HEURIGUIDE: CALIBRATING LLM AUTHORITY FOR TEST-TIME HEURISTIC DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have recently emerged as a powerful tool for automated heuristic design (AHD) in combinatorial optimization. Most existing LLM-based AHD algorithms employ an outer agent to optimize a strategy, an algorithmic procedure that constructs a solution, during development and then deploy it unchanged, missing the opportunity to adapt the strategy to individual test-time instances. We therefore systematically investigate placing an inner LLM agent directly into the test-time solving process. Since LLM-generated output can be unstable, the central design question is how to calibrate the inner agent's authority at test time. By imposing progressively greater procedural guidance on the inner agent, we propose three inner-agent organizations: a *solution generator* that directly generates a solution for each test instance, a *strategy adapter* that selects and adapts existing strategies, and a *strategy router* that only selects among strategies curated by the outer agent. We evaluate these three agent organizations on HeuriGym, a benchmark comprising nine combinatorial optimization problems across four domains. Averaged across all nine problems, QCI increases monotonically with procedural guidance, from 0.599 for the solution generator to 0.787 for the strategy adapter and 0.805 for the strategy router, compared with 0.728 for the outer-agent-only baseline. The strategy router also performs best on most individual problems. However, on E-graph Extraction, a problem for which the baseline fails entirely, success requires allowing the inner agent to generate beyond the curated strategies. Together, these results suggest using strong procedural guidance by default while relaxing procedural guidance when curated strategies prove insufficient.

1 INTRODUCTION

Large language models (LLMs) have emerged as a powerful tool for automated heuristic design (AHD) in combinatorial optimization. Combinatorial optimization seeks the best solution from a large discrete search space, and many practically important problems in this class are NP-hard. Rather than relying on a human to handcraft every rule, LLM-based AHD systems ask an LLM to propose, repair, and evolve solver programs (Romera-Paredes et al., 2024; Chen et al., 2026; Novikov et al., 2025; Yu et al., 2026). This approach has greatly expanded the space of heuristics that can be explored during development. However, the practical goal is not only to discover one strong strategy, but also to solve test instances that differ substantially from the demonstration instances in size, structure, and difficulty.

Most existing LLM-based AHD algorithms employ an outer agent to optimize a strategy, an algorithmic procedure that constructs a solution, during development and then deploy it unchanged, missing the opportunity to adapt the strategy to individual test-time instances. A fixed strategy may branch on instance features, but it cannot choose among independently discovered procedures or create a new procedure when an instance falls outside what was encountered during development. This leaves potential gains from specialization on the table.

A narrow set of learned strategies may underfit the diversity of test instances. At the other extreme, generating a new solver for every test instance may waste time and compute resources to rediscover procedures that may have already been verified. We therefore seek a practical middle ground:

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.

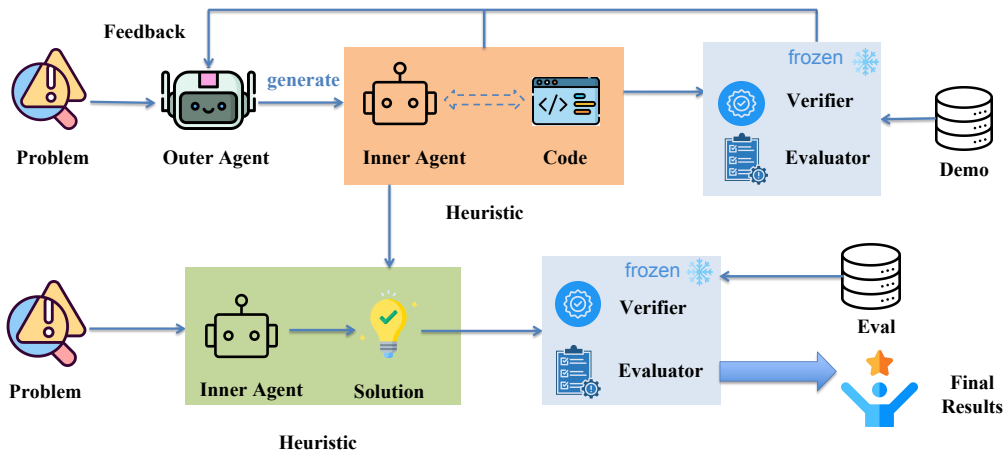


Figure 1: The HeuriGuide workflow. During development (top), an outer agent iteratively improves what will be deployed, guided by a frozen verifier and evaluator on demonstration instances; at test time (bottom), the deployed heuristic is frozen, an inner agent acts inside it on each new instance, and the same frozen judges score the final solutions.

enough authority to respond to instance-level differences, but proper structure to ensure reliable solutions.

We introduce HeuriGuide, a framework that provides guidance on how to calibrate LLM authority for test-time heuristic design. During development, an outer agent and an inner agent work together to discover a portfolio of strategies or iterate code. At test time, an inner agent inspects the current instance, and its organization determines what it is allowed to produce. HeuriGuide provides three organizations: a *solution generator* that writes the solution directly, a *strategy adapter* that can select an existing strategy or write an adapted one, and a *strategy router* that only selects among frozen, verifier-tested strategies. These three organizations form a controlled ladder from unrestricted generation to procedural reuse, allowing us to separate the value of generating, adapting, and choosing at test time.

Across the nine HeuriGym problems, the main trend is clear: more procedural guidance works better. QCI rises from 0.599 for the solution generator to 0.787 for the strategy adapter and 0.805 for the strategy router, compared with 0.728 for the outer-agent-only benchmark baseline. This work makes three contributions:

- We introduce HeuriGuide, the first framework to systematically calibrate the authority of a test-time inner agent for heuristic design.
- In HeuriGuide, the strategy adapter and the strategy router outperform the benchmark baseline and match the expert reference on several problems; with the same underlying model (GPT-5.4), HeuriGuide also makes E-graph Extraction solvable, a problem on which the benchmark reports a QCI of zero.
- Comparing the three organizations, we show that a constrained test-time action space is generally more effective than free-form generation, yet the advantage can reverse where the curated strategies cannot cover the test instances: strong procedural guidance by default, relaxed where they prove insufficient.

2 Related Work

LLM-based Heuristic Discovery. Traditional metaheuristics search the solution space through human-designed representations, neighborhoods, and update rules, usually balancing intensification against diversification (Blum & Roli, 2003). Hyper-heuristics move the search up one level by selecting or generating heuristics rather than solutions (Burke et al., 2013). Closely related, learning-to-search methods learn policies for sequential construction decisions (Daumé et al., 2009), while

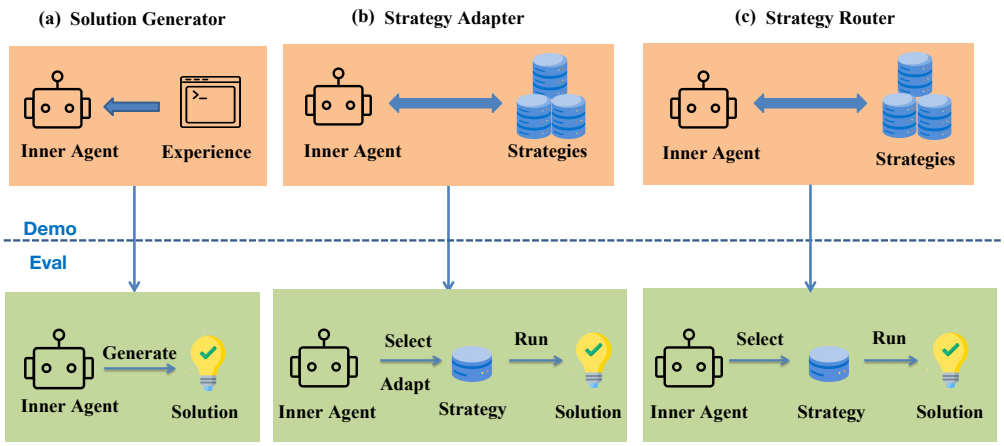


Figure 2: The three inner-agent organizations during development (top) and at test time (bottom), ordered left to right by increasing procedural guidance: (a) the solution generator constructs each solution directly from learned experience; (b) the strategy adapter selects from the portfolio and may write new strategies; (c) the strategy router only selects among frozen, verifier-tested strategies.

metareasoning and algorithm selection study how to choose computations or solvers under limited resources (Russell, 1991; Lieder et al., 2014; Rice, 1976; Xu et al., 2008). LLM-based AHD inherits these ideas, but changes how new heuristics are proposed. An LLM can write and revise executable code from evaluator feedback, rather than search only through a fixed neighborhood or hand-written grammar. EoH, FunSearch, and ReEvo evolve heuristic code; EoH-S evolves complementary heuristic sets across problem instances; HSEvo and MEoH improve diversity or optimize multiple criteria; MOTIF extends the target from one heuristic to interacting solver components; and AlphaEvolve and LaGO scale or decompose evaluator-guided program improvement (Liu et al., 2024; Romera-Paredes et al., 2024; Ye et al., 2024; Liu et al., 2025; Dat et al., 2025; Yao et al., 2025; Kiet et al., 2026; Novikov et al., 2025; Yu et al., 2026). EoH-S focuses on constructing a complementary portfolio during development, whereas HeuriGuide studies how an inner agent may use or move beyond such a portfolio at test time. RAISE searches for adversarial instances to make this development loop more robust (Liu et al., 2026), while HeuriGym supplies executable environments across multiple combinatorial domains (Chen et al., 2026). Taken together, these methods mainly improve what is discovered before deployment. HeuriGuide instead asks how that experience should be used when a new instance must actually be solved.

Experiential Learning and Test-time Adaptation. Experiential learning turns execution histories, such as execution results, tool outputs, and feedback, into knowledge that an agent can reuse without changing the underlying model weights. ExpeL and Reflexion retain feedback-derived experience for later trials (Zhao et al., 2024; Shinn et al., 2023). GEPA reflects over full system trajectories to evolve prompts (Agrawal et al., 2026). TextGrad propagates natural-language critiques through a compound system, while Self-Refine iteratively improves an individual test-time output using self-feedback (Yuksekgonul et al., 2025; Madaan et al., 2023). Test-time adaptation, more broadly, changes a trained system after it observes test inputs instead of treating deployment as a fixed forward pass (Wang et al., 2021). Related work also optimizes LM pipelines or harness code around model calls (Khattab et al., 2024; Lee et al., 2026). In HeuriGuide, the model weights remain fixed; what can adapt is the decision procedure around the model. HeurAgenix is the closest work in combinatorial optimization. It first develops a set of heuristics and then dynamically selects a heuristic for each problem state (Yang et al., 2025). Its test-time degree of freedom is therefore selection, closest to our strategy router. Within HeuriGuide, we hold the benchmark, verifier, and development protocol fixed while varying the inner agent: it may act as a solution generator that writes the answer directly, a strategy router that selects a frozen strategy, or a strategy adapter that may also generate and repair strategy code.

Table 1: Structure of the routing table: one signature entry shown in full; support counts the demonstration measurements behind each mean cost, and an unseen signature falls back to the global default.

Routing table	
signature (prop_1=large, prop_2=few, ...)	
winner	s_2
candidates	s_2: mean cost 412, support 4
	s_5: mean cost 428, support 4
	s_1: mean cost 517, support 3
signature (prop_1=small, ...)	
	⋮
global default: s_2, with its own candidates	

3 THE HEURIGUIDE FRAMEWORK

As shown in Figure 1, HeuriGuide separates heuristic design into development and test-time solving. During development, HeuriGuide learns from demonstration instances guided by a frozen verifier and evaluator, and distills this experience into a deployable heuristic; once development ends, the heuristic is frozen. At test time, an inner agent solves each new instance guided by the frozen heuristic. Since all three organizations share this two-phase setting and evaluation protocol, this setup enables a structured comparison of alternative test-time action spaces for the inner agent. As illustrated in Figure 2, the inner agent may generate a solution, adapt a strategy, or select among verified strategies. We first describe the solution generator, followed by the two portfolio-based organizations.

Solution Generator During development, the solution generator (Figure 2a) distills feedback from demonstration attempts into experience encoded in a problem-specific harness. At test time, the inner agent constructs a fresh solution for each instance guided by the frozen harness. Its defining interface is *direct construction*: the inner agent returns a solution rather than a strategy identifier or executable strategy.

The development experience combines verifier errors, evaluator costs, and execution metadata about response acceptance, repair, and fallback use. Across generations, the outer agent uses these signals to refine the instance representation, local validity checks, and the policies for repair, retry, and fallback. The resulting experience is retained in the harness's prompt and control logic, without updating the model weights or building a reusable strategy portfolio.

At test time, the harness presents the current instance to the inner agent, parses and checks its response, and may request a bounded retry or improvement before falling back. The frozen verifier and evaluator then judge the final output. Unlike the strategy router and the strategy adapter, the inner agent directly generates a solution, providing the greatest instance-level flexibility but the least procedural guidance.

Strategy Router The strategy router (Figure 2c) spans both phases: the demonstration runs build its two assets, the frozen portfolio and a routing table, and at test time the router uses them to commit to one strategy per instance. During the demonstration runs the embedded inner agent explores, and after every generation the recorded outcomes of its attempts are distilled into the routing table (Table 1): for each categorical feature signature, a top-k (k=3) candidate list of strategies with mean cost and support, whose first entry is the winning strategy, plus a global default, the best strategy over all demonstration instances.

At test time the router follows Algorithm 1. The router first executes the fallback strategy and stores its solution (FALLBACK), so a candidate answer exists before any model call, though on hard instances its validity is not guaranteed. The router then looks up the signature of the instance's categorical features (LOOKUP): an exact match returns that entry's winning strategy and an unseen

Algorithm 1 Test-time execution of the strategy router and strategy adapter

Require: Instance x, portfolio $\mathcal{P}$, routing table $\mathcal{T}$, mode $m \in \{\text{ROUTER}, \text{ADAPTER}\}$
$y^\star \leftarrow \text{FALLBACK}(x)$
$q^\star \leftarrow -\infty$
$p_{\text{table}} \leftarrow \text{LOOKUP}(\mathcal{T}, \text{FEATURES}(x))$
$p_{\text{agent}} \leftarrow \text{SELECT}(x, \mathcal{P}, \mathcal{T})$ {one LLM call}
$\mathcal{C} \leftarrow \{p_{\text{agent}}, p_{\text{table}}\}$
if $m = \text{ADAPTER}$ **then**
  $\mathcal{C} \leftarrow \mathcal{C} \cup \text{UNTRIED}(\mathcal{P})$ {overlapped with LLM calls}
end if
for all $p \in \text{UNIQUE}(\mathcal{C})$ **do**
  $y \leftarrow \text{RUN}(p, x)$
  if $\text{VERIFY}(y)$ **then**
    $q \leftarrow \text{SCORE}(y)$
    if $q > q^\star$ **then**
      $(y^\star, q^\star) \leftarrow (y, q)$
    end if
  end if
end for
if $m = \text{ADAPTER}$ **then**
  while budget remains and no failure fingerprint has reached the repetition limit **do**
    $p \leftarrow \text{GENERATEORREPAIR}(x, \textit{failures}, \textit{casebook})$ {one LLM call}
    $y \leftarrow \text{SANDBOXRUN}(p, x)$
    if $\text{VERIFY}(y)$ **then**
      $q \leftarrow \text{SCORE}(y)$
      if $q > q^\star$ **then**
        $(y^\star, q^\star) \leftarrow (y, q)$
      else
        break {valid but no gain}
      end if
    else
      Record the failure fingerprint in the casebook
    end if
  end while
end if
return $y^\star$ {return fallback if no candidate verified}

signature returns the global default, giving the candidate p_{table}. The same table serves the inner agent as reference (SELECT): given the features, the portfolio, and the table's evidence, the agent commits to one strategy and a few parameters in a single LLM call, giving p_{agent}. The runtime executes the agent's choice, keeps the table's recommendation as a backup candidate, and retains whichever valid solution the frozen judges score higher. Portfolio code never changes: the agent decides what to try, while deterministic programs decide every element of the solution.

Strategy Adapter The strategy adapter (Figure 2b) inherits the router's entire selection skeleton and adds the defining capability, *adaptation*: the adapter-only loop of Algorithm 1. After the primary choice runs, the agent writes a new strategy (GENERATEORREPAIR), the frozen verifier checks it, and the agent rewrites on the returned error; while a generation call is in flight, the runtime executes the untried portfolio strategies (UNTRIED). Returned code runs in an isolated subprocess with a memory cap (SANDBOXRUN) and competes under the same keep-best-valid rule as portfolio strategies.

Adaptation is supported by three mechanisms. A casebook remembers failures: during the demonstration runs, every failed generation is recorded as a failure fingerprint, and at test time the accumulated casebook is appended to every generation prompt, so new attempts avoid mistakes already made during development. A circuit breaker cuts losses: the loop's guard in Algorithm 1 stops spending calls when generation keeps failing with the same fingerprint on an instance. Finally,

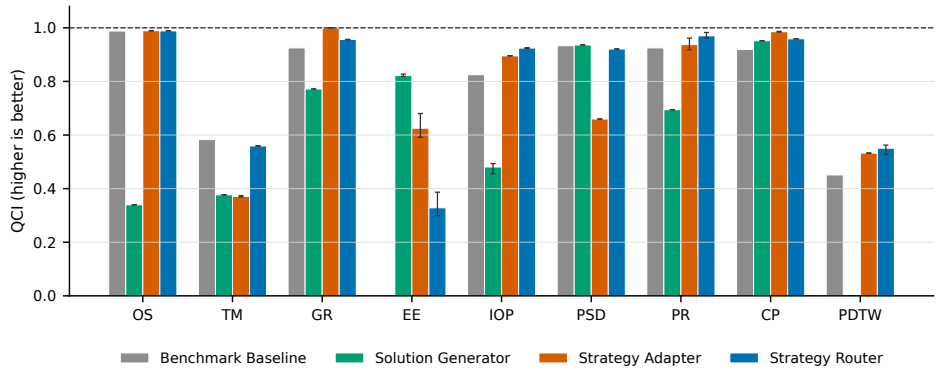


Figure 3: Per-problem QCI for the three organizations and the benchmark baseline: bars show the mean of three complete evaluations, whiskers the min–max range. The strategy router and the strategy adapter approach the expert reference (QCI close to 1) on OS, PR, CP, and GR.

development absorbs what adaptation invents: a generated strategy that won on its demonstration instance is handed back to the outer agent, which may write it into the next generation's frozen portfolio, with a syntax gate and strict elitism ensuring absorption never degrades the portfolio; the strategy thus becomes a permanent portfolio member that future runs can simply select at test time.

4 Evaluation

4.1 Experimental Setup

Benchmark We evaluate all nine HeuriGym problems: Operator Scheduling (OS), Technology Mapping (TM), Global Routing (GR), E-graph Extraction (EE), Intra-operator Parallelism (IOP), Protein Sequence Design (PSD), Pedigree Reconstruction (PR), Crew Pairing (CP), and Pickup-and-Delivery with Time Windows (PDTW).

Implementation and Protocol All three organizations use GPT-5.4 with $G{=}5$ iterations, the same frozen verifiers, evaluators, and instances, and the same evaluation protocol, all executed in Docker containers pinned to eight CPU cores per problem: three complete evaluations per problem, reported per problem as the best of the three (Table 2). The benchmark baseline row in Table 2 reports the official scores released by HeuriGym. The strategy router and the strategy adapter additionally evolve $K{=}3$ candidate solvers per generation. HeuriGym leaves a reward-hacking loophole on Protein Sequence Design: the raw instance file contains the answer, the native residue identities, which a test-time inner agent could simply copy. We therefore add a frozen input mask that hides these fields from the solver in all three organizations, while the frozen verifier and evaluator still read the original files.

Metric Let $x_1, \ldots, x_N$ be the evaluation instances of a problem. A frozen, task-specific verifier checks every constraint of a produced solution, and a frozen evaluator returns its objective cost c_n. Following the benchmark's released evaluation pipeline (Chen et al., 2026), each verified solution is scored against the expert reference cost $c_n^\star$ by the capped ratio $s_n = \min\big(1,\, c_n^\star / c_n\big)$ (task-specific variants of this normalization apply on some problems, e.g., a log-scale ratio for intra-operator parallelism, and the ratio is inverted for maximization), and a problem is summarized by

$$\text{coverage} = \hat{N}/N, \qquad \text{quality} = \Big(\prod\nolimits_{n=1}^{\hat{N}} s_n\Big)^{1/\hat{N}}, \tag{1}$$

$$QCI = \frac{2 \cdot \text{quality} \cdot \text{coverage}}{\text{quality} + \text{coverage}}, \tag{2}$$

where $\hat{N}$ is the number of instances that pass verification. Coverage (called *yield* in the benchmark paper) is the fraction of instances solved validly; quality is the geometric mean of normalized scores

Table 2: Per-problem QCI, best of three evaluations for each problem; bold marks the best HeuriGuide organization. The nine-problem rises monotonically with procedural guidance, from 0.599 to 0.787 to 0.805, against 0.728 for the baseline.

Method	OS	TM	GR	EE	IOP	PSD	PR	CP	PDTW	Overall
Benchmark Baseline	.988	.583	.925	.000	.825	.933	.925	.919	.451	.728
Solution Generator	.340	.377	.772	**.828**	.493	**.936**	.694	.952	.000	.599
Strategy Adapter	**.989**	.374	**1.000**	.680	.896	.660	.962	**.987**	.533	.787
Strategy Router	**.989**	**.560**	.956	.387	**.926**	.921	**.983**	.960	**.562**	**.805**

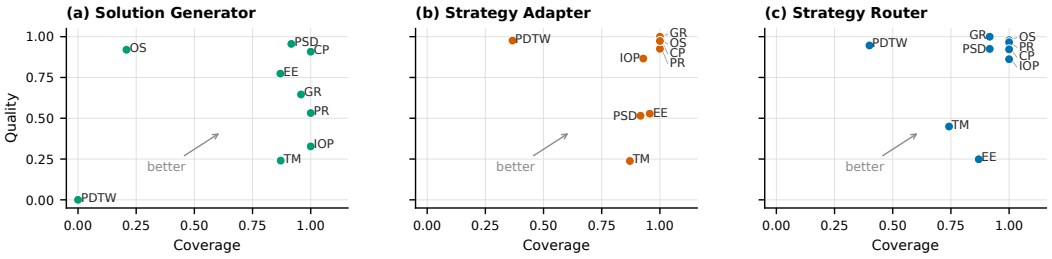


Figure 4: Per-problem quality (score of the valid solutions) and coverage (fraction of instances solved validly) for (a) the solution generator, (b) the strategy adapter, and (c) the strategy router. Points near the left fail to produce valid solutions (PDTW; OS in (a)); points near the bottom produce valid but weak ones (TM).

over those instances; and QCI, their harmonic mean, is the benchmark's per-problem score.[1] QCI punishes both failure axes: invalid solutions and weak solutions alike pull it down.

4.2 Main Results

The strategy router delivers the best overall score. It reaches 0.805 nine-problem QCI (Table 2), improving on the benchmark baseline by 10.6% relative and winning seven of nine problems against the baseline; the strategy adapter follows at 0.787 and the solution generator at 0.599, so the mean rises monotonically with procedural guidance. This ordering is stable: for every organization, the nine-problem mean varies by no more than 0.014 across the three evaluations. Since every score is normalized against the expert reference cost, a QCI close to 1 means matching the expert reference; the strategy router and the strategy adapter reach this level on OS, PR, CP, and GR (Figure 3).

Quality and coverage separate two failure modes (Figure 4): points low on the vertical axis produce valid but weak solutions (TM in all three panels), while points far to the left fail to produce valid solutions at all. The strategy router on PDTW is the clearest case of the latter: its quality is 0.946 but its coverage only 0.400, because a schedule must satisfy pairing, capacity, and time-window constraints at once, and on 18 of its 30 instances no portfolio strategy produces a valid solution; the solution generator on OS (quality 0.92, coverage 0.21) tells the same story.

4.3 Selection Drives Overall Performance

The advantage of the strategy router rests on accurate selection. For each instance we define η as the score of the strategy the router executed divided by the score of the best portfolio strategy for that instance, obtained by replaying every frozen strategy offline with no model calls. Pooled over the

[1] HeuriGym's official evaluation script computes exactly these quantities, namely geometric-mean quality over valid instances and the harmonic combination it names QCI, and it produced the reported reference numbers. The HeuriGym paper instead describes an arithmetic-mean quality termed yield/QYI; we follow the official script throughout.

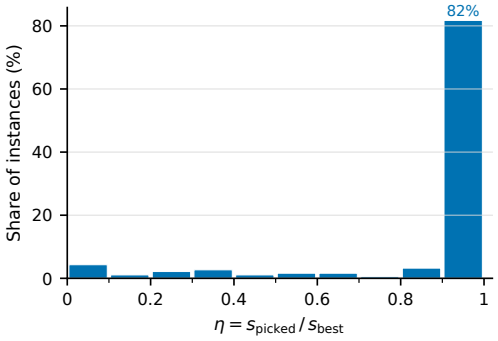


Figure 5: Selection quality of the strategy router: distribution of η, the score of the executed strategy divided by the best portfolio strategy's score on that instance, over the 186 scoreable instances; 82% reach $\eta \geq 0.9$ and 56% attain the per-instance best, so the router behaves close to an oracle over its own portfolio.

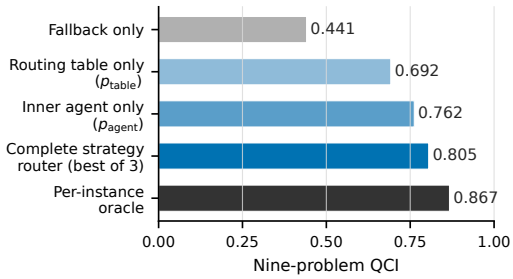


Figure 6: Selection-policy ablation of the strategy router. The fallback alone, the routing table alone (p_{table}), and the inner agent alone (p_{agent}) each fall short of the complete router (best of three, Table 2); the per-instance oracle bounds what any selection policy could achieve.

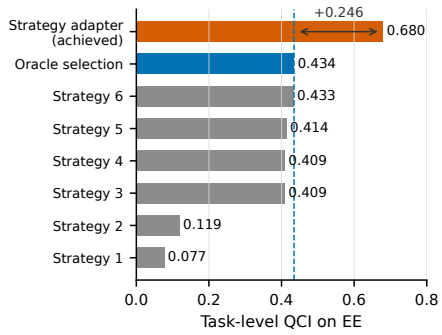


Figure 7: Selection ceiling on EE: task QCI of each frozen strategy, their per-instance oracle (dashed, 0.434), and the strategy adapter (0.680). The six strategies fail on the same instances, so no selection-only policy can exceed a ceiling that the adapter surpasses via generated code.

186 scoreable instances of the nine problems (Figure 5), 82% reach $\eta \geq 0.9$ and 56% attain exactly the per-instance best: at test time the router behaves close to an oracle over its own portfolio.

We ablate the strategy router by executing one component of Algorithm 1 at a time: *fallback only* keeps the initial fallback write and runs no candidate; *routing table only* always executes the lookup result p_{table}; *inner agent only* always executes the agent's choice p_{agent}. As Figure 6 shows, the three reduced policies reach nine-problem means of 0.441, 0.692, and 0.762, all below the complete router's 0.805 (Table 2), while a per-instance oracle over the portfolio attains 0.867. The inner agent is thus the largest single contributor, and the router's score is not inherited from its safety net.

4.4 Code Generation Overcomes Portfolio Gaps

Selection is only one half of the strategy adapter; the other half is generation: the adapter-only loop of Algorithm 1 writes new strategies when the selected ones fall short. On E-graph Extraction this half is decisive.

E-graph Extraction is the suite's hardest problem (the benchmark reports 0.000) and the one problem where the trend inverts, rising monotonically from 0.387 (strategy router) through 0.680 (strategy adapter) to 0.828 (solution generator). The strategy adapter's gain over the router is provably not a selection effect. Figure 7 replays every strategy of the adapter's frozen EE portfolio on every scored

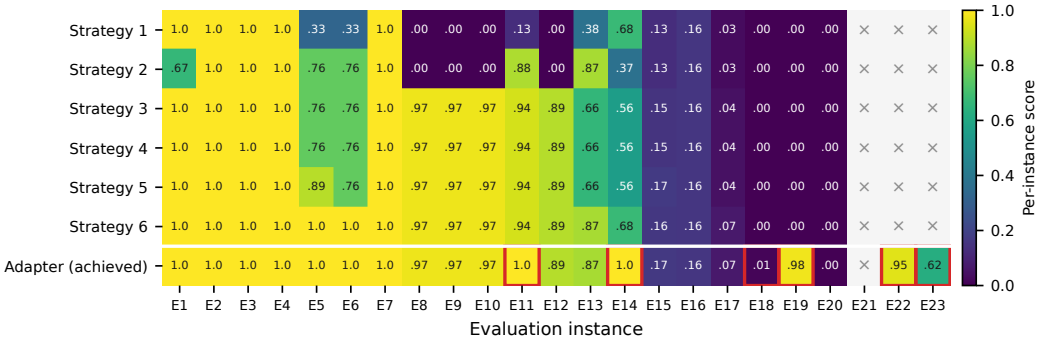


Figure 8: Per-instance EE scores for the six frozen strategies (rows, numbered as in Figure 7) and the strategy adapter (bottom row). Numbered cells are valid solutions (.00: near-zero score); ×: no valid solution; red outline: solution from a generated strategy. Generated strategies improve on the best frozen one on six instances (E11, E14: 1.0; E19: near zero to 0.98), recovering two of the three unsolved instances (E22, E23).

instance under the evaluation protocol, with no model calls: the best fixed strategy attains 0.433 task QCI, and even oracle per-instance selection, which picks the highest-scoring strategy for every instance, attains only 0.434. The two numbers nearly coincide because the six strategies succeed and fail on the same instances, so switching among them offers almost nothing. No routing policy over this portfolio, however accurate, can exceed that ceiling, yet the adapter reaches 0.680.

To see what carries the adapter above the ceiling, we examine EE at the instance level (Figure 8). The red-outlined cells, solutions from generated strategies, improve on the best frozen strategy on six instances (E11 and E14 reach 1.0; E19 rises from near zero to 0.98); most notably, they recover two of the three instances that defeat every frozen strategy (E22 and E23). Those three failures all occur on the large machine-learning compiler graphs (`resnet50`, `vgg`, `nasrnn`) and share a common cause: all six frozen strategies use one extraction routine and fail with the same verifier error, a root e-class left with no selected node. The generated code builds around exactly that constraint, extracting root-first, tracking reachability, detecting cycles, and ending with a self-repair pass that guarantees every reachable e-class receives one node. Where no amount of selection can help, generated code turns unsolvable instances into solved ones.

The solution generator pushes generation furthest: constructing every solution directly, it achieves the best EE score, 0.828. A portfolio that cannot cover this instance diversity leaves selection no room to help, so on this problem generating beats selecting.

5 Conclusion

We introduce HeuriGuide to systematically calibrate the authority of an inner LLM at test time for automated heuristic design. HeuriGuide comprises direct solution generation, strategy adaptation, and strategy selection with increasing procedural guidance. Across all nine HeuriGym problems, the strategy router performs best overall and improves QCI from 0.728 for the benchmark baseline to 0.805, while the strategy adapter also exceeds the baseline at 0.787. Both portfolio-based organizations reach expert-level performance on several problems. Selection among verified procedures is the strongest default when the learned portfolio already covers the relevant test instances: the LLM can specialize its choice to each instance without taking on the cost and uncertainty of constructing a new procedure. However, less authority is not always better. On E-graph Extraction, the frozen strategies share a common failure mode, so even oracle selection cannot overcome the portfolio's coverage gap. In this case, the strategy adapter and solution generator perform better by producing solutions beyond the existing portfolio. Taken together, our results show that test-time authority should be calibrated rather than simply maximized or minimized: select when verified strategies are sufficient, and allow adaptation or direct generation when they are not.

AI USE STATEMENT

We used generative AI tools only to polish the language of this paper; they were not used for any other part of the research. LLMs (GPT-5.4) appear in this work only as the object of study. We have reviewed all AI-assisted edits and take full responsibility for the content of this paper.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026.

Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, September 2003. ISSN 0360-0300. doi: 10.1145/937503.937505. URL https://doi.org/10.1145/937503.937505.

Edmund Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64:1695–1724, 07 2013. doi: 10.1057/jors.2013.71.

Hongzheng Chen, Yingheng Wang, Yaohui Cai, Hins Hu, Jiajie Li, Shirley Huang, Chenhui Deng, Rongjian Liang, Shufeng Kong, Haoxing Ren, Samitha Samaranayake, Carla P. Gomes, and Zhiru Zhang. HeuriGym: An agentic benchmark for LLM-crafted heuristics in combinatorial optimization. In *International Conference on Learning Representations*, 2026.

Pham Vu Tuan Dat, Long Doan, and Huynh Thi Thanh Binh. Hsevo: Elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using llms. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(25):26931–26938, April 2025. ISSN 2159-5399. doi: 10.1609/aaai.v39i25.34898. URL http://dx.doi.org/10.1609/aaai.v39i25.34898.

Hal Daumé, III, John Langford, and Daniel Marcu. Search-based structured prediction, 2009. URL https://arxiv.org/abs/0907.0786.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines. In *International Conference on Learning Representations (ICLR)*, 2024.

Nguyen Viet Tuan Kiet, Dao Van Tung, Tran Cong Dao, and Huynh Thi Thanh Binh. MOTIF: Multi-strategy optimization via turn-based interactive framework. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

Yoonho Lee, Rahul Nair, Qi Zhang, Kyle Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses, 2026.

Falk Lieder, Dillon Plunkett, Jessica B. Hamrick, Stuart J. Russell, Nicholas J. Hay, and Thomas L. Griffiths. Algorithm selection by rational metareasoning as a model of human strategy selection. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pp. 2870–2878, Cambridge, MA, USA, 2014. MIT Press.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model, 2024. URL https://arxiv.org/abs/2401.02051.

Fei Liu, Yilu Liu, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. Eoh-s: Evolution of heuristic set using llms for automated heuristic design, 2025. URL https://arxiv.org/abs/2508.03082.

Fei Liu, Alessio Figalli, Patrick Owen, and Nicola Serra. Raise: LLM-based automated heuristic design with robust adversary instance search, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Alexander Novikov et al. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025.

John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

Stuart Russell. Principles of metareasoning. *Artif. Intell.*, 49(1–3):361–395, May 1991. ISSN 0004-3702. doi: 10.1016/0004-3702(91)90015-C. URL https://doi.org/10.1016/0004-3702(91)90015-C.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=uXl3bZLkr3c.

Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

Xianliang Yang, Ling Zhang, Haolong Qian, Lei Song, and Jiang Bian. Heuragenix: Leveraging LLMs for solving complex combinatorial optimization challenges, 2025.

Shunyu Yao, Fei Liu, Xi Lin, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. Multi-objective evolution of heuristic using large language model, 2025. URL https://arxiv.org/abs/2409.16867.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large language models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems*, 2024.

Mingyang Yu, Ruixin Yang, and Chuchu Fan. Heuristic search as language-guided program optimization, 2026.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639:609–616, 2025.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. ISBN 978-1-57735-887-9. doi: 10.1609/aaai.v38i17.29936. URL https://doi.org/10.1609/aaai.v38i17.29936.