Mirage Discoveries: Separating Signal From Selection Noise in LLM-Driven Program Search

Anonymous authors
Paper under double-blind review

Abstract

Autonomous discovery systems increasingly pair language models with evaluators to generate, test, and select new programs end to end. When the same evaluation regime both guides search and decides acceptance, the search optimizes against its own evaluator: as more candidates are tried, those selected increasingly exploit noise in the evaluation slice rather than transfer to unseen data. We measure this winner's curse in LLM-guided evolution of learning-to-rank pipelines over noisy logged user behavior. Fitness-based acceptance identifies the held-out best candidate in 5 of 12 runs, close to the roughly 4 expected by chance, and equal-budget random search transfers as well as guided search. We introduce audited acceptance, which separates search from deployment: candidates are optimized on a cheap fitness fold but enter the deployable set only when their improvement persists on an audit fold, disjoint from both the fitness fold and the final test fold, by a margin calibrated to the paired-bootstrap noise floor of the search evaluator, with all audit usage counted and reported. The same criterion yields a pre-search headroom gate that determines whether a search space contains enough measurable signal to justify autonomous optimization at all. In a search space that passes the gate, all three discovered executable pipelines outperform an Optuna-tuned LambdaMART baseline by up to 0.0057 NDCG@10 on roughly 60k untouched test queries, and the discovered programs are readable code that recovers the structure of the original challenge winners. As autonomous experimentation scales, our results suggest that where a search space sits relative to the evaluator's noise floor, and whether acceptance is audited against held-out data, together decide whether these systems find real improvements or only appear to.

1 Introduction

Autonomous agents that run their own machine learning experiments end to end are now an established industrial pattern. One recent system closes the loop from hypothesis generation through experiment execution and accepts results by statistical significance against a rolling baseline measured on the same experimental stream that drives the search (Gao et al., 2026). The other runs many asynchronous experiments against a recommendation system, accumulating cross-experiment learnings in a shared memory that feeds subsequent hypotheses (Li et al., 2026a). As described, neither documents evaluating a candidate's acceptance on a data regime disjoint from the one that drove its selection; we do not claim neither system has one. Even so, this is not an unusual design: it is the default architecture for autonomous discovery loops in the FunSearch and AlphaEvolve lineage (Romera-Paredes et al., 2024; Novikov et al., 2025), where a language model proposes program edits, an evaluator scores them, and the highest-scoring candidates are both selected as parents for further search and, eventually, accepted as the loop's output.

The architecture has a weakness wherever the evaluator's score is noisy relative to the effect sizes it is asked to detect, the common case for behavioral data (clicks, purchases, revenue) in ranking and recommendation. An evaluator that guides the search and decides what the search keeps is being asked to judge its own homework. As the number of candidates tried grows, the ones that survive selection are disproportionately those whose apparent gains are noise the evaluator scored well, a form of the winner's curse familiar from multiple-comparisons statistics but rarely measured inside an autonomous loop's own reporting.

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.

We measure the failure mode directly and propose a fix. Using LLM-guided evolutionary search over executable learning-to-rank pipelines as a testbed, where every candidate's held-out performance can be checked against tens of thousands of disjoint queries, we show that (1) the score a loop selects on is a poor predictor of held-out quality once the search space's achievable effects sit near the evaluator's own noise floor, (2) naive, fitness-gated acceptance ships candidates that are frequently no better than chance, and (3) auditing every acceptance decision on a regime the search never touched, at a margin calibrated to the evaluator's own resampling noise, recovers reliable acceptance without a better search algorithm. The same calibration, applied before any search spend, gives a cheap pre-flight test for whether a search space is worth searching at all.

Our contributions: (1) a measured account of selection-fold overfitting in an LLM-guided discovery loop, reporting the rank agreement between selection-fold and held-out scores across dozens of independently evaluated candidates rather than a single before/after comparison; (2) audited acceptance, a rule that decouples what the search optimizes from what it may ship, together with a pre-search headroom gate built on the same statistic; (3) a controlled test, in a search space whose achievable effects clear the evaluator's noise floor, of audited acceptance against naive acceptance, with audit usage reported rather than assumed free; and (4) an open, audited system that improves relevance and conversion over a tuned baseline with no detectable revenue cost, producing discovered programs that are readable code.

2 Related Work

Autonomous ML experimentation. Recent systems close the loop from hypothesis to execution to result analysis, with human-in-the-loop checkpoints and rolling-baseline significance gating (Gao et al., 2026), or run large numbers of asynchronous experiments against production recommendation systems with a shared cross-experiment memory (Li et al., 2026a); an earlier industrial ranking agent followed the same pattern (Kumar et al., 2026). A closed-loop harness that continually reallocates a production recommender's resources goes further and removes the intermediate fitness signal entirely: each proposed configuration is deployed directly and judged by the live A/B experiment that also serves it (Azhar et al., 2026). As described, all three accept results from the same stream, live or offline, that generated the candidates, with no disjoint regime held back beforehand. We add the measurement none of them reports: what fraction of a loop's selected improvements survive a data regime it never touched, and what changes once acceptance is gated on that regime. Agentic AutoML systems that iterate over ML pipelines (Jiang et al., 2025; Liu et al., 2026b; Hollmann et al., 2023) and reflective prompt optimizers (Agrawal et al., 2026) are typically evaluated by single-objective validation accuracy on the fold that drove the iteration.

LLM-guided program evolution. FunSearch (Romera-Paredes et al., 2024) and AlphaEvolve (Novikov et al., 2025) pair an LLM mutation operator with evolutionary search over programs and automated evaluators; open engines reproduce the pattern (Cemri et al., 2026; Liu et al., 2026a). Reported results are almost always evaluated on the data used to select candidates. We run the same class of loop for supervised, multi-objective learning-to-rank and audit every acceptance decision against held-out data.

Leaderboards and model-selection overfitting. Repeatedly selecting on a validation signal biases the selected model's estimated performance (Cawley & Talbot, 2010). The Ladder algorithm (Blum & Hardt, 2015) keeps a competition leaderboard reliable under adaptive submissions by updating the incumbent only when a submission's improvement clears a margin; its simplest analyzed variant uses a fixed step, and its treatment of practical deployment also weighs score uncertainty. Sequential peeking at an A/B test inflates false positives unless the stopping rule is accounted for (Johari et al., 2017). Audited acceptance applies the same update-only-past-a-margin discipline inside an autonomous loop rather than a public leaderboard: the margin here is a paired-bootstrap standard error of the search's own audit fold at a fixed multiplier ($z_{0.95} = 1.645$), the adversary is the loop's own search rather than external submitters, and the number of audit queries spent is counted and reported. We do not claim the calibrated-margin idea itself is unprecedented; the contribution is applying and measuring this discipline end to end inside an autonomous search loop's real acceptance decisions.

Reusable holdouts and adaptive data analysis. An evaluation set queried repeatedly by an adaptive process loses its validity as an estimate of generalization; the reusable holdout (Dwork et al., 2015) shows that a query-budgeted, noise-added interface preserves holdout validity under repeated,

adaptive queries, and a re-collected ImageNet-style test set showed accuracy drops whose cause is debated between adaptive overfitting to the reused test set and genuine differences in the newly collected images (Recht et al., 2019). Audited acceptance applies the same discipline inside a single loop's own acceptance decision: every audit query is counted, and the threshold for accepting an improvement is calibrated to the evaluator's own resampling noise rather than to a significance level chosen independently of the data. The headline comparison we report comes from a test fold that is never used for selection and is kept separate from the audit fold used during search.

Confidence-gated cascades. A parallel line of work in LLM-as-a-judge evaluation asks when a cheap, decision-only judge can be trusted and when a costlier evaluator is warranted. JEV-as-a-Judge (Li et al., 2026b) exposes a decision and a confidence statistic derived from its output label probabilities (accuracy tracks this statistic even where the statistic itself is not well calibrated); a frozen cascade that accepts confident verdicts and escalates uncertain ones retains 99% of a much stronger judge's accuracy at a fraction of its cost. FrugalGPT (Chen et al., 2023) cascades generative models on a similar principle. Audited acceptance is the same idea aimed at a different resource: not a choice between judges of differing capability, but between evaluation regimes of differing statistical power. The fitness fold is the cheap first pass; the disjoint audit fold is the escalation, spent only when the fitness fold reports an apparent gain. Section 6 returns to this.

Automated loss and pipeline discovery for ranking. LambdaMART (Burges, 2010) remains a standard for feature-based ranking, where gradient-boosted trees still match or beat neural rankers (Qin et al., 2021). Automated loss search has mostly targeted vision (Gonzalez & Miikkulainen, 2020; Li et al., 2019; 2022; Liu et al., 2021), with LambdaLoss (Wang et al., 2018) the hand-derived counterpart for ranking metrics; unsupervised evolutionary search has been applied to lexical retrieval with a scalar fitness (Nian et al., 2026). We search executable ranking pipelines, features, model family, loss, and ensemble, under three competing behavioral objectives, with every result subject to the audit in Section 3.

3 Method

3.1 Setting and notation

An autonomous discovery loop maintains a population of candidate programs. At each iteration, a generative model proposes a modification to a parent program, and an evaluator scores the candidate on a *fitness fold* D_{fit}, cheap enough to query on every iteration. The score $f(\theta)$ both determines which candidates seed further search and, in naive designs, which candidates the loop reports or deploys. When D_{fit} is small relative to the variance of the metric it computes, $f(\theta)$ is a noisy estimate of a candidate's true quality, and a process that tries many candidates and keeps the best-scoring ones is, by construction, biased toward candidates whose $f(\theta)$ overestimates them: the winner's curse.

3.2 Measuring the winner's curse

We quantify this rather than assume it. For every candidate a loop evaluates, we also score it on a held-out regime D_{held} the search never uses for selection. If selection were unbiased with respect to D_{held}, a candidate's fitness-fold rank would predict its held-out rank; if selection amplifies noise, the two ranks decorrelate as the achievable true effect shrinks relative to the fitness fold's own measurement noise. We report two quantities across all evaluated candidates rather than a single before/after estimate: the rank agreement (Spearman correlation) between fitness-fold and held-out score, and the fraction of independent search runs in which the best fitness-fold candidate is also the best held-out candidate.

3.3 Audited acceptance

We decouple search from deployment with an explicit second decision. Search proceeds as before, using $f(\theta)$ on D_{fit} to guide generation. Separately, an incumbent accepted candidate θ^* is replaced by a new candidate θ only when its improvement over θ^* persists on an *audit fold* D_{aud}, disjoint from both D_{fit} and a further-held-out *test fold* D_{test} reserved for final reporting, by a margin calibrated to the evaluator's own paired-bootstrap noise floor. Writing $g(\theta) = \text{metric}(\theta, D_{\text{aud}})$ per query group and $\hat{\sigma}$ the bootstrap standard error of $g(\theta) - g(\theta^*)$ over paired resamples of query groups, θ replaces

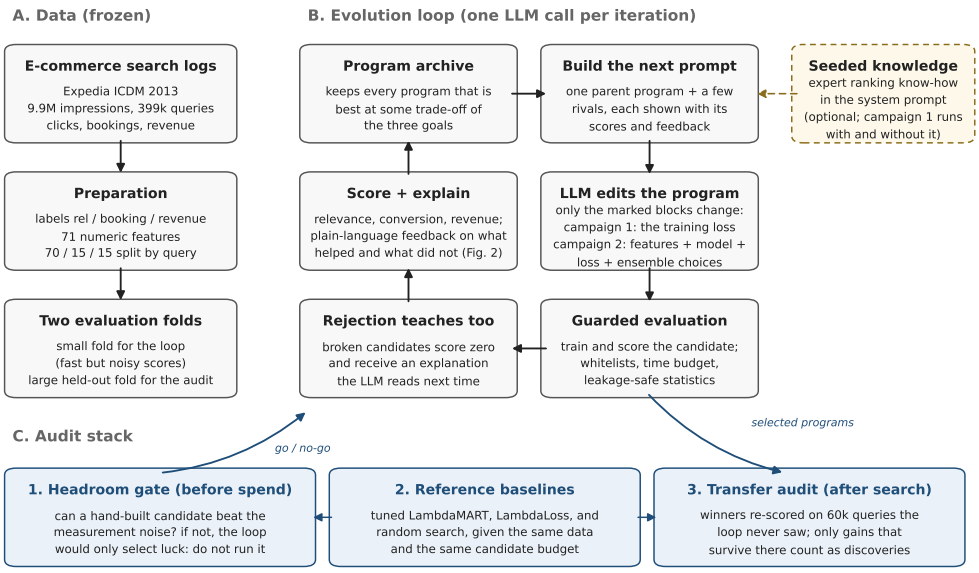

Figure 1: The autonomous discovery loop and its audit stack (the acceptance and stopping criteria applied to every run). Lane A: frozen data preparation, producing the small fitness fold and the large audit and test folds. Lane B: the loop itself, one language-model call per iteration, from archive through prompt, edit, guarded evaluation, and feedback back to the archive. Lane C: the audit stack, applied identically to every run, comprising the pre-search headroom gate, reference baselines evaluated on equal terms, and the transfer audit that decides what counts as a discovery.

θ^* only if

$$g(\theta) - g(\theta^*) - z_{0.95}\,\hat{\sigma} \;>\; 0.$$

The paired bootstrap resamples queries, treating the query as the unit of noise: 10,000 resamples for every reported comparison, 2,000 inside the acceptance replay of Section 5.2, and a few hundred for the noise-gated feedback shown to the language model during search. The $z_{0.95} = 1.645$ threshold is a nominal one-sided criterion under the bootstrap's normal approximation, not a procedure-wide error guarantee: it is applied at every candidate that already looks promising within a run, and we do not correct across those repeated checks. An audit query is spent only when a candidate already looks like an improvement on the fitness fold, so audit consumption tracks how often the search believes it has found something, not the total number of candidates tried. We report this count directly (Section 5.2), as a measure of audit usage, not as a statistically independent validation count.

The same calibration, applied before any search iteration, gives a *headroom gate*: a single, hand-built reference candidate is scored against the current best on D_{fit}, and the search runs only if that candidate's improvement already clears the fitness fold's bootstrap noise floor by a fixed factor, $2.5\times$ in Section 5.3, chosen empirically rather than derived from a formal criterion. The gate is a sufficient-condition check on one candidate, not a proof about the space; our evidence validates only its positive prediction, a space that cleared it did pay off, and we have not tested a space that fails the gate to confirm the negative one. A space in which a literature-informed candidate cannot clear the bar is plausibly, but not provenly, unlikely to yield a result that survives audited acceptance, so the gate is a cheap first check before committing generation budget, not a substitute for running the search.

4 Experimental Setup

The discovery loop, dataset, both search spaces, and the pre-search headroom gate were first described in a non-archival workshop paper (Anonymous, 2026), which reported the pipeline discoveries of Table 1 and Appendix A after a post-hoc transfer audit of each completed search's final selection. That report did not measure the systematic rank agreement between selection-fold and

held-out scores across all selected programs (Section 5.1), did not test audited against naive acceptance inside the loop (Section 5.2), and did not separate an audit fold from the test fold or calibrate an acceptance margin to the search evaluator's own noise (Section 3). Those additions, and the winner's-curse measurement they enable, are this paper's contribution; the underlying system, data splits, and the pipeline-space transfer numbers are shared with the earlier report.

Task and data. We use the public training file of the Expedia Personalized Hotel Search challenge (ICDM 2013; Liu et al., 2013): 9.9M impressions, 399,540 queries, split 70/15/15 by query into train, validation, and test, so no query spans folds. Each shown hotel gets a graded relevance label (5 if booked, 1 if clicked, 0 otherwise). Three objectives are computed from behavioral labels: relevance (graded NDCG@10), conversion (binary booking NDCG@10), and revenue (pooled, dollar-weighted capture of booked revenue in the top 10, normalized against the ideal ordering). During search, each candidate trains on a fixed 8,000-query subsample of the training split, so an evaluation takes 5 to 40 seconds, and is scored on the fitness fold D_{fit}, a fixed 6,000-query subsample of the validation split. The audit fold D_{aud} is the remaining 53,902 validation queries, disjoint from D_{fit} by construction. The test fold D_{test} is the full 59,902-query held-out test split, disjoint from both; it is never used for selection, and the headline comparison of Section 5.3 touches it once. Where we also use it as the measurement regime D_{held} of Section 3.2, each program is scored on it once, post hoc, and no decision depends on the result. "Full scale" below means retraining a fixed program on the entire 280k-query training split.

Two search spaces. We evolve two different objects with the same loop, the same audit machinery, and the same data; the search space is the manipulated factor. In the first (objective space), the loop mutates only the LambdaMART gradient function computed per query, with training configuration, features, and evaluation metrics frozen. In the second (pipeline space), it mutates feature-construction code, behind leakage-guarded, out-of-fold helpers, and a pipeline specification (one to three members from XGBoost, LightGBM, or scikit-learn (Chen & Guestrin, 2016; Ke et al., 2017) with ranking, pointwise, or binary losses, combined by a per-query blending rule). Both use the same generative model (Claude Sonnet 5, one call per iteration) and the same iteration budget, and both are scored after the fact with the same audit machinery from Section 3; the searches themselves ran under naive, fitness-based selection, which is what lets Section 5.2 replay them under both rules. The prompt can carry a curated page of learning-to-rank priors ("seeded domain knowledge"): in the objective-space runs we ablate it (Section 5.1); in the pipeline-space runs it is always present.

Baselines. An Optuna-tuned LambdaMART (Akiba et al., 2019; Burges, 2010) (40 trials, retuned at every data scale reported, since optimal settings change with data volume), untuned default LambdaMART, a hand-derived LambdaLoss objective (Wang et al., 2018), and an equal-generation-budget random search over the same space as the evolutionary loop.

5 Results

5.1 Selection-fold score does not predict held-out quality when effects are small

In the objective search space, evolved candidates beat the default baseline on their own fitness fold and improve steadily. On held-out queries the picture reverses: final selections gain $+0.0004$ to $+0.0014$ NDCG@10, the best evolved objective beats its own starting program by $+0.0002$ (two-sided $p = 0.76$), equal-budget random search transfers as well or better ($p = 0.03$ in its favor), and every method sits significantly below the tuned baseline ($p < 0.001$). The mechanism is measurable. Component-removal analysis puts a single mechanism's estimated effect at 0.0005 to 0.008 NDCG@10, most below 0.005, against a fitness-fold measurement noise (95% bootstrap half-width) of ± 0.007 to 0.009 NDCG@10, so selecting the best score selects luck. We then collect every program that any of the twelve objective-space runs kept as its best pick at some point during search (41 distinct programs in total, 2 to 5 per run), and compare each program's fitness-fold NDCG@10 to its NDCG@10 on the 53,902-query audit fold. We ask two questions. First, do the two scores rank programs the same way? This depends on how much validation data the run searched on: with only 1,600 queries, the two rankings show no significant association (Spearman 0.31, $p = 0.23$, 17 programs); with 6,000 queries they show a significant positive association (Spearman 0.53, $p = 0.007$, 24 programs). Second, is each run's actual final pick, the program with the best fitness-fold score, also the best program on the audit fold? Only in 5 of the 12 runs (6 of 12 on the test fold), close to the roughly 4 of 12 we would expect from picking one of 2 to 5 candidates at random; this comparison is

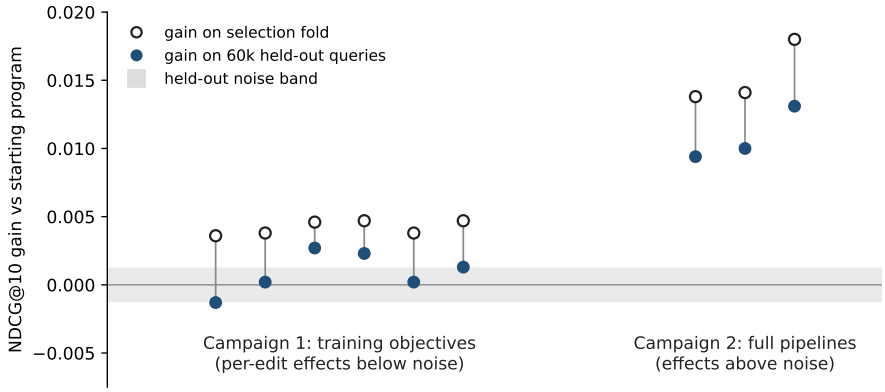

Figure 2: Per-run NDCG@10 gain over the starting program, on the selection fold (open) versus held-out queries (filled). Gains in the objective search space (left) collapse into the noise band under audit; gains in the pipeline search space (right) survive it.

too small to establish equivalence to chance, but it gives no evidence that the naive pick beats it. So even at the larger fold, where the overall ranking shows a real association, whether the naive rule's one actual pick is the best available program is itself close to a coin flip. Figure 2 contrasts per-run gains on the selection fold against gains on held-out data across both search spaces: objective-space gains collapse into the noise band under audit, while pipeline-space gains survive it.

Two further observations from this campaign set up what follows. First, fold size mattered more than anything else we tried: growing the fitness fold's validation side from 1,600 to 6,000 queries cut its noise in half and roughly tripled how much of the apparent gain survived transfer. That was the only change in this campaign that helped at all, so Section 5.2 asks what an audit adds on top of that larger fold, not whether a bigger fold could replace it. Second, extra guidance changed how fast the search went, not what it found: giving the prompt curated domain knowledge let two of three runs match, within 1 to 3 iterations, what an equal-budget random search needed its entire run to reach (unseeded runs took 7 to 23 iterations), and a richer diagnostic feedback channel changed nothing measurable. Neither change affected what transferred to held-out data. Analysis was worth more than search here: removing the best objective's components one at a time (five evaluations) showed that a per-query adaptive blend weight was actively harmful, and fixing it to a constant improved all three metrics.

5.2 Audited acceptance versus naive acceptance

We test this directly by replaying each pipeline-space run candidate by candidate under two acceptance rules: naive, which replaces the incumbent whenever fitness-fold score improves, and audited, which replaces it only when the improvement clears the audit-fold noise floor (Section 3), spending $N_{\text{boot}} = 2{,}000$ resamples and a one-sided $z_{0.95} = 1.645$ margin per check. Across all three runs, audited acceptance never ships a program worse than naive acceptance on the untouched test fold, and beats it in two of three: a tie on run s0 (both ship the same candidate, test NDCG@10 0.4316, 8 audit queries spent); a win on run s1 ($+0.0016$, shipping at iteration 23 rather than naive's iteration 31, 29 audit queries spent, 2 accepted); and a win on run s2 ($+0.0010$, shipping at iteration 25 rather than iteration 40, 32 audit queries spent, 2 accepted). Pooled across the three runs, audited acceptance improves untouched-test NDCG@10 by $+0.0009$ on average, at a median cost of 29 audit queries per run, never the full 40-candidate population. Unlike Campaign 1, the gap between fitness-fold and audit-fold gain does not widen over the course of these runs (Appendix B); this is the expected signature of a search space that clears the headroom gate, where a candidate's fitness-fold score is already a reasonably faithful guide, rather than evidence against the mechanism. What audited acceptance changes here is not whether the loop eventually finds a good program, but which one it ships and when: by requiring held-out confirmation rather than chasing further fitness-fold gains, it repeatedly stops earlier, at a candidate whose advantage is already established, rather than continuing to a later one whose apparent edge is fitness-fold noise.

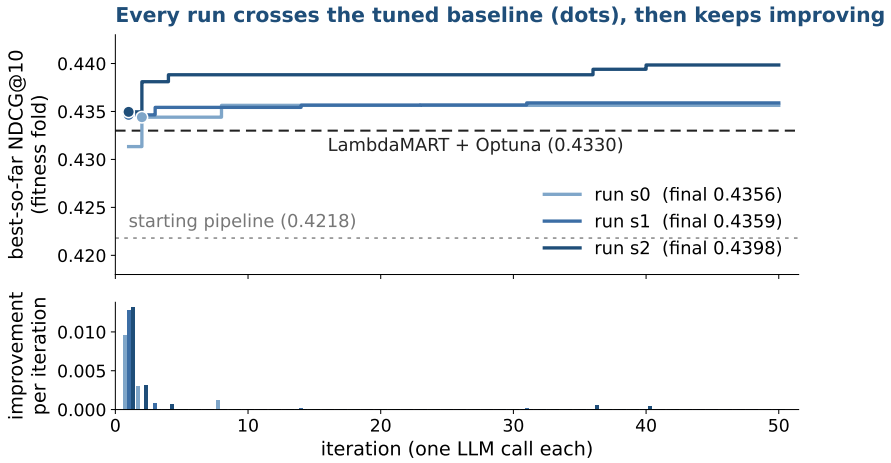

Figure 3: Best-so-far fitness-fold NDCG@10 per iteration (top) and each iteration's increment (bottom), for the three independent pipeline-space runs. All three cross the tuned baseline within a handful of iterations and continue to find increments through iteration 50; Table 1 reports what survives on held-out data.

Table 1: Pipeline-space transfer: for each run, the naive rule's final selection (highest fitness-fold score) is trained under the search's own budget and scored on the 6,000-query fitness fold (Val) and the 59,902-query test fold; Section 5.2 compares this naive selection to the audited alternative. All three runs beat the tuned baseline on test NDCG@10 and booking NDCG@10; revenue is within noise. Bold: best per column.

Method	Val NDCG@10	Test NDCG@10	Test booking	Test revenue
Starting pipeline	0.4218	0.4222	0.4443	0.4105
LambdaMART + Optuna	0.4330	0.4296	0.4527	0.4231
Pipeline run s0	0.4356	0.4316	0.4545	**0.4257**
Pipeline run s1	0.4359	0.4322	0.4553	0.4215
Pipeline run s2	0.4398	**0.4352**	**0.4582**	0.4239

5.3 Discovering pipelines that survive the audit

The headroom gate applied before this campaign required a hand-built reference candidate from the challenge literature, within-query rank features and count encodings, to beat the fitness fold's noise floor by 2.5×. It did, by $+0.013$ NDCG@10. Building that candidate also surfaced three successively subtler label-leakage bugs in target encoding (costing -0.10 and -0.13 NDCG@10 before the out-of-fold, query-split construction fixed them), each caught by the audit before any search depended on it. All three independent pipeline-space runs (50 iterations, 40 programs retained in the archive per run; roughly ten dollars of language-model spend each) then discover pipelines that beat the tuned LambdaMART baseline on the 59,902-query test fold (Table 1). Roughly 70% of each run's selection-fold gain survives on held-out queries. The best run's improvement over the tuned baseline is significant on relevance and booking, and positive but within noise on revenue; the same pattern holds at full data scale (Table 2, Appendix A).

All three runs independently converge on the same structure: within-query percentile ranks of price, star rating, review and location scores; count encodings of property and destination identifiers; and a diverse three-model ensemble combined by per-query z-score weighting, the structure the original 2013 challenge winners engineered by hand (Liu et al., 2013) (Appendix C shows the code). One caveat on provenance: rank and count features were present in the loop's seeded domain knowledge, so their presence is partly prompted; the specific constructions, model choices, and several unprompted inventions (per-query price normalization, quantile-filling a mostly-missing location feature) are the search's own. On the original competition's hidden Kaggle test set, scored by one

Table 2: Significance of the best run's improvement over the tuned baseline, by paired query bootstrap (10,000 resamples, one-sided). n.s.: not significant at $p = 0.05$.

Metric	Δ	95% CI	p
NDCG@10	$+0.0057$	$[0.0041, 0.0073]$	$\leq 10^{-4}$
Booking NDCG@10	$+0.0056$	n/a	$\leq 10^{-4}$
Revenue	$+0.0008$	$[-0.0034, 0.0047]$	n.s.
NDCG@10, full scale	$+0.0075$	$[0.0060, 0.0091]$	$\leq 10^{-4}$

late submission each with no leaderboard iteration, the discovered pipeline reaches 0.5196 private NDCG@38 (20th of 340 teams, top 6%), against 0.5140 (30th) for the tuned baseline. The winning hand-built ensembles remain ahead (0.5398), and our submission benefits from hindsight, since the winners' published lessons are part of the seeded knowledge.

6 Discussion

The two search spaces share the loop, the audit machinery, and the data, and the same loop under the same audit produced a mirage in one and a durable discovery in the other. Whether an autonomous loop's output is trustworthy is not a property of the loop or the language model driving it; it is a property of the relationship between the search space's achievable effect sizes and the evaluator's own measurement noise. That relationship is cheap to check before committing any search budget, and acceptance auditing keeps it honest for the rest of the run, not only at the end.

Limitations. Our evidence comes from one dataset, chosen because public learning-to-rank data with both conversion and revenue labels is scarce (Kabir et al., 2024). We have not tested a second dataset, and make no claim beyond ranking and recommendation. That said, the audit machinery itself needs only a fitness fold, a larger disjoint audit fold, a paired bootstrap, and a reporting fold untouched by both, so it does not depend on anything specific to this dataset, and the qualitative finding, that acceptance criteria bind before generation quality does, may well hold in other settings that evaluate on noisy behavioral signals.

Our formalism assumes fixed folds and does not address a moving target. Production loops that continually reallocate resources under a changing operating point face a related but distinct problem: tracking a shifting optimum, rather than avoiding overfitting to a fixed noisy signal. Two recent systems report exactly this tension: one sees an offline win erased by a concurrent baseline refresh (Gao et al., 2026), and another explicitly assumes the best configuration drifts from cycle to cycle (Azhar et al., 2026).

A few further scope notes. Two search spaces is a coarse manipulation; intermediate spaces (features plus loss, model plus loss) would map the headroom boundary more finely. Three runs per condition are enough for the paired-bootstrap headline, which every run cleared, but not enough to characterize how variable the search itself is. Revenue is a proxy metric, and discovered pipelines are audited for statistical transfer, not online effects. We compare against equal-budget random search and a tuned single-model baseline; a hand-tuned ensemble and an off-the-shelf AutoML baseline on the same features would further isolate what the loop's guidance contributes beyond a wider search space, and are left for future work. We did not run a controlled comparison against other LLM-guided discovery engines; the audit itself is engine-agnostic, and two engine swaps at double budget (a meta-evolution variant (Liu et al., 2026a) and a recalibrated AdaEvolve) landed inside the spread of the original three pipeline-space runs.

Audit cost. Every audit here spends a full paired-bootstrap query against the audit fold whenever a candidate looks like a fitness-fold improvement (Section 3). That is conservative: it treats every promising candidate alike regardless of how large its fitness-fold margin is. The parallel to confidence-gated cascading noted in Section 2 suggests cheaper policies may be possible; we leave designing and testing one to future work.

7 Conclusion

As autonomous systems take on more of the research loop, the evaluator deciding what to keep becomes as consequential as the generator proposing candidates. We show that when acceptance is decided by the same signal that guides search, a loop can appear to make steady progress while shipping candidates that do not clearly beat chance. The same diagnosis that explains this failure, comparing a search space's achievable effects to the evaluator's noise, also pointed us to where to search next: a space with effects above that noise floor, audited throughout with its own query budget reported, is where this loop's acceptance decisions held up. Search-space design and acceptance discipline read here as complementary levers, not substitutes for one another. Code, configs, seeds, discovered programs, and audit scripts will be released on publication.

AI use statement

Generative AI tools (large language models) were used to aid and polish the writing of this manuscript, to assist with retrieval and discovery of related work, and to assist with research ideation and execution, including implementing experiment and analysis code under author direction and review. Generative AI tools were not used to generate synthetic data (all reported results use the public Expedia dataset), to produce or verify formal mathematical proofs, or for translation. A large language model is additionally the object of study and a component of the method described in Section 3, serving as the mutation operator inside the autonomous discovery loop being evaluated; this methodological role is distinct from the authoring-assistance uses disclosed above. All AI-assisted code and text were reviewed and verified for correctness by the authors, who take responsibility for the final content of this work.

Reproducibility statement

Code, configuration files, random seeds, per-iteration logs, discovered programs, and the audit scripts used to produce every reported number will be released at an anonymized repository upon publication. The train/validation/test split (Section 4) partitions the public Expedia dataset by query identifier so no query spans folds; the fitness, audit, and test folds used throughout are pairwise disjoint subsets of this split, as described in Section 4 and implemented in the released data preparation script.

References

L. Agrawal, S. Tan, D. Soylu, et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026.

T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2019.

Anonymous. EvoRank: LLM-guided evolution of multi-objective learning-to-rank pipelines. GenAIECommerce Workshop, co-located with ACM RecSys, 2026. Non-archival workshop proceedings; details omitted for anonymity.

Muhammad Rafay Azhar, Yuhang Zhou, Gilbert Jiang, Yuchen Wang, Rahul Sharma, Matthew DeSousa, Jiayi Liu, Xin Guo, Lizhu Zhang, and Xiangjun Fan. CORAL: An LLM-native harness for production recommender systems. *arXiv preprint arXiv:2609.02730*, 2026.

Avrim Blum and Moritz Hardt. The Ladder: A reliable leaderboard for machine learning competitions. In *International Conference on Machine Learning (ICML)*, 2015.

Christopher J. C. Burges. From RankNet to LambdaRank to LambdaMART: An overview. Technical Report MSR-TR-2010-82, Microsoft Research, 2010.

Gavin C. Cawley and Nicola L. C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11:2079–2107, 2010.

M. Cemri, S. Agrawal, A. Gupta, et al. AdaEvolve: Adaptive LLM driven zeroth-order optimization. *arXiv preprint arXiv:2602.20133*, 2026.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016.

Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth. The reusable holdout: Preserving validity in adaptive data analysis. *Science*, 349(6248):636–638, 2015.

Erwin Gao, Vinodh Kumar Sunkara, Jingyi Guan, et al. Agentic ML Exploration (A-MLE) for Ads ranking. *arXiv preprint arXiv:2609.08248*, 2026.

Santiago Gonzalez and Risto Miikkulainen. Improved training speed, accuracy, and data utilization through loss function optimization. In *IEEE Congress on Evolutionary Computation*, 2020.

N. Hollmann, S. Müller, and F. Hutter. Large language models for Automated Data Science: Introducing CAAFE for Context-Aware Automated Feature Engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Z. Jiang, D. Schmidt, D. Srikanth, et al. AIDE: Ai-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.

Ramesh Johari, Pete Koomen, Leonid Pekelis, and David Walsh. Peeking at A/B tests: Why it matters, and what to do about it. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2017.

M. A. Kabir, M. A. Hasan, A. Mandal, D. Tunkelang, and Z. Wu. A survey on E-Commerce Learning to Rank. *arXiv preprint arXiv:2412.03581*, 2024.

G. Ke, Q. Meng, T. Finley, et al. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

A. Kumar, E. Gao, M. Levi, et al. Ranking Engineer Agent (REA): The autonomous ai agent accelerating Meta's Ads ranking innovation. Engineering at Meta blog, `https://engineering.fb.com/2026/03/17/developer-tools/ranking-engineer-agent-rea-autonomous-ai-system-accelerating-meta-ads-ranking-innovation/`, 2026. Accessed 17 August 2026.

C. Li, X. Yuan, C. Lin, et al. AM-LFS: AutoML for loss function search. In *International Conference on Computer Vision (ICCV)*, 2019.

H. Li, T. Fu, J. Dai, et al. AutoLoss-Zero: Searching loss functions from scratch for generic tasks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Ming Li, Dai Li, Xuying Ning, Bo Sun, Rui Li, Yi Zhang, Silvia Gong, Xuan Cao, Cornelia Carapcea, Qunshu Zhang, Zhigang Wang, Yinglong Xia, and Andy Wang. Auto-RecSys: Harnessing autonomous research agents for industry-scale recommender system. *arXiv preprint arXiv:2609.10922*, 2026a.

Yubo Li, Yidi Miao, Ramayya Krishnan, and Rema Padman. JEV-as-a-Judge: Accept when confident, escalate when unsure. *arXiv preprint arXiv:2609.26550*, 2026b.

P. Liu, G. Zhang, B. Wang, et al. Loss function discovery for object detection via convergence-simulation driven search. In *International Conference on Learning Representations (ICLR)*, 2021.

S. Liu, S. Agarwal, M. Maheswaran, et al. EvoX: Meta-evolution for automated discovery. *arXiv preprint arXiv:2602.23413*, 2026a.

X. Liu, B. Xu, Y. Zhang, et al. Combination of diverse ranking models for personalized Expedia hotel searches. *arXiv preprint arXiv:1311.7679*, 2013.

Z. Liu, J. Chai, X. Zhu, et al. ML-Agent: Reinforcing LLM agents for autonomous machine learning engineering. In *International Conference on Machine Learning (ICML)*, 2026b.

J. Nian, F. Li, D. H. Park, and Y. Fang. RankEvolve: Automating the discovery of retrieval algorithms via LLM-driven evolution. *arXiv preprint arXiv:2602.16932*, 2026.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Z. Qin, L. Yan, H. Zhuang, et al. Are Neural Rankers still outperformed by Gradient Boosted Decision Trees? In *International Conference on Learning Representations (ICLR)*, 2021.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do ImageNet classifiers generalize to ImageNet? In *International Conference on Machine Learning*, pp. 5389–5400, 2019.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, et al. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

X. Wang, C. Li, N. Golbandi, M. Bendersky, and M. Najork. The LambdaLoss framework for ranking metric optimization. In *ACM International Conference on Information and Knowledge Management (CIKM)*, 2018.

A Full-scale comparison

Table 3 retrains fixed programs on the entire 280k-query training split and scores them on the 59,902-query test fold, adding NDCG@38, the original challenge metric. The comparison carries its own methodology lesson: against a baseline tuned on the small search fold the discovered pipelines win easily; re-tuning the baseline at full scale reverses that verdict; and only a comparison with comparable capacity on both sides isolates the discovered structure, which then wins on all four numbers. The margin splits across both components: the discovered features alone, under the same tuned single model, reach 0.4587 NDCG@10 ($+0.0043$); the two-member ensemble adds $+0.0032$.

Table 3: Full-scale comparison (280k training queries, 59,902 test queries). The last row gives the pipeline's XGBoost member the baseline's full-scale tuned settings, scales its LightGBM member to match (255 leaves, 300 trees), and drops the third member; it had no search or tuning of its own.

Method	NDCG@10	NDCG@38	Booking@10	Revenue@10
Tuned baseline, small-fold tuning	0.4349	0.4983	0.4588	0.4300
Tuned baseline, full-scale tuning	0.4544	0.5132	0.4805	0.4565
Pipeline s1, small-fold tuning	0.4490	0.5080	0.4738	0.4483
Pipeline s1, baseline's settings, 2 members	**0.4619**	**0.5185**	**0.4883**	**0.4585**

B Fitness-fold versus audit-fold gain over the course of search

For each pipeline-space run, every time the naive rule updates its pick, we compare that pick's gain on the audit fold to its gain on the fitness fold. Across all three runs, the two stay close throughout the search: the audit-fold gain is always between 93% and 128% of the fitness-fold gain, and it does not drift lower as the search runs longer (the second half of each run averages 99%, 105%, and 96% of the fitness-fold gain, no lower than the first half's 103%, 111%, and 101%). Naive selection is not getting more fooled by noise the longer this search runs. That fits the rest of the paper: this search space already cleared the headroom gate, so its real effects are larger than the fitness fold's noise, and the widening gap between what looks good and what holds up, seen in Campaign 1 (Section 5.1), is what happens when effects are smaller than that noise, not a general property of longer searches.

C A discovered program

Figure 4 excerpts the feature block and pipeline specification of the program discovered by pipeline-space run s1. Full programs for all runs are in the released repository.

```
# run s1 (excerpt; "..." and "# 5 cols" are editorial)
for c in ["price_usd", "prop_starrating", ...]:   # 5 cols
    out[f"{c}_qrank"] = df.groupby("qid")[c].rank(pct=True)
for c in ["prop_id", "srch_destination_id"]:
    out[f"{c}_count"] = stats.count(df, c)

PIPELINE = {"models": [
    {"family": "xgb",     "objective": "rank:ndcg",  ...},
    {"family": "lgbm",    "objective": "lambdarank", ...},
    {"family": "sklearn", "objective": "ert",        ...}],
  "ensemble": {"type": "zscore_weighted",
               "weights": [0.5, 0.35, 0.15]}}
```

Figure 4: The discovered program is readable code: feature excerpt and pipeline specification from run s1. Within-query percentile ranks, count encodings through a leakage-guarded helper, and a three-member ensemble blended by per-query z-scores.