EvoMaestro: Toward Interpretable and Steerable LLM-Driven Program Evolution

Feng Liang
feng011@e.ntu.edu.sg
Nanyang Technological University
Singapore, Singapore

Sizhe Cheng
sizhe003@e.ntu.edu.sg
Nanyang Technological University
Singapore, Singapore

Yikai Li
liyi0084@e.ntu.edu.sg
Nanyang Technological University
Singapore, Singapore

Ruijie He
herj25@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Xiaolin Wen
xiaolin004@e.ntu.edu.sg
Nanyang Technological University
Singapore, Singapore

Yong Wang
yong.wang@ntu.edu.sg
Nanyang Technological University
Singapore, Singapore

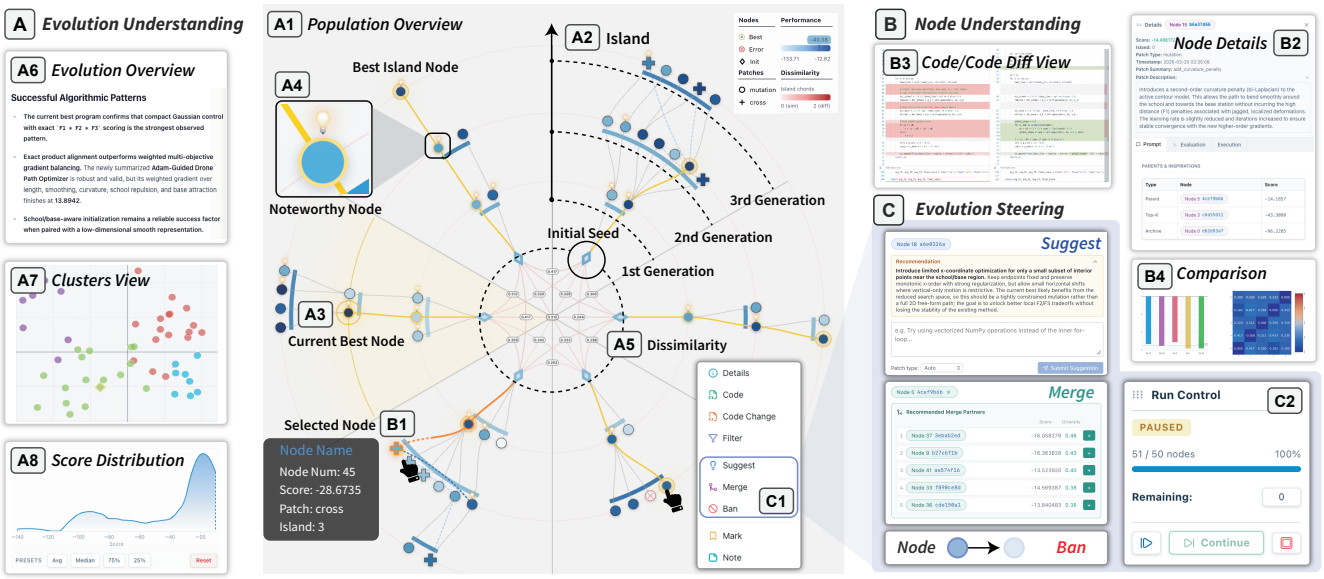

Figure 1: The EvoMaestro interface supports LLM-driven program evolution through three stages: (A) *Evolution Understanding* overviews the population structure, performance, and strategies (A1-A8); (B) *Node Understanding* enables multi-level inspection of individual nodes, including metadata (B1, B2), code explanations (B3), and comparison (B4); and (C) *Evolution Steering* allows interactive interventions such as *Suggest*, *Merge*, *Ban*, and *pacing control* of the evolution process.

Abstract

Recent program evolution systems use large language models (LLMs) to generate and iteratively improve populations of programs, producing striking advances in mathematics, algorithm design, and scientific computing. Yet these systems largely operate as fully automated black boxes. As populations grow, domain experts must make sense of the scores, code changes, reasoning, and algorithmic ideas across many programs, while current interfaces provide limited support for understanding or redirecting the evolution. We characterize this need to understand and steer populations of evolving algorithmic ideas as *semantic oversight*. A formative study with 8 domain experts yields six design requirements for this emerging human-computer interaction problem. We then propose a steerable program evolution framework that lets expert judgments shape subsequent evolution. Built on this framework, EvoMaestro is an interactive visual analytics system that organizes evolution information from population overview to source code. It helps experts locate and compare noteworthy programs, guide evolution with natural language, combine promising ideas, and prune unproductive directions. A seven-day system demonstration illustrates how an expert applied these capabilities throughout a long-running evolution process. A within-subjects study with 12 participants shows that EvoMaestro improves users' understanding of evolution processes, reduces cognitive workload, and supports expert steering. These findings suggest that preserving expert agency in

UIST '26, Detroit, MI, USA

ACM ISBN 979-8-4007-2856-3/2026/11
https://doi.org/10.1145/3830398.3830626

open-ended LLM-driven search requires support for both informed judgment and the ability to shape subsequent automation.

CCS Concepts

• **Human-centered computing → Interactive systems and tools**; *Visualization systems and tools.*

Keywords

LLM-driven program evolution, visual analytics

ACM Reference Format:

Feng Liang, Sizhe Cheng, Yikai Li, Ruijie He, Xiaolin Wen, and Yong Wang. 2026. EvoMaestro: Toward Interpretable and Steerable LLM-Driven Program Evolution. In *The 39th Annual ACM Symposium on User Interface Software and Technology (UIST '26), November 02–05, 2026, Detroit, MI, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3830398.3830626

1 Introduction

Advances in large language models (LLMs) have transformed how programmers write code, from inline completions to multi-file edits guided by natural-language (NL) intent [1, 30]. Recent systems (e.g.,[16, 23, 25]) push this further: instead of assisting programmers, large language models (LLMs) serve as the engine of evolutionary search over programs. They maintain a *population* of candidate programs and iteratively improve them over successive generations: an LLM reads select programs, reasons about their algorithmic strategies, and proposes modifications, while an evaluation function scores the results and feeds them back into the population. This paradigm has produced striking results, including novel mathematical findings [9, 25], improvements on long-standing algorithms [23], and new activation functions in deep learning (DL) [31]. As open-source frameworks [16, 26] lower the entry barrier, LLM-driven program evolution is becoming a powerful tool for researchers across fields.

However, these systems operate as fully automated black boxes, lacking transparency and controllability during execution. Once a run is launched, experts have limited ability to monitor, understand, or steer the evolution process. This poses two key challenges (**C1**, **C2**). First, experts struggle to *understand* the evolution (**C1**): a run produces a rapidly growing tree of program variants, each carrying scores, code diffs, and error logs, and the sheer volume overwhelms human inspectors. Unlike classical evolutionary algorithms (EAs), LLM-driven evolution generates intentional *code edits* accompanied by rich metadata (e.g., reasoning chains and strategy descriptions), but existing interfaces (e.g., ShinkaEvolve UI [16]) fail to effectively surface this information for analysis. Second, experts lack mechanisms to *steer* the evolution (**C2**): current systems do not support pausing, redirecting, combining solutions, or pruning unproductive lineages during execution. Although experts possess domain knowledge that could redirect unproductive search directions and amplify promising ones, there is no effective channel to incorporate such human input into the evolution process.

To address these challenges, we present a novel interactive visual analytics system for interpretable and steerable LLM-driven program evolution. Specifically, we first conducted a formative study with 8 domain experts to derive six concrete design requirements (§ 4). Informed by these requirements, we propose a steerable program evolution framework that extends the standard LLM-driven evolution process by enabling expert interventions (e.g., manually guiding program generation) during evolution to support steering (**C2**) and exposing evolution details (e.g., population structure, performance, and strategies) to support interpretation (**C1**). Built on this framework, the EvoMaestro interface incorporates a novel visual design of population overview and multiple coordinated analytical views to further support both intuitive evolution understanding (**C1**) and smooth expert steering interactions (**C2**). The interface organizes evolution information through progressive disclosure from population-level overviews down to LLM-generated line-level code annotations, enabling experts to build situational awareness without being overwhelmed by raw code and scores. We demonstrate EvoMaestro on a real-world optimization task and evaluate it through an in-depth user study with 12 participants against the state-of-the-art baseline (i.e., ShinkaEvolve Web UI [16]). The results show that EvoMaestro improves users' understanding of the evolution process, reduces cognitive workload, and supports effective expert steering.

In summary, we make the following contributions:

- We characterize *semantic oversight* of LLM-driven program evolution as an emerging human-computer interaction (HCI) problem: experts must understand and steer populations of evolving algorithmic ideas. A formative study with 8 scientists and engineers yields six design requirements.
- We propose a steerable program evolution framework and EvoMaestro[1]. The framework lets expert judgments shape subsequent evolution process; the interface helps experts locate noteworthy nodes and inspect strategies from the population level to source code.
- We demonstrate sustained expert steering on a real-world optimization task and evaluate EvoMaestro through a within-subjects study with 12 participants. The study shows that EvoMaestro significantly improves experts' understanding of the evolving population, enables effective expert-guided steering, and reduces cognitive workload. We distill two lessons for preserving expert agency: support informed judgment and action on that judgment.

2 Related Work

Our work relates to existing research on LLM-Driven Program Evolution, Visualization for Evolutionary Algorithms, and Human Oversight of Iterative Program Generation.

2.1 LLM-Driven Program Evolution

Recent studies use LLMs as mutation operators in evolution processes, generating code variants through prompting rather than syntactic manipulation. FunSearch [25] demonstrated that LLM-driven evolution could discover novel mathematical constructs. AlphaEvolve [23] scaled up this approach for scientific and algorithmic discovery. DeepEvolve [19] extends AlphaEvolve with deep research capabilities. ShinkaEvolve [16] improves the search efficiency. Open-source implementations [16, 18, 26] have broadened

[1] Opensourced on GitHub (https://github.com/ifsheldon/EvoMaestro)

the access. These systems are already producing real scientific impact [23, 25, 31]. Although they generate code, their application domains are diverse: code can represent candidate solutions to problems in mathematics, computational chemistry, physics, biology and other fields. This approach applies when solutions can be (1) expressed as executable programs and (2) evaluated automatically through safe execution using a computable, sufficiently reliable metric. Its potential users therefore include scientists and engineers who can judge domain-specific ideas but may need support to connect those ideas with generated code and evaluation results. Existing program evolution systems primarily operate as automated pipelines and provide limited support for expert steering during a run. EvoMaestro builds directly on these systems, providing an interactive layer that makes the evolution process legible and steerable by domain experts.

2.2 Visualization for Evolutionary Algorithms

For traditional EAs, early work has visualized population dynamics [15, 33], fitness landscapes, multi-objective trade-offs [7, 11, 13, 20, 22, 34, 35] and neuroevolution trajectories [2, 32]. NAS-Navigator [29] and VisEvol [5] provide visual steering with domain knowledge injection and real-time intervention in neural architecture search, representing the closest design precedents to our work. These systems operate on search spaces where solutions are numerical vectors, modular components, or architectural blocks. LLM-driven program evolution differs from traditional EAs in that each mutation is a *reasoned code edit* accompanied by rich metadata (e.g., reasoning chains, LLM-generated summaries) with no analog in classical EAs, and code diffs are structurally complex. ShinkaEvolve [16] and OpenEvolve [26] provide a web UI for visualizing their evolutions, but they are limited to post-hoc dashboards and do not support real-time interaction or steering. LLM4AD [18] provides pre-run configurations and simple control (e.g., start, resume) over an evolution on its UI. In our work, we not only visualize the evolution process but also enable real-time steering.

2.3 Human Oversight of Iterative Program Generation

Existing HCI research on human-AI collaboration in programming primarily studies a user working with one assistant or one active artifact through code suggestions and chat-based editing [1, 17, 21, 30]. Recent taxonomies of human-LLM roles [4] also focus on dyadic archetypes. By contrast, in program evolution, the expert oversees persistent alternatives and shifts from co-author to curator and strategist: the expert must allocate attention, compare strategies, inspect evidence, follow lineages, and decide where to intervene. Mixed-initiative program synthesis provides a closer precedent. ASSUAGE [12] exposes parallel assembly-synthesis instances and lets users guide them through a predefined vocabulary of constraints and decompositions. Our system works with open-ended algorithmic ideas: LLM-generated programs can introduce strategies outside predefined categories, and experts can describe suggestions and new directions in NL. Steering such open-ended ideas requires *semantic oversight*: experts must understand the strategies represented by generated programs, compare them across lineages, check them against code and evaluation results, decide which to preserve, combine, prune, or redirect, and monitor their effects.

Several HCI frameworks help organize these needs. Endsley's situation awareness (SA) framework [8] describes how operators perceive a system's state, comprehend what it means, and project what may happen next. Sheridan's levels of automation (LOA) [27] and Issak et al.'s trajectory of control [14] describe movement between passive monitoring and active intervention. Pareek et al. [24] show that multi-agent interfaces must balance useful process visibility and trust against information overload. Our system supports semantic oversight through population-level attention routing, multi-level inspection of algorithmic strategies, and interventions that shape future generations.

3 Background: LLM-Driven Program Evolution

This section introduces the key concepts and terminology of LLM-driven program evolution. All existing systems share a four-stage loop (Fig. 2, Bottom Row), which continues until a termination condition is met, such as a time budget, a target score, or a maximum number of nodes. In all existing systems, the loop is fully automatic.

① **Population Database.** An evolution maintains a *population* of candidate programs stored in a population database. Each candidate is represented as a *node* in an evolution tree, carrying the program's source code together with metadata such as performance scores, code diffs, and LLM reasoning chains. In many systems [16, 23, 25], the population is partitioned into *islands*, sub-populations that evolve independently. Optionally, *migrations* move nodes among islands. These mechanisms, inspired by habitat isolation in nature, help maintain program diversity and improve performance.

② **Parent Selection.** In each generation, the system selects one or more nodes from the population as *parents*, the starting points for producing the next generation of candidates. The automatic and probabilistic selection typically favors nodes with higher scores, ensuring that good nodes are more likely to be refined further.

③ **Node Generation.** New nodes are created by prompting an LLM to modify the selected parents. In a *mutation*, the LLM reads a single parent, reasons about its algorithm, and proposes a targeted code edit. In a *crossover*, the LLM reads two or more parents and combines their ideas into a new program. In both cases, the LLM produces a *patch* (code edits) applied to generate the new node, optionally with a *patch description* summarizing the intent in NL. Unlike classical EAs, where mutations are random perturbations, each LLM-generated patch is deliberate and semantically meaningful.

④ **Node Evaluation.** An *evaluation function* executes the newly generated program and assigns one or more numeric scores measuring solution quality. This function is provided by the user and measures domain-specific metrics (e.g., prediction accuracy and latency). The evaluation stage can also generate additional metadata. For example, ShinkaEvolve [16] computes an embedding vector from each node's code for similarity comparison across the population. The evaluated node, together with its scores and metadata, is then added back to the population database, and the loop repeats.

4 Formative Study

To inform the design of EvoMaestro, we conducted a formative study with engineers and researchers who regularly solve optimization problems and could benefit from LLM-driven evolution. This study aimed to understand three key aspects: (1) how they approach iterative algorithm improvement with LLMs, (2) what information they seek when examining an LLM-driven evolution run, and (3) what intervention capabilities they expect to effectively steer such processes. We used the web UI of ShinkaEvolve [16] as the baseline system for this study, as it is one of the most advanced open-source LLM-driven program evolution tools [18, 26].

4.1 Participants and Procedures

We recruited eight participants (**P1-P8**; 2 female, 6 male) from universities and companies across three countries via direct outreach and referrals (see Appendix A.1). All participants have substantial experience in algorithm design or optimization, including software engineers (P1, P2, P6) and graduate students (P3-P5, P7, P8) in DL and mathematics. All are proficient in reading code and regularly use LLM-based programming tools, with four (P1, P4-P6) having more than five years of competitive programming experience. We did not require prior experience in EA, as the target users are domain experts who seek to apply EA techniques to their problems, rather than EA specialists. Before the participant interviews, we collaborated with E1, a co-author and domain expert. He is a PhD student in theoretical computer science with six years of competitive programming experience and had extensive prior experience with the baseline system. We interviewed him about his experience with and feedback on the baseline. Based on his feedback and our literature review, the research team developed the two design probes. As E1 was a co-author, we did not include him among the recruited participants.

We conducted a semi-structured interview with each participant, with each session lasting approximately 50 to 60 minutes (see Appendix A.2 for details). The two design probes covered alternative visualizations of evolution results and interaction mechanisms for steering an evolution process. In each session, participants explored a pre-generated ShinkaEvolve [16] evolution with 50 nodes using a think-aloud protocol, followed by semi-structured questions. We then presented the probes and gathered feedback.

4.2 Findings and Design Requirements

We conducted a thematic analysis following Braun and Clarke [3]. The first author performed iterative open coding on interview notes, and themes were refined through team discussion. Our analysis revealed six themes, which we distilled into six design requirements. We organize them into two groups to reflect the two core problems identified from interview feedback: *information overload* (DR1–DR4) and *lack of human-in-the-loop interaction* (DR5–DR6).

4.2.1 Problem 1: Information Overload. Participants reported being overwhelmed by the volume of information on the interface, yet paradoxically found it insufficient for understanding what they cared about most: what algorithmic strategies the nodes were using, which ones were promising, and how the search was progressing.

DR1: Surface algorithmic intent of node generation. Participants consistently reported that raw code and scores did not convey the algorithmic strategy behind each node. P1 noted: *"there is lots of information, but none of it helps me understand [what is going on]."* While the baseline did include patch descriptions, they were buried and easy to miss. When participants found them, both P1 and P6 identified them as the most helpful information. E1 exhibited a telling workaround: rather than reading code directly, he routinely copied it into ChatGPT to ask about the algorithmic idea. The system should present the *idea* behind each node, not just its code and score, in a way that is immediately visible and scannable.

DR2: Support multi-level comparison between nodes. Comparing nodes was a major pain point. P1 reported that *"comparing code is tedious, and it's hard to compare the differences in thinking between solutions,"* and noted that *"it's hard to tell at a glance whether performance improved or regressed"* when the parent and child had similar scores. The baseline only supported comparing a node with its parent; E1 resorted to manually writing down scores for comparison. Participants wanted comparison at multiple levels: across islands (P5), side-by-side code diffs (P1, P6), and strategy-level clustering (P5). E1 likewise wanted comparison across islands. The system should enable comparison at multiple abstraction levels.

DR3: Provide population-level overviews. Multiple participants desired a global overview beyond the tree structure. P1 and P6 articulated this clearly: *"it would be best to have a global summary, like a report, summarizing what directions all these nodes explored."* P5 found it *"difficult to find the top-K nodes."* E1 asked for the global average score, which was not available. The system should aggregate individual node information into overviews.

DR4: Prioritize decision-relevant information. Even when relevant information was present, participants struggled to focus on what mattered. P7 and P8 stated: *"too much information with no focus."* The baseline mixed global statistics and local node details in the same tabbed panels (P2), and its visual encoding was cluttered with color schemes for performance and island membership. Participants converged on filtering as a solution: P1, P5, P7, and P8 all suggested dimming nodes below a performance threshold. P4 and P3 suggested novelty metrics to highlight qualitatively different nodes, and P6 wanted *"performance jumps"* to be visually salient. The system should highlight nodes most likely to merit attention and allow experts to customize what counts as noteworthy.

4.2.2 Problem 2: Lack of Human-in-the-Loop Interaction. Participants consistently expressed a desire to intervene in evolution.

DR5: Enable targeted expert intervention. Participants wanted to track nodes of interest and act on them, with P1 requesting bookmarking. E1 had similarly developed a manual workaround by writing down node IDs on paper. P7 wanted to *"choose a subset of nodes to continue evolving"* and prevent others from evolving to save resources. P8 shared the same idea. We presented one design probe that included a suggestion dialog for NL-guided mutation and a multi-select panel for merging. Reactions were largely positive: P1 found the *"suggest"* feature promising but noted that the system should *"provide recommendation options"* rather than requiring guidance from scratch. P5 is skeptical about merging, questioning whether it might *"go against the original intent of evolutionary computation,"* while P4 and P6 responded positively. The system should

allow experts to intervene, including guiding mutations, combining ideas from multiple nodes, and shaping which lineages continue.

DR6: Support flexible evolution pacing. Participants did not want a binary switch between full automation and manual control. P5 and P8 noted that the desired involvement depends on situations and problems. P7 articulated a similar vision: *"fully automatic generation of some nodes, but with checkpoints where you can intervene at any time."* This points to a need for flexible pacing: the system should support a spectrum of involvement levels, from passive monitoring through throttled observation to step-by-step control, with the ability to shift between modes as the situation demands.

All DRs map to the theoretical frameworks introduced in Section 2. DR1–DR4 address information overload by supporting the three levels of Endsley's SA [8]: DR3–DR4 support perception (what is happening in the population), DR1–DR2 support comprehension (why nodes differ), and DR3 also supports projection (which directions are promising). DR5–DR6 address the lack of steering, moving the system from Sheridan's [27] Level 9 toward Level 5.

5 EvoMaestro

Guided by the six design requirements (§ 4), we propose a framework for steerable program evolution and EvoMaestro, an interactive visual analytics system built upon this framework (Fig. 2). The framework (§ 5.1) extends the standard evolution loop with intervention points and an expert review prioritization stage, enabling steering. The interface (§ 5.2) organizes information through progressive disclosure to support understanding of an evolution at different levels of detail and provides views for steering the evolution.

5.1 Steerable Program Evolution

This subsection describes how our framework transforms the fully automatic four-stage loop (①–④, § 3) into a steerable process. We first describe the four actions experts can take to intervene in a running evolution (DR5, DR6), then explain how these actions are enabled by our modifications to the loop stages.

When the expert identifies a node worth refining, they can use ***Suggest*** to select it as the parent node and provide NL guidance to steer the next mutation. The system feeds this guidance into the LLM's prompt, producing a new child node. To help the expert formulate their guidance, the system also generates a recommendation that the expert can adopt, modify, or ignore. Additionally, the expert can use ***Merge*** to select two nodes and direct the system to combine them into a new node with optional NL instructions. The system recommends merge candidates based on performance and dissimilarity (details in Appendix B.2). If the expert identifies a node pursuing an unproductive strategy, they can use ***Ban*** to remove it from future parent selection, thereby pruning unproductive lineages. For the entire loop, the expert can flexibly control the pace, pausing, stepping through one node at a time, or letting the evolution run automatically. To enable these actions, our framework extends the standard four-stage loop at key stages (Fig. 2, Top):

① **Population Database.** The expert can *ban* nodes in the population database, preventing them from being selected in stage ②. Outside the loop, *Maestro Agent*, an LLM assistant, reads from the population database to help experts understand code, compare nodes, and identify patterns across the evolution (§ 5.2.2).

②–③ **Parent Selection & Node Generation.** These stages are automatic in the standard loop. Ours adds intervention points that allow the expert to override these stages through *Suggest* or *Merge*.

④ **Node Evaluation.** At this stage, besides running evaluations, existing systems compute embeddings from each node's raw code for similarity comparison (§ 3). EvoMaestro instead computes *reasoning-based embeddings* from each node's patch description and reasoning chain pair, capturing the similarity of *algorithmic intent* rather than syntactic structure. These embeddings underpin the clustering, dissimilarity chords, and comparison views throughout the interface. Experts can toggle to code-based embeddings when they want to compare implementations instead of strategies.

⑤ **Expert Review Prioritization (new stage).** The framework adds a fifth stage after node evaluation. A prioritization function consumes scores and embeddings from Node Evaluation and flags noteworthy nodes for inspection. By default, the function flags nodes whose evaluation score improves by at least 15% relative to their parent. Experts can adjust the threshold or replace the function with one based on other signals, such as embedding dissimilarity. Unlike ShinkaEvolve's novelty-based rejection sampling [16], these flags affect only interface highlighting and do not alter scores, reject candidates, or affect parent selection.

Pacing control. The expert can switch at any time between three modes: Auto (free-running), Wait (pauses between generations), and Manual (one node at a time).

The steerable evolution loop provides the mechanisms for experts to intervene, while the EvoMaestro interface surfaces these steering capabilities alongside features that support understanding the evolution at different levels of detail, which we describe next.

5.2 EvoMaestro Interface

The EvoMaestro interface (Fig. 1) is designed around a progressive workflow: the expert starts by gaining a global overview of the evolving population (§5.2.1), then inspects specific nodes to understand their strategies and compare them (§5.2.2), and finally acts on those insights by steering the evolution (§5.2.3).

5.2.1 Evolution Understanding (DR3, DR4). Before drilling into individual nodes, the expert typically needs to *get oriented* about the evolving population: what strategies have been explored, which nodes are performing well, and where the search might be heading. They can then *analyze the population structure* to identify clusters of similar strategies and the overall score distribution.

Getting oriented. When the expert opens EvoMaestro, a *radial tree* in *Population Overview* (Fig. 1 A1) provides an immediate visual summary of the full evolution history. Nodes are arranged in concentric rings by generation and grouped into angular sectors by island (A2), with a sequential color scale encoding performance so that high-scoring nodes stand out at a glance. Solid lines connect parent and child nodes, tracing evolutionary lineages; cross-island links indicate migrations where a node's offspring was assigned to a different island. We provide design rationale details in Appendix B.1. To guide their focus, the best node in each island is marked with a golden circle ◉ (A3), and noteworthy nodes prioritized for

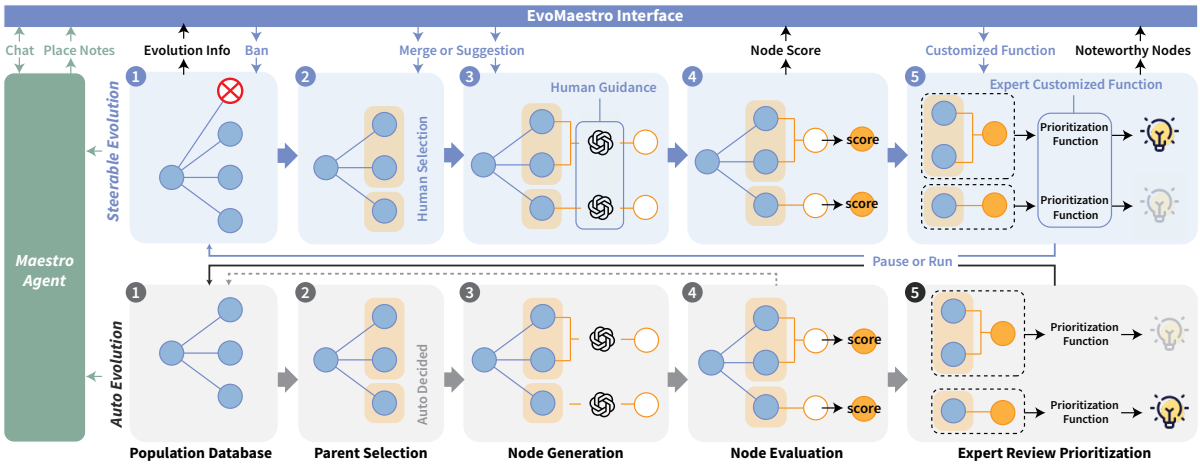

Figure 2: Steerable Program Evolution Framework. Stages ①–④ correspond to the standard loop (§ 3); ⑤ is added by EvoMaestro. The steerable loop (Top) extends the automatic loop (Bottom) with Ban, Merge, Suggest, Expert Review Prioritization, and Pacing Control. The Maestro Agent (left) provides conversational code explanation and annotation.

expert review (§ 5.1) are flagged with a lightbulb 💡 (A4). To understand how different the top strategies are from each other, they can read the *Dissimilarity Chords* (A5), color-coded arcs connecting the best node of each island. For a narrative summary without inspecting individual nodes, a sidebar panel (A6) provides an LLM-generated *Evolution Overview* describing explored, promising, and failed strategies. To reduce clutter, they can click on the encoded averages (the global one in the legend, per-island ones on seed nodes, and per-generation ones on arcs) to dim all nodes below that average; error and timeout nodes can be toggled off.

Analyzing population structure. Two additional views help the expert understand patterns across the population. *Clusters View* (Fig. 1A7) projects nodes into a scatter plot using PCA on their reasoning-based embeddings (§ 5.1), revealing clusters of nodes that share similar strategies. *Score Distribution* (A8) shows a density plot of scores; the expert can set a threshold to filter out low-scoring nodes, which are then dimmed on the tree.

5.2.2 Node Understanding (DR1, DR2). Beyond the population level, experts need to understand individual nodes. EvoMaestro supports this progressively: *inspecting nodes* via hover tooltips and lineage paths for quick triage, *understanding a node's strategy* through the detail panel and Maestro Chat with line-level annotations, and *comparing nodes* through dissimilarity chords and side-by-side views.

Inspecting nodes. Hovering over a node on the tree reveals a brief tooltip (Fig. 1B1) showing its name, score, patch type, and island assignment, allowing the expert to quickly triage nodes without leaving the tree view. Clicking a node selects it and draws an orange lineage path from the initial seed to that node, letting the expert trace its evolutionary history at a glance. The last segment of this path uses a solid/dashed encoding: the solid half is drawn closer to the higher-scoring node of the pair, instantly revealing whether performance improved or regressed.

Understanding a node's strategy. The detail panel (Fig. 1B2) shows the node's performance scores, patch description, and metadata such as island assignment, node ID, and parent lineage. The patch description provides a high-level understanding of the *idea* that the patch pursued without requiring the expert to read the code. For a deeper understanding, they can open the full source code (Fig. 1B3) and invoke *Maestro Agent* through the *Maestro Chat* to explain it. *Maestro Chat* operates in two modes: in *Code view*, the expert asks the agent to explain a single node's algorithmic strategy; in *Code diff*, they view two nodes side by side and ask the agent to compare them. Additionally, *Maestro Agent* can place *line-level annotations* directly on the code regions and highlight them. This anchors abstract strategy descriptions to concrete code locations, which is more effective for supporting human judgment [28].

Comparing nodes. *Dissimilarity Chords* (Fig. 1A5) helps compare the best node in each island. Experts can select multiple nodes for detailed comparison in *Comparison Panel* (Fig. 1B4), which shows performance comparisons and a dissimilarity heatmap. They can also open a side-by-side code diff for any pair, with *Maestro Chat* available to explain the differences. The dissimilarity values throughout the interface are computed from reasoning-based embeddings (§ 5.1) for comparing algorithmic strategies.

5.2.3 Evolution Steering (DR5, DR6). Having understood the population and identified promising or problematic nodes, the expert can act on those insights by guiding and combining nodes through *Suggest* and *Merge*, pruning unproductive lineages through *Ban*, and controlling the pace of the evolution. The steering mechanisms described in Section 5.1 are accessed through a unified context menu (Fig. 1C1) that appears when right-clcking any node, along with *Mark* and *Note* for bookmarking and annotating nodes.

Guiding, combining, and pruning nodes. Selecting *Suggest* opens a modal (Fig. 1C, top) where the expert writes NL guidance, with an AI-generated recommendation as a starting point. Selecting *Merge* opens a modal (Fig. 1C, middle) listing recommended merge partners ranked by score and dissimilarity. Both actions produce new child nodes that appear on the tree in real time. Selecting *Ban* (Fig. 1C, bottom) marks the node as banned, visually dimming it on the tree and removing it from future parent selection.

Controlling the pace. *Run Control* (Fig. 1C2) allows switching between modes (§ 5.1), in which new nodes appear with a dashed border during evaluation, transitioning to solid once complete.

6 System Demonstration

To assess EvoMaestro in a complex real-world setting, we present a system demonstration with E1, a co-author and domain expert who contributed feedback during the formative phase. E1 was particularly suited to this extended demonstration because he combined extensive experience with the baseline system, relevant algorithm-design expertise, strong motivation to apply LLM-driven program evolution in his research in theoretical computer science and algorithm design, and willingness to commit substantial time. We adapted an optimization problem from the ICPC 2025 Online Winter Challenge[2] as the task. It aims to optimize traffic routing in a large-scale AI computing cluster interconnected by optical circuit switches (OXCs). Appendix C provides the task details in depth. E1 used EvoMaestro for approximately 16 hours over 7 days for the task. The following describes his workflow, which we reconstructed from several types of data, including his contemporaneous activity log of important actions and reasoning, a short post-study summary, system logs of evolution actions, *Maestro Chat* histories, and generated node, score, and lineage artifacts. During the reconstruction, we triangulated his rationale with recorded actions and generated artifacts, and followed up with E1 when details were unclear. We did not log every low-level UI action (e.g., tab switch or node click).

To kickstart the evolution process, E1 generated an initial seed program using GPT-5. During the study, E1 interacted with EvoMaestro through a recurring analysis loop: (1) understanding the current evolution process, (2) inspecting specific nodes for steering, (3) intervening to inject domain knowledge, and (4) setting the system to evolve automatically before re-observing the results. We illustrate each step of this loop with concrete examples (Fig. 3).

Understanding the current evolution process. At the beginning of each cycle, E1 typically sought a global overview, including best nodes and overall evolutionary trend. For example, after the initial 92 nodes were generated automatically, he first located the best nodes (both global and island-level) in *Population Overview* (Fig. 3A), highlighted by golden circles (Fig. 3A1), then skimmed the arcs (Fig. 3A2) to preview the evolutionary trend. After confirming that the evolutionary trend was generally improving (i.e., arc colors became deeper across generations), he examined the *Evolution Overview* (Fig. 3A3) to obtain a strategy summary. It showed that the top-performing nodes had converged on two strategies (multi-attempt randomized greedy search and structural deterministic routing) and that earlier time-bounded search loops had largely failed, whereas successful nodes used adaptive attempt budgets. This raised a key question: were the top-performing nodes fully utilizing the 50-second execution time limit? To answer this, he proceeded to inspect specific nodes.

Inspecting nodes for steering. When something at the global level caught his attention, E1 inspected individual nodes to verify hypotheses or seek explanations. To investigate the time-budgeting question, he first used the *Score Distribution* (Fig. 3B1) to filter out low-performing nodes and focus on top candidates. He then examined the detail panels of selected nodes (N-54, 67, 75, 81, and 91, the best node) and reviewed their evaluation results across test cases (Fig. 3B2). The longest runtime was 20s, far below the limit, confirming that the node was leaving significant computation time unused. He then used *Maestro Chat* (Fig. 3B3) with line-level annotations to understand the best node's search logic in detail. Surprised by the sophisticated algorithm (precomputation, pruning, and steal mechanisms) implemented in the best node (N-91), he further explored other nodes (N-65, 67, 81), guided by the noteworthy-node flags 💡. He compared their performance and dissimilarity and found that although they have similar scores, N-91 is distinct from the other three nodes, as shown in *Comparison Panel* (Fig. 3B4). He further used *Maestro Chat* to understand their code details.

Intervening with domain knowledge. Once E1 identified an actionable insight during node inspection, he intervened in the evolution process. Continuing the above example, he used *Suggest* (Fig. 3C1) on N-91 (44k) to inject time-budgeting knowledge: *"continuously search until the exact moment before the 50s timeout,"* producing N-94 (44.5k). He then generated 4 new nodes (N-95 - N-98) using *Run Control* (Fig. 3C2). Based on insights from node inspection, he found the deterministic waterfilling algorithm in N-65 complementary to N-91, and thus used *Merge* (Fig. 3C3) to combine them, producing N-99 (45.0k). Beyond these targeted interventions, E1 also explored speculative ideas using *Suggest*, adapting his strategies based on observed patterns. For example, by inspecting improvement trends through arc and node colors, he observed that one lineage (N-26->29->48) lagged behind the overall generational progress (Fig. 3A4 and C4). After discussing this in *Maestro Chat*, he concluded that this lineage introduced excessive complexity with only marginal gains, and used *Ban* to remove N-48.

Resuming automatic evolution. After each round of intervention, E1 resumed automatic evolution. Before resuming, he typically used *Ban* to prune unproductive nodes, preventing the system from revisiting dead ends. He then set a node budget in *Run Control* (e.g., 20 new nodes) and left to do other work. Upon returning, he examined the *Population Overview*, *Evolution Overview*, and *Score Distribution* to assess whether the injected knowledge had taken effect, thereby initiating the next analysis cycle.

This cycle was repeated over the 7-day period. E1 progressively shifted from active steering to passive monitoring as the evolution matured. By the end, the system had generated over 200 nodes with the best score of 45.8k (N-144) across multiple distinct strategies (Fig. 3D). Throughout the process, he relied entirely on observation and steering through EvoMaestro, without writing code. In his post-study summary, E1 reported that EvoMaestro shifted his work from tedious code review toward active human-AI collaboration. He identified *Merge*, *Suggest*, and *Maestro Chat* as central to this shift and expressed interest in using EvoMaestro in future research.

7 User Study

We conducted a within-subjects user study to compare EvoMaestro against the baseline. Our goals were to assess whether EvoMaestro improves experts' ability to (1) understand the state and strategies of an evolving population, (2) identify decision-relevant information, and (3) intervene effectively in an evolution. Accordingly, this

[2]https://icpc.global/regionals/finder/Winter-Challenge-2026

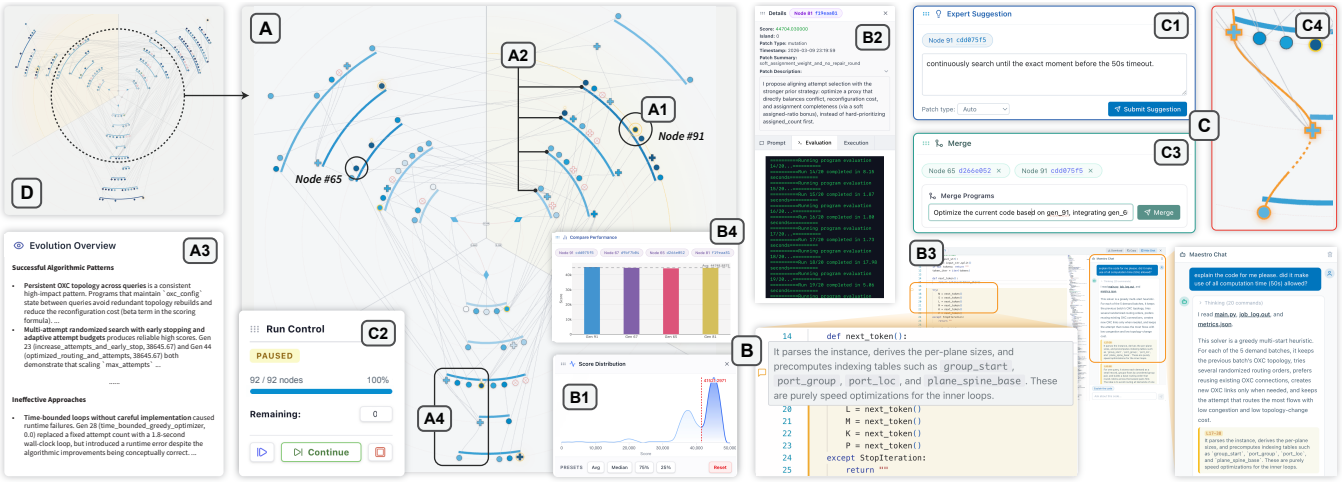

Figure 3: The expert's workflow during the system demonstration. (A) Understanding: *Population Overview* shows the evolution tree with best nodes highlighted (A1), per-generation score trends on arcs (A2), *Evolution Overview* summarizing strategies (A3), and a banned node (A4). (B) Inspecting nodes: *Score Distribution* filters low-performing nodes (B1); the detail panel shows evaluation runtimes (B2); Maestro Chat explains node code with line-level annotations (B3); and *Comparison* reveals dissimilarity among top nodes (B4). (C) Steering: *Suggest* injects time-budgeting guidance (C1); *Run Control* sets the node budget (C2); and *Merge* combines complementary nodes (C3). (D) The final evolution tree with the best node highlighted.

comparison evaluates understanding, workload, and steering support, not final program quality. We describe the study setup first and then report the results.

7.1 Study Setup

With the same recruitment process as the formative study (§ 4.1), we recruited 12 participants (**U1–U12**; 4 female, 8 male). Three (U3, U6, U9) were also participants in the formative study. All participants had experience in algorithm design or optimization, including software engineers (U3, U9) and graduate students (U1, U2, U4–U8, U10–U12) spanning DL, physics, statistics, HCI, and mechanical engineering. All are fluent in Python and daily users of LLM-based programming tools. Most had basic familiarity with LLM-based evolutionary systems; two (U2, U11) had no prior exposure and one (U8) had long-term hands-on experience with similar systems. Participant details are provided in Table 2 (Appendix D.1).

Study design: We used a within-subjects design comparing EvoMaestro against the baseline ShinkaEvolve [16] web UI. Each participant used both conditions on two multi-objective optimization tasks of comparable complexity (details in Appendix D.2), counterbalanced using a 2×2 Latin square (Appendix D.3). In the EvoMaestro condition, the evolution remained live so that participants could execute real-time interventions.

Procedure: Each 80-minute session comprised a tutorial on LLM-driven evolution, followed by two 25-minute task sessions (one per condition). In each task session, participants first spent 15 minutes on sensemaking tasks: identifying the most promising nodes, describing algorithmic strategies, and spotting evaluation weaknesses. They then spent 10 minutes on intervention tasks: in the EvoMaestro condition, writing a NL suggestion and initiating a merge; in the baseline condition, describing their intended interventions in writing, capturing intervention *intent* for qualitative comparison. After each condition, participants completed a questionnaire and NASA-TLX. A post-study interview compared experiences across both conditions. The details are provided in Appendix D.5.

Measures: We designed a custom 7-point Likert questionnaire to assess three aspects mapped to our design requirements. *Understandability* (5 items, both conditions) measured whether each system helped participants understand the population's evolution, discern strategies, locate notable nodes, compare nodes, and identify individual node algorithms, covering DR1–DR4. *Usability* (3 items, both conditions) assessed ease of use, information organization, and task effectiveness. These shared items enable paired statistical comparison between conditions. Three additional *Intervenability* items (EvoMaestro only) assessed intervention identification (DR5), pacing control (DR6), and intervention effectiveness (DR5), and one item assessed the value of *Maestro Chat* for code understanding. These EvoMaestro-only items capture capabilities with no baseline equivalent. We also administered the NASA-TLX [10] after each condition to measure cognitive workload and collected open-ended responses and a post-study interview for qualitative analysis. Full questionnaire items are provided in Appendix D.4. For shared items, we use the Wilcoxon signed-rank test with Cliff's delta (δ) for effect sizes [6] where $|\delta| < 0.33$ is small, < 0.474 medium, and ≥ 0.474 large. EvoMaestro-only items are reported descriptively. Qualitative analysis followed the same approach in the formative study (§ 4.2).

7.2 Results

We report the results from our Likert questionnaires and NASA-TLX, followed by qualitative findings from the open-ended questions and post-study interviews.

7.2.1 Closed-ended Questions. Overall, EvoMaestro significantly outperformed the baseline on all eight shared Likert items and received consistently high ratings on the EvoMaestro-only items.

Understandability & Usability. Fig. 4 (left) summarizes the comparison on shared Likert items. EvoMaestro scored significantly higher than the baseline on all eight shared items. The strongest are task effectiveness ($\delta = 0.694$, large) and information organization ($\delta = 0.639$, large), followed by locating abnormal nodes (DR4: $\delta = 0.521$, large) and comparing nodes (DR2: $\delta = 0.486$, large). Overall, the participants rated EvoMaestro's understandability at a median of 6.0 (mean 6.20) versus 5.0 (mean 4.65) for the baseline, and usability at 7.0 (mean 6.31) versus 5.0 (mean 4.61). The largest improvements were in usability items, suggesting that our progressive disclosure and visual encodings make the evolution process not only more comprehensible but also much easier to work with.

EvoMaestro-only items. The Intervenability and *Maestro Chat* items were consistently rated highly (Fig. 4, middle). AI-assisted code explanation (*Maestro Chat*) received a median of 7.0 (mean 6.42, SD 0.90), the highest rating of any individual item, reflecting participants' strong appreciation for LLM-mediated code understanding. The three intervenability items received medians of 6.0–6.5, indicating that participants found the steering mechanisms effective. EvoMaestro's design helps experts see *where* to intervene, while the act of formulating effective guidance remains more challenging, as identifying intervention needs (DR5, median 6.5) scored slightly higher than conducting interventions (DR5, median 6.0).

Cognitive workload (NASA-TLX). Fig. 4 (right) summarizes the NASA-TLX results. EvoMaestro significantly reduced overall cognitive workload (the mean of all six subscales per participant) compared to the baseline (median 3.08 vs. 4.25; $p = 0.004$, $\delta = -0.417$, medium effect). For the subscales, the largest improvements were in *frustration* ($p = 0.004$, $\delta = -0.535$, large), *temporal demand* ($p = 0.016$, $\delta = -0.479$, large), and *mental demand* ($p = 0.043$, $\delta = -0.347$, medium). The frustration reduction is the largest effect, which we attribute to the combination of progressive disclosure design (reducing information overload) and steering mechanisms (enabling experts to act on their observations rather than passively watching). Both systems enabled participants to complete the tasks, as the Performance and Effort subscales did not differ, but EvoMaestro did so with much less cognitive strain.

7.2.2 Qualitative Findings. We organize the qualitative findings from post-study interviews and open-ended responses around three concerns: understanding, steering, and usability.

Evolution Understanding. The feedback has a consistent contrast between the two systems regarding information structuring. The baseline presented all data at once, which overwhelmed users. U4 and U9 reported confusion and cognitive overload due to the large number of nodes and complex visualizations. EvoMaestro's progressive disclosure approach relieves users of information overload. U5 noted a significantly reduced mental load, describing the experience as *"peeling away layers."* U4 found the radial tree *"much clearer than ShinkaEvolve's style of piling everything together."* These findings suggest that the value of our design lies not in showing less information but in revealing it in the right order: population-level orientation first, then node-level details on demand.

Evolution Steering. All participants appreciated the ability to intervene. U5 noted that *"we can add constraints to the LLM, and this is visible on the system, making it easy to trust,"* and U10 noted that *Ban* resolved their concern about *"the AI wasting tokens repeating the same strategy."* Despite this appreciation, their comments revealed a nuanced relationship between automation and human control. Rather than replacing automation, participants used LLM-generated recommendations as a starting point: U7 described *"checking the merge recommendations, examining the recommended nodes in detail, then finding further inspiration,"* and U4 noted that the recommendations *"deepened my understanding of the algorithms,"* suggesting they served as learning aids rather than just shortcuts. Looking forward, participants wanted even richer automation support, such as *"global recommended actions"* (U9) and guiding groups of related nodes simultaneously (U7, U8).

Overall Usability. NL guidance is crucial for improving the usability of such complex systems with dense information, as *Maestro Chat* was the highest-rated feature (median 7.0). U5 noted its usefulness at different levels: *"not only can it provide global understanding, but also understanding of individual lines or functional blocks, saving a tremendous amount of time."* However, the system's usability did not rely on *Maestro Chat* alone. U4, who had competitive programming experience but a limited optimization background, chose not to use it because of its verbose responses but still found the system accessible through interactive steering: *"while playing around with Merge, I actually came to understand a bit of what optimization researchers do."* The system offers interactions that suit users with different backgrounds: experts with strong algorithmic knowledge can leverage *Maestro Chat* for deep code understanding, while those newer to the problem domain can build their understanding through hands-on interaction with the steering features.

8 Discussion

Designing and evaluating EvoMaestro highlights the importance of expert agency in open-ended LLM-driven search. We discuss how understanding and steering support this agency, then examine current limitations and future directions.

8.1 Lessons Learned

LLM-driven program evolution creates two linked challenges for expert agency. A growing population can exceed what an expert can inspect linearly, limiting their ability to judge what the system is doing. Meanwhile, full automation leaves domain insights unused when experts cannot redirect the search. Our experience suggests that preserving expert agency requires both forming an informed judgment and making that judgment affect subsequent automation.

Understanding preserves expert agency. Information overload cannot be addressed by simple information hiding or summarization. Scores identify relative performance, but do not explain how a candidate works or whether its strategy makes sense for the problem. Semantic oversight therefore follows three steps: (1) route attention using population structure, filters, noteworthy-node flags, and lineage; (2) compare algorithmic strategies at an appropriate level; and (3) inspect code, diff, logs, and evaluation results when verification is needed. *Maestro Chat* and its explanations help

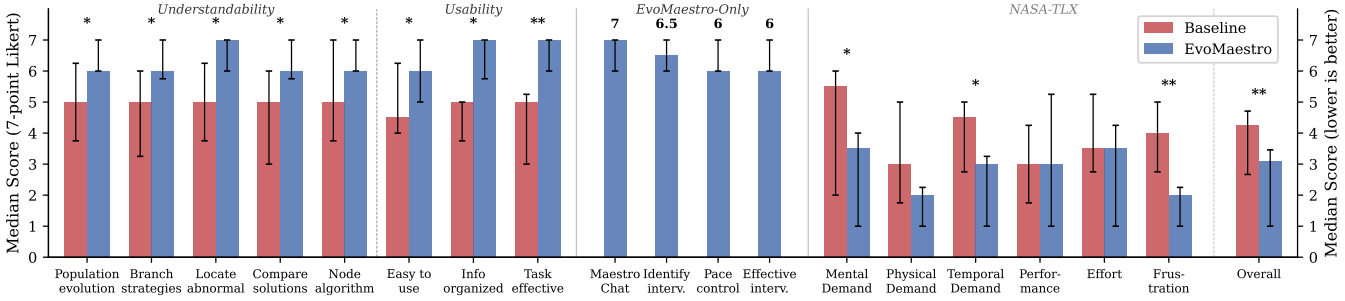

Figure 4: Evaluation results (detailed values in Tables 4 and 5 in Appendix D.6). Bars show median scores with interquartile ranges (Q1–Q3). Left: Shared Likert items (7-point scale; higher is better). Middle: EvoMaestro-only Likert items (numbers indicate medians). Right: NASA-TLX scores (lower is better); its Performance subscale is participants' perceived success in completing the task. Asterisks mark significant differences.

experts move between code and strategy, while the code and evaluation results remain available for checking. In our evaluation, some of the strongest improvements concerned information organization, locating abnormal nodes, and comparing nodes, while overall cognitive workload decreased. *Maestro Chat* was the highest-rated feature, suggesting that when LLM agents produce complex artifacts, dedicating another agent to explanation can substantially reduce the cognitive cost of human oversight. More broadly, by presenting supporting evidence in a useful order, such interfaces can help experts form their own judgment about a candidate and identify when closer inspection is needed.

Steerability turns automation into collaboration with trust. Forming a judgment is only one part of agency, because understanding alone leaves the expert just as an observer. Experts also need to act on it. Collaboration requires trust and a way for domain knowledge to change what the automated process does next. In EvoMaestro, the significance of its intervention features is not that selection, rejection, and recombination are new to EA, but that experts can use these operations based on algorithmic ideas to change which ideas later generations develop. The immediate effects of these interventions are visible in the population, so experts can inspect how the population responds before deciding whether to intervene again. Participants rated the intervention support highly. They consistently linked the ability to intervene with their trust in the system's output. U1 called interventions *"necessary for producing better results and trusting those results,"* and U5 valued that the constraints added by the expert *"are visible in the system, making it easy to trust."* The system demonstration also illustrates how an expert moved between active intervention and passive monitoring. The broader lesson is that collaboration does not require constant human control but trust and a path for expertise to affect automation when judgment is needed.

Together, these lessons connect agency to task-level alignment in LLM-driven search. Initial prompts and evaluators encode the task objective, but may not anticipate every failure mode or concern that emerges during open-ended search. Without understanding, experts cannot verify whether emerging strategies remain consistent with their intent; without steering, they cannot redirect the search when strategies diverge. Task-level alignment should therefore be supported as an ongoing process of inspection, correction, and monitoring, rather than only through an initial specification.

8.2 Limitations and Future Directions

We also acknowledge our limitations and discuss future directions.

System demonstration with a co-author. E1 was a co-author who provided feedback during the formative phase. Although he had not seen the final EvoMaestro system before the demonstration, his prior involvement gave him expectations about the system's capabilities. Together with his strong motivation and substantial time investment, this may have encouraged deeper and more persistent engagement than might be expected from other users and may have shaped his positive reflections. We therefore use the demonstration only to illustrate how sustained expert steering unfolded through his workflow. Future case studies with independent domain experts are needed to examine whether these usage patterns generalize.

Reliability of generated explanations. LLM-generated patch descriptions, summaries, chat responses and annotations may be plausible yet incorrect. EvoMaestro allows for progressive in-depth inspection, but experts may still miss errors, especially outside their areas of expertise. Future systems should expose stronger provenance and support cross-checking and validation across models.

Scalability. Our evaluation involved evolution trees with 50 nodes, and the system demonstration produced approximately 200 nodes. Filtering helped experts focus at these scales, but does not establish that the interface can support thousands of nodes. Larger populations may require hierarchical aggregation, collapsed lineages, or focus-and-context techniques.

Future work can examine how our approach can be extended or transferred to other domains and iterative multi-agent systems.

Interaction extensions. Participants wanted to guide groups of related nodes, suggesting future support for recommending and steering node groups. The system could also suggest pacing changes, such as automatic pausing when a population converges.

Beyond software engineering. Programs can represent candidate solutions outside software engineering when they can be executed and evaluated with metrics (§ 2). Potential users may have deep domain knowledge without equivalent programming expertise or the time to inspect every generated program directly.

The HCI challenge is therefore to connect domain concepts, algorithms, code, and evaluation results. Our studies cover algorithmic and numerical optimization, and interfaces for other fields may require different representations and interventions. Evaluating the approach in these fields remains future work.

From co-author to curator: Beyond program evolution. Most HCI work on LLM-assisted programming studies a single developer working with a single assistant [1, 21, 30]. However, as LLM-based multi-agent systems advance beyond program evolution, these lessons learned may also inform the design of other iterative multi-agent systems that maintain persistent revision histories. Users in such systems similarly need to locate, compare, combine, and redirect alternatives generated along different trajectories. Evaluating this transfer remains future work.

9 Conclusion

Existing LLM-driven program evolution systems operate as fully automated black boxes that provide limited support for domain experts to understand and steer an evolution. Through a formative study with 8 domain experts, we identified two core challenges (understanding and steering) and derived six design requirements. We then developed a steerable program evolution framework that enables expert intervention and EvoMaestro, which supports semantic oversight through multi-level inspection and interventions based on algorithmic ideas. A within-subjects study with 12 participants showed significantly higher ratings across all shared understandability and usability items, high ratings for intervention support, and reduced cognitive workload. A system demonstration illustrated sustained expert steering across more than 200 nodes over 7 days. In future work, we plan to evaluate the approach with independent domain experts, larger populations, tasks in other fields, and stronger support for verifying generated explanations. More broadly, our work suggests that preserving expert agency in open-ended AI search requires both helping experts form informed judgments and allowing them to shape subsequent automation.

Acknowledgments

We used LLMs for proofreading and LaTeX programming but wrote and edited the text content of this paper manually. This study is supported by the Ministry of Education, Singapore, under Academic Research Funds (NTU Tier 1, RG104/25 and Tier 2, Proposal ID: T2EP202220049). Any opinions, findings, conclusions, or recommendations expressed are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

References

[1] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proc. ACM Program. Lang.* 7, OOPSLA1 (April 2023). doi:10.1145/3586030

[2] N. Boukhelifa, W. Cancino, A. Bezerianos, and E. Lutton. 2013. Evolutionary Visual Exploration: Evaluation With Expert Users. *Computer Graphics Forum* 32, 3pt1 (2013), 31–40. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12090 doi:10.1111/cgf.12090

[3] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101. arXiv:https://doi.org/10.1191/1478088706qp063oa doi:10.1191/1478088706qp063oa

[4] Shreya Chappidi, Jatinder Singh, and Andra V. Krauze. 2026. Who Does What? Archetypes of Roles Assigned to LLMs During Human-AI Decision-Making. arXiv:2602.11924 [cs.HC] https://arxiv.org/abs/2602.11924

[5] A. Chatzimparmpas, R. M. Martins, K. Kucher, and A. Kerren. 2021. VisEvol: Visual Analytics to Support Hyperparameter Search through Evolutionary Optimization. *Computer Graphics Forum* 40, 3 (2021), 201–214. doi:10.1111/cgf.14300 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14300.

[6] Norman Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin* 114, 3 (1993), 494–509. doi:10.1037/0033-2909.114.3.494

[7] Karen Daniels, Georges Grinstein, Adam Russell, and Mason Glidden. 2012. Properties of normalized radial visualizations. *Information Visualization* 11, 4 (Oct. 2012), 273–300. doi:10.1177/1473871612439357

[8] Mica R. Endsley. 1995. Toward a Theory of Situation Awareness in Dynamic Systems. *Human Factors* 37, 1 (1995), 32–64. doi:10.1518/001872095779049543 _eprint: https://doi.org/10.1518/001872095779049543.

[9] Bogdan Georgiev, Javier Gómez-Serrano, Terence Tao, and Adam Zsolt Wagner. 2025. Mathematical exploration and discovery at scale. https://arxiv.org/abs/2511.02864 _eprint: 2511.02864.

[10] Sandra G. Hart and Lowell E. Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In *Human Mental Workload*, Peter A. Hancock and Najmedin Meshkati (Eds.). Advances in Psychology, Vol. 52. North-Holland, 139–183. doi:10.1016/S0166-4115(08)62386-9

[11] Zhenan He and Gary G. Yen. 2016. Visualization and Performance Metric in Many-Objective Optimization. *IEEE Transactions on Evolutionary Computation* 20, 3 (2016), 386–402. doi:10.1109/TEVC.2015.2472283

[12] Jingmei Hu et al. 2021. Assuage: Assembly Synthesis Using A Guided Exploration. In *The 34th Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '21)*. Association for Computing Machinery, New York, NY, USA, 134–148. doi:10.1145/3472749.3474740

[13] Amin Ibrahim, Shahryar Rahnamayan, Miguel Vargas Martin, and Kalyanmoy Deb. 2016. 3D-RadVis: Visualization of Pareto front in many-objective optimization. In *Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC)* (Vancouver, BC, Canada). IEEE Press, 736–745. doi:10.1109/CEC.2016.7743865

[14] Alayt Issak, Jeba Rezwana, and Casper Harteveld. 2026. "Control Is a Trajectory, Not a Point": Conceptualizing Control in Human-AI Co-Creativity. doi:10.1145/3772318.3790861

[15] A. Kerren and T. Egger. 2005. EAVis: a visualization tool for evolutionary algorithms. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*. 299–301. doi:10.1109/VLHCC.2005.33

[16] Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. 2025. ShinkaEvolve: Towards Open-Ended And Sample-Efficient Program Evolution. https://arxiv.org/abs/2509.19349 _eprint: 2509.19349.

[17] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2024. A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3597503.3608128

[18] Fei Liu, Rui Zhang, Zhuoliang Xie, Rui Sun, Kai Li, Qinglong Hu, Ping Guo, Xi Lin, Xialiang Tong, Mingxuan Yuan, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. 2026. LLM4AD: A Platform for Algorithm Design with Large Language Model. https://arxiv.org/abs/2412.17287 _eprint: 2412.17287.

[19] Gang Liu, Yihan Zhu, Jie Chen, and Meng Jiang. 2025. Scientific Algorithm Discovery by Augmenting AlphaEvolve with Deep Research. arXiv:2510.06056 [cs.AI] https://arxiv.org/abs/2510.06056

[20] Yuxin Ma, Zherui Zhang, Ran Cheng, Yaochu Jin, and Kay Chen Tan. 2025. ParetoLens: A Visual Analytics Framework for Exploring Solution Sets of Multi-Objective Evolutionary Algorithms [Application Notes]. *IEEE Computational Intelligence Magazine* 20, 1 (2025), 78–94. doi:10.1109/MCI.2024.3487134

[21] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2024. Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI '24)*. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3613904.3641936

[22] Deepak Nagar, Palaniappan Ramu, and Kalyanmoy Deb. 2021. Interpretable Self-Organizing Maps (iSOM) for Visualization of Pareto Front in Multiple Objective Optimization. In *Evolutionary Multi-Criterion Optimization: 11th International Conference, EMO 2021, Shenzhen, China, March 28–31, 2021, Proceedings* (Shenzhen, China). Springer-Verlag, Berlin, Heidelberg, 645–655. doi:10.1007/978-3-030-72062-9_51

[23] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. 2025. AlphaEvolve: A coding agent for scientific and algorithmic discovery. https://arxiv.org/abs/2506.13131 _eprint: 2506.13131.

[24] Advait Pareek, Zijian Govers, et al. 2026. Sensemaking in Multi-Agent LLM Interfaces. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing*

Systems. Association for Computing Machinery. TODO: verify authors, DOI, and page numbers.
[25] 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. 2024. Mathematical discoveries from program search with large language models. *Nature* 625, 7995 (Jan. 2024), 468–475. doi:10.1038/s41586-023-06924-6
[26] Asankhaya Sharma. 2025. OpenEvolve: an open-source evolutionary coding agent. https://github.com/algorithmicsuperintelligence/openevolve
[27] Thomas B. Sheridan. 1992. *Telerobotics, automation, and human supervisory control*. MIT Press, Cambridge, MA, USA.
[28] Venkatesh Sivaraman, Eric Mason, Mengfan Li, Jessica Tong, Andrew King, Jeremy Kahn, and Adam Perer. 2026. Intelligent Reasoning Cues: A Framework and Case Study of the Roles of AI Information in Complex Decisions. doi:10.48550/arXiv.2602.00259
[29] Anjul Tyagi, Cong Xie, and Klaus Mueller. 2023. NAS-Navigator: Visual Steering for Explainable One-Shot Deep Neural Network Synthesis. *IEEE Transactions on Visualization and Computer Graphics* 29, 1 (2023), 299–309. doi:10.1109/TVCG.2022.3209361
[30] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems (CHI EA '22)*. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3491101.3519665
[31] Alex Vitvitskyi, Michael Boratko, Matej Grcic, Razvan Pascanu, Deep Shah, and Petar Veličković. 2026. Mining Generalizable Activation Functions. https://arxiv.org/abs/2602.05688 _eprint: 2602.05688.
[32] Rui Wang, Jeff Clune, and Kenneth O. Stanley. 2018. VINE: an open source interactive data visualization tool for neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (Kyoto, Japan) *(GECCO '18)*. Association for Computing Machinery, New York, NY, USA, 1562–1564. doi:10.1145/3205651.3208236
[33] A.S. Wu, K.A. De Jong, D.S. Burke, J.J. Grefenstette, and C. Loggia Ramsey. 1999. Visual analysis of evolutionary algorithms. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, Vol. 2. 1419–1425 Vol. 2. doi:10.1109/CEC.1999.782649
[34] Xiaoru Yuan, Donghao Ren, Zuchao Wang, and Cong Guo. 2013. Dimension Projection Matrix/Tree: Interactive Subspace Visual Exploration and Analysis of High Dimensional Data. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2625–2633. doi:10.1109/TVCG.2013.150
[35] Liangli Zhen, Miqing Li, Dezhong Peng, and Xin Yao. 2020. Objective reduction for visualising many-objective solution sets. *Inf. Sci.* 512, C (Feb. 2020), 278–294. doi:10.1016/j.ins.2019.04.014

A Formative Study Details

A.1 Participants

Table 1: Formative study participants. EA familiarity is self-reported during the interview: "None" indicates no prior exposure to EAs; "Basic" indicates familiarity with core concepts (e.g., population, mutation, selection) but no implementation experience. None had implemented or published EA-related work. All participants are daily users of LLM-based programming tools.

ID	Background	Domain	EA Famil.
P1†	Software Engineer	Database Optimization	Basic
P2	Software Engineer	Backend Optimization	None
P3	PhD Student	DL	None
P4†	PhD Student	Mathematics and DL	None
P5†	PhD Student	DL	None
P6†	Software Engineer	Backend Optimization	Basic
P7	MSc Student	DL	None
P8	MSc Student	DL	None

†Over 5 years of competitive programming experience.

A.2 Procedure

Each formative study session followed a seven-stage protocol. Sessions were conducted with audio and screen interactions recorded with consent.

Stage 1: Consent and warm-up (2 min). We obtained informed consent and established rapport.

Stage 2: Background questions (8 min). We asked participants about their research domain, their experience iteratively improving algorithms with LLM tools, how they manage multiple nodes, and how they evaluate node quality. These questions were designed to surface current workflow pain points before introducing the evolution concept.

Stage 3: Concept introduction (5 min). We introduced LLM-driven program evolution using a three-part visual handout covering the evolutionary loop (population, selection, LLM mutation, evaluation), the distinction between random and reasoned mutations, and the island model for maintaining diversity. We did not mention EvoMaestro or any specific design ideas at this stage.

Stage 4: System walkthrough and free exploration with think-aloud (15 min). After a brief orientation to the ShinkaEvolve baseline interface[3], participants freely explored a pre-run evolution result containing approximately 50 nodes across multiple islands, generated for a drone path optimization task with competing objectives (flight efficiency, noise mitigation, and signal stability). We asked participants to think aloud as they explored and used minimal prompts (e.g., "What are you looking at right now?" "What would you want to know that you can't see?") only when they fell silent for more than ten seconds.

Stage 5: Semi-structured follow-up (10 min). We asked targeted questions covering information usefulness, code comprehension difficulty, strategies for identifying promising nodes, understanding of the overall evolution trajectory, desire to intervene, opportunities for expert knowledge injection, and preferences for pacing and control.

Stage 6: Design probes (8 min). We showed two design probes framed as "ideas we are exploring": Probe A demonstrated information overload reduction (score-emphasis tree with performance color coding, noteworthy-node highlighting, and per-node strategy summaries); Probe B demonstrated expert steering (a suggestion dialog for NL-guided mutation and a merge panel for multi-parent recombination). We collected reactions, suggestions, and concerns for each probe.

Stage 7: Wrap-up (2 min). We asked participants to summarize in one sentence what they found most difficult about understanding the evolution run and invited any remaining thoughts.

B Design and System Details

B.1 Radial Layout Design

Design rationale. Evolution histories usually begin with a few seed nodes and expand as descendants accumulate. The radial tree places early generations near the center and later generations on larger circumferences, giving denser outer generations more space. In contrast, a top-down tree uses a rectangular band at each depth, leaving much of the band unused near sparse early generations while later

[3]Available at https://shinka-mop.reify.ing

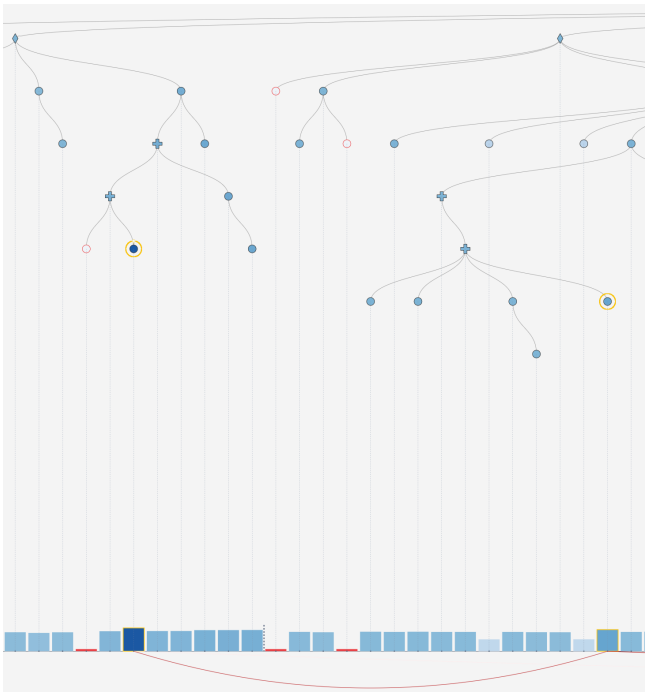

Figure 5: Alternative tree-and-bar design considered during design iteration. Each node has a unique horizontal position, and a dotted guide links it to an aligned score bar below the tree. Arcs connect node pairs whose dissimilarity exceeds a threshold. As the population grew, the lineage tree, score bars, and arcs competed for horizontal space.

branching histories compete for the same width. The radial layout also maps the evolution structure directly: radius encodes generation depth, angular sectors group islands, and edges trace lineage. EvoMaestro also provides a sortable node table for exact score and metadata lookup, while the radial tree serves as the process view for generation, lineage, island context, and intervention history. Exact angular positions within a sector are layout-driven and carry no data meaning. Euclidean distance is not designed to encode strategy similarity; only its radial component reflects generation difference.

Design alternative. We also implemented a top-down tree in which each node had a unique horizontal coordinate, with an aligned score bar below it and arcs between bars whose node dissimilarity exceeded a threshold (Fig. 5). The threshold limited the visible arcs instead of showing links between every pair. During design iteration, E1 found the tree difficult to read when the population exceeded approximately 50 nodes because the tree, bars, and arcs competed for horizontal space. We therefore selected the radial tree for the population overview.

B.2 Merge Recommendation

Given a node s selected for *Merge*, EvoMaestro first excludes incorrect, banned, and already selected nodes from the candidate partners. For each remaining candidate c, it computes a quality value $q(c)$ from the program's evaluation score and a dissimilarity value $d(s, c) = 1 - \cos(\mathbf{e}_s, \mathbf{e}_c)$ from the selected embedding source. Experts can use embeddings of either the source code or the generated reasoning, depending on whether they want to compare implementations or algorithmic strategies. The quality and dissimilarity values are separately min–max normalized to $\hat{q}(c)$ and $\hat{d}(s, c)$. Candidates from a different island receive a small dissimilarity preference of 0.1, and candidates retained in the evolution archive receive a small quality preference of 0.05; both normalized values remain capped at 1. The archive is a bounded set of correct programs retained by the evolution engine for subsequent parent and inspiration selection. The final recommendation score is

$$r(c) = \lambda \hat{d}(s, c) + (1 - \lambda)\hat{q}(c), \tag{1}$$

where experts can adjust λ to balance dissimilarity and quality; the default is 0.6. EvoMaestro presents the five highest-ranked candidates. Candidates without the selected embedding are assigned a dissimilarity value of 0.5 before normalization; when no usable embeddings are available, the method ranks candidates by quality alone.

This ranking does not determine whether two strategies are semantically complementary. It narrows the candidate set to programs that combine strong evaluation results with different implementations or algorithmic strategies. The expert then inspects their scores, strategy similarity, and code differences before selecting a partner, and can provide optional natural-language guidance about which ideas the merged child should preserve or combine.

C Task Details: OXC Routing Optimization

This appendix provides an overview of the task and the full mathematical formulation of the OXC routing optimization task used in Section 6.

C.1 Task Overview

Adapted from the ICPC 2025 Online Winter Challenge[4], the task aims to optimize traffic routing in a large-scale AI computing cluster interconnected by OXCs. The goal is to evolve programs that jointly determine (1) the physical topology configuration of OXCs and (2) traffic routing strategies over the configured topology. Each program (evolution node) represents a candidate routing policy and is evaluated using a metric balancing two competing objectives: (1) minimizing peak link congestion and (2) minimizing the physical reconfiguration cost by OXCs switching. Each program must complete execution within 50s during evaluation. The search space is highly complex, as it involves combinatorial optimization over both network topology and traffic assignment. Effective solutions require coordinating discrete topology decisions with continuous traffic distribution under performance constraints. The task is challenging, as the program must jointly optimize topology configuration and traffic routing across a large-scale network, requiring strong algorithmic design skills to navigate the complex search space effectively.

C.2 Mathematical Formulation

Network architecture. The multi-plane network comprises N Groups, M OXCs, S Spines per Group, and L Leaves per Group. To

[4]https://icpc.global/regionals/finder/Winter-Challenge-2026

ensure scalability, the network is partitioned into P independent planes, with K parallel links between Spines and OXCs within the same plane. Cross-plane links are strictly prohibited. The total number of ports per OXC, R, is:

$$R = N \cdot (S/P) \cdot K$$

A specific port on an OXC is uniquely mapped by Group i, Spine j, and link k as:

$$Idx_{port} = i \cdot (S/P) \cdot K + (j \bmod S/P) \cdot K + k$$

A valid routing path for real-time queries must symmetrically traverse the network: $LeafA \leftrightarrow SpineA \leftrightarrow OXC \leftrightarrow SpineB \leftrightarrow LeafB$.

Dual optimization objectives. Under strict temporal execution budgets, the system must dynamically balance two competing objectives:

(1) **Minimize Maximum Flow Conflict:** Alleviate network bottlenecks by minimizing the peak active flows $C(e)$ passing through any physical link $e \in E$:

$$\min\left(\max_{e \in E} C(e)\right)$$

(2) **Minimize OXC Adjustment Cost:** Reduce the time penalty of physically rotating micro-mirrors, measured by the edit distance between successive port connection states T_{t-1} and T_t:

$$\min D(T_t, T_{t-1}) = |T_{t-1} \setminus T_t| + |T_t \setminus T_{t-1}|$$

D Evaluation Study Details

D.1 Participants

Table 2: Evaluation study participants. LLM-Evo Exp. indicates self-reported experience with LLM-based evolutionary systems: N = None, B = Basic (understand concepts or short-term use), M = Medium (used similar systems with extended time), E = Expert (developing or researching). All participants are daily users of LLM-based programming tools.

ID	Background	Domain	LLM-Evo Exp.
U1	MSc Student	Physics	B
U2	PhD Student	DL	N
U3	Software Engineer	Backend Optimization	B
U4	MSc Student	HCI	B
U5	MSc Student	Statistics	B
U6	PhD Student	DL	B
U7	MSc Student	DL	B
U8	MSc Student	DL	M
U9	Software Engineer	Database Optimization	B
U10	MSc Student	DL	B
U11	MSc Student	DL	N
U12	MSc Student	Mechanical Engineering	B

D.2 Tasks

Task A: Drone Multi-Objective Path Optimization (MOP). A drone must fly from a fixed start point to an end point through a series of intermediate waypoints. The evolvable function outputs waypoint coordinates that balance three competing objectives: flight path length, noise penalty from proximity to schools, and signal degradation from distance to base stations.

Task B: Server Routing on Undirected Graphs (GraphMOP). In an edge computing network modeled as an undirected graph, tasks generated at terminal nodes must be routed to server nodes for processing. The evolvable function determines server assignments and routing paths, balancing two competing objectives: end-to-end latency (transmission delay plus queuing delay) and system energy consumption.

Both tasks were chosen because they involve multi-objective optimization with non-trivial trade-offs, require domain reasoning to interpret nodes (not just numerical scores), and produce evolution trees with diverse algorithmic strategies amenable to expert analysis. Each task[5,6] was seeded with a pre-run evolution containing approximately 50 nodes across 6 islands. In the EvoMaestro condition, the evolution remained live, allowing participants to execute interventions and observe new nodes in real time.

D.3 Counterbalancing

To control for ordering and task effects, we counterbalanced the assignment of conditions and tasks (§ D.2) using a 2×2 Latin square design:

Table 3: Counterbalanced assignment of conditions and tasks.

Group	Session 1	Session 2
G1 ($n = 3$)	Baseline + Task A	EvoMaestro + Task B
G2 ($n = 3$)	EvoMaestro + Task A	Baseline + Task B
G3 ($n = 3$)	Baseline + Task B	EvoMaestro + Task A
G4 ($n = 3$)	EvoMaestro + Task B	Baseline + Task A

D.4 Measures

We collected four types of data after each condition:

D.4.1 Custom Likert Questionnaire (7-point scale). The following items were shared across both conditions (1 = Strongly Disagree, 7 = Strongly Agree):

Understandability (5 items, both conditions).

- **L1** (DR3) The system's visualization design allows me to clearly understand how the entire code population evolves over generations.
- **L2** (DR1) The system allows me to easily discern the overall algorithmic strategies adopted by the LLM across different branches.
- **L3** (DR4) The system helps me easily locate code that exhibits abnormal or special performance.
- **L4** (DR2) The system enables me to effectively compare different solutions and understand the differences between them.

[5]Web link for ShinkaEvolve Task A and Web link for ShinkaEvolve Task B
[6]Web link for EvoMaestro Task A and Web link for EvoMaestro Task B

L5 (DR1) The system allows me to clearly see the algorithm used by each node.

Usability (3 items, both conditions).

L6 The system is easy to use for me.
L7 The information provided by the system is organized very clearly on the screen, and I can easily find what I need.
L8 The system information effectively helps me complete my tasks and operational scenarios.

The following items were collected only in the EvoMaestro condition:

Maestro Chat (1 item, EvoMaestro only).

L9 AI-assisted code explanations (like Maestro Chat) helped me understand solutions that would be difficult for me to grasp on my own.

Intervenability (3 items, EvoMaestro only).

L10 (DR5) The system enables me to accurately identify where intervention is needed in the current evolutionary tree.
L11 (DR6) The system allows me to flexibly control the evolution pace, pausing to observe or stepping through as needed.
L12 (DR5) The system effectively supports me in conducting effective interventions within the evolutionary tree.

D.4.2 NASA-TLX. The NASA Task Load Index [10] was administered after each condition (both conditions, 7-point scale where 1 = Very Low and 7 = Very High; for Performance, 1 = Good and 7 = Poor):

(1) Mental Demand: How mentally demanding was the task?
(2) Physical Demand: How physically demanding was the task?
(3) Temporal Demand: How hurried or rushed was the pace of the task?
(4) Performance: How successful were you in accomplishing what you were asked to do?
(5) Effort: How hard did you have to work to accomplish your level of performance?
(6) Frustration: How insecure, discouraged, irritated, stressed, and annoyed were you?

D.4.3 Open-ended Questions. The following questions were asked in both conditions:

O1 Please describe in detail how you identified the well-performing branch from the evolutionary tree or scatter plot. Which visual cues provided the most critical guidance?
O2 How did you identify the first critical code variant that triggered the issue in this branch?
O3 How did you understand the algorithm logic of code variants through the interface?
O4 What was the most cognitively demanding part of the experiment?
O5 At what moment did the system make you feel like you had a clear global view?
O6 In this human-AI co-evolution process, do you feel your sense of control over the algorithmic direction was strengthened or weakened?

The following questions were asked only in the EvoMaestro condition:

O7 Please explain the design rationale behind your intervention prompt. Based on which specific bottleneck or logical flaw did you decide to issue this instruction to the LLM?
O8 Did you use Maestro Chat to understand code or code differences? If so, describe a specific moment when it helped. If not, why not?

D.4.4 Post-study Interview. A semi-structured interview after both conditions captured preference comparisons, feature-level feedback, and reflections on how each tool shaped understanding.

D.5 Procedure

Each evaluation session lasted approximately 80 minutes and followed this protocol:

(1) **Introduction and consent (5 min).** We explained the study goals and obtained informed consent.
(2) **Tutorial on LLM-driven evolution (5 min).** Using the same visual handout from the formative study (Appendix A.2), we introduced the evolutionary loop, reasoned mutations, and the island model.
(3) **Condition 1 — Tool training (5 min).** We demonstrated the assigned interface's key features.
(4) **Condition 1 — Tasks (25 min).** Participants explored the evolution result following a structured task sheet (in the EvoMaestro condition, the evolution was live, allowing real-time interventions):
   - *Sensemaking tasks (15 min):* Participants were asked to identify the three most promising nodes and explain why, describe the major algorithmic strategies that emerged during evolution, identify any nodes exploiting evaluation function weaknesses, and find a low-scoring node with a promising idea worth developing further.
   - *Intervention tasks (10 min):* In the EvoMaestro condition, participants selected a node and wrote a NL suggestion to guide the next mutation, then selected 2–3 nodes and initiated a multi-parent merge. In the baseline condition, participants instead wrote down what modifications they would make and which nodes they would combine, capturing intervention *intent* without tool support for qualitative comparison.
(5) **Condition 1 — Questionnaire (5 min).** Participants completed the custom Likert questionnaire (Understandability, Usability, and, for EvoMaestro, Intervenability and Maestro Chat items) and open-ended questions, followed by the NASA-TLX.
(6) **Break (5 min).**
(7) **Condition 2 (35 min).** Steps 3–5 repeated with the other interface and task.
(8) **Post-study interview (10 min).** A semi-structured interview covering preference comparison between the two interfaces, feature-level feedback on EvoMaestro, suggestions for missing features, and how participants' understanding of the evolution differed between conditions.

D.6 Detailed Results

Table 4: Paired comparison of shared Likert items (7-point scale; higher is better). All differences are significant at $p < 0.05$ (Wilcoxon signed-rank test). Effect sizes: $|\delta| < 0.33$ = small, < 0.474 = medium, ≥ 0.474 = large.

Item	Ours Med	Base Med	p	δ
Understandability				
Population evolution (DR3)	6.0	5.0	.016	$.451^{m}$
Branch strategies (DR1)	6.0	5.0	.039	$.403^{m}$
Locate abnormal (DR4)	7.0	5.0	.023	$.521^{L}$
Compare solutions/nodes (DR2)	6.0	5.0	.020	$.486^{L}$
Node algorithm (DR1)	6.0	5.0	.031	$.368^{m}$
Usability				
Easy to use	6.0	4.5	.027	$.472^{m}$
Info organized	7.0	5.0	.016	$.639^{L}$
Task effective	7.0	5.0	.002	$.694^{L}$

^{m}Medium effect. ^{L}Large effect.

Table 5: NASA-TLX results (7-point scale; lower is better). Significant differences are marked with *. The Performance subscale measures participants' perceived success in completing the study task, not generated-program performance.

Subscale	Ours Med	Base Med	p	δ
Mental Demand	3.5	5.5	$.043^{*}$	$-.347^{m}$
Physical Demand	2.0	3.0	.070	$-.354^{m}$
Temporal Demand	3.0	4.5	$.016^{*}$	$-.479^{L}$
Performance	3.0	3.0	.992	.056
Effort	3.5	3.5	.254	−.174
Frustration	2.0	4.0	$.004^{*}$	$-.535^{L}$
Overall	**3.08**	**4.25**	$\mathbf{.004^{*}}$	$\mathbf{-.417^{m}}$

^{m}Medium effect. ^{L}Large effect.