SEARCH-GUIDED REASONING TRACE CURATION FOR VERILOG GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in reasoning-capable large language models (LLMs) have improved Verilog generation. However, most training pipelines distill natural-language reasoning traces whose intermediate design decisions are weakly structured and are primarily selected according to the correctness of the final generated RTL. We present **MCTS-SFT**, a verifier-guided framework for curating reasoning traces for supervised fine-tuning (SFT) of Verilog generation models. MCTS-SFT formulates trace construction as search over intermediate reasoning states and uses Monte Carlo Tree Search (MCTS) to propagate functional feedback from complete Verilog rollouts to earlier design decisions. Within this framework, we introduce **Plan-MCTS**, which searches over compact hardware-design plans that explicitly represent interfaces, timing, state, behavior, and update rules. We further study **plan-preserving supervision**, in which the searched structured state is retained as part of the downstream SFT target. Our experiments find that Plan-MCTS achieves the highest functional trace yield, with 84.2% correct traces compared with 81.9% for one-shot structured planning and 82.0% for unguided tree exploration. For identical Plan-MCTS trajectories and Verilog outputs, retaining the explicit searched plan during SFT raises pass@10 from 52.56% to 57.7%, demonstrating that the representation of intermediate design decisions materially affects downstream learning. In a separate 6,000-trace study, MCTS-curated supervision improves pass@10 over full-response chain-of-thought distillation by up to 3.7 points while reducing downstream generation length by approximately 50% on VerilogEval-v2. These results demonstrate that structured hardware planning, verifier-guided search, and explicit intermediate-state supervision provide complementary mechanisms for constructing more effective and token-efficient training data for Verilog generation.

1 INTRODUCTION

Large language models (LLMs) have recently made substantial progress on long-horizon, multi-step tasks, driven in part by reasoning models that generate extended intermediate traces before producing a final response (Guo et al., 2025; Wang et al., 2024). These reasoning trajectories have become an increasingly important source of supervision for distillation and fine-tuning, enabling smaller models to inherit aspects of the reasoning behavior of stronger teachers. However, the quality of the intermediate traces remains underexplored. Long reasoning sequences can introduce substantial token overhead and may contain repetitive, inconsistent, or weakly relevant content (**?**Qu et al., 2025), motivating the construction of effective reasoning supervision.

This issue is particularly relevant for hardware description language (HDL) generation. Recent work has applied reasoning-capable LLMs to functional Verilog generation and has shown that fine-tuning on chain-of-thought (CoT) responses, often combined with executable feedback, can substantially improve final design correctness (Yubeaton et al., 2025; Zhu et al., 2025; Wang et al., 2025b). Existing approaches, however, generally curate complete reasoning responses and evaluate them according to the final generated Verilog, providing limited control over the quality of the intermediate reasoning that is ultimately used for training. Moreover, these traces are typically expressed as free-form natural language, despite RTL design being naturally organized around explicit decisions concerning interfaces, timing, state, behavior, and signal updates. This motivates a reasoning-trace

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

curation process that can both evaluate intermediate design decisions using functional feedback and represent them in a form that provides effective supervision for downstream fine-tuning.

To address this problem, we present **MCTS-SFT**, a verifier-guided framework for curating Verilog reasoning traces for supervised fine-tuning. MCTS-SFT formulates intermediate reasoning as a sequence of design states and applies Monte Carlo Tree Search (MCTS) to estimate their quality from terminal testbench feedback. We study two forms of intermediate representation. *Thought-MCTS* searches directly over free-form reasoning steps, while *Plan-MCTS* maintains a compact structured hardware plan whose updates correspond to explicit design decisions. The resulting search trajectories are then converted into prompt-response examples for SFT, allowing the student model to learn from reasoning paths selected using executable feedback. In Plan-MCTS, we additionally retain the searched plan itself as part of the training response, directly exposing the intermediate hardware decisions discovered during curation.

Our main contributions are:

1. We introduce **MCTS-SFT**, a verifier-guided framework for curating reasoning traces for Verilog supervised fine-tuning. MCTS-SFT uses terminal testbench feedback to assign value to intermediate reasoning states, enabling executable verification to guide training-data construction without a separate process-reward model.
2. Within MCTS-SFT, we introduce **Plan-MCTS**, a structured curation strategy that represents intermediate reasoning as explicit hardware design decisions rather than accumulated free-form text. Controlled comparisons show that structured planning is independently useful, while verifier-guided search provides additional benefit for discovering functionally correct training trajectories.
3. We demonstrate that **preserving the searched hardware plan as SFT supervision** is critical to downstream learning. For identical Plan-MCTS trajectories and Verilog outputs, explicit plan supervision improves pass@10 from 52.56% to 57.7%; in the larger 6k-trace study, MCTS-curated SFT also improves correctness over full-response CoT distillation while reducing generation length by approximately 50% on VerilogEval-v2.

2 Background

2.1 Training LLMs for Verilog Generation

Specializing LLMs for Verilog generation has become an active research area as a means of addressing limitations in hardware design capabilities (Yang et al., 2025). Early work focused on domain-specific pretraining and supervised fine-tuning over curated Verilog corpora (Thakur et al., 2024), followed by instruction-tuning approaches that expanded prompt-response datasets through filtering and augmentation strategies (Zhao et al., 2024; Liu et al., 2024). More recent methods incorporate functional feedback directly into post-training. VeriRL applies reinforcement learning using similarity to reference implementations (Teng et al., 2025), while subsequent work introduced module-specific testbench generation to enable functionality-based rewards (Wang et al., 2025a). Other approaches have explored more specialized reward formulations, including hierarchical (Chen et al., 2025) and entropy-aware objectives (Shi et al., 2025).

2.2 Reasoning LLMs for Verilog Generation

The emergence of reasoning LLMs (Guo et al., 2025) has motivated training pipelines that incorporate explicit reasoning traces into Verilog generation. VeriThoughts (Yubeaton et al., 2025) distills DeepSeek-R1 reasoning traces for SFT after verifying the resulting Verilog against reference solutions. QiMeng-CodeV-R1 (Zhu et al., 2025) similarly constructs reasoning-based training data with functional checking and applies DAPO reinforcement learning, while VeriReason (Wang et al., 2025b) uses GRPO with dataset refinement and structural Verilog rewards. These approaches substantially improve functional correctness, with CodeV-R1 reporting 12–20% gains over prior state-of-the-art results on VerilogEval (Zhu et al., 2025), but also expose challenges including long and inefficient reasoning trajectories (Wang et al., 2025b; Yubeaton et al., 2025). ReasoningV adapts the amount of reasoning to problem complexity (Qin et al., 2025), while SiliconMind-V1 introduces a

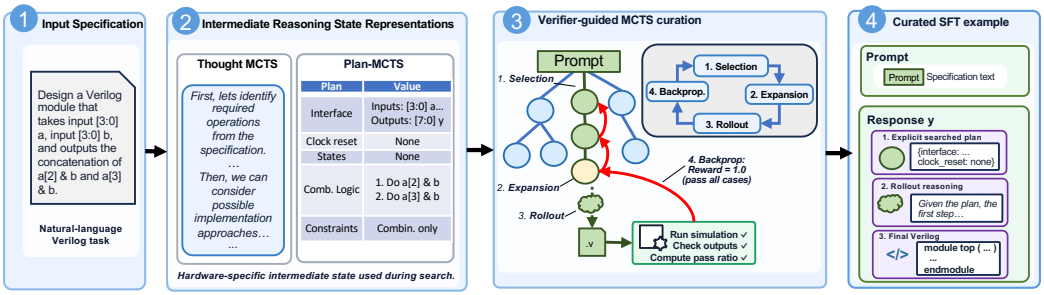


Figure 1: MCTS-SFT trace curation process, including state representations and MCTS stages.

multi-agent refinement process that incorporates LLM-generated testbench feedback into SFT without external EDA tools (Chen et al., 2026). QiMeng-CRUX further introduces a structured thought representation for GRPO, using confidence-based rewards to optimize intermediate reasoning structure (Huang et al., 2026). Independent analysis of RTL reasoning traces also identifies substantial repetition and token overhead (DeLorenzo et al., 2025b) Recent work has also begun to explicitly assign value to intermediate RTL reasoning steps. StepPRM-RTL (Vijayaraghavan et al., 2026) represents RTL construction as a sequence of rationale–code-edit pairs and trains a step-level process reward model (PRM) to score intermediate decisions. The learned PRM is then used to guide MCTS exploration of alternative reasoning trajectories, with the resulting trajectories incorporated into a retrieval-augmented fine-tuning procedure. This provides process-level credit assignment during RTL reasoning, but requires learning an auxiliary reward model from intermediate steps.

Our work addresses this problem from a complementary direction. Rather than training a separate process reward model, MCTS-SFT uses executable feedback from complete Verilog rollouts as the sole source of search value and propagates this terminal feedback through MCTS to earlier reasoning states. Moreover, Plan-MCTS represents these states as persistent, hardware-specific design plans and studies the searched state representations that should be retained as downstream SFT supervision. Therefore, our focus is on using terminal functional verification to discover and value intermediate design states without requiring direct process-level labels or a learned verifier.

2.3 Inference-Level Verilog Reasoning Optimization

A complementary line of work improves Verilog generation at inference time through structured prompting, candidate selection, or iterative refinement. Abstractions-of-Thought (DeLorenzo et al., 2025a) uses a staged prompting procedure that transitions from higher-level to lower-level intermediate representations. VRank (Zhao et al., 2025b) clusters sampled Verilog candidates according to simulated behavior and selects outputs using self-consistency, while VFocus (Zhao et al., 2025a) combines CoT sampling, reasoning-length filtering, simulation-based ranking, and refinement of promising candidates. Other agentic and feedback-driven systems similarly use compiler diagnostics, simulation feedback, generated testbenches, or iterative repair, while evolutionary approaches refine candidate implementations with respect to correctness or hardware metrics (Min et al., 2026; Ping et al., 2026; Hsin et al., 2026). These methods demonstrate the value of search and reasoning structure for improving final RTL candidates, but primarily operate at inference time. In contrast, MCTS-SFT uses verifier-guided search offline to explore intermediate reasoning sequences, curate training trajectories, and leverage them for supervised fine-tuning.

3 Framework

Given a natural-language Verilog specification x, MCTS-SFT constructs reasoning supervision by using executable feedback to evaluate and curate intermediate design decisions before supervised fine-tuning. For each specification, a teacher model explores intermediate reasoning states, selected states are rolled out into complete reasoning traces and Verilog implementations, and the resulting code is evaluated by a functional verifier. Monte Carlo Tree Search (MCTS) propagates terminal verification feedback to earlier reasoning states, allowing otherwise non-executable design decisions to

influence subsequent exploration. The resulting trajectories are then converted into prompt-response examples for fine-tuning a student model.

MCTS-SFT denotes the complete curation and fine-tuning framework. Within this framework, we consider two intermediate-state representations. *Thought-MCTS* searches over accumulated free-form reasoning, while *Plan-MCTS* searches over structured hardware design states. Our primary formulation uses Plan-MCTS and retains the searched hardware plan as part of the downstream SFT supervision. Figure 1 summarizes the complete pipeline.

3.1 Verifier-Guided Reasoning-Trace Curation

We formulate reasoning-trace construction for a specification x as a finite-horizon decision process

$$\mathcal{M}_x = (\mathcal{S}, \mathcal{A}, T, R), \tag{1}$$

where a state contains the specification together with its current intermediate reasoning representation, an action modifies that representation, T defines the corresponding transition, and R is obtained by verifying a complete Verilog rollout. At reasoning step t,

$$S_t = (x, h_t), \qquad S_{t+1} = T(S_t, a_t), \tag{2}$$

where h_t denotes the reasoning state constructed after t decisions. The interpretation of h_t depends on the search representation: Thought-MCTS uses an accumulated natural-language reasoning prefix, whereas Plan-MCTS uses the structured hardware plan introduced in Section 3.2.

A central difficulty is that an intermediate state cannot be functionally verified in isolation. Decisions concerning timing, state organization, or control logic acquire executable meaning only after they are instantiated in a complete RTL implementation. We therefore evaluate an intermediate state through a rollout model,

$$(r, v) \sim \pi_\theta^{\text{rollout}}(\cdot \mid x, h_t), \tag{3}$$

which materializes the current state into a complete reasoning trace r and Verilog implementation v. The implementation is then compiled and evaluated against the corresponding testbench, producing terminal reward

$$R(v) = \begin{cases} \alpha_{\text{NC}}, & \text{if } v \text{ does not compile,} \\ \dfrac{N_{\text{pass}}}{N_{\text{total}}}, & \text{otherwise,} \end{cases} \tag{4}$$

where $\alpha_{\text{NC}} < 0$ penalizes non-compilable designs and $N_{\text{pass}}/N_{\text{total}}$ measures functional correctness.

Verifier-guided tree search. MCTS repeatedly performs selection, expansion, rollout, and backpropagation over this intermediate-state space. During selection, the next explored action is chosen according to the UCT criterion

$$a^* = \arg\max_{a \in \mathcal{A}_{\text{tree}}(S)} \left[Q(S,a) + c_{\text{uct}} \sqrt{\frac{\log N(S)}{1 + N(S,a)}} \right], \tag{5}$$

where $Q(S,a)$ estimates the downstream functional value of action a, $N(S)$ and $N(S,a)$ denote state and edge visitation counts, and c_{uct} controls exploration.

When an expandable state is reached, the teacher proposes multiple candidate next decisions, each defining a successor state through Eq. 2. A selected successor is rolled out according to Eq. 3, and its terminal verifier reward is propagated through the visited path. We maintain a running action-value estimate

$$Q(S,a) \leftarrow Q(S,a) + \frac{R(v) - Q(S,a)}{N(S,a)}. \tag{6}$$

Repeated simulations use executable feedback to score complete implementations and guide subsequent exploration toward intermediate decisions associated with stronger functional outcomes. Additional search details, pseudocode, and generation parameters are provided in Appendix A.2.

3.2 PLAN-MCTS: STRUCTURED HARDWARE DESIGN STATES

Plan-MCTS instantiates the intermediate state h_t as a structured hardware design plan rather than an accumulated prose trace. The plan contains seven hardware-specific fields: `interface`, `clock_reset`, `state_elements`, `combinational_logic`, `fsm`, `edge_cases`, and `implementation_constraints`. Together, these fields capture the principal architectural and behavioral decisions required to translate the natural-language specification into RTL while remaining independent of the exact syntax of the final implementation.

At step t, the current plan is denoted by p_t. An expansion action proposes a structured update

$$p_{t+1} = \mathrm{Update}(p_t, \Delta p_t), \tag{7}$$

where Δp_t identifies a target field, proposes a design decision, and provides a concise rationale. The teacher determines which aspect of the current plan should be refined and generates alternative updates during expansion. Consequently, branches in the search tree correspond to alternative hardware design decisions rather than alternative continuations of unrestricted natural-language reasoning.

The structured plan is intentionally an intermediate design representation rather than executable RTL. A selected plan is therefore evaluated by conditioning the rollout model on both the specification and accumulated plan,

$$(r, v) \sim \pi_\theta^{\mathrm{rollout}}(\cdot \mid x, p_t), \tag{8}$$

after which the resulting Verilog implementation is functionally verified and its reward is backpropagated through the sequence of plan updates.

This gives the plan two complementary roles within MCTS-SFT. First, it defines a hardware-specific state space in which MCTS can explore and assign value to intermediate design choices. Second, it provides an explicit record of the decisions associated with the resulting implementation, which can subsequently be exposed to the student during fine-tuning.

For comparison, Thought-MCTS applies the same verifier-guided search procedure while representing h_t as an accumulated free-form reasoning prefix. This isolates the effect of structured hardware representation from the effect of MCTS itself. Likewise, Plan-CoT and Plan-RandomTree introduced in Section 4 distinguish structured planning from verifier-guided search and from branching without reward-guided selection, respectively.

3.3 PLAN-PRESERVING SUPERVISED FINE-TUNING

For a curated Plan-MCTS trajectory, let (x, p, r, v) denote the input specification, searched hardware plan, rollout reasoning, and terminal Verilog implementation. The corresponding training target is

$$y^{\mathrm{plan}} = \mathrm{Serialize}(p, r, v), \tag{9}$$

where $\mathrm{Serialize}(\cdot)$ retains the explicit field structure of p together with the materialized reasoning and final implementation. The resulting (x, y^{plan}) pairs are used for standard autoregressive SFT of the student model.

This final stage separates two distinct aspects of reasoning-trace curation. *Search quality* determines which intermediate design states and terminal implementations are discovered through verifier-guided exploration, while *supervision quality* determines how those states are represented to the downstream model. Our controlled experiments separately evaluate these effects by comparing planning with and without verifier-guided search and, for fixed Plan-MCTS trajectories, varying only the intermediate representation retained in the SFT response. Additional details relating to the SFT training configurations are included in the Appendix.

4 EXPERIMENTS

We evaluate MCTS-SFT through controlled and extended-scale experiments addressing three questions: **RQ1:** How do structured planning and verifier-guided search affect the discovery of functional training trajectories? **RQ2:** How does the intermediate representation retained during SFT

affect downstream learning? **RQ3:** How does search-curated reasoning supervision affect downstream correctness, generation efficiency, and transfer at larger scale?

We use two complementary protocols. A **controlled 1k-trace study** isolates the effects of structured planning, verifier-guided search, and SFT-target representation under a unified teacher model. An **extended 6k-trace study** evaluates the broader MCTS-SFT pipeline across multiple Verilog benchmarks.

4.1 Experimental Setup

Data and verification. We construct training examples from natural-language Verilog specifications sampled from the VeriThoughts training set (Yubeaton et al., 2025). Generated implementations are compiled and evaluated using automatically constructed testbenches following prior Verilog reasoning work (Zhu et al., 2025). A trajectory is considered functionally correct when its terminal Verilog implementation passes all associated testbench cases.

Controlled 1k-trace study. We sample 100 specifications and generate 10 trajectories per specification for each curation strategy, yielding 1,000 examples per dataset. DeepSeek-R1 is used for every LLM invocation, and all methods share the same verifier, generation budget, student model, and SFT configuration. This matched setting is used for the component-level comparisons in RQ1 and RQ2.

We compare five curation strategies. **CoT** independently samples complete free-form reasoning responses. **Thought-MCTS** applies verifier-guided search over free-form reasoning states. **Plan-CoT** generates a complete structured hardware plan in a single step before rollout, providing a planning baseline without tree search. **Plan-RandomTree** matches the structured state representation, branching process, rollout procedure, and search budget of Plan-MCTS, but selects nodes uniformly rather than using verifier-guided values. **Plan-MCTS** incrementally constructs the structured hardware plan using the search procedure in Section 3. Together, these controls separate the effects of structured representation, tree exploration, and verifier-guided selection.

For RQ2, we additionally construct three SFT targets from identical Plan-MCTS trajectories: **rollout only**, which omits the searched plan; **rendered plan**, which prepends a natural-language rendering of the plan; and **explicit plan**, which preserves the structured hardware representation directly. Since the underlying trajectories, terminal Verilog implementations, rewards, and dataset sizes are fixed, this comparison isolates the effect of SFT-target representation. We also include **Summary-MCTS**, which compresses accumulated free-form reasoning before subsequent expansion, as a control for reasoning verbosity.

Extended 6k-trace study. We additionally evaluate a larger-scale setting using 600 specifications and 10 trajectories per specification, yielding 6,000 examples for each full curation strategy. We compare CoT, Thought-MCTS, and Plan-MCTS, together with answer-only SFT and a 600-example Best-of-N Plan-MCTS baseline.

This experiment utilizes a mixed teacher configuration: DeepSeek-R1 is used for CoT generation, Thought-MCTS, and Plan-MCTS rollouts, while DeepSeek-V3 constructs intermediate Plan-MCTS states. It also uses a *rollout-only* Plan-MCTS SFT target rather than the explicit plan-preserving target studied in the controlled experiment. We therefore use the 6k study for pipeline-level comparisons of downstream accuracy and generation efficiency, while component-level conclusions are drawn from the unified-teacher 1k study.

Supervised fine-tuning. For all datasets, we fine-tune the same 7B Qwen2.5 student model using an identical SFT procedure. We train for three epochs with learning rate 5×10^{-5}, maximum sequence length 16,384, and LoRA rank 64 on a single NVIDIA H200 GPU. Additional generation and training details are provided in Appendix A.1.

Evaluation. For the controlled study, we evaluate on VerilogEval-v1-Human (Liu et al., 2023) using $n = 10$ generations per problem. The extended study additionally evaluates VerilogEval-v2 Code Completion and Spec-to-RTL using $n = 20$ generations per problem. We report pass@k functional correctness and average generated response length. For RQ1, we report *functional trace*

Table 1: **RQ1: Functional trace discovery under the controlled 1k-trace setting.** Each strategy generates 1,000 trajectories from the same 100 specifications using DeepSeek-R1 for all LLM calls. A correct trace passes all associated testbench cases.

Strategy	State representation	Exploration	Correct	Yield
CoT	Free-form	Independent	731 / 1,000	73.1%
Thought-MCTS	Free-form	Verifier-guided	836 / 1,000	83.6%
Plan-CoT	Structured plan	One-shot	819 / 1,000	81.9%
Plan-RandomTree	Structured plan	Random tree	820 / 1,000	82.0%
Plan-MCTS	Structured plan	Verifier-guided	**842 / 1,000**	**84.2%**

yield, defined as the proportion of curated trajectories whose terminal implementation passes all testbench cases.

To examine transfer beyond the predominantly single-module VerilogEval setting, we additionally evaluate on 172 non-agentic multi-module tasks from CVDP-002 and CVDP-003.

5 Results

We organize the results around the three research questions in Section 4: the effects of planning and verifier-guided search during curation, the representation retained during SFT, and downstream performance at larger scale.

5.1 RQ1: How Do Structured Planning and Verifier-Guided Search Affect Functional Trace Discovery?

Table 1 reports functional trace yield in the controlled 1k-trace setting, where all methods use the same teacher, specifications, generation budget, and verifier.

Through these results, we can see that structured planning alone results in functional trace yield. Plan-CoT reaches 81.9% functional yield, compared with 73.1% for independent CoT sampling. Random tree exploration provides essentially no additional benefit (82.0%), whereas verifier-guided Plan-MCTS achieves the highest yield at 84.2%. Thus, the hardware-specific representation provides a strong basis for trace construction, with verifier-guided selection contributing additional improvements over planning or branching alone.

Thought-MCTS also attains a high functional yield of 83.6%, indicating that verifier-guided search can identify strong terminal implementations even with free-form reasoning states. However, curation-time correctness alone does not determine the usefulness of a trajectory for downstream learning, motivating the representation analysis in RQ2.

5.2 RQ2: How Does Intermediate Representation Affect Downstream SFT?

Table 2 evaluates the controlled SFT datasets. The three Plan-MCTS target variants use identical search trajectories, Verilog implementations, rewards, and dataset sizes, isolating the effect of how the searched plan is presented to the student.

The fixed-trajectory comparison shows that supervision quality depends strongly on the retained intermediate representation. Plan-MCTS with rollout-only supervision obtains 52.56% pass@10; adding a natural-language rendering of the plan raises this to 55.10%, while retaining the explicit structured plan reaches 57.70%. As the underlying trajectories and terminal programs are identical, the 5.14-point improvement from rollout-only to explicit-plan supervision is attributable to the intermediate information exposed during SFT. The explicit representation also outperforms the rendered plan at pass@1/5/10, indicating that preserving the structured hardware fields is more effective than expressing the same information as additional prose.

The remaining controls reinforce this conclusion. Plan-CoT reaches 57.05% pass@10 without tree search, highlighting that structured planning is independently valuable, while explicit-plan Plan-MCTS obtains the strongest pass@5 and pass@10. In contrast, Summary-MCTS remains close

Table 2: **RQ2: Effect of reasoning and SFT-target representation.** All models are trained on 1,000 examples and evaluated on VerilogEval-v1-Human with $n = 10$. The three Plan-MCTS target variants use identical search trajectories, terminal Verilog implementations, and verifier rewards.

Method / SFT target	pass@1	pass@5	pass@10
CoT	31.22	46.91	52.56
Thought-MCTS	27.37	46.71	51.28
Summary-MCTS	29.23	46.43	51.92
Plan-CoT	**32.37**	50.40	57.05
Plan-RandomTree	28.53	47.44	53.21
Plan-MCTS, rollout only	29.17	46.48	52.56
Plan-MCTS + rendered NL plan	30.00	47.60	55.10
Plan-MCTS + explicit plan	31.50	**50.80**	**57.70**

Table 3: Trace characteristics in the controlled study. Lower Distinct-2 indicates greater lexical repetition. The three Plan-MCTS target variants share the same terminal trajectories and therefore have the same mean verifier reward.

Strategy / target	Mean reward	Mean tokens	Distinct-2
CoT	0.571	1,021	0.789
Thought-MCTS	0.754	3,121	0.607
Plan-CoT	0.755	1,649	0.747
Plan-RandomTree	0.739	1,556	0.740
Plan-MCTS, rollout only	**0.780**	1,561	0.743
Plan-MCTS + rendered NL plan	**0.780**	1,681	0.745
Plan-MCTS + explicit plan	**0.780**	1,759	0.748

to Thought-MCTS, suggesting that the benefit is not explained simply by compressing free-form reasoning.

Trace characteristics. Table 3 further characterizes the controlled datasets. Thought-MCTS produces substantially longer and more repetitive traces than the plan-based methods despite its high curation yield. More importantly, the three Plan-MCTS targets have identical terminal rewards (final verilog designs), but markedly different downstream performance. This further reinforces the role of curated intermediate reasoning structures within the SFT procedure for effective Verilog generation.

5.3 RQ3: Does Search-Curated Supervision Improve Downstream Correctness and Efficiency at Larger Scale?

We next evaluate the extended 6k-trace setting across VerilogEval-v1-Human and both VerilogEval-v2 tasks. As described in Section 4, this experiment uses a mixed-teacher configuration and rollout-only Plan-MCTS target, so we interpret it as a pipeline-level evaluation rather than a component attribution study.

We find that Plan-MCTS achieves the strongest functional performance across all three benchmarks. Relative to full-response CoT distillation, it improves pass@1/5/10 by 2.6/3.2/3.2 points on VerilogEval-v1-Human and by 1.8/2.6/3.7 points on VerilogEval-v2 Code Completion. Table 5 indicates that gains on CVDP (Spec-to-RTL) problems are smaller, but remain positive at each reported pass@k.

A primary advantage of MCTS-SFT appears in *generation efficiency*. As shown in Table 4, on the two VerilogEval-v2 prompt sets, the downstream response of models trained via CoT produces averages of 7,317 and 7,313 tokens, whereas Plan-MCTS reduces these to 3,622 and 3,710, corresponding to 50.5% and 49.2% reductions while maintaining or improving functional accuracy. Thought-MCTS is less consistent in accuracy and generates longer responses than Plan-MCTS across all three benchmarks. The answer-only and Best-of-N controls likewise do not match the full Plan-MCTS dataset, although the smaller Best-of-N training set prevents attributing this difference solely to its selection procedure.

Table 4: **RQ3: Downstream Verilog generation after SFT in the extended study.** VE-v1-H denotes VerilogEval-v1-Human, VE-v2-CC denotes VerilogEval-v2 Code Completion, and VE-v2-S2R denotes VerilogEval-v2 Spec-to-RTL. Tokens report mean generated response length ± standard deviation. Bold denotes the highest pass@k value for each benchmark.

		VE-v1-H				VE-v2-CC				VE-v2-S2R			
Training data	**# SFT**	**p@1**	**p@5**	**p@10**	**Tokens**	**p@1**	**p@5**	**p@10**	**Tokens**	**p@1**	**p@5**	**p@10**	**Tokens**
Baseline (no SFT)	0	21.4	40.9	46.6	1132 ± 1640	25.1	40.1	46.0	456 ± 665	24.1	38.6	43.7	422 ± 649
CoT (answer only)	6,000	36.1	48.1	52.6	1888 ± 4217	35.7	50.1	54.4	442 ± 466	34.0	49.0	55.2	451 ± 627
Best-of-N (Plan-MCTS)	600	33.5	49.4	54.4	3143 ± 3045	35.3	50.7	55.4	3556 ± 3178	30.3	48.0	54.0	3760 ± 3217
CoT (full response)	6,000	36.9	52.5	57.2	3162 ± 3159	41.5	54.8	58.8	7317 ± 1757	41.2	55.6	60.3	7313 ± 1715
Thought-MCTS	6,000	35.9	53.5	57.8	4521 ± 3057	36.9	55.3	60.4	4348 ± 3038	41.3	55.8	60.3	4483 ± 3017
Plan-MCTS	6,000	**39.5**	**55.7**	**60.4**	2993 ± 3173	**43.3**	**57.4**	**62.5**	3622 ± 2962	**41.4**	**55.9**	**60.6**	3710 ± 2973

Table 5: Preliminary multi-module evaluation on 172 CVDP-002/003 tasks. Absolute success rates remain low, and we therefore interpret this experiment as evidence of transfer rather than a comprehensive evaluation of multi-module RTL generation.

Model	**Overall**	**Easy**	**Medium**
CoT-SFT	9.30% (16/172)	15.73%	2.41%
Thought-MCTS-SFT	11.63% (20/172)	**20.22%**	2.41%
Plan-MCTS-SFT	**12.21% (21/172)**	19.10%	**4.82%**

Transfer to multi-module RTL generation. Finally, we evaluate transfer beyond the predominantly single-module VerilogEval setting using 172 non-agentic CVDP-002/003 tasks. Plan-MCTS obtains the highest overall success rate at 12.21%, compared with 11.63% for Thought-MCTS and 9.30% for CoT-SFT. On medium-difficulty tasks, it solves four examples compared with two for each baseline, although Thought-MCTS performs slightly better on the easy subset. Given the low absolute success rates of the student model, we treat these results as preliminary evidence of transfer rather than evidence of strong multi-module capability.

6 Conclusion

We present MCTS-SFT, a verifier-guided framework for curating reasoning traces for supervised fine-tuning of Verilog generation models. The framework addresses a central limitation of conventional CoT distillation: intermediate hardware decisions are typically represented as free-form text and receive little direct supervision despite their importance to final RTL correctness. By formulating reasoning as a sequence of intermediate states and propagating terminal testbench feedback through MCTS, MCTS-SFT provides a mechanism for discovering and valuing these otherwise non-verifiable design decisions.

Our controlled experiments show that the strongest performance arises from the combination of structured hardware planning, verifier-guided search, and explicit preservation of the searched plan during SFT. Structured planning is independently valuable, while Plan-MCTS further improves functional trace discovery beyond one-shot planning and unguided tree exploration. Moreover, when the underlying trajectories and Verilog outputs are held fixed, retaining the explicit structured plan produces substantially stronger downstream performance than either omitting the plan or rendering it as less-structured natural language. In the larger 6k-trace evaluation, Plan-MCTS also improves Verilog correctness over full-response CoT distillation while reducing downstream generation length by approximately 50% on VerilogEval-v2.

These results indicate that effective reasoning supervision depends not only on whether a trajectory reaches a correct final program, but also on which intermediate design states are discovered and how those states are represented during training. This suggests a broader direction for hardware reasoning models in which executable feedback is used to shape intermediate design representations that are both functionally grounded and effective for downstream learning.

AI USE STATEMENT

Large language models are a central component of the research methodology in this work. We use LLMs as teacher models to generate, expand, and curate reasoning trajectories and Verilog implementations that form the training datasets evaluated in our experiments. LLM-based tools were also used in a limited capacity during manuscript preparation to assist with formatting and editorial refinement. All scientific claims, experimental design choices, analyses, figures, tables, and conclusions were determined by the authors.

ETHICS STATEMENT

This work studies methods for improving language-model-based Verilog generation using automatically generated reasoning traces and functional verification. The experiments use existing research datasets, benchmarks, and model-generated training data and do not involve human subjects or private personal information. As with other LLM code-generation systems, generated RTL may contain functional or implementation errors if used without appropriate verification. Our methodology incorporates executable testing during data curation, but such verification should not be interpreted as guaranteeing correctness or suitability for deployment in safety- or security-critical hardware.

REPRODUCIBILITY STATEMENT

We provide the methodological and experimental details needed to reproduce the main results throughout the paper and appendices. Section 3 defines the MCTS-SFT and Plan-MCTS formulations, while Appendix A.2 provides additional MCTS algorithm and state-representation details. The experimental setup specifies the controlled and extended-scale evaluation protocols, and Appendix A.1 reports the available curation, training, model, and evaluation configurations and hyperparameters. Additional ablations and generalization experiments are included in the appendices to clarify the conditions under which the reported findings were obtained. We also document the distinctions between teacher configurations, search strategies, and SFT-target representations so that the controlled comparisons can be reconstructed consistently.

REFERENCES

Mu-Chi Chen, Yu-Hung Kao, Po-Hsuan Huang, Shao-Chun Ho, Hsiang-Yu Tsou, I Wu, En-Ming Huang, Yu-Kai Hung, Wei-Po Hsin, Cheng Liang, et al. SiliconMind-V1: Multi-Agent Distillation and Debug-Reasoning Workflows for Verilog Code Generation. *arXiv preprint arXiv:2603.08719*, 2026.

Zhirong Chen, Kaiyan Chang, Zhuolin Li, Xinyang He, Chujie Chen, Cangyuan Li, Mengdi Wang, Haobo Xu, Yinhe Han, and Ying Wang. ChipSeek-R1: Generating human-surpassing RTL with LLM via hierarchical reward-driven reinforcement learning. *arXiv preprint arXiv:2507.04736*, 2025.

Matthew DeLorenzo, Kevin Tieu, Prithwish Jana, Piyush Jha, Dileep Kalathil, Vijay Ganesh, and Jeyavijayan Rajendran. Abstractions-of-Thought: Intermediate Representations for LLM Reasoning in Hardware Design. *arXiv preprint arXiv:2505.15873*, 2025a.

Matthew DeLorenzo, Kevin Tieu, and Jeyavijayan Rajendran. Tracing the Logic: Evaluating LLM Reasoning Paths in RTL Generation. In *2025 IEEE 43rd International Conference on Computer Design (ICCD)*, pp. 776–781. IEEE, 2025b.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Wei-Po Hsin, Ren-Hao Deng, Yao-Ting Hsieh, En-Ming Huang, and Shih-Hao Hung. EvolVE: Evolutionary Search for LLM-based Verilog Generation and Optimization. *arXiv preprint arXiv:2601.18067*, 2026.

Lei Huang, Rui Zhang, Jiaming Guo, Yang Zhang, Di Huang, Shuyao Cheng, Pengwei Jin, Chongxiao Li, Zidong Du, Xing Hu, et al. QiMeng-CRUX: Narrowing the Gap Between Natural Language and Verilog via Core Refined Understanding eXpression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 31193–31201, 2026.

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

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

Kyungjun Min, Kyumin Cho, Junhwan Jang, and Seokhyeong Kang. Revolution: An evolutionary framework for RTL generation driven by large language models. In *2026 31st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 282–288. IEEE, 2026.

Heng Ping, Peiyu Zhang, Shixuan Li, Wei Yang, Anzhe Cheng, Shukai Duan, Xiaole Zhang, and Paul Bogdan. COEVO: Co-Evolutionary Framework for Joint Functional Correctness and PPA Optimization in LLM-Based RTL Generation. *arXiv preprint arXiv:2604.15001*, 2026.

Haiyan Qin, Zhiwei Xie, Jingjing Li, Liangchen Li, Xiaotong Feng, Junzhan Liu, and Wang Kang. ReasoningV: Efficient Verilog code generation with adaptive hybrid reasoning model. *arXiv preprint arXiv:2504.14560*, 2025.

Xiaoye Qu, Yafu Li, Zhaochen Su, Weigao Sun, Jianhao Yan, Dongrui Liu, Ganqu Cui, Daizong Liu, Shuxian Liang, Junxian He, et al. A Survey of Efficient Reasoning for Large Reasoning Models: Language, Multimodality, and Beyond. *arXiv preprint arXiv:2503.21614*, 2025.

Jiahe Shi, Zhengqi Gao, Ching-Yun Ko, and Duane Boning. EARL: Entropy-Aware RL Alignment of LLMs for Reliable RTL Code Generation. *arXiv preprint arXiv:2511.12033*, 2025.

Fu Teng, Miao Pan, Xuhong Zhang, Zhezhi He, Yiyao Yang, Xinyi Chai, Mengnan Qi, Liqiang Lu, and Jianwei Yin. VERIRL: Boosting the LLM-based Verilog Code Generation via Reinforcement Learning. In *2025 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–9. IEEE, 2025.

Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. Verigen: A large language model for Verilog code generation. *ACM Transactions on Design Automation of Electronic Systems*, 29(3):1–31, 2024.

Prashanth Vijayaraghavan, Apoorva Nitsure, Luyao Shi, Ehsan Degan, and Vandana Mukherjee. Stepprm-rtl: Stepwise process-reward guided llm fine-tuning for enhanced rtl synthesis. *arXiv preprint arXiv:2606.04246*, 2026.

Ning Wang, Bingkun Yao, Jie Zhou, Yuchen Hu, Xi Wang, Nan Guan, and Zhe Jiang. Insights from verification: Training a Verilog generation LLM with reinforcement learning with testbench feedback. *arXiv preprint arXiv:2504.15804*, 2025a.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510/.

Yiting Wang, Guoheng Sun, Wanghao Ye, Gang Qu, and Ang Li. VeriReason: Reinforcement learning with testbench feedback for reasoning-enhanced Verilog generation. *arXiv preprint arXiv:2505.11849*, 2025b.

Guang Yang, Wei Zheng, Xiang Chen, Dong Liang, Peng Hu, Yukui Yang, Shaohang Peng, Zhenghan Li, Jiahui Feng, Xiao Wei, et al. Large language model for Verilog code generation: Literature review and the road ahead. *arXiv preprint arXiv:2512.00020*, 2025.

Patrick Yubeaton, Andre Nakkab, Weihua Xiao, Luca Collini, Ramesh Karri, Chinmay Hegde, and Siddharth Garg. VeriThoughts: Enabling automated Verilog code generation using reasoning and formal verification. *arXiv preprint arXiv:2505.20302*, 2025.

Yang Zhao, Di Huang, Chongxiao Li, Pengwei Jin, Ziyuan Nan, Tianyun Ma, Lei Qi, Yansong Pan, Zhenxing Zhang, Rui Zhang, et al. CodeV: Empowering LLMs with HDL Generation through Multi-Level Summarization. *arXiv preprint arXiv:2407.10424*, 2024.

Zhuorui Zhao, Bing Li, Grace Li Zhang, and Ulf Schlichtmann. VFocus: Better Verilog Generation from Large Language Model via Focused Reasoning. In *2025 IEEE 38th International System-on-Chip Conference (SOCC)*, pp. 1–6. IEEE, 2025a.

Zhuorui Zhao, Ruidi Qiu, Chao Lin, Grace Li Zhang, Bing Li, and Ulf Schlichtmann. Vrank: Enhancing Verilog code generation from large language models via self-consistency. In *2025 26th International Symposium on Quality Electronic Design (ISQED)*, pp. 1–7. IEEE, 2025b.

Yaoyu Zhu, Di Huang, Hanqi Lyu, Xiaoyun Zhang, Chongxiao Li, Wenxuan Shi, Yutong Wu, Jianan Mu, Jinghua Wang, Yang Zhao, et al. QiMeng-CodeV-R1: Reasoning-enhanced verilog generation. *arXiv preprint arXiv:2505.24183*, 2025.

A Appendix

A.1 Implementation and Experimental Configuration

This appendix provides the implementation and hyperparameter details omitted from the main text. Because the controlled 1k-trace study and the extended 6k-trace study use different teacher configurations, we describe them separately.

A.1.1 Reasoning-Trace Curation

Extended 6k-trace study. This experiment uses 600 natural-language specifications sampled from the VeriThoughts training set and generates 10 trajectories per specification, yielding 6,000 traces for each full curation strategy. For MCTS, the maximum number of search iterations per specification is 10 and the maximum number of child nodes generated at an expansion is 3. Intermediate reasoning-node generations are limited to 2,048 tokens, while terminal rollouts are allowed up to 16,184 tokens. Curation-time generation uses temperature 1.0.

The functional reward uses automatically constructed testbenches adapted from the CodeV-R1 (Zhu et al., 2025) verification open-source material. CoT and the alternative trace-generation strategies are sampled 10 times per specification to match the number of generated trajectories. The answer-only baseline is obtained by post-processing CoT examples to retain only the final answer, while the Best-of-N Plan-MCTS baseline contains 600 traces selected according to average node score.

Table 6: Curation configuration reported for the extended 6k-trace study.

Parameter	**Setting**
Source specifications	600 VeriThoughts prompts
Trajectories per specification	10
Full dataset size	6,000 traces
Maximum MCTS iterations	10
Maximum child nodes per expansion	3
Intermediate-node token limit	2,048
Terminal-rollout token limit	16,184
Curation temperature	1.0
Best-of-N dataset size	600 traces
Best-of-N selection criterion	Average node score

Table 7: Controlled 1k-trace study configuration.

Parameter	Setting
Source specifications	100 VeriThoughts prompts
Trajectories per specification	10
Dataset size per strategy	1,000 traces
Teacher model	DeepSeek-R1
Teacher assignment	All LLM calls and stages
Downstream benchmark	VerilogEval-v1-Human
Generations per evaluation problem	$n = 10$

Table 8: Supervised fine-tuning configuration.

Parameter	Setting
Training framework	LlamaFactory
Learning rate	5×10^{-5}
Maximum sequence length	16,384 tokens
Training epochs	3
Adaptation method	LoRA
LoRA rank	64
Training hardware	1 NVIDIA H200 GPU

Teacher models. In the extended study, DeepSeek-R1 is used for CoT generation, Thought-MCTS, and the reasoning rollouts used by Plan-MCTS. DeepSeek-V3 is used to generate the intermediate non-reasoning Plan-MCTS planning states. All experiments access these models through OpenRouter.

DeepSeek-V3 was used for intermediate Plan-MCTS updates to reduce expansion cost. An LLM is invoked during each expansion, while only the compact plan decision is retained from intermediate or unselected branches. DeepSeek-R1 is then used to materialize selected plans into reasoning and final Verilog.

Controlled 1k-trace study. The controlled study uses 100 specifications sampled from the larger 600-specification set and generates 10 traces per specification, yielding 1,000 examples for each strategy. To remove the teacher-model confound, DeepSeek-R1 is used for every LLM call, including CoT generation, Thought-MCTS expansion and rollout, Plan-CoT planning and rollout, Plan-RandomTree expansion and rollout, and Plan-MCTS expansion and rollout. All controlled methods use the same reward, inference settings, generation budget, student model, and SFT configuration.

A.1.2 Supervised Fine-Tuning

All experiments use LlamaFactory framework scripts open-sourced via VeriThoughts (Yubeaton et al., 2025) for supervised fine-tuning. All compared datasets use the same optimization configuration: a learning rate of 5×10^{-5}, maximum sequence length of 16,384 tokens, three training epochs, and LoRA with rank 64. Training is performed on a single NVIDIA H200 GPU.

The controlled experiments retain the same SFT configuration for all compared datasets. The student LLM used in this study is Qwen2.5-Coder-7B-Instruct.

A.1.3 Downstream Evaluation

For the extended 6k-trace study, downstream evaluation is performed on VerilogEval-v1-Human and the VerilogEval-v2 Code Completion and Spec-to-RTL tasks. Generation is limited to 8,192 tokens, with temperature 0.5 and top-$p = 0.9$. Each benchmark problem is sampled $n = 20$ times. The final complete Verilog module is extracted from each model response using the same post-processing procedure across all evaluated models.

For the controlled 1k-trace study, downstream comparisons are performed on VerilogEval-v1-Human with $n = 10$ generations per problem.

Table 9: Downstream generation configuration reported for the extended study.

Parameter	Setting
Benchmarks	VerilogEval-v1-Human, VerilogEval-v2
VerilogEval-v2 tasks	Code Completion, Spec-to-RTL
Maximum generation length	8,192 tokens
Temperature	0.5
Top-p	0.9
Generations per problem	$n = 20$
Output processing	Consistent final-module extraction

A.2 Additional Algorithm and Formulation Details

This appendix provides additional details of the verifier-guided search procedure described in Section 3. We retain the notation introduced in the main text: for a specification x, an intermediate state is $S_t = (x, h_t)$, where h_t denotes either a free-form reasoning state or a structured hardware plan. The state transition, rollout, verifier reward, UCT selection rule, and action-value update are defined in Section 3.1; here we provide the remaining search mechanics and the full formulation of the Thought-MCTS comparison.

A.2.1 MCTS Search Procedure

For each specification, search begins from the root

$$S_0 = (x, h_0), \tag{10}$$

where h_0 is an empty intermediate representation. The search terminates when the current state is judged ready for Verilog generation or when a maximum search depth D is reached. For each specification, MCTS performs M simulations, each consisting of selection, expansion, rollout, and backpropagation.

Selection. Starting from S_0, search recursively selects among previously expanded actions using the UCT rule defined in Eq. 5. For a state S, $\mathcal{A}_{\text{tree}}(S)$ denotes the set of actions that have already been expanded, $N(S)$ is the number of visits to the state, and $N(S, a)$ is the number of times action a has been selected from S.

We use UCT rather than a prior-weighted PUCT rule because the actions in our setting are variable-length reasoning or planning generations. Calibrated action priors over these multi-token outputs are not available from the teacher models and are not directly comparable across candidate actions.

Expansion. When selection reaches an expandable leaf state $S_\ell = (x, h_\ell)$, the expansion model proposes K candidate actions:

$$\left\{a_\ell^{(1)}, \ldots, a_\ell^{(K)}\right\} \sim \pi_\theta^{\text{expand}}(\cdot \mid S_\ell). \tag{11}$$

Each candidate defines a successor state

$$S_{\ell+1}^{(k)} = T(S_\ell, a_\ell^{(k)}) = \left(x, h_\ell \oplus a_\ell^{(k)}\right), \tag{12}$$

where $\oplus$ is representation-specific. For Thought-MCTS, the candidate action is a free-form reasoning step; for Plan-MCTS, it is a structured update to the hardware plan. One newly created child is then selected for rollout.

Rollout and verification. The selected child is passed to the rollout model, which completes the reasoning process and generates final Verilog as described in Eq. 3. The resulting implementation is evaluated using the terminal verifier reward in Eq. 4. The terminal state therefore associates the specification and intermediate search representation with a materialized reasoning trace and final Verilog implementation.

Backpropagation. The terminal verifier reward is propagated along every state–action edge visited during the simulation. In addition to the running action-value update in Eq. 6, the visitation counts are updated as

$$N(S) \leftarrow N(S) + 1, \tag{13}$$

$$N(S, a) \leftarrow N(S, a) + 1. \tag{14}$$

Consequently, functional feedback observed only after producing complete Verilog contributes to the value estimates of the intermediate reasoning or planning actions that led to that implementation.

A.2.2 THOUGHT-MCTS: FREE-FORM REASONING SEARCH

Thought-MCTS applies the same selection, rollout evaluation, and backpropagation procedure as Plan-MCTS, but directly represents the search state as accumulated natural-language reasoning. After t reasoning decisions,

$$h_t^{\text{thought}} = (z_1, z_2, \ldots, z_t), \tag{15}$$

where each z_i is an LLM-generated reasoning segment. An expansion action proposes the next reasoning segment,

$$a_t^{\text{thought}} = z_{t+1} \sim \pi_\theta^{\text{expand}}(\cdot \mid x, z_1, \ldots, z_t), \tag{16}$$

and the transition appends it to the existing prefix:

$$h_{t+1}^{\text{thought}} = h_t^{\text{thought}} \circ z_{t+1}, \tag{17}$$

where $\circ$ denotes sequence concatenation.

During expansion, candidate reasoning steps are generated conditioned on the specification and accumulated prefix. During rollout, the same prefix conditions generation of the remaining reasoning and final Verilog:

$$(r_T, v_T) \sim \pi_\theta^{\text{rollout}}(\cdot \mid x, z_1, \ldots, z_t). \tag{18}$$

Thus, Thought-MCTS changes only the representation searched by MCTS: intermediate states consist directly of natural-language reasoning rather than explicit hardware-design fields. Because this text is retained in later expansion and rollout contexts, longer reasoning prefixes also increase the amount of intermediate context carried through search.

A.2.3 ADDITIONAL PLAN-MCTS SEARCH DETAILS

As defined in Section 3.2, Plan-MCTS instead represents h_t as a partial hardware plan p_t and each action as a structured update Δp_t. The expansion model may determine which plan field to update based on the current state and may append additional decisions to fields that require further specification. As a result, multiple search steps may refine the same aspect of the design rather than requiring each field to be populated exactly once.

Candidate plan updates are otherwise processed using the same selection, rollout, verification, and backpropagation procedure described above. During rollout, the accumulated plan is supplied as design context to the rollout model, which materializes it into natural-language reasoning and final Verilog:

$$(r_T, v_T) \sim \pi_\theta^{\text{rollout}}(\cdot \mid x, p_T). \tag{19}$$

The verifier evaluates only v_T, and the resulting terminal reward is propagated through the plan-update actions encountered along the selected search trajectory.

A.3 ADDITIONAL CONTROLLED ANALYSES

A.3.1 COMPRESSION VERSUS HARDWARE-SPECIFIC STRUCTURE

The controlled experiments in Section 5.2 show that preserving the structured hardware plan substantially improves downstream SFT. To determine whether this effect can instead be explained by reducing the length of the intermediate search state, we additionally evaluate **Summary-MCTS**.

Summary-MCTS is identical to Thought-MCTS except that, after each generated thought, an additional LLM call condenses the accumulated reasoning before the next expansion. It therefore

Table 10: Comparison of reasoning-state compression and hardware-specific structure in the controlled study.

Method / representation	pass@1	pass@5	pass@10
Thought-MCTS, full free-form prefix	27.37	46.71	51.28
Summary-MCTS, compressed free-form prefix	29.23	46.43	51.92
Plan-MCTS, rollout only	29.17	46.48	52.56
Plan-MCTS + rendered NL plan	30.0	47.6	55.1
Plan-MCTS + explicit structured plan	31.5	50.8	57.7

provides a compression control without introducing the hardware-specific interface, timing, state, behavior, or implementation fields used by Plan-MCTS.

Summarization modestly improves Thought-MCTS at pass@1 by 1.86 points and at pass@10 by 0.64 points, but does not reproduce the gains obtained from explicit Plan-MCTS supervision. The explicit structured-plan target exceeds Summary-MCTS by 2.27/4.37/5.78 points at pass@1/5/10. These results indicate that compression alone is insufficient to explain the benefit of the structured hardware representation.

The natural-language and explicit-plan targets provide a complementary comparison. Both are derived from the same Plan-MCTS trajectories and underlying Verilog implementations, but retaining the explicit structured representation improves pass@1/5/10 by 1.5/3.2/2.6 points over the rendered natural-language plan. Thus, the organization of the intermediate hardware decisions also affects their effectiveness as SFT supervision.

A.4 Additional Generalization Experiments

Beyond the controlled and extended-scale evaluations, we conducted two additional studies to examine whether the observed benefits extend across teacher models and larger training sets.

A.4.1 Generalization to a Second Teacher Model

To examine dependence on the teacher model, we additionally use GPT-5.4-mini for reasoning-trace curation while retaining the same Qwen student. Table 11 reports downstream performance for Thought-MCTS and Plan-MCTS.

Table 11: Generalization to GPT-5.4-mini as the curation teacher.

Teacher strategy	pass@1	pass@5	pass@10
Thought-MCTS	33.21	45.90	50.64
Plan-MCTS	33.78	46.21	50.00

Both search representations transfer to the second teacher. Plan-MCTS is slightly stronger at pass@1 and pass@5, while Thought-MCTS is slightly stronger at pass@10. The small differences do not establish a consistent ranking between the two variants, but indicate that the verifier-guided curation framework is not restricted to the teacher configuration.

Table 12: Scaling standard Thought-MCTS supervision to larger training datasets.

Training configuration	pass@1	pass@5	pass@10
Qwen2.5 baseline	25.6	38.9	42.6
SFT, 10k MCTS traces	38.2	51.8	56.6
SFT, 15k MCTS traces	41.7	54.9	59.6
SFT, 25k MCTS traces	43.7	54.1	58.0
SFT, 40k MCTS traces	45.6	56.0	58.9
SFT, 60k MCTS traces	49.0	60.0	63.3

A.4.2 SCALING MCTS-CURATED SUPERVISION

We additionally examine whether MCTS-curated supervision continues to benefit the student as the training dataset grows. For this experiment, we scale the standard Thought-MCTS pipeline to 6,000 unique prompts and up to 60,000 reasoning traces. Table 12 reports the resulting pass@k performance.

Pass@1 increases consistently with dataset scale, reaching 49.0 at 60k traces, corresponding to a 23.4-point improvement over the base model and a 10.8-point improvement over the 10k-trace model. Pass@5 and pass@10 also improve substantially overall, although the intermediate results are not strictly monotonic. These results provide evidence that the benefits of MCTS-curated supervision persist at substantially larger prompt and dataset scales.