LibPilot: Knowledge-Constrained Dual-Agent Reasoning for Standard Cell Library Tuning

Yanfang Liu[1], Zijin Cheng[2], Mingjun Wang[1], Rongliang Fu[1*], Chao Wang[2], Bei Yu[1]
[1]The Chinese University of Hong Kong [2]Southeast University

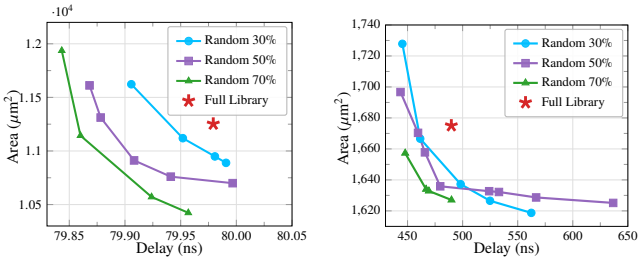
Abstract

Modern logic synthesis is frequently constrained by the scale of contemporary standard cell libraries. Providing industrial synthesizers with hundreds of cell variants expands the mapping search space exponentially, often trapping optimization in suboptimal local minima. Recent library tuning methods mitigate this by searching for advantageous cell subsets, yet they operate as black-box combinatorial optimizers with poor sample efficiency and no principled understanding of inter-cell relationships. We propose **LibPilot**, which reframes library tuning as a knowledge–constrained reasoning task rather than stochastic search. An offline stage constructs a Library Knowledge Graph that encodes equivalence, absorption, and decomposition relations among cell variants. Online, a dual-agent architecture diagnoses Power, Performance, and Area (PPA) bottlenecks from post-synthesis reports and prescribes graph-constrained cell substitutions, producing interpretable, design-aware library adjustments at every iteration. A multi-stage Pareto exploration strategy balances global design-space coverage with local Pareto-front refinement. Across seven industrial circuits at 7 nm, 28 nm, and 45 nm, LibPilot achieves up to 50.85% timing improvement, 13.40% area reduction, and 29.93% power savings over the full-library baseline, while requiring 3× fewer synthesis invocations than the state-of-the-art reinforcement learning approach.

1 Introduction

Logic synthesis serves as the fundamental bridge between abstract Register Transfer Level (RTL) descriptions and physical layout implementations in modern Application-Specific Integrated Circuit (ASIC) design flows [1, 2]. The primary objective of this phase is to transform a technology-independent behavioral model into a highly optimized gate-level netlist. The quality of the technology mapping step directly governs the achievable Power, Performance, and Area (PPA) of the fabricated silicon [3]. As semiconductor manufacturing scales into the deep sub-micron regime, stringent physical constraints and narrowing timing margins amplify the algorithmic complexity of synthesis optimization. Consequently, discovering synthesis strategies that satisfy conflicting multi-objective PPA constraints remains a persistent challenge in Electronic Design Automation (EDA) [4].

At the core of the technology mapping phase lies the standard cell library, which provides the fundamental building blocks for physical netlist realization [5]. To accommodate multi-objective constraints at advanced technology nodes, modern foundries supply extensively characterized libraries comprising hundreds or thousands of cell variants that span diverse Boolean logic structures, variable drive strengths, and multiple threshold voltages. Although this combinatorial scale offers granular flexibility for PPA optimization, a counter-intuitive phenomenon consistently emerges in practice: providing a larger cell library to the synthesizer does not necessarily yield superior post-mapping results [6]. Contemporary technology mappers employ greedy heuristics to traverse the exponential mapping space; presenting an unconstrained full library overwhelms these heuristics and causes the optimizer to settle in suboptimal local minima. As illustrated in Fig. 1, we synthesize two representative designs under three randomly sampled library subsets (30%, 50%, and 70% of the full library) and compare the resulting delay and area against the full-library baseline. Across both designs and technology nodes, numerous partial-library configurations Pareto-dominate the full-library result, achieving simultaneously lower delay and smaller area. This observation motivates the central hypothesis of our work: a strategically curated library subset, informed by structural and electrical knowledge of the target design, can consistently outperform the full library in post-mapping PPA.



(a) design1, 7nm (b) design2, 28nm

Figure 1. Post-synthesis delay versus area for two designs mapped with Synopsys Design Compiler under different library sampling ratios. *Random X%* denotes synthesis using a randomly sampled X% subset of the full standard cell library. The Full Library baseline (red star) is Pareto-dominated by multiple partial-library configurations across both designs and technology nodes.

Motivated by the promise of strategically restricted libraries, prior research has explored two complementary directions: dynamic cell extension and discrete subset selection. Cell extension methods synthesize novel composite logic gates to enrich the available mapping space [7]; however, these generative techniques introduce prohibitive overhead in physical cell layout and subsequent electrical characterization [8]. Standard cell library tuning, by contrast, focuses on extracting an optimal subset from pre-characterized foundry libraries. Recent work [6] has adopted reinforcement learning (RL) to navigate this combinatorial space with preliminary success.

We contend, however, that standard cell library tuning is fundamentally a structured reasoning task, not a stochastic combinatorial search problem, and that it demands two forms of understanding

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

ICCAD '26, San Jose, CA, USA

ACM ISBN 979-8-4007-2873-0/2026/11
https://doi.org/10.1145/3831252.3834206

that existing methods lack. First, it requires inter-cell relational knowledge: cells within a technology library are linked by equivalence, absorption, and decomposition relationships (e.g., an AND2 driving a NOR2 is functionally replaceable by a single AOI21), yet RL formulations treat each cell's inclusion as an independent binary action, discarding this relational structure entirely. Second, it requires design-specific diagnostic insight: an effective tuning decision must trace an observed PPA bottleneck back to the specific cells responsible and reason about what substitution will alleviate it, whereas RL agents learn solely from scalar reward signals and cannot attribute an observed PPA change to specific cell-level decisions. Without relational knowledge, stochastic methods cannot anticipate the downstream mapping consequences of cell exclusion, leading to frequent functional violations; without diagnostic insight, they explore blindly, resulting in poor sample efficiency and opaque decision logic.

These observations motivate a fundamentally different formulation. Rather than searching the 2^N binary inclusion space through trial and error, we encode inter-cell relationships as a Library Knowledge Graph that defines legal substitution pathways. Large Language Model (LLM)-based agents then consume structured post-synthesis reports and query this graph to prescribe interpretable, design-aware library adjustments. This reformulation constrains the search from an exponential binary space to the subspace of graph-legal substitutions, where every decision is grounded in both structural domain priors and empirical PPA feedback. Building on this insight, we present **LibPilot**, a standard cell library tuning framework whose dual-agent architecture decouples the reasoning task into bottleneck diagnosis and graph-constrained compensation. The framework further integrates an automated functional completeness check and a multi-stage Pareto exploration strategy to guarantee mapping legality and to promote convergence toward the global Pareto frontier. Overall, the primary contributions of this paper are as follows:

- We construct a Library Knowledge Graph from full technology library data, building three explicit equivalence, absorption, and decomposition relations that equip the downstream agents with structured domain priors.
- We introduce a collaborative dual-agent reasoning architecture in which a diagnostic Ban Agent identifies PPA bottlenecks from post-synthesis reports and a prescriptive Replace Agent queries the relational graph to prescribe targeted cell replacements. This architecture improves sample efficiency and decision interpretability relative to state-of-the-art RL-based method.
- We employ an Upper Confidence Bound for Trees (UCT)-guided logic-family injection module complements the agents by introducing previously absent cell families to escape local optima.
- We implement a multi-stage Pareto exploration strategy that integrates coarse-grained search, hypervolume-based filtering, and fine-grained refinement, balancing broad design-space coverage with targeted local exploitation.
- We evaluate LibPilot on seven industrial circuits across 7nm, 28nm, and 45nm technology nodes. Experimental results demonstrate that our framework achieves up to 50.85% timing improvement, 13.40% area reduction, and 29.93% power savings over the full-library baseline, while requiring 3× fewer synthesis iterations compared to state-of-the-art RL-based approach.

2 Preliminaries

2.1 Standard Cell Library

A standard cell library provides pre-designed gates and components optimized for a specific fabrication process. Cells fall into two primary functional categories: combinational and sequential. Combinational cells, including basic logic gates (such as AND, OR, and XOR) and composite gates (such as full adders), perform stateless Boolean operations. Sequential cells, mainly flip-flops and latches, contain memory elements required for state retention and synchronous circuit execution.

Beyond logical functionality, standard cells are characterized by their physical and electrical properties. For any given logic function, a modern technology library typically provides multiple cell variants that realize identical Boolean logic but differ in sizing and driving capability. A cell with higher driving capability can charge and discharge capacitive loads more rapidly, thereby reducing signal delay; this performance advantage, however, incurs a larger silicon footprint and higher power consumption. This diversity in cell variants establishes a discrete design space that enables logic synthesis tools to fine-tune PPA trade-offs in the final implementation.

2.2 Logic Synthesis

Logic synthesis is the automated process of transforming a high-level circuit description, typically specified at the RTL, into a functionally equivalent, technology-specific gate-level netlist [9]. This process is conventionally partitioned into three phases: translation, logic optimization, and technology mapping.

The translation phase initially parses the RTL code to construct a technology-independent representation, often modeled as a Boolean network or a Directed Acyclic Graph (DAG). Subsequently, logic optimization applies algebraic and Boolean transformations, such as constant propagation, common subexpression elimination, and AIG rewriting, to reduce logic complexity while preserving functional equivalence [10]. The final phase, technology mapping, binds the optimized logic network to a concrete library of standard cells. This phase bridges the theoretical design with its physical realization by selecting appropriate cells from the provided library. Modern technology mappers evaluate large combinations of cell variants to realize the circuit topology, aiming to optimize PPA by minimizing gate count and interconnections, reducing power dissipation, and strictly adhering to timing constraints. Because the mapping procedure determines both the gate selections and their physical interconnections, the quality of cell selection profoundly influences the overall cost, performance, and manufacturability of the resulting integrated circuit.

2.3 LLM for EDA Applications

LLMs have recently been adopted within the EDA ecosystem for tasks ranging from HDL code generation [11, 12] to autonomous toolflow orchestration [13]. Beyond syntactic fluency, LLMs offer structured, context-aware reasoning over domain-specific constraints: they can ingest heterogeneous textual specifications (e.g., timing reports, cell datasheets) and synthesize multi-step decision plans in a single inference pass without task-specific training. This zero-shot reasoning capacity makes LLMs well-suited to design-space

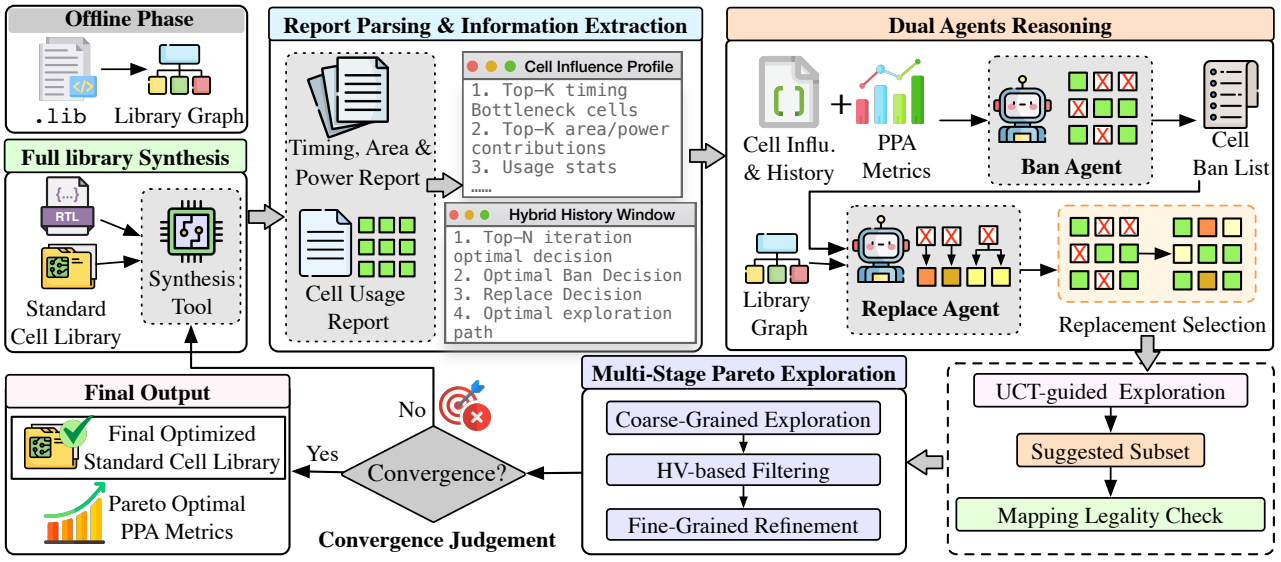


Figure 2. The workflow of the proposed LibPilot framework. The system initiates with an offline Library Graph construction, followed by a closed-loop online optimization cycle. During each iteration, the dual-agent engine diagnoses PPA bottlenecks and prescribes cell replacements utilizing historical context and EDA domain priors. The generated subset subsequently undergoes UCT-guided global exploration, mapping legality check, and a multi-stage Pareto exploration mechanism to iteratively converge upon the optimal technology library.

exploration problems where the action space is combinatorially large yet governed by formal domain rules—precisely the setting of standard cell library tuning. LibPilot leverages this capability by pairing LLM agents with a pre-constructed relational graph that encodes the formal structure of the technology library, confining the model's reasoning to graph-legal substitutions.

3 Problem Formulation

Let $\mathcal{P} = \{c_1, c_2, \ldots, c_N\}$ denote a complete standard cell library comprising N pre-characterized cell variants. Each variant c_i belongs to exactly one logic function family $F \in \mathcal{F}$; members of the same family realize identical Boolean logic but differ in drive strength, threshold voltage, or sizing. Given a target circuit $\mathcal{A}$ specified at the RTL and an industrial technology mapper Φ, the synthesizer produces a gate-level netlist $\mathcal{N}_X = \Phi(\mathcal{A}, \mathcal{P}(X))$ from a restricted library $\mathcal{P}(X) = \{c_i \in \mathcal{P} \mid x_i = 1\}$, where $X \in \{0,1\}^N$ is a binary decision vector controlling cell inclusion.

The quality of $\mathcal{N}_X$ is characterized by three post-synthesis metrics: worst negative slack $WNS(\mathcal{N}_X)$, total silicon area $Area(\mathcal{N}_X)$, and total power consumption $Power(\mathcal{N}_X)$. We formulate library tuning as the following constrained multi-objective optimization problem:

$$\begin{aligned} \min_{X\in\{0,1\}^N} \quad & \{-WNS(\mathcal{N}_X),\ Area(\mathcal{N}_X),\ Power(\mathcal{N}_X)\}, \\ \text{s.t.} \quad & \mathcal{L}(\mathcal{P}(X)) = \text{TRUE}, \\ & WNS(\mathcal{N}_X) \geq \tau_{\text{target}}, \\ & x_i = 1, \quad \forall\, c_i \in \mathcal{P}_{\text{prot}}, \end{aligned} \tag{1}$$

where the three constraints enforce the following requirements. *Mapping legality* $\mathcal{L}(\mathcal{P}(X))$ asserts that the Boolean functions realizable by $\mathcal{P}(X)$ form a functionally complete set, preventing unmapped logic during synthesis. *Timing validity* requires the post-synthesis worst negative slack to meet or exceed the design-specified threshold τ_{target}. *Structural preservation* mandates that every cell in the protected set $\mathcal{P}_{\text{prot}} = \mathcal{P}_{\text{util}} \cup \mathcal{P}_{\text{seq,mand}}$—encompassing physical utility cells and mandatory sequential elements—remains unconditionally included to guarantee implementation legality and DFT compliance.

A decision vector $X^\star$ is *Pareto optimal* if no feasible X can improve any objective without degrading another. The solution quality of the discovered Pareto front $\mathcal{F}_{\text{pareto}}$ is measured by the Hypervolume (HV) indicator, which quantifies the volume of the objective space dominated by all non-dominated solutions relative to a worst-case reference point.

4 Proposed Methodologies

The central design principle of LibPilot is to decompose library tuning into two cognitively distinct sub-tasks—diagnosis (which cells cause PPA bottlenecks) and prescription (what graph-legal substitutions alleviate them)—and to constrain every reasoning step with structured domain knowledge rather than relying on prior-free trial and error. This principle is instantiated through the closed-loop, knowledge-driven architecture illustrated in Fig. 2.

4.1 Library Knowledge Graph Construction

The foundation of the LibPilot framework is the transformation of unstructured physical and logical data from standard technology libraries into a structured relational representation. This offline preprocessing phase systematically extracts formal domain priors from the complete library $\mathcal{P}$, enabling the subsequent LLM agents to perform knowledge-constrained deduction rather than stochastic exploration. This procedure is illustrated in Fig. 3.

Functional Logic Analysis. After parsing the technology library, we partition the available standard cell variants into three mutually exclusive subsets based on their functional attributes and operational constraints: $\mathcal{P} = \mathcal{P}_{\text{comb}} \cup \mathcal{P}_{\text{seq}} \cup \mathcal{P}_{\text{util}}$.

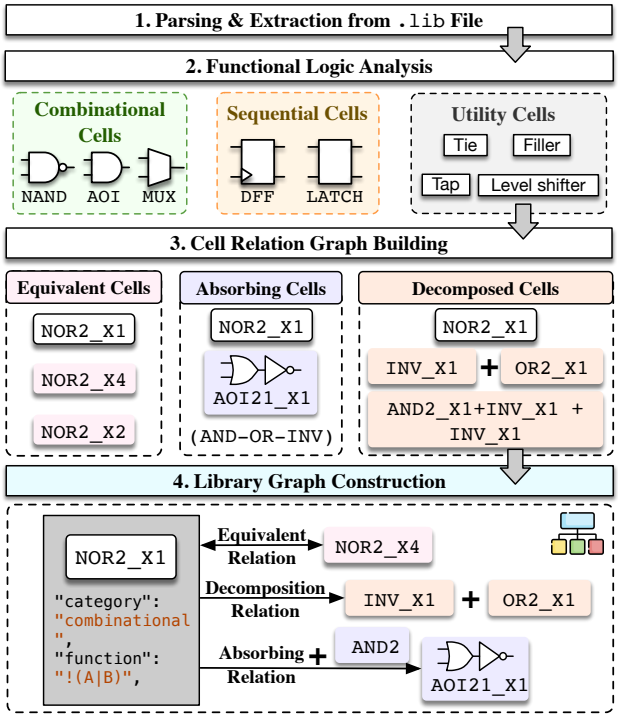


Figure 3. Construction flow of the Library Knowledge Graph. The procedure parses the `.lib` file, classifies cells into three functional categories, canonicalizes Boolean expressions, and automatically populates three relational edge sets through structural pattern matching.

The subset $\mathcal{P}_{\text{comb}}$ comprises **combinational cells** (e.g., NAND, AOI), which exhibit extensive functional substitutability and constitute the primary search space for PPA optimization. $\mathcal{P}_{\text{seq}}$ consists of **sequential cells** (predominantly flip-flops and latches). Due to their strict sensitivity to setup and hold timing margins and Design for Testability (DFT) constraints, their optimization is handled conservatively and restricted to drive strength adjustments. Finally, $\mathcal{P}_{\text{util}}$ encompasses **utility cells** (e.g., tie, filler, and tap cells) indispensable for physical implementation legality. To preserve the structural integrity of the synthesized netlist $\mathcal{N}$, these cells are permanently protected. Formally, the elimination decisions generated by the Ban Agent at any iteration k must satisfy $C_{\text{ban}}^{(k)} \cap \mathcal{P}_{\text{util}} = \emptyset$.

Automated Graph Formulation. To facilitate rule-constrained derivation of the compensation set $C_{\text{rep}}^{(k)}$, we implement an automated extraction pipeline that derives relational mappings directly from the Boolean signatures of the cells. We formalize the Library Knowledge Graph as a directed multigraph $\mathcal{G} = (\mathcal{P}, \mathcal{E})$, where each node represents a specific cell variant $c_i \in \mathcal{P}$, and the edges $\mathcal{E}$ denote functionally valid topological transformations. Rather than relying on manual enumeration, our engine autonomously populates three distinct relational edge sets for any given logic cell c_i:

First, the **Equivalent Cell** set $\mathcal{R}_{\text{EC}}(c_i)$ collects variants with identical Boolean logic but varying drive strengths or threshold voltages (e.g., mapping NOR2_X1 to NOR2_X4), providing the LLM agents with direct timing-versus-area trade-off options. Second, the **Absorbing Cell** set $\mathcal{R}_{\text{AC}}(c_i)$ contains structurally complex gates capable of subsuming c_i alongside an adjacent logic operation. For instance, an AND2 driving a NOR2 can be absorbed by a single AOI21, effectively reducing the logical stage depth. Third, the **Decomposed Cell** set $\mathcal{R}_{\text{DC}}(c_i)$ captures functional decompositions, where a composite gate is factored into a series of simpler elements (e.g., decomposing a NOR2 into an OR2 followed by an INV).

Collectively, these relations define the edge set $\mathcal{E}$ of the graph $\mathcal{G}$. Each node c_i encapsulates physical parameters (area, delay, leakage power), while the edges encode legal substitution pathways. This automated graph architecture equips the Replace Agent with explicit domain knowledge, ensuring that every replacement decision within $C_{\text{rep}}^{(k)}$ is both functionally equivalent and physically realizable.

4.2 Dual-Agent Reasoning Engine

The core optimization loop of LibPilot is driven by a collaborative dual-agent architecture that decouples library tuning into a *diagnostic* phase, which identifies cells to remove, and a *prescriptive* phase, which selects replacements. This separation is motivated by two failure modes of simpler alternatives. A single-agent formulation that directly outputs the target subset requires simultaneous reasoning about removal, compensation, and functional completeness within one inference pass, resulting in excessive prompt complexity and frequent output-parsing failures. A ban-only strategy without explicit compensation causes monotonic shrinkage of the allowed set, progressively degrading mapper flexibility. The decoupled design reduces each agent's decision space while guaranteeing that every removal is paired with a graph-constrained compensation.

PPA Bottleneck Diagnosis. During each synthesis iteration k, the synthesizer produces $\mathcal{N}^{(k)} = \Phi(\mathcal{A}, \mathcal{P}^{(k)})$, from which the framework extracts $\text{WNS}(\mathcal{N}^{(k)})$, $\text{Area}(\mathcal{N}^{(k)})$, and $\text{Power}(\mathcal{N}^{(k)})$. By parsing the worst-slack timing paths and aggregating per-cell area and power contributions, the system constructs a compact *Cell Influence Profile* that ranks every instantiated variant $c_i \in \mathcal{P}_{\text{used}}(\mathcal{N}^{(k)})$ by timing criticality, area share, and power share. The system appends a hybrid history window that interleaves the most recent iteration summaries with the highest-quality legal configurations discovered so far, providing both local continuity and global anchoring for cross-iteration reasoning. This compressed representation serves as the primary input to both agents.

Ban Agent. As shown in the upper half of Fig. 4, the Ban Agent produces the elimination set $C_{\text{ban}}^{(k)} \subset \mathcal{P}_{\text{used}}(\mathcal{N}^{(k)})$. Receiving the Cell Influence Profile, the Library Knowledge Graph $\mathcal{G}$, and the current optimization directive, the agent correlates observed PPA bottlenecks with specific cell usages through structured reasoning over the provided context. Under severe timing violations ($\text{WNS} < 0$), the agent targets cells that appear frequently on critical paths with excessive delay; under an area-reduction directive, it targets high-drive variants over-utilized in non-critical paths. Each decision is accompanied by a justification and a specification of the desired replacement property that the Replace Agent subsequently consumes.

All outputs undergo post-hoc rule enforcement. Utility and structurally protected cells are unconditionally excluded from the ban list. Cells whose instance count exceeds a configurable fraction of the total population are shielded to prevent catastrophic remapping. Sequential cell bans are permitted only when at least one functionally identical sibling remains in the allowed set. If the LLM fails to

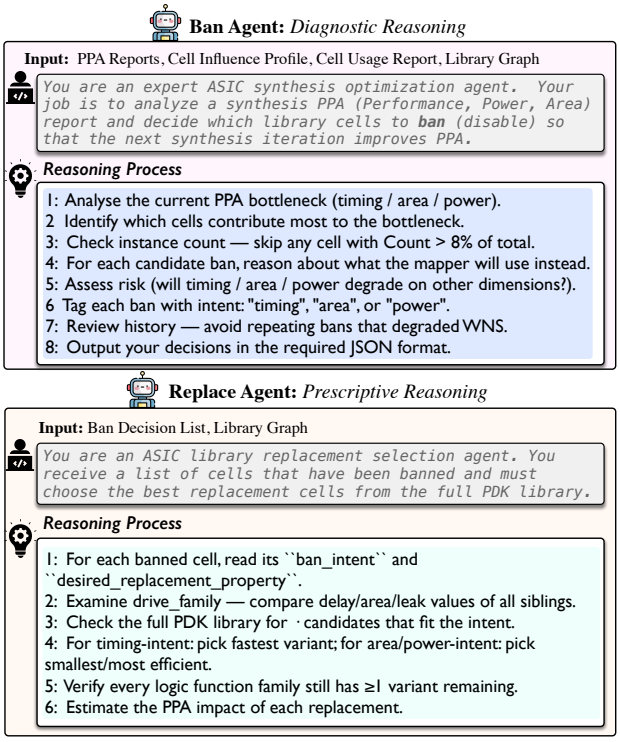


Figure 4. Reasoning process of the dual agents. The Ban Agent performs diagnostic reasoning to identify suboptimal cells, and the Replace Agent performs prescriptive reasoning to select graph-constrained substitutions.

produce a parseable response, an empty ban list is returned so that the optimization loop never stalls on a parsing failure.

Replace Agent. In the lower half of Fig. 4, the Replace Agent prescribes a compensation set $C_{\text{rep}}^{(k)} \subset \mathcal{P}$ for each banned cell. Using $\mathcal{G}$, it maps eliminated cells to legal substitution candidates from the three relational categories: a higher-drive equivalent from $\mathcal{R}_{\text{EC}}$ or an absorption configuration from $\mathcal{R}_{\text{AC}}$ for timing-critical cells; a lower-drive equivalent or a decomposition from $\mathcal{R}_{\text{DC}}$ for area-inefficient cells, provided sufficient timing margin remains. To guide selection, the agent receives a structured candidate summary that marks each variant as currently present, newly banned, or importable from the full library, enabling precise navigation of the substitution space.

Should the LLM fail to produce a valid plan, a fallback activates in strict priority: same-family drive substitution, then absorption via upper cells, then decomposition into simpler primitives. All replacements are validated against the full library, and sequential replacements are further constrained to preserve the original logic function. The updated library subset is computed as $\mathcal{P}^{(k+1)} = \left(\mathcal{P}^{(k)} \setminus C_{\text{ban}}^{(k)}\right) \cup C_{\text{rep}}^{(k)}$. A no-shrink safeguard guarantees $|\mathcal{P}^{(k+1)}| \geq |\mathcal{P}^{(k)}|$: if the updated subset is smaller, cells are back-filled from the full library by drive-family proximity. This collaborative decoupling pushes the iterative refinement toward the Pareto frontier while preserving Boolean equivalence at every step.

4.3 UCT-Driven Logic Injection

The dual-agent engine refines cells already present in the netlist but cannot discover logic function families that lack any representative in the active subset. Because the Cell Influence Profile is constructed solely from mapper-selected cells, absent families never surface through ban-and-replace operations; the search consequently remains susceptible to local optima.

To overcome this limitation, LibPilot incorporates a UCT-guided global exploration module that operates at the granularity of logic function families $\mathcal{F}$, reducing the action space by an order of magnitude relative to individual cell selection. Candidate families are classified as either *absent* (no variant present in the current subset) or *incomplete* (present but missing certain drive-strength variants). A configurable ratio governs the mix between the two categories, with incomplete families receiving a discounted prior to favor genuinely novel logic introductions. For each candidate family F, the algorithm computes: $\text{UCT}(F) = \bar{R}(F) + \lambda\sqrt{\frac{\ln N_k}{n(F)}}$, where $\bar{R}(F)$ is the empirical reward aggregated from historical configurations containing any variant of F, N_k is the total iteration count, and $n(F)$ is the visitation count of F. The first term exploits historically advantageous structures; the second encourages exploration of undervisited functions. Top-scoring families are *materialized* into concrete cell variants by instantiating a combination of drive strengths according to the current optimization directive. The resulting injection set $C_{\text{UCT}}^{(k)}$ is merged into the active library:

$$\mathcal{P}^{(k+1)} = \left(\mathcal{P}^{(k)} \setminus C_{\text{ban}}^{(k)}\right) \cup C_{\text{rep}}^{(k)} \cup C_{\text{UCT}}^{(k)}. \tag{2}$$

Two design choices ensure practical efficiency. First, the module activates only upon detecting diminishing returns across recent iterations, avoiding unnecessary perturbation during productive local search. Second, no additional synthesis invocations are incurred: the observed PPA outcome is attributed to injected families via weighted decomposition, and reward statistics are updated through exponential moving averages. This cross-iteration learning refines the exploration policy exclusively from outcomes already produced by the main loop.

4.4 Mapping Legality Check

Aggressive cell elimination may render the remaining subset functionally incomplete, causing the synthesizer to fail on unmappable logic. LibPilot therefore validates every candidate subset before synthesis invocation by verifying the presence of universal logic primitives (e.g., NAND/NOR families), fundamental inverting structures, and essential sequential elements (e.g., D-flip-flops and scan-chain registers); utility cells such as TIEHI and TIELO are permanently protected from elimination. Whenever a missing Boolean function is detected, an automated repair routine retrieves the lowest-cost variant fulfilling that function from the full library and appends it to the subset, ensuring that every synthesis invocation operates on a provably complete library without manual intervention.

4.5 Multi-Stage Pareto Exploration

To systematically navigate the combinatorial search space of standard-cell selection and mitigate premature convergence to local minima, LibPilot employs a three-stage exploration strategy that balances broad design-space coverage with localized Pareto-front exploitation. Algorithm 1 details the complete procedure.

Algorithm 1: Multi-Stage Pareto Exploration

Input: Circuit $\mathcal{A}$, full library $\mathcal{P}$, relation graph $\mathcal{G}$, mapper Φ
Output: Pareto-optimal library-subset front $\mathcal{F}_{\text{pareto}}$

1 $\mathcal{F}_{\text{pareto}} \leftarrow \emptyset$;
2 $\mathcal{S}_{\text{coarse}} \leftarrow \emptyset$;
// Stage 1: Coarse-Grained Exploration
3 **for** ***each*** *directive* $d \in \mathcal{D}$ **do**
4 $\quad \mathcal{S}_d \leftarrow$ Execute R_c iterations on $\mathcal{P}$ with ratio α_{coarse} under d;
5 $\quad \mathcal{S}_{\text{coarse}} \leftarrow \mathcal{S}_{\text{coarse}} \cup \mathcal{S}_d$;
// Stage 2: HV-based Filtering
6 Rank all legal $\mathcal{P}_{\text{cand}} \in \mathcal{S}_{\text{coarse}}$ by calculated HV value ;
7 $\mathcal{S}_{\text{fine}} \leftarrow$ top K subsets; Update $\mathcal{F}_{\text{pareto}}$ via non-dominated sorting;
// Stage 3: Fine-Grained Refinement
8 **for** ***each*** *seed* $\mathcal{P}_{\text{seed}} \in \mathcal{S}_{\text{fine}}$ **do**
9 $\quad \mathcal{P}^{(0)} \leftarrow \mathcal{P}_{\text{seed}}$; $\mathcal{P}_{\text{best}} \leftarrow \mathcal{P}_{\text{seed}}$;
10 $\quad$ **for** $k = 0$ **to** $R_f - 1$ **do**
11 $\quad\quad \mathcal{N}_k \leftarrow \Phi(\mathcal{A}, \mathcal{P}^{(k)})$; Update $\mathcal{F}_{\text{pareto}}$ with PPA($\mathcal{N}_k$);
12 $\quad\quad$ **if** score($\mathcal{N}_k$) *improves* **then** $\mathcal{P}_{\text{best}} \leftarrow \mathcal{P}^{(k)}$;
13 $\quad\quad$ **else** $\mathcal{P}^{(k)} \leftarrow \mathcal{P}_{\text{best}}$;
14 $\quad\quad d_k \leftarrow \text{AdaptiveStrategy}(\text{PPA}(\mathcal{N}_k))$;
15 $\quad\quad C_{\text{ban}}^{(k)} \leftarrow \text{BanAgent}(\mathcal{N}_k, \alpha_{\text{fine}}, d_k)$;
16 $\quad\quad C_{\text{rep}}^{(k)} \leftarrow \text{ReplaceAgent}(C_{\text{ban}}^{(k)}, \mathcal{G})$;
17 $\quad\quad C_{\text{UCT}}^{(k)} \leftarrow \emptyset$;
18 $\quad\quad$ **if** *diminishing returns detected* **then**
19 $\quad\quad\quad C_{\text{UCT}}^{(k)} \leftarrow \text{UCTInject}(\mathcal{P}^{(k)}, \mathcal{G})$;
20 $\quad\quad \mathcal{P}^{(k+1)} \leftarrow (\mathcal{P}^{(k)} \setminus C_{\text{ban}}^{(k)}) \cup C_{\text{rep}}^{(k)} \cup C_{\text{UCT}}^{(k)}$;
21 **return** $\mathcal{F}_{\text{pareto}}$

Hybrid History Window. At each iteration, both the Ban Agent and the Replace Agent receive a curated history window rather than the raw iteration log. The default hybrid policy blends the k_r most recent records with the k_b globally best legal records, subject to a total budget $k_{\max}$, thereby preserving local search continuity while anchoring decisions to competitive configurations observed earlier in the search.

Coarse-Grained Exploration. After initializing the Pareto front and the candidate pool, the framework launches parallel trajectories, each governed by a distinct optimization directive $d \in \mathcal{D}$ targeting timing recovery, area reduction, power minimization or balanced PPA metrics (line 3). Within every trajectory, the Ban Agent operates under a high elimination ratio α_{coarse} for R_c iterations, removing a substantial fraction of instantiated cells per iteration to force the mapper Φ into significant structural rewiring (line 4). All resulting subsets are aggregated into the coarse candidate set $\mathcal{S}_{\text{coarse}}$ (line 5).

HV-Based Filtering. Once the coarse phase completes, all legally synthesized configurations in $\mathcal{S}_{\text{coarse}}$ are ranked by their HV contribution to the multi-objective PPA space (line 6). The top K subsets are selected as seeds for the subsequent fine-grained stage, and the global Pareto front $\mathcal{F}_{\text{pareto}}$ is simultaneously updated through non-dominated sorting over all evaluated candidates (line 7).

Fine-Grained Refinement. Starting from each selected seed (lines 8–9), the search transitions to a conservative regime with $\alpha_{\text{fine}} \ll \alpha_{\text{coarse}}$. At every iteration k, the current subset $\mathcal{P}^{(k)}$ is synthesized through the mapper Φ and the resulting PPA vector updates $\mathcal{F}_{\text{pareto}}$ (line 10). A greedy-accept mechanism retains the best-performing configuration: if the composite score improves, $\mathcal{P}_{\text{best}}$ is updated accordingly; otherwise the search rolls back to $\mathcal{P}_{\text{best}}$ to prevent error accumulation (lines 11–13). An adaptive strategy selector then adjusts the optimization directive d_k based on the current PPA profile (line 14), after which the Ban Agent identifies localized bottleneck cells for removal (line 15) and the Replace Agent substitutes them with logically equivalent drive-strength variants drawn from the relation graph $\mathcal{G}$ (line 16). When diminishing returns are detected across consecutive iterations, the UCT module injects previously absent logic families into the subset to escape local optima (lines 17–19). The updated subset is then assembled by applying all three operators jointly (line 20), and the procedure repeats until the iteration budget R_f is exhausted. Upon termination, the accumulated Pareto front $\mathcal{F}_{\text{pareto}}$ is returned.

Table 1. PPA Comparison with Four LLM Backbones.

Design PDK	LLM Backbones	WNS↑ (ps)	Area↓ (μm^2)	Power↓ (mW)
dp_ram 7 nm	Deepseek-v3.2 [14]	-0.48	10441.15	14.88
	GPT-5.4-mini [15]	-0.45	10441.15	14.92
	GLM-5 [16]	-0.46	10445.58	14.86
	Qianwen-3.6-plus [17]	-0.48	10353.90	14.85
rvg 28 nm	Deepseek-v3.2 [14]	-58.96	6621.80	31.48
	GPT-5.4-mini [15]	-57.67	6690.60	30.32
	GLM-5 [16]	-53.16	6176.39	30.83
	Qianwen-3.6-plus [17]	-52.88	6104.20	30.37
fsm 45 nm	Deepseek-v3.2 [14]	-68.00	6394.72	13.92
	GPT-5.4-mini [15]	-70.50	7242.19	14.84
	GLM-5 [16]	-67.23	7696.03	15.02
	Qianwen-3.6-plus [17]	-56.54	6091.70	12.87

5 Experimental Results

5.1 Experimental Setup

PDKs and Benchmark Circuits. We assemble a benchmark suite of seven industrial RTL designs that span diverse circuit functionalities and structural complexities. The designs include a control-intensive finite-state machine (`fsm`, 3,672 cells), a memory-oriented dual-port RAM (`dp_ram`, 13,628 cells), an arithmetic floating-point subtractor (`fpu_sub`, 8,412 cells), a network-processing packet pipeline (`np_pkt`, 5,127 cells), a register-file adapter (`ada_rf`, 8,185 cells), a random vector generator (`rvg`, 8,526 cells), and an ingress buffer (`ing_buf`, 4,480 cells). Total cell counts range from 3,672 to 13,628, while sequential cell ratios vary from 5.63% (`ing_buf`) to 20.65% (`rvg`), covering a wide spectrum of combinational-sequential compositions, logic depths, and timing criticality profiles.

Each design is synthesized with three commercially characterized standard cell libraries at the industrial 7 nm, 28 nm, and 45 nm technology nodes, producing 21 design-technology configurations in total. This multi-node setup enables rigorous assessment of framework robustness across substantially different library sizes, physical constraint regimes, and PPA trade-off landscapes.

Baselines and Configurations. We compare LibPilot against two baselines and conduct ablation studies on its key components.

(1) **Full Library** is the default industrial flow that supplies the complete, unmodified standard cell library to the synthesizer, serving as the PPA reference for all methods.

Table 2. Detailed PPA comparison results on seven designs across three PDKs. Higher WNS and lower area/power indicate better performance. Best and second-best entries for each design and metric are highlighted accordingly.

Design	Full library #DC=1			MapTune[6] #DC=120			LibPilot w/o Dual-Agents #DC=40			LibPilot w/o UCT Module #DC=40			LibPilot (Ours) #DC=40		
	WNS↑ (ps)	Area↓ (μm^2)	Power↓ (mW)	WNS↑ (ps)	Area↓ (μm^2)	Power↓ (mW)	WNS↑ (ps)	Area↓ (μm^2)	Power↓ (mW)	WNS↑ (ps)	Area↓ (μm^2)	Power↓ (mW)	WNS↑ (ps)	Area↓ (μm^2)	Power↓ (mW)
7 nm Library															
rvg	-8.87	13288.33	28.81	**-7.10**	13425.26	29.00	-8.76	13979.30	29.42	-8.12	13348.05	**28.74**	-7.76	**13235.61**	28.81
dp_ram	-0.50	11896.58	15.06	-0.49	11752.41	15.31	-0.50	10484.77	14.92	**-0.48**	10506.70	14.87	**-0.48**	**10353.90**	**14.85**
ada_rf	-0.47	8926.23	13.17	-0.43	10398.22	13.47	-0.45	9227.16	13.23	-0.30	**8274.21**	**13.03**	**-0.27**	8318.53	**13.03**
fsm	-12.56	5500.28	**7.83**	**-10.73**	**3820.19**	8.00	-12.39	5083.40	7.85	-11.68	5161.55	7.88	-11.71	4763.34	**7.83**
fpu_sub	-30.27	9132.68	11.91	-31.93	**8651.42**	**11.70**	-30.46	9585.71	12.00	-30.16	9034.93	11.87	**-29.53**	8913.86	11.83
np_pkt	-67.66	5469.25	6.82	-70.25	**5194.45**	6.83	-67.62	5488.15	6.98	**-67.17**	5388.77	6.85	-67.27	5395.07	**6.81**
ing_buf	-2.63	4349.51	3.98	-3.68	5026.02	4.11	-2.78	**4270.42**	3.99	-2.37	4483.64	3.97	**-1.75**	4426.02	**3.95**
28 nm Library															
rvg	-63.41	6117.17	43.34	-53.28	6153.97	31.47	-62.43	7110.43	47.92	-53.43	**6031.62**	30.98	**-52.88**	6104.20	**30.37**
dp_ram	-94.29	6888.29	24.41	**-85.28**	8780.56	24.79	-94.25	7278.64	24.86	-92.46	**6061.10**	**24.18**	-91.26	6390.09	24.35
ada_rf	-433.76	4765.95	19.56	-214.77	**4382.41**	20.42	-415.90	5357.77	20.87	-256.50	4902.66	18.07	**-213.20**	4511.05	**15.88**
fsm	-103.23	1835.44	11.01	-96.65	3134.12	12.98	-103.11	1856.23	11.06	-90.70	**1762.11**	**10.95**	**-84.34**	1765.13	10.98
fpu_sub	-90.41	**4224.53**	**16.80**	-87.56	5304.10	18.50	-90.17	4398.03	16.98	-86.44	4588.04	17.20	**-86.07**	4458.13	17.15
np_pkt	**-156.59**	2624.33	10.60	-156.33	2741.89	10.47	-157.59	2555.53	10.81	-163.88	**2414.79**	10.57	-164.55	2563.85	**10.44**
ing_buf	-243.83	1810.62	5.80	-217.21	1947.08	5.94	-242.18	1902.35	5.91	-216.46	1761.23	**5.78**	**-215.37**	**1738.55**	5.80
45 nm Library															
rvg	-44.29	17950.21	**51.53**	-42.73	17820.94	52.15	-44.21	19090.78	52.62	-38.40	**17601.71**	52.62	**-35.73**	17808.97	52.23
dp_ram	-69.06	19318.21	29.90	**-54.63**	19595.23	29.98	-68.68	20906.46	29.95	-59.32	**17935.16**	30.36	-56.47	18076.41	**29.50**
ada_rf	-71.89	13615.36	25.92	**-64.02**	14644.78	26.42	-72.00	13678.97	25.93	-94.28	**13311.67**	**25.57**	-75.54	13552.43	25.67
fsm	-85.64	6923.45	14.44	-75.12	6168.31	**12.77**	-84.00	7269.55	14.84	-58.88	6160.45	12.91	**-56.54**	**6091.70**	12.87
fpu_sub	-175.28	**11653.99**	**19.40**	-164.69	16863.34	21.64	-174.61	12469.66	19.92	-164.87	13009.68	20.47	**-162.53**	13432.39	20.34
np_pkt	-330.11	**6806.48**	12.52	-333.38	8380.37	13.10	-330.76	7102.31	12.51	-328.50	7323.25	**12.35**	**-327.57**	7188.84	12.44
ing_buf	-126.77	**5478.84**	6.68	**-110.22**	5805.45	6.70	-126.73	6652.58	7.15	-122.25	5645.39	**6.62**	-120.12	5612.22	6.67

(2) **MapTune** [6] is the state-of-the-art RL-based library tuning framework. We adopt its best exploration mode from the latest publicly released implementation.

To isolate component contributions, we also evaluate:

(1) *LibPilot w/o Dual-Agents* replaces the Ban and Replace Agents with random elimination coupled with rule-based drive-strength fallback, isolating the effect of knowledge-constrained reasoning.
(2) *LibPilot w/o UCT Module* retains the complete dual-agent engine but deactivates UCT-driven logic injection, quantifying the impact of global exploration on convergence quality.

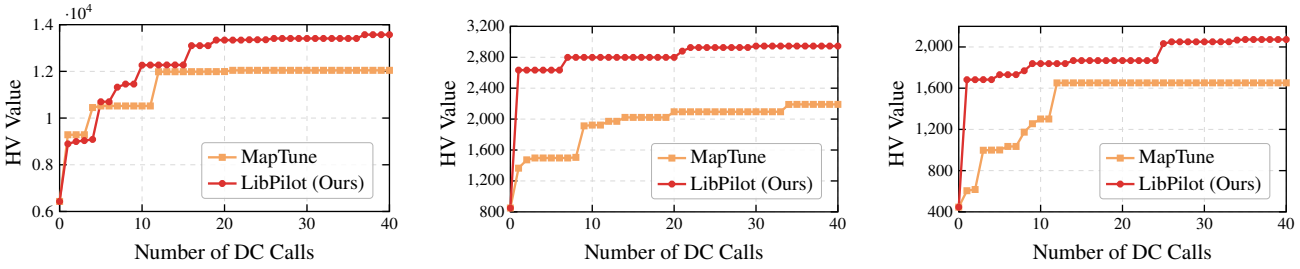
Implementation Details. All synthesis experiments are conducted using Synopsys **Design Compiler (DC)** as the industrial technology mapper. For each design–technology pair, the clock period constraint is set to an aggressive target that induces non-trivial timing violations under the full-library baseline, thereby creating a challenging optimization regime for all evaluated methods. Specifically, we adopt clock periods of 80 ps, 200 ps, and 400 ps for the 7 nm, 28 nm, and 45 nm nodes, respectively. The offline Library Knowledge Graph construction, including `.lib` parsing, functional logic classification, and automated relational edge population, is performed once per technology library and adds small one-time overhead relative to the iterative synthesis loop.

For the LLM reasoning backbone, we evaluate four frontier-class models: DeepSeek-v3.2 [14], GPT-5.4-mini [15], GLM-5 [16], and Qwen-3.6-plus [17]. As shown in TABLE 1, Qwen-3.6-plus consistently achieves a good PPA trade-off across all three configurations, delivering a low area and power while maintaining competitive WNS. We therefore adopt Qwen-3.6-plus as the default backbone for all subsequent experiments unless otherwise stated. The Ban Agent enforces an instance-ratio protection threshold of 8%, preventing the removal of any cell type that accounts for more than 8% of total mapped instances.

5.2 PPA Comparison

Improvement over Full Library Baseline. LibPilot achieves higher WNS than the unmodified Full Library in 19 of 21 configurations, with gains up to 220.6 ps (`ada_rf`, 28 nm). These timing improvements are generally accompanied by simultaneous area and power reductions. On `dp_ram` at 7 nm, area decreases by 13.0% and power by 1.4% alongside tighter slack. On `rvg` at 28 nm, power drops by 30.0% with a 10.5 ps WNS gain, Pareto-dominating the full library on all three objectives. In the two configurations where WNS marginally degrades, both area and power improve correspondingly, reflecting a principled multi-objective trade-off.

Comparison with MapTune. LibPilot outperforms MapTune in 49 of 63 individual PPA entries (77.8%), while requiring only a fixed budget of 40 DC invocations per configuration. MapTune, in contrast, averages 120 invocations to reach HV convergence, making LibPilot



(a) 7nm, `dp_ram` (b) 28nm, `rvg` (c) 45nm, `fsm`

Figure 5. The comparison of HV convergence with DC Calls on three different designs across 7nm, 28nm, and 45nm.

3× more sample-efficient. By metric, LibPilot attains lower area in 17/21, lower power in 18/21, and higher WNS in 14/21 configurations, with full Pareto dominance in 9 of 21 cases. The advantage is especially pronounced at 28 nm, where LibPilot reduces power on all 7 designs and area on 6 of 7. Even when MapTune leads on a single metric, the regressions on remaining objectives typically yield an inferior overall trade-off: on `dp_ram` at 28 nm, MapTune gains 6.0 ps in WNS but incurs 37.4% larger area and higher power.

Component Ablation. The two ablated variants reveal complementary contributions. Removing the dual agents (*w/o Dual-Agents*) consistently inflates area, as uninformed cell elimination cannot identify low-impact removal candidates; on `dp_ram` at 45 nm, area increases by 15.7% relative to the complete framework. Disabling UCT (*w/o UCT*) preserves competitive area and power in many cases but degrades timing: LibPilot achieves higher WNS in 17 of 21 configurations, with notable gains of 43.3 ps on `ada_rf` at 28 nm, confirming that logic-family injection is critical for expanding the synthesizer's optimization space along timing-critical paths. Integrating both components yields the most balanced PPA profile across all three technology nodes.

5.3 HV Convergence Comparison

Fig. 5 compares the HV convergence of LibPilot and MapTune on three representative configurations spanning all technology nodes. Two consistent trends emerge. First, LibPilot converges significantly faster: on all three designs, it reaches a near-final HV value within approximately 15 DC invocations, whereas MapTune requires over 30 calls to plateau. Second, LibPilot attains a higher converged HV in every case. On `dp_ram` at 7 nm, the final HV of LibPilot exceeds that of MapTune by roughly 12.5%; on `fsm` at 45 nm, the gap widens to over 25%. These results confirm that the structured, LLM-guided search of LibPilot explores the Pareto front more effectively per DC invocation than the RL-based exploration of MapTune, consistent with the 3× sample-efficiency advantage.

5.4 Ablation Study on Multi-Stage Exploration

TABLE 3 isolates the contribution of each component in the Multi-Stage Pareto exploration by removing the history window, the coarse-grained stage, and the fine-grained stage in turn. The HV value is computed with the Full Library PPA as the reference point.

Effect of History Window. Removing the history window causes the HV degradation across all three configurations: on `rvg` at 28 nm, HV drops from 1771.37 to 57.92, a 96.7% reduction. Without historical context, the agents repeatedly revisit ineffective cell removals, wasting the limited DC budget on redundant explorations.

Table 3. Ablation Study on Multi-Stage Pareto Exploration.

Variants	HV↑ Value	WNS↑ (*ps*)	Area↓ (μm^2)	Power↓ (m*W*)
`dp_ram` (7 nm Library)				
Full (Ours)	**6.48**	**-0.48**	**10353.90**	**14.85**
w/o History Window	1.12	-0.49	11148.68	14.91
w/o Coarse-grained	0.76	-0.49	11356.30	14.92
w/o Fine-grained	0.80	-0.49	11011.98	14.97
`rvg` (28 nm Library)				
Full (Ours)	**1771.37**	**-52.88**	**6104.20**	**30.37**
w/o History Window	57.92	-60.37	6115.28	33.26
w/o Coarse-grained	0	-55.38	6119.82	43.03
w/o Fine-grained	0	-55.41	6123.22	38.95
`fsm` (45 nm Library)				
Full (Ours)	**3.80e4**	**-56.54**	**6091.70**	**12.87**
w/o History Window	5.29e3	-67.02	6592.73	13.58
w/o Coarse-grained	3.51e3	-75.77	6350.26	13.82
w/o Fine-grained	2.14e3	-74.94	6546.56	13.91

HV is calculated using the full-library PPA metrics as the reference point.

Effect of Coarse- and Fine-grained Stages. Both stages are indispensable. Disabling the coarse-grained stage removes the ability to prune entire cell families early, forcing the search to operate in a prohibitively large space; on `rvg` at 28 nm, HV collapses to zero, indicating that no solution improves over the full library within the given budget. Likewise, removing the fine-grained stage eliminates targeted cell-level refinement, also yielding zero HV on the same configuration. On `fsm` at 45 nm, although both ablated variants still achieve nonzero HV, the complete framework attains an HV that is 10.8–17.8× higher, together with the best WNS, area, and power. These results demonstrate that coarse-grained exploration efficiently narrows the search space while fine-grained exploration refines solutions along the Pareto front, and combining the two stages with history awareness is essential for multi-objective improvement.

6 Conclusion

We presented LibPilot, a standard cell library tuning framework that couples a library knowledge graph with dual-agent LLM reasoning, UCT-guided logic-family injection, legality checks, and multi-stage Pareto exploration. By interpreting structured post-synthesis reports instead of relying solely on black-box search, LibPilot targets PPA bottlenecks with interpretable, graph-constrained cell updates. On seven industrial designs across 7 nm, 28 nm, and 45 nm libraries, LibPilot improves PPA and reduces synthesis iterations relative to full-library baselines and RL-based tuning. Future work includes broader mapper and LLM-backend studies and tighter integration with downstream physical design.

Acknowledgments

The research work described in this paper was conducted in the JC STEM Lab of Intelligent Design Automation funded by The Hong Kong Jockey Club Charities Trust.

References

[1] A. Mishchenko *et al.*, "Abc: A system for sequential synthesis and verification," *URL http://www. eecs. berkeley. edu/alanmi/abc*, vol. 17, 2007.

[2] G. Huang, J. Hu, Y. He, J. Liu, M. Ma, Z. Shen, J. Wu, Y. Xu, H. Zhang, K. Zhong *et al.*, "Machine learning for electronic design automation: A survey," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 5, pp. 1–46, 2021.

[3] W. Haaswijk, E. Collins, B. Seguin, M. Soeken, F. Kaplan, S. Süsstrunk, and G. De Micheli, "Deep learning for logic optimization algorithms," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–4.

[4] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.

[5] H. J. Touati, *Performance-oriented technology mapping*. University of California, Berkeley, 1990.

[6] M. Liu, D. Robinson, Y. Li, and C. Yu, "MapTune: Advancing ASIC technology mapping via reinforcement learning guided library tuning," in *IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–10.

[7] R. Fu, C. Wang, B. Yu, and T.-Y. Ho, "TeMACLE: A technology mapping-aware area-efficient standard cell library extension framework," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 8, pp. 3034–3045, 2025.

[8] Z. Chen, C. Guo, S. Wang, G. Feng, Z. Song, Z. Wu, X. Yin, W. Song, L. Zhang, Z. Yan, and C. Zhuo, "ZlibBoost: An efficient and flexible open-source framework for standard cell characterization," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 31, no. 4, 2026.

[9] G. D. Micheli, *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education, 1994.

[10] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proceedings of the IEEE*, vol. 78, no. 2, pp. 264–300, 2002.

[11] M. Wang, B. Sun, J. Mu, F. Gu, B. Han, T. Yang, X. Zhang, S. Liu, Y. Wen, H. Wang *et al.*, "Moss: Multi-modal representation learning on sequential circuits," in *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2025, pp. 1–7.

[12] S. Liu, W. Fang, Y. Lu, J. Wang, Q. Zhang, H. Zhang, and Z. Xie, "Rtlcoder: Fully open-source and efficient llm-assisted rtl code generation technique," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 4, pp. 1448–1461, 2024.

[13] H. Wu, Z. He, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu, "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, 2024.

[14] A. Liu, A. Mei, B. Lin, B. Xue, B. Wang, B. Xu, B. Wu, B. Zhang, C. Lin, C. Dong *et al.*, "Deepseek-v3. 2: Pushing the frontier of open large language models," *arXiv preprint arXiv:2512.02556*, 2025.

[15] OpenAI, "Introducing gpt-5.4 mini and nano," 2026. [Online]. Available: https://openai.com/zh-Hans-CN/index/introducing-gpt-5-4-mini-and-nano/

[16] A. Zeng, X. Lv, Z. Hou, Z. Du, Q. Zheng, B. Chen, D. Yin, C. Ge, C. Huang, C. Xie *et al.*, "Glm-5: from vibe coding to agentic engineering," *arXiv preprint arXiv:2602.15763*, 2026.

[17] Qwen, "Qwen3.6-plus: Towards real world agents," 2026. [Online]. Available: https://qwen.ai/blog?id=qwen3.6