentage points. This contrast is consistent with approximately 68% of a design's best observed result being reachable in one intervention: direct operations can access an effective intervention immediately, whereas recipes gain from selecting and composing reliable actions. An effective deployment could therefore begin with recipes and escalate to direct operations when a measured bottleneck remains unresolved. Further study should establish which design evidence could trigger this change in agent control and how the remaining budget should be divided between action breadth and continued recipe search.

Unlike the fitted heuristic, the LLM-based harness improves unseen designs because physical bottlenecks vary too widely for a fixed selection rule to remain well calibrated. Among the LLMs tested, model choice changed cost by roughly twentyfold in the multi-iteration study but mean improvement by under a percentage point, and no LLM dominated across designs in the single-iteration study. This result places the main opportunity in the harness rather than in model selection alone: broader recipes, better evidence retrieval, and explicit uncertainty handling could improve coverage and generalisation. The present catalogue remains weighted toward placement and routing, with only *lut_cone* restructuring combinational logic. Critical-cone technology remapping, don't-care-based resynthesis, or deeper pipelining could reach logic-dominated bottlenecks, but netlist-rewriting actions would require formal or broader equivalence checking beyond the sampled simulation used here. Structural and functional validity also does not ensure a frequency gain when an action crosses a tool boundary, so a commit-only-if-improved check should be reapplied at each boundary.

The harness evidence remains bounded by one device family, toolchain, and tool version, and by a knowledge base distilled from the eight known designs. Larger benchmark sets, other vendors, and independently constructed or continuously updated knowledge bases are needed to test whether the harness is truly transferable. Finally, the same checkpoint-and-validate architecture could target power and area improvement, at the added complexity of multi-objective trade-offs, and its MCP-based interface provides a natural boundary for connecting additional capability, including front-end RTL generators.

V. CONCLUSION

This paper presented LATCH, a two-layer framework separating LLM orchestration from deterministic execution for post-route FPGA frequency improvement across sixteen designs. In a single attempt with seven LLMs, expert recipes passed functional checks in 98.2% of trials, direct EDA operations in 71.4% and constructed procedures in 17.9%, though passing direct operations had a higher median gain of +3.4% against +3.2% for expert recipes. Within twelve DCPs per run, mean improvement rose from 10.5% (sequential) to 16.0% (branching) and 20.3% (knowledge base), matching the fitted heuristic's 19.9%. Per-design best gains spanned 0.4% to 102.2%, from iterated recipe selection on twelve designs and one direct-operation attempt on four.

ACKNOWLEDGMENT

The authors thank the AMD University Programme for Vivado licenses, and the FPL'26 Agentic FPGA Backend Optimization Competition Organisers for setting the challenge, providing benchmarks, and drafting the baseline software. We used Claude Sonnet 5, DeepSeek v4 Flash & v4 Pro, Gemini 3.6 Flash, OpenAI GPT-5.6 Luna, GPT-5.6 Sol, GPT-5.6 Terra as part of the framework development and experimental evaluation; we used Claude Opus 5 to proofread the author-written manuscript text.

REFERENCES

- [1] Z. Di, R. Tao, J. Mai, L. Chen, and Y. Lin, "LEAPS: Topological-layout-adaptable multi-die FPGA placement for super long line minimization," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 3, pp. 1259–1272, Mar. 2024. [Online]. Available: <https://doi.org/10.1109/TCSI.2023.3340554>
- [2] C. Xu, G. Liu, R. Zhao, S. Yang, G. Luo, and Z. Zhang, "A parallel bandit-based approach for autotuning FPGA compilation," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 157–166. [Online]. Available: <https://www.csl.cornell.edu/~zhiruz/pdfs/datuner-fpga2017.pdf>
- [3] W. L. Neto, Y. Li, P.-E. Gaillardon, and C. Yu, "FlowTune: End-to-end automatic logic optimization exploration via domain-specific multiarmed bandit," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 6, pp. 1912–1925, Jun. 2023. [Online]. Available: <https://doi.org/10.1109/TCAD.2022.3213611>
- [4] Z. Di, R. Tao, L. Chen, Q. Wu, and Y. Lin, "Imbalanced large graph learning framework for FPGA logic elements packing prediction," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 4, pp. 2034–2038, Apr. 2024. [Online]. Available: <https://doi.org/10.1109/TCSII.2023.3334247>
- [5] G. Huang *et al.*, "Machine learning for electronic design automation: A survey," *ACM Transactions on Design Automation of Electronic Systems*, vol. 26, no. 5, Jun. 2021. [Online]. Available: <https://doi.org/10.1145/3451179>
- [6] T. Martin, G. Grewal, and S. Areibi, "A machine learning approach to predict timing delays during FPGA placement," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Jun. 2021, pp. 124–127. [Online]. Available: <https://doi.org/10.1109/IPDPSW52791.2021.00026>
- [7] Y.-K. Lin, Z. Xiong, and D. Z. Pan, "Differentiable timing-driven FPGA placement with smooth optimization and ML-based delay calibration," in *2025 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, Oct. 2025, pp. 1–9. [Online]. Available: <https://doi.org/10.1109/ICCAD66269.2025.11240642>
- [8] H. Jiang *et al.*, "Critical path aware timing-driven global placement for large-scale heterogeneous FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2026, early Access. [Online]. Available: <https://doi.org/10.1109/TCAD.2026.3678497>
- [9] N. Kapre, B. Chandrashekar, H. Ng, and K. Teo, "Driving timing convergence of FPGA designs through machine learning and cloud computing," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2015, pp. 119–126.
- [10] E. Ustun, S. Xiang, J. Gui, C. Yu, and Z. Zhang, "LAMDA: Learning-assisted multi-stage autotuning for FPGA design closure," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines*, 2019, pp. 74–77. [Online]. Available: <https://www.csl.cornell.edu/~zhiruz/pdfs/lamda-fccm2019.pdf>
- [11] A. Mirhoseini *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, Jun. 2021. [Online]. Available: <https://doi.org/10.1038/s41586-021-03544-w>
- [12] J. Pan, G. Zhou, C.-C. Chang, I. Jacobson, J. Hu, and Y. Chen, "A survey of research in large language models for electronic design automation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 30, no. 3, Feb. 2025. [Online]. Available: <https://doi.org/10.1145/3715324>
- [13] J. Nainani, C.-T. Ho, A. Dhurka, and H. Ren, "Timing analysis agent: Autonomous multi-corner multi-mode (MCM) timing debugging with timing debug relation graph," 2025, arXiv:2504.11502. [Online]. Available: <https://arxiv.org/abs/2504.11502>
- [14] Y. Ouyang, Y. Luo, D. Zuo, and Y. Ma, "Retrieve, schedule, reflect: LLM agents for chip QoR optimization," 2026, arXiv:2603.13767. [Online]. Available: <https://arxiv.org/abs/2603.13767>
- [15] C. Liu and D. Chitnis, "EESizer: LLM-based AI agent for sizing of analog and mixed signal circuit," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 73, no. 6, pp. 3825–3834, Jun. 2026. [Online]. Available: <https://doi.org/10.1109/TCSI.2025.3646359>
- [16] M. Youngman, C. Sestito, and T. Prodromakis, "LLMs in digital EDA: A perspective on shifting roles from generation to orchestration," 2026, arXiv:2608.27184. [Online]. Available: <https://arxiv.org/abs/2608.27184>
- [17] B.-Y. Wu *et al.*, "EDA corpus: A large language model dataset for enhanced interaction with OpenROAD," in *2024 IEEE LLM Aided Design Workshop (LAD)*, 2024, arXiv:2405.06676. [Online]. Available: <https://arxiv.org/abs/2405.06676>
- [18] W. Fu *et al.*, "A generalize hardware debugging approach for large language models semi-synthetic, datasets," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 72, no. 2, pp. 623–636, Feb. 2025. [Online]. Available: <https://doi.org/10.1109/TCSI.2024.3487486>
- [19] P. Liu, N. Xu, J. Tang, Y. Cao, and C. Ding, "Bridging the last mile of circuit design: PostEDA-Bench, a hierarchical benchmark for PPA convergence and DRC fixing," 2026, arXiv:2605.06936. [Online]. Available: <https://arxiv.org/abs/2605.06936>
- [20] Q. Li *et al.*, "PDAGENT-Bench: Characterizing, grounding, and architecting LLM/VLM agents for VLSI physical design," 2026, arXiv:2606.17253. [Online]. Available: <https://arxiv.org/abs/2606.17253>
- [21] H. Sami, M. ul Islam, S. Charas, A. Gandhi, P.-E. Gaillardon, and V. Tenace, "Nexus: A lightweight and scalable multi-agent framework for complex tasks automation," 2025, arXiv:2502.19091. [Online]. Available: <https://arxiv.org/abs/2502.19091>
- [22] A. Ghose, A. B. Kahng, S. Kundu, and Z. Wang, "ORFS-agent: Tool-using agents for chip design optimization," in *2025 ACM/IEEE 7th Symposium on Machine Learning for CAD (MLCAD)*, 2025, pp. 1–13. [Online]. Available: <https://doi.org/10.1109/MLCAD65511.2025.11189204>
- [23] Z. Chen, Z. Song, Y. Li, Q. Sun, and C. Zhuo, "FluxEDA: A unified execution infrastructure for stateful agentic EDA," 2026, arXiv:2603.25243. [Online]. Available: <https://arxiv.org/abs/2603.25243>
- [24] H. Lin, K. Li, L. Wang, and Z. Wang, "PACT: Post-route agentic checkpoint tuning for FPGA timing closure," 2026, arXiv:2608.23602v2; accepted to FPT 2026. [Online]. Available: <https://arxiv.org/abs/2608.23602v2>
- [25] H. Wu *et al.*, "ChatEDA: A large language model powered autonomous agent for EDA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 10, pp. 3184–3197, Oct. 2024. [Online]. Available: <https://doi.org/10.1109/TCAD.2024.3383347>
- [26] W. Fang *et al.*, "Dr. RTL: Autonomous agentic RTL optimization through tool-grounded self-improvement," 2026, arXiv:2604.14989. [Online]. Available: <https://arxiv.org/abs/2604.14989>
- [27] R. Krishna, S. Krishna, and D. Chin, "Design conductor 2.0: An agent builds a TurboQuant inference accelerator in 80 hours," 2026, arXiv:2605.05170. [Online]. Available: <https://arxiv.org/abs/2605.05170>
- [28] C. Lavin and A. Kaviani, "RapidWright: Enabling custom crafted implementations for FPGAs," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 133–140. [Online]. Available: <https://doi.org/10.1109/FCCM.2018.00030>

- [29] AMD, “Agentic FPGA backend optimization competition @ FPL’26,” GitHub repository, 2026, accessed: Sep. 29, 2026. [Online]. Available: https://github.com/Xilinx/fpl26_optimization_contest
- [30] RapidWright, “Pre-implemented modules – Part I,” RapidWright Documentation, 2026, accessed: Sep. 29, 2026. [Online]. Available: https://www.rapidwright.io/docs/PreImplemented_Modules_Part_I.html



Shu Gu received the M.Eng. (Hons.) degree in electronics and computer science from the University of Edinburgh, Edinburgh, U.K., in 2025. He is currently pursuing a Ph.D. degree with the School of Engineering at the same university. His research interests include electronic design automation, FPGA design and timing optimisation, and the application of large language models to hardware design workflows.



Matthew Youngman (Graduate Student Member, IEEE) received the M.Eng. degree in general engineering, specialising in electronics, from Durham University, Durham, U.K., in 2024. Following a software engineering role in industry, he is currently pursuing a Ph.D. degree with the School of Engineering, University of Edinburgh, Edinburgh, U.K. His research interests include large language model multi-agent systems for backend electronic design automation, particularly how agents can coordinate deterministic tools across the EDA design flow.



Yuqin Zhao is a Research Associate at the APRIL AI Hub, University of Edinburgh, U.K. He received his Ph.D. in Electronic and Electrical Engineering from the University of Sheffield, U.K. His research focuses on high-performance heterogeneous computing and AI-assisted hardware design, particularly FPGA/SoC systems, hardware–software task allocation, high-level synthesis, and design automation. His current research interests include generative AI and large language models for electronic design automation, intelligent hardware–software co-design,

FPGA acceleration, and AI-assisted hardware verification.



Cristian Sestito (Member, IEEE) is a Research Fellow at the Centre for Electronics Frontiers, University of Edinburgh, U.K. He earned his B.Sc. (2016) and M.Sc. (2019) in Electronic Engineering, followed by a Ph.D. in Information and Communication Technologies (2023), from the University of Calabria, Italy. His PhD focused on the design and FPGA implementation of convolutional neural networks, with particular emphasis on dilated and transposed convolutions. During 2021–2022, he was a Visiting Scholar at Heriot-Watt University, Edinburgh, U.K. where he worked on neural network compression techniques.

His current research interests include digital AI accelerator design, software simulators for neuromorphic AI, and AI-assisted electronic design automation frameworks.



Ali Siddique is currently a Research Associate in AI for Circuits and Systems Design Automation with the Institute for Integrated Micro and Nano Systems, University of Edinburgh, U.K. Previously, he was a Research Associate with the Advanced Processor Technologies Group, University of Manchester, U.K., where he worked on neuromorphic and edge AI hardware within the EU-funded CONVOLVE project. His research focuses on neuromorphic computing, spiking neural networks, online and on-chip learning, FPGA-based AI accelerators, and hardware–software co-design. His research interests also include agentic AI for hardware design automation and hardware-aware optimization using metaheuristic methods such as Grey Wolf Optimization. He is particularly interested in developing efficient and deployable learning systems that combine algorithmic flexibility with low-cost hardware implementation.



Ramesh Krishnamurthy is a Research Associate in AI for Electronics Testing and Verification Automation with the APRIL AI Hub at the University of Edinburgh, U.K. He received the B.E. degree in Electronics and Communication Engineering from Visvesvaraya Technological University, India, in 2014, the M.Sc. degree in Embedded Systems from the University of Twente, The Netherlands, in 2021, and the Ph.D. degree in Computer Science from the University of Antwerp, Belgium, in 2025.

His doctoral research focused on formal analysis of nonlinear model-predictive control, combining learning-based controller approximation, formal verification, and worst-case execution-time analysis for embedded and safety-critical systems. His current research interests include formal verification, AI-assisted hardware verification, agentic AI for electronic design automation, and trustworthy computing systems.



Danial Chitnis received an MSc in Microelectronics at Bristol, UK, in 2008, and a DPhil in Engineering Science from the University of Oxford in 2013, and then joined the Department of Biomedical Physics at UCL, London, as a postdoctoral researcher. In 2017, he joined The University of Edinburgh as a Chancellor’s Fellow in Electronics. His research focuses on using machine learning and artificial intelligence to improve the design and testing of integrated circuits and electronic systems.



Themis Prodromakis (Senior Member, IEEE) received the bachelor’s degree in electrical and electronic engineering from the University of Lincoln, U.K., the M.Sc. degree in microelectronics and telecommunications from the University of Liverpool, U.K., and the Ph.D. degree in electrical and electronic engineering from Imperial College London, U.K. He then held a Corrigan Fellowship in nanoscale technology and science with the Centre for Bio-Inspired Technology, Imperial College London, and a Lindemann Trust Visiting Fellowship

with the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, USA. He was a Professor of nanotechnology at the University of Southampton, U.K. He holds the Regius Chair of Engineering at the University of Edinburgh. He is currently a Royal Academy of Engineering Chair in emerging technologies and a Royal Society Industry Fellowship. His background is in electron devices and nanofabrication techniques. His current research interests include memristive technologies for advanced computing architectures and biomedical applications. He is a fellow of the Royal Society of Chemistry, the British Computer Society, the IET, and the Institute of Physics.