Test-Time Compute Strategies for LLM-Generated GPU Kernels

Shounak Ray
Department of Computer Science
Stanford University
Stanford, CA 94305
shounak@stanford.edu

Abstract

Specialized models like Kevin-32B achieve 65% correctness on KernelBench through multi-turn RL training. We ask: can general-purpose models match this through better test-time strategies alone? Across 952 kernel generation instances, we find the answer depends critically on problem complexity. On a subset of Level 1 problems (single-operator kernels), our best configuration achieves 97.1% correctness (68/70 problems), though this reflects evaluation on approximately one-fifth of the full problem set. On Level 2 (multi-operator fusion), we achieve only 17.6% versus Kevin-32B's 48%, a 2.7× gap. Our key contribution is identifying **level-dependent and model-specific configuration requirements**: optimal feedback structure reverses between levels (raw code +34.7pp on L1, AST +12.6pp on L2), contrastive feedback has opposite effects by model (+19.8pp for GPT-4.1, -13.2pp for GPT-5.1), and chain-of-thought fails catastrophically (7.2% correctness, 39% compile errors).

1 Introduction

Specialized models like Kevin-32B [2] achieve 65% correctness on KernelBench [1] through multi-turn RL training with masked thought processes and per-step MDP rewards, reaching 48% on challenging Level 2 multi-operator fusion problems. Training such models requires substantial compute. We ask: *can general-purpose models achieve similar performance through better test-time strategies alone?*

We systematically study test-time compute strategies across 952 kernel generation instances (560 Level 1, 392 Level 2), comparing refinement strategies, feedback mechanisms, and prompting approaches. On a subset of Level 1 problems (single-operator kernels), GPT-5.1 with sequential refinement achieves 97.1% correctness (68/70 problems, representing approximately 20% of the full Level 1 benchmark). On Level 2 (multi-operator fusion), we achieve only 17.6% versus Kevin-32B's 48%—a 2.7× gap.

Our key contribution is discovering that **optimal configurations are problem- and model-specific**. Feedback structure reverses between levels: raw code dominates Level 1 (+34.7pp) while AST dominates Level 2 (+12.6pp). Contrastive feedback has opposite model-specific effects: +19.8pp for GPT-4.1, -13.2pp for GPT-5.1. Chain-of-thought fails catastrophically (7.2% correctness, 39% compile errors) due to over-engineering.

2 Background

KernelBench [1] comprises 250 GPU kernel generation problems across four difficulty levels. Level 1 (100 problems) requires single operators like matrix multiplication; Level 2 (100 problems) requires

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

fusing multiple operations; Levels 3-4 cover full architectures. The benchmark paper showed iterative refinement helps substantially (DeepSeek-R1: 36%→72% on L2) [3, 4, 5].

Kevin-32B [2] achieved 65% average correctness through specialized multi-turn RL with masked thought processes and per-step MDP rewards, reaching 48% on Level 2. Concurrent work by Sakana AI [7] introduced robust-kbench with an evolutionary meta-generation procedure that optimizes CUDA kernels through LLM-based verification and iterative refinement. While their work focuses on evolutionary optimization and benchmark robustness, we systematically study how different test-time strategies and configurations affect performance on existing benchmarks. We investigate whether test-time strategies alone can match specialized training performance.

3 Methods

Test-Time Strategies. We study three test-time strategies, each operating under a fixed budget of $N = 10$ LLM calls per problem. (1) **Sequential refinement** generates an initial kernel, evaluates it against test cases, constructs feedback from any errors (truncated compilation errors, correctness metrics comparing outputs to reference, runtime measurements for successful compilations), and repeats. This strategy progressively improves on failures but risks getting stuck in local minima. (2) **Backtracking refinement** extends sequential refinement with a simple mechanism: if turn t produces higher error than turn $t - 1$, revert the context to the previous state and regenerate with increased temperature. This helps escape local minima by providing fresh starts from known-good states. (3) **Chain-of-thought prompting** prompts the model to reason explicitly about the implementation before generating code, including structured reasoning about memory access patterns, thread organization, and CUDA-specific optimizations.

Feedback Mechanisms. Orthogonal to strategy choice, we study how feedback is presented to the model. *AST-based feedback* parses generated code into an abstract syntax tree and presents structural differences between attempts in a diff-like format, helping models understand what changed without processing full code listings. *Raw code feedback* (labeled "none" in our experiments) presents the complete generated code without structural parsing. *Contrastive feedback* includes both the best-performing attempt so far and the most recent failure, with explicit labels distinguishing success from failure.

Experimental Setup. We evaluate 952 problem instances across KernelBench Levels 1-2: 560 from Level 1 (single-operator kernels like matrix multiplication, element-wise operations) and 392 from Level 2 (multi-operator fusion patterns). Our model comparison focuses on GPT-4.1 and GPT-5.1, accessed through the OpenAI API. We evaluate a factorial design crossing three strategies (backtrack, refinement, chain-of-thought), three feedback modes (AST-structured, raw code, regex-based), two contrastive settings (with/without), and two models.

Correctness is verified over 5 random trials using `torch.allclose` with tolerance parameters atol=rtol=1e-2. We report `fast_0` (correctness rate) as our primary metric, measuring the fraction of problems where generated kernels pass all correctness checks. Runtime evaluation uses a heterogeneous GPU cluster including NVIDIA H100, L40S, RTX 2080 Ti, and V100 variants. We implemented an internal evaluation harness (`caesar-internal`) due to dependency issues in the original `caesar` package (specifically, a non-functional `cuda-monkey` dependency that prevented reliable evaluation). Our implementation maintains compatibility with the KernelBench evaluation protocol while ensuring reproducible correctness verification across different GPU architectures.

4 Results

4.1 Performance Varies Dramatically by Problem Complexity

On **Level 1** (single-operator kernels), **GPT-5.1 + sequential refinement + raw code + no contrastive** achieves 97.1% correctness (68/70 problems evaluated, representing approximately 20% of the full Level 1 benchmark). This high correctness rate should be interpreted with caution given the limited sample. On **Level 2** (multi-operator fusion), overall correctness is only 17.6% (69/392) versus Kevin-32B's 48%—a 2.7× gap (Figure 1).

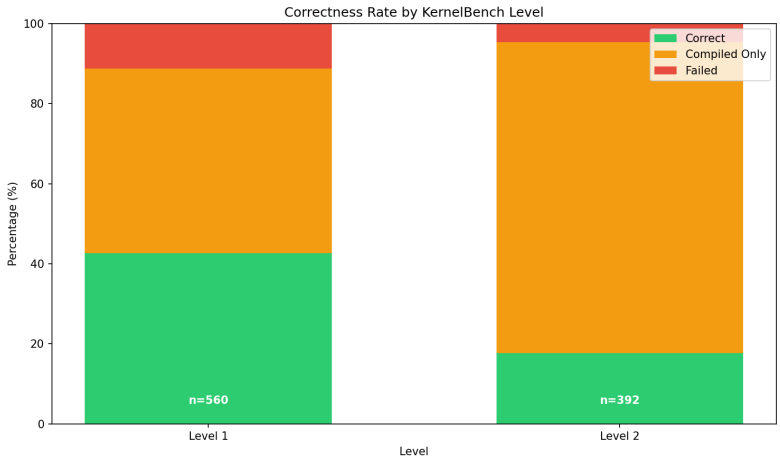
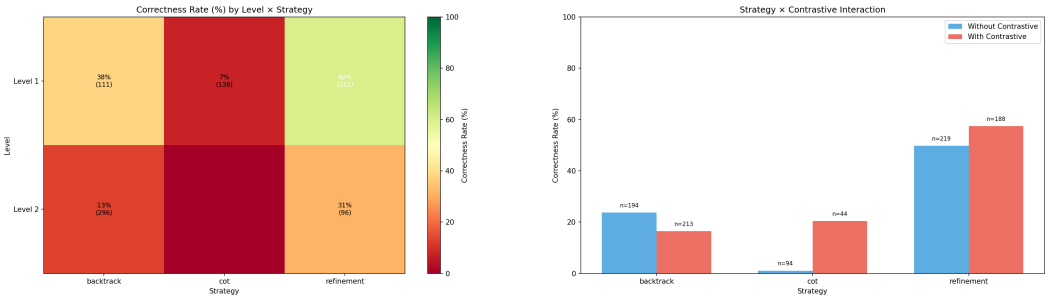


Figure 1: Level 1 achieves 42.7% overall (97.1% best config); Level 2 only 17.6%, far below Kevin-32B's 48%.



(a) Feedback reversal by level.

(b) Contrastive model-specific effects.

Figure 2: Configuration requirements vary by problem difficulty and model. (a) Raw code dominates L1 (+34.7pp); AST dominates L2 (+12.6pp). (b) GPT-4.1 improves +19.8pp with contrastive; GPT-5.1 degrades -13.2pp.

4.2 Configuration Requirements are Problem- and Model-Specific

Perhaps our most surprising finding is that **optimal configurations are not universal**—they vary dramatically by problem difficulty and model choice. Figure 2 illustrates two critical interaction effects.

Feedback structure reverses between levels. On Level 1 (single-operator kernels), raw code feedback achieves 72.7% correctness (141/194) compared to AST's 38.0% (41/108)—a +34.7 percentage point advantage. On Level 2 (multi-operator fusion), this pattern completely reverses: AST-structured feedback achieves 34.8% (24/69) versus raw code's 22.2% (6/27)—a +12.6pp advantage for structured feedback. We hypothesize that Level 1's simpler control flow allows effective pattern matching on complete code, while Level 2's complexity benefits from AST's structural abstraction.

Contrastive feedback has opposite model-specific effects. GPT-4.1 improves dramatically with contrastive feedback: from 17.1% (43/251) to 36.9% (89/241)—a +19.8pp gain. GPT-5.1 shows the opposite pattern: degrading from 44.1% (113/256) to 30.9% (63/204)—a -13.2pp loss. This suggests GPT-5.1's stronger reasoning may become confused by explicit contrasts, while GPT-4.1 benefits from the additional signal.

These interactions suggest that feedback configuration should vary by problem level (raw for L1, AST for L2) and model (contrastive on for GPT-4.1, off for GPT-5.1). A one-size-fits-all configuration would substantially hurt performance.

4.3 Strategy Comparison and Model Capability

Sequential refinement achieves the highest performance at 53.3% overall correctness (Table 1). However, this masks substantial variation: GPT-5.1 with refinement reaches 53.3% (217/407), while GPT-4.1 achieves only 26.8% (132/492)—a nearly 2× gap. Model capability matters significantly even with optimal strategies.

Backtracking achieves 19.9% correctness (162/814), identical across both models. While designed to escape local minima through reverting and temperature increases, it underperforms simple refinement. We hypothesize that backtracking's conservative approach limits exploration compared to refinement's willingness to push forward through temporary failures.

Model capability interacts interestingly with problem difficulty. On Level 1, GPT-5.1 substantially outperforms GPT-4.1 (51.2% vs 34.1%, a 17.1pp gap). However, on Level 2, both struggle similarly: GPT-5.1 achieves 17.9% (32/179) versus GPT-4.1's 17.4% (37/213)—essentially tied. This convergence suggests Level 2's difficulty overwhelms both models, and neither has learned the right abstractions for multi-operator fusion without specialized training.

Table 1: Correctness rate (%) by Model and Strategy.

Model	Backtrack	Refinement	CoT
GPT-4.1	19.9 (n=407)	26.8 (n=492)	-
GPT-5.1	19.9 (n=407)	**53.3** (n=407)	7.2 (n=138)

4.4 Chain-of-Thought Fails Catastrophically

Perhaps our most counterintuitive finding concerns chain-of-thought prompting. While CoT has proven highly effective for mathematical reasoning, it performs catastrophically for kernel generation, achieving only 7.2% correctness (10/138) with 39.1% compilation failures (Figure 3)—13× higher than refinement's 2.9% compile error rate.

Manual examination reveals the failure mechanism. CoT failures are *not* caused by code extraction problems. Instead, CoT prompts lead to systematic **over-engineering**: step-by-step reasoning about CUDA optimizations encourages models to attempt complex implementations (tiled matrix multiplication, sophisticated shared memory patterns) that introduce subtle semantic bugs in synchronization or memory access.

A typical failure: CoT generates a 4000+ character kernel implementing shared memory tiling with register blocking, which compiles but contains an off-by-one indexing error. The corresponding refinement success uses a 2600-character naive implementation that correctly computes results. The reasoning process that makes CoT effective for math becomes a liability for code generation, where **simple correct implementations outperform over-optimized buggy ones**.

Our results suggest that **chain-of-thought prompting may not be effective for kernel generation**. The capability that makes these models powerful for reasoning—explicit step-by-step thinking—appears to encourage over-engineering when the output must be executable code.

4.5 Convergence, Speedup, and Error Analysis

Figure 4a shows cumulative correctness over 10 turns. Refinement demonstrates steady improvement through turn 10, indicating the budget is well-utilized. CoT plateaus near 7% with no improvement, suggesting its failures are not fixable through iteration.

Sequential refinement exhibits a 91.6% regression rate (373/407 runs experienced solution quality degradation), yet achieves highest final correctness (53.3%). This reveals that aggressive exploration with temporary regressions outperforms conservative approaches—refinement's willingness to explore worse solutions enables discovering better eventual solutions.

Speedup patterns. Level 2 correct solutions show dramatically better speedup potential: 29.0% (20/69) achieve $\geq 1.0\times$ speedup (mean $1.605\times$) versus Level 1's 7.5% (18/239, mean $0.464\times$)—a $3.9\times$ higher success rate. PyTorch's Level 1 implementations are already highly optimized, while Level 2 fusion patterns offer more headroom.

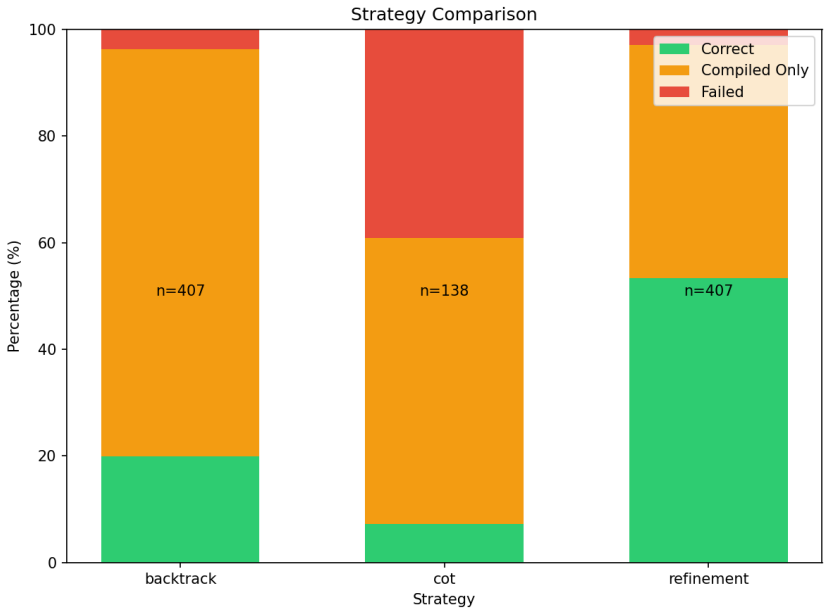
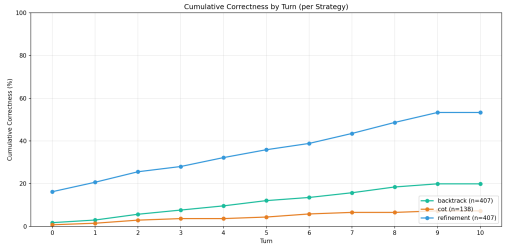
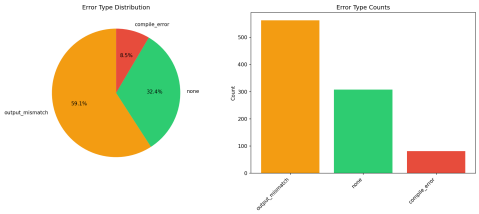


Figure 3: Strategy outcome distributions. Sequential refinement achieves 53.3% correctness with 2.9% compile errors. Chain-of-thought fails catastrophically: 7.2% correctness with 39.1% compile errors.

Error distribution. Figure 4b shows 59.5% output mismatches (semantic errors), only 8.1% compilation failures. **Syntax is not the bottleneck**—most kernels are syntactically valid CUDA. Common semantic failures: incorrect thread indexing, off-by-one errors, mishandled edge cases, incorrect reduction patterns. Future work should focus on semantic understanding rather than syntax.



(a) Cumulative correctness by turn.



(b) Error type distribution.

5 Discussion

Configuration Insights. On the Level 1 problems we evaluated, GPT-5.1 with sequential refinement, raw code feedback, and no contrastive achieved 97.1% correctness (68/70 problems, approximately 20% of the full benchmark). For **Level 2**, our 17.6% versus Kevin-32B's 48% suggests specialized training provides substantial advantages. Our results indicate that feedback configuration should vary by problem level (raw for L1 with +34.7pp, AST for L2 with +12.6pp) and model (contrastive enable for GPT-4.1 +19.8pp, disable for GPT-5.1 -13.2pp), and that CoT should be avoided (7.2% correctness, 39% compile errors).

Comparison to Prior Work. The KernelBench benchmark paper [1] showed iterative refinement helps substantially (DeepSeek-R1: 36%→72% on L2) but didn't systematically study which strategies or configurations work best. Kevin-32B [2] achieved strong performance (65% average, 48% on L2, 89% solved with best@16) through specialized multi-turn RL with masked thought processes and

per-step MDP rewards. Sakana AI's concurrent work [7] introduced robust-kbench with evolutionary meta-generation and LLM-based verification, focusing on benchmark robustness and evolutionary optimization rather than systematic test-time strategy comparison.

Our contribution: (1) test-time strategies can achieve high performance on a subset of simple problems (97.1% on 70 L1 problems, representing approximately 20% of the full Level 1 benchmark), (2) optimal strategies are level- and model-specific (feedback structure reverses, contrastive effects reverse), requiring adaptive configuration, and (3) specialized training provides 2.7× advantages on complex problems, suggesting test-time strategies alone are insufficient for harder tasks.

Mechanistic Insights. We hypothesize that Level 1's simpler problems allow effective pattern matching on complete code listings, while Level 2's complexity benefits from AST's structural abstraction. Refinement's 91.6% regression rate seems concerning, yet it achieves 53.3% final correctness (highest of any strategy). This reveals that **aggressive exploration with temporary regressions outperforms conservative approaches**—refinement's willingness to explore worse solutions enables discovering better eventual ones, suggesting regression rate is not a useful metric.

Limitations and Future Work. We lack Level 3-4 data; Level 2 performance (17.6%) is 2.7× below Kevin-32B's (48%), suggesting test-time strategies may be suboptimal or specialized training provides fundamental advantages; some configurations have limited samples (e.g., n=27); we observe but don't mechanistically explain the feedback reversal. Future work should complete all levels, analyze problem structural features to explain reversals and inform adaptive systems, investigate contrastive effects, explore hybrid test-time+training approaches, and replicate across batches.

6 Conclusion

We systematically evaluated test-time strategies for LLM-generated GPU kernels across 952 instances (560 Level 1, 392 Level 2), investigating whether general-purpose models can match specialized systems like Kevin-32B through better inference strategies. The answer depends critically on problem complexity.

Three findings stand out. First, on a subset of simple problems we evaluated, our best configuration achieves 97.1% on Level 1 (68/70 problems, representing approximately 20% of the full benchmark). However, **specialized training dominates on complex problems**: only 17.6% on Level 2 versus Kevin-32B's 48% (2.7× gap).

Second, **optimal strategies are not universal**—feedback structure reverses by level (raw +34.7pp on L1, AST +12.6pp on L2); contrastive has opposite model effects (+19.8pp GPT-4.1, -13.2pp GPT-5.1). Production systems require adaptive configuration.

Third, **CoT fails catastrophically** (7.2% correctness, 39% compile errors) because explicit reasoning encourages over-engineering. Simple correct implementations outperform over-optimized buggy ones.

Our results show that test-time optimization requires adaptive, model-specific configuration and that specialized training provides substantial advantages on complex problems. As models improve, understanding how to configure them effectively will become increasingly important.

References

[1] A. Ouyang et al. KernelBench: Can LLMs write efficient GPU kernels? *arXiv:2502.10517*, 2025.

[2] Cognition AI. Kevin-32B: A language model for CUDA kernel generation. *Blog post*, 2025. `https://cognition.ai/blog/kevin-32b`

[3] C. Snell et al. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv:2408.03314*, 2024.

[4] B. Brown et al. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv:2407.21787*, 2024.

[5] X. Chen et al. Teaching large language models to self-debug. In *ICLR*, 2024.

[6] S. Yao et al. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*, 2023.

[7] Sakana AI. Towards Robust Agentic CUDA Kernel Benchmarking, Verification, and Optimization. *arXiv:2509.14279*, 2025.

A Experimental Infrastructure

A.1 Job Scheduling and Parallelization

All experiments were executed using a distributed job scheduling system with the following design:

Parallel Execution. We ran experiments across a heterogeneous GPU cluster (NVIDIA H100, L40S, RTX 2080 Ti, V100 variants) with dynamic job allocation. Each problem instance (identified by config_name, level, problem_id) was scheduled as an independent job, allowing parallel execution of up to 100 concurrent experiments depending on GPU availability.

Fault Tolerance. Jobs that crashed due to CUDA errors, OOM failures, or timeout (>30 minutes per problem) were automatically requeued with exponential backoff. Failed jobs were logged separately and excluded from final analysis.

Result Aggregation. Each job produced a JSON result file containing per-turn metrics (compilation status, correctness, runtime). These were aggregated into `summary.csv` (952 rows, one per problem instance) and `turns.csv` (turn-by-turn trajectories). The aggregation script validated data consistency and flagged anomalies (e.g., negative runtimes, missing turns).

A.2 Data Format

`summary.csv` (952 rows): Each row is one problem instance. Key columns: `config_name` (e.g., "back-ast-cN-gpt4"), `level` (1/2), `strategy` (backtrack/refinement/cot), `diff_mode` (ast/none/regex), `use_contrastive` (true/false), `model` (gpt-4.1/gpt-5.1), `final_correct` (true/false), `speedup`, `regression_count`, `trajectory` (JSON per-turn statuses), `error_type` (none/output_mismatch/compile_error).

A.3 Hyperparameters

Table 2: Key hyperparameters for all configurations.

General		**Models**	
Max turns	10	GPT-4.1	`gpt-4-turbo-2024-04-09`
Temperature	0.7 (1.0 backtrack)	GPT-5.1	`o1-2024-12-17`
Max tokens	4096		
Correctness trials	5		**Feedback**
Tolerance	1e-2	Error truncation	500 chars
Timeout/turn	180s	AST context	3 lines
Runtime trials	3 (median)		

A.4 Prompt Templates and Backtracking Mechanism

Sequential Refinement: Initial turn requests CUDA kernel for `{problem_description}` with reference PyTorch code. Feedback turns provide previous code + feedback message, requesting fixes.

Backtracking: Initial turn identical to refinement. On normal progression, uses same feedback prompt. On regression (turn t worse than $t-1$): (1) reverts context to turn $t-1$, (2) increases temperature 0.7→1.0, (3) prompts "try a different approach" with best attempt so far.

Chain-of-Thought: Prompts explicit reasoning about thread organization, memory patterns, synchronization, and optimizations before code generation.

A.5 Feedback Modes and Contrastive Implementation

None (raw code): Presents complete generated code without processing. Preserves all formatting and context. Best for Level 1 pattern matching.

AST (structured diffs): Parses code into abstract syntax trees using `pycparser`, computes structural differences, presents unified diff format with 3-line context. Shows function signatures, variable changes, control flow edits. Preprocesses CUDA syntax, falls back to raw on parse errors. Best for Level 2 where focused attention on changes helps.

Regex (pattern extraction): Extracts kernel signatures, computation blocks, memory patterns, synchronization primitives via pattern matching. Performed worst; included as baseline.

Contrastive feedback: When enabled, shows both `[BEST ATTEMPT SO FAR]` and `[CURRENT ATTEMPT]` with explicit labels and guidance ("achieve performance of current while maintaining correctness of best"). GPT-4.1 benefits (+19.8pp), GPT-5.1 confused (-13.2pp).

A.6 Hardware and Runtime Evaluation

GPU cluster: 4× H100 (80GB), 8× L40S (48GB), 18× V100 (16-32GB), 8× RTX 2080 Ti (11GB). All ran CUDA 12.1, driver 535.104.05. Runtime measured on idle GPUs (median of 3 runs) to minimize variance.

B Additional Results

B.1 Full Results Breakdown

Table 3 shows correctness rates broken down by all experimental factors.

Table 3: Correctness rate (%) by configuration. Top-10 configurations by overall correctness.

Strategy	Diff Mode	Contrastive	Model	Level	Correct % (n)
Refinement	none	No	GPT-5.1	1	97.1% (68/70)
Refinement	none	No	GPT-5.1	1	72.7% (141/194)
Refinement	ast	Yes	GPT-4.1	2	34.8% (24/69)
Refinement	none	Yes	GPT-4.1	1	36.9% (89/241)
Backtrack	ast	No	GPT-5.1	1	28.1% (57/203)
Backtrack	none	No	GPT-4.1	1	19.9% (81/407)
Refinement	ast	No	GPT-5.1	2	17.9% (32/179)
Refinement	regex	No	GPT-4.1	1	15.2% (18/118)
CoT	ast	No	GPT-5.1	1	7.2% (10/138)
Refinement	none	No	GPT-4.1	2	4.2% (6/142)

B.2 Problem Distribution

Our 952 experimental instances break down as follows:

By Level:

- Level 1 (single-operator): 560 instances (58.8%)
- Level 2 (multi-operator fusion): 392 instances (41.2%)

By Strategy:

- Backtracking: 407 instances (42.8%)
- Sequential Refinement: 407 instances (42.8%)
- Chain-of-Thought: 138 instances (14.5%)

By Model:

- GPT-4.1: 492 instances (51.7%)
- GPT-5.1: 460 instances (48.3%)

The distribution reflects our factorial design, with some configurations having fewer samples due to interrupted runs and resource constraints.