PAPER AGENT NETWORK: LITERATURE-GROUNDED MULTI-AGENT PROGRAM EVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Using a research paper in program search requires turning its methods into executable candidates and adapting them to a target task. A paper can initialise an approach whose implementation continues to develop through evaluation and findings from other approaches. We introduce the Paper Agent Network (PAN), a literature-grounded multi-agent system that organises this process around persistent coding agents. Each agent starts from a selected paper and develops a revisable method route through tool use, evaluator feedback, and accumulated implementation history. A shared result board and queued reports connect these routes; a progress-based controller allocates work and introduces reserve agents. The design separates admitting an improved artifact from sharing a finding that may inform later work. We evaluate the complete system on 23 tasks spanning mathematics, systems, engineering, planning, scientific code, and prediction against four open program-evolution frameworks. The reported comparison contains twelve best or tied-best results. Two process analyses make method and information use concrete. A denoising construction appears in executable artifacts, and a peer's scheduling report motivates a change to a route that improves through later work. The 17-task literature comparison shows that the positive denoising case does not imply a general benefit from literature access. These observations establish method and information use within the implemented system; interaction's independent performance contribution remains unresolved. Persistent paper-informed routes provide a concrete setting for studying how published methods are implemented, exchanged, and adapted during computational problem solving.

1 INTRODUCTION

Language models coupled with executable evaluators have produced useful mathematical constructions and algorithmic improvements through iterative program search (Romera-Paredes et al., 2024; Novikov et al., 2025). Open frameworks support this search through population diversity, candidate reuse, and adaptive allocation (Lange et al., 2025; Assumpção et al., 2025; Cemri et al., 2026). We study how to organise this process around research papers, so that paper-informed approaches can accumulate implementation experience and exchange findings under a shared evaluator.

Research papers offer methodological starting points through their representations, assumptions, and optimisation strategies. We treat each paper as an initial *methodological prior* that conditions an agent's proposals. A persistent agent develops this direction through accumulated implementations, failures, and evaluator feedback. Different papers can encourage different proposal distributions, although methods may overlap or prove unsuitable. An initially weak implementation need not exhaust an approach's potential, so the agent retains its experience while adapting or departing from its starting method. A paper thus supplies an implementable hypothesis about the task; continued evaluation determines which parts of that hypothesis remain useful.

Paper-based tools and multi-agent research provide complementary foundations for this organisation. Paper2Agent constructs interactive agents around papers and associated code, including joint analysis with method and data resources from different papers (Miao et al., 2025). Other multi-agent systems study hypothesis generation and experimental collaboration (Ghafarollahi & Buehler, 2024; Su et al., 2024; Lyu et al., 2026). Our focus is the development of multiple paper-informed approaches through repeated implementation under one evaluator. Each approach needs a continu-

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

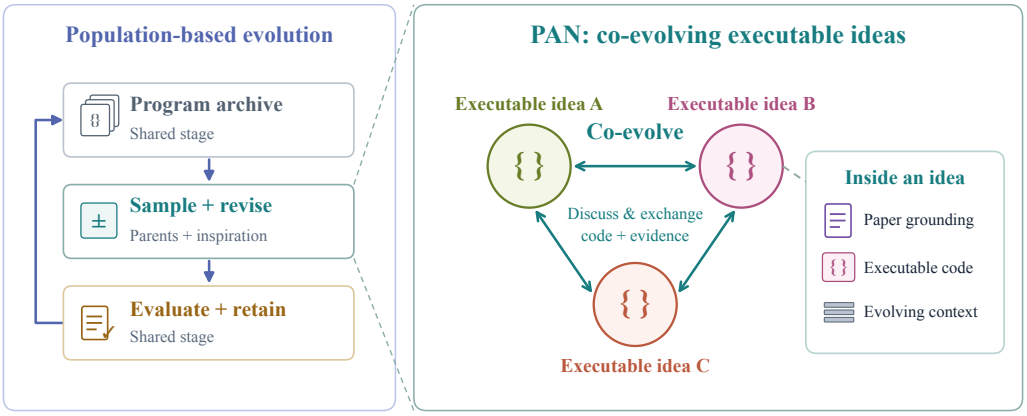

Figure 1: **Program evolution and paper-informed routes. Left:** a schematic of population-based evolution; existing frameworks support diversity and reuse (Sharma & contributors, 2025; Lange et al., 2025; Cemri et al., 2026). **Right:** papers initialise approaches that agents implement, evaluate, and revise within persistent contexts. Reports and reference programs connect these continuing routes. "Discuss" denotes queued report exchange; pairwise dialogue was disabled.

ing history, while findings need to remain available across approaches. A report can inform another route without transferring an entire program. The receiving agent can adapt the finding to its own approach and retain the result in its continuing implementation history.

We introduce the *Paper Agent Network* (PAN) to coordinate these persistent paper-informed routes (Figure 1). Each agent develops an approach through tool use and evaluation. A shared result board connects routes through tested artifacts and findings, and a synchronous controller allocates work using progress and introduces reserve paper agents. The design separates selecting a better artifact from sharing a useful finding. Board admission requires sufficient score improvement; a report can still reach another route when its artifact is not admitted. Delayed initial sharing gives routes time for independent work, while reference programs support subsequent reuse. These choices organise the development and exchange of paper-informed methods. Diversity depends on the programs that routes develop, so distinct paper identities alone cannot establish diverse implementations.

We evaluate the complete system against four open program-evolution frameworks on 23 tasks, covering mathematical, systems, engineering, planning, scientific-code, and prediction objectives. PAN obtains twelve best or tied-best results in the reported comparison, with performance varying across tasks. The comparison measures outcomes of the adapted systems. A 17-task literature comparison further examines what paper access contributes under the recorded configurations. Its near-neutral aggregate accompanies both a positive denoising case and tasks where the full-paper configuration performs worse, making compatibility between a source method and the target task central to interpreting the results. Relating the two comparisons is essential to interpreting the paper-agent idea. A system can perform strongly against competing frameworks while gaining little from its paper inputs. We therefore examine which successful constructions contain identifiable literature content and which strengths persist when paper instructions and retrieval tools are absent.

Two process cases connect these outcomes to the development of individual routes. A denoising paper supplies an identifiable matrix construction that appears in executable artifacts. In transaction scheduling, a peer report motivates a change whose first implementation remains weaker than the sender's; the receiving route subsequently improves during continued search. The latter observation gives report exchange a concrete role in route development, although it does not isolate exchange's effect on performance. Together, the cases show why a paper-agent study needs to trace both the methods entering programs and the use of information over time.

2 Related work

Multi-agent scientific research. Multi-agent scientific systems distribute research activities across interacting roles. SciAgents combines literature-derived knowledge graphs with multi-agent reasoning to generate materials research hypotheses (Ghafarollahi & Buehler, 2024), while VirSci studies collaborative scientific idea generation (Su et al., 2024). EvoScientist develops research and experimental agents with persistent memory and evolving strategies (Lyu et al., 2026). The AI Scientist automates a broader pipeline from ideas through experiments and manuscript generation (Lu et al., 2024). PAN focuses on continuing implementation routes for one executable task. Artifacts and evaluator scores provide a common interface across initial methods.

Evaluator-guided program evolution. Evaluator-guided evolution develops candidate programs through measured feedback and reuse. FunSearch couples language-model proposals with program evaluation and a database of successful programs (Romera-Paredes et al., 2024). AlphaEvolve extends evolutionary coding to mathematical problems and computational infrastructure (Novikov et al., 2025); OpenEvolve provides an open implementation of this paradigm (Sharma & contributors, 2025). ShinkaEvolve combines parent sampling, novelty rejection, and bandit-based model selection (Lange et al., 2025). CodeEvolve uses island populations, inspiration-based crossover, and meta-prompting (Assumpção et al., 2025). These methods already support diversity and reuse. PAN makes a persistent, paper-conditioned agent the unit of continuing work.

Adaptive allocation of search. Adaptive allocation directs further computation using the progress of search populations. AdaEvolve uses accumulated improvement to adapt local exploration, allocate work across populations, and introduce new tactics during stagnation (Cemri et al., 2026). CostAda additionally conditions progress credit on realised token cost and remaining budget (Zhang et al., 2026). PAN's local/global gain mixture and logarithmic cost penalty are adapted from CostAda. The unit being routed is a persistent agent initially associated with a paper, and plateau and retirement rules govern changes to the active team. This inherited allocation rule directs work across the evolving paper-informed routes.

Literature access and research implementation. Literature tools support evidence synthesis, method access, and research implementation. PaperQA2 studies literature retrieval, synthesis, and contradiction detection (Skarlinski et al., 2024). LMR-Bench tests paper-to-code reproduction through masked functions in research repositories (Yan et al., 2025). Paper2Agent converts papers and associated code into tested tools exposed through the Model Context Protocol (MCP). Its joint AlphaGenome and ADHD GWAS example connects method and data MCPs to a downstream co-scientist for cross-paper analysis (Miao et al., 2025). PAN uses paper text to initialise multiple persistent coding routes that repeatedly revise artifacts under one objective and exchange reports through a controller. ADRS motivates automated systems research through executable verifiers (Cheng et al., 2025); AutoLab evaluates sustained research and engineering under wall-clock budgets (Xu et al., 2026). We use subsets of these tasks under a different budget protocol. The resulting scores assess the implemented network under our candidate-cap protocol; direct comparison with their published leaderboards would also require matching the resource conditions.

3 Paper Agent Network

PAN coordinates persistent paper-conditioned processes through a shared board and queued relays (Figures 1 and 2). A synchronous controller schedules work and changes membership using measured progress. We use *paper* for a source of methods, *agent* for the persistent tool-using process, *route* for its evolving approach, and *report* for an attempt's method and measured outcome. A route can change without changing the agent's paper identifier. In each work turn, a route receives pending findings and revises its artifacts. Evaluation then supplies scores and reports for decisions about future work, board admission, and exchange.

3.1 Paper-conditioned proposal processes

A persistent agent carries both an initial method and the experience acquired while developing it. To describe how these sources of information enter candidate generation, let τ denote a task, p_i the paper assigned to agent i, $h_{i,t}$ its local implementation and interaction history, and B_t the shared

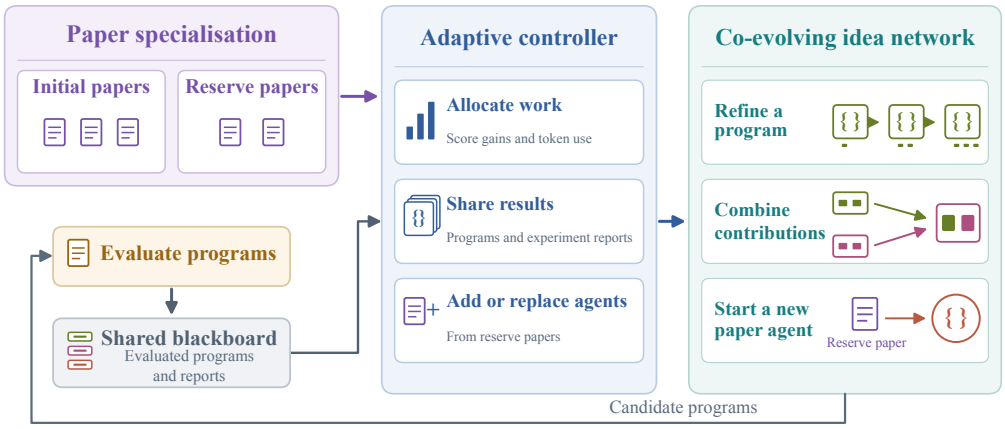

Figure 2: **Implementation of the paper-agent network.** The controller schedules persistent routes, receives evaluated artifacts and reports, and introduces reserve agents. Board admission retains sufficiently improved artifacts; queued relays can also deliver findings from non-admitted attempts. Arrows show the available information paths, including selected reference programs.

evidence available at time t. We interpret an agent's candidate generation as an implicit proposal distribution $q_\theta(x \mid \tau, p_i, h_{i,t}, B_t)$. A conceptual description of the network is

$$q_t(x) = \sum_{i \in \mathcal{A}_t} w_{i,t}\, q_\theta(x \mid \tau, p_i, h_{i,t}, B_t), \qquad w_{i,t} \geq 0, \quad \sum_i w_{i,t} = 1, \tag{1}$$

where $\mathcal{A}_t$ is the active team. Weights represent allocation of proposal opportunities; an individual deterministic scheduling decision gives a one-hot allocation. The implementation estimates neither densities nor mixture weights. Papers supply initial priors, histories develop individual routes, and shared evidence couples subsequent proposals.

3.2 Task specification and paper-based specialisation

A task supplies a seed artifact x_0, an executable evaluator f, an artifact interface, and a nominal evaluation cap B. We write the search objective as $\max_{1 \leq t \leq B} f(x_t)$ over recorded candidate evaluations. For minimisation tasks, f denotes the sign-reversed objective in this notation; tables retain the original metric and direction.

The corpus layer retrieves candidates from alphaXiv and, for broader coverage, OpenAlex and Semantic Scholar. Topic and term overlap filter the pool before a model selects an initial team of three papers and up to three reserves. Pools and selections are cached per task. A cached index describes each paper's methods, parameters, assumptions, and page references. An agent can then request the complete relevant pages. Built without the task statement, the index is reusable across tasks.

3.3 Agent state, evaluated reports, and exchange

An agent retains its paper, transcript, working artifacts, and evaluation history. Its tools support shell execution, file writing, SEARCH/REPLACE edits, and corpus reads. Candidates can be programs or generated data. Handover notes and tool-output elision compress contexts. The adapter supports separate workspaces or a shared run directory, without security sandboxes.

A scheduled agent implements or revises an artifact and submits it to the scoring wrapper, which records evaluator output and freezes the artifact under a content digest. The adapter obtains the score and artifact path from this record. A report associates the measurement with the assigned paper, proposed method, and findings. This ties a reported score to an evaluated artifact; it does not verify the explanation or attribution to the paper.

Evaluation, board admission, and relay are separate operations. A valid report enters the board when its score exceeds the current bar by the task-specific threshold δ. Smaller improvements can

still update routing progress, and non-admitted reports can inform later work through relays. Thus an attempt need not produce a new best program to contribute information to another route. For example, a candidate below the global bar can improve its agent's local best, affect routing credit, and supply a method report to other routes.

The first two rounds relay the score bar without other agents' method descriptions. Later relays queue the latest score and method summary for the next scheduled work turn, avoiding an extra turn per broadcast. For stalled or converging routes, the controller also selects reference programs, filtering near-duplicate source text. A newcomer receives a summary of prior attempts and the best available program. These are the evaluated interactions.

3.4 Work allocation and network membership

Let s be an agent's latest valid score, b_i its previous best, and b the previous global bar. The controller measures nonnegative normalised improvement as

$$g(s, z) = \frac{\max(s - z, 0)}{\max(|s|, |z|, 1)}. \tag{2}$$

An uninitialised best receives zero gain. Following CostAda (Zhang et al., 2026), let ρ be the fraction of the configured *token* budget remaining, c the turn's recorded token cost, and c_0 a reference cost. The implemented utility combines these quantities as

$$u_i = \frac{(1 - \rho)g(s, b) + \rho g(s, b_i)}{1 + (1 - \rho)\log(1 + c/c_0)}. \tag{3}$$

An exponential moving average with coefficient 0.9 smooths this signal. Thus local improvement receives more weight early, while global improvement and cost receive more weight as the token budget drains. The candidate cap and token budget are separate controls.

Routes are ranked by smoothed utility and then by best score, with limits on consecutive retasks. Five reports without admission trigger a reserve opportunity; a separate trigger attempts to admit a newcomer when at most 35% of the evaluation budget remains. A reserve fills a vacancy or the one extra slot, otherwise replacing a weak route while protecting the leader when possible. Without a suitable reserve, an existing route continues. Source-text overlap after removing seed lines supplies a heuristic similarity signal, not a semantic measure of algorithmic diversity.

Appendices B and C document operation roles and controller settings, including defaults, membership rules, and the configuration details available for reconstructing the search process.

4 Experimental protocol

Tasks and comparison systems. We assess whether the complete system develops effective artifacts across varied executable objectives. The 23-task evaluation comprises six mathematical and signal-processing tasks and five ADRS systems tasks from the AdaEvolve/CostAda/AlphaEvolve collection, six AutoLab engineering tasks, TravelPlanner, the Ewald problem from SciCode, and four prediction tasks from MLE-Bench (Cemri et al., 2026; Zhang et al., 2026; Cheng et al., 2025; Xu et al., 2026; Xie et al., 2024; Tian et al., 2024; Chan et al., 2025). GPU-timed tasks are excluded because shared accelerators prevented reliable timing. AutoLab CPU timings use host-calibrated anchors. We compare with ShinkaEvolve, CodeEvolve, AdaEvolve, and OpenEvolve (Lange et al., 2025; Assumpção et al., 2025; Cemri et al., 2026; Sharma & contributors, 2025) through adapters to the same task evaluators. The tasks use the recorded model identifier `gpt-5.6-luna` for candidate generation. Baseline search code uses its default settings subject to model compatibility, evaluator adaptation, and the cap; Appendix B.2 lists the changes.

Budget, replication, and comparison scope. The comparison uses a nominal cap of 30 scored candidates per task. We report means of per-run best scores, with three replicates for PAN and two per baseline (Table 1). Thus n counts search runs, not the number of benchmark instances. These small samples support descriptive comparisons. The cap counts wrapper evaluations; tools, token use, and offline computation are not equalised.

Metrics. For system m and task j, let S_{mj} denote the reported mean task score. We report the mean normalised score, average rank, and best-or-tied-best task counts. Normalisation uses

Table 1: **Task-level program-search outcomes.** PAN achieves twelve best or tied-best results across 23 tasks. Entries are mean per-run best scores under the nominal 30-evaluation cap, with three runs for PAN and two per baseline. Lower is better for ↓; all other metrics are maximised. Bold marks ties within 10^{-6} relative tolerance before rounding. Cloudcast is scaled by 10^3.

Task	PAN	ShinkaEvolve	CodeEvolve	AdaEvolve	OpenEvolve
Mathematical construction					
Heilbronn (convex)	**0.0284**	0.0206	0.0272	0.0225	0.0282
Circle packing (square)	2.6293	2.6315	**2.6331**	2.6272	2.6277
Circle packing (rect.)	**2.3622**	2.3558	2.3482	2.3586	2.3570
Min–max distance	**0.0776**	0.0769	0.0761	0.0773	0.0733
Signal processing	**0.728**	0.675	0.656	0.689	0.658
Heilbronn (triangle)	**0.0342**	0.0305	0.0337	0.0332	0.0323
Systems research (ADRS)					
Prism (model placement)	26.239	26.225	**26.256**	26.238	26.197
EPLB (MoE balancing)	**0.140**	0.129	0.129	0.137	0.129
LLM–SQL (prefix cache)	**0.7264**	0.7260	0.7209	0.7072	0.7208
Transaction scheduling	3718	3701	4071	**4118**	3743
Cloudcast (broadcast)	1.6037	1.5921	1.6036	**1.6153**	1.5940
Engineering optimisation (AutoLab)					
Sorting network	**1.000**	0.850	0.850	0.850	0.850
Adversarial splay	**0.527**	0.051	0.009	0.008	0.006
Levenshtein (CPU-timed)	0.477	0.473	0.457	**0.639**	0.480
BVH ray tracing (CPU-timed)	0.387	0.404	0.466	**0.472**	0.409
Adaptive compression	0.379	0.478	0.369	**0.481**	0.379
ResNet bit-flip attack	0.983	0.981	**0.988**	0.975	0.906
Planning and scientific code					
TravelPlanner, final pass rate ↑	**0.797**	0.428	0.361	0.350	0.372
SciCode Ewald, normalised step score ↑	**0.955**	0.636	0.545	0.727	0.909
Prediction from data					
Pizza requests, AUC ↑	0.704	0.691	0.717	**0.726**	0.708
Spooky author, log loss ↓	0.374	**0.370**	0.410	0.418	0.419
Nomad2018, RMSLE ↓	**0.0538**	0.0613	0.0609	0.0602	0.0610
Denoising documents, RMSE ↓	0.0422	0.0557	0.0621	**0.0214**	0.0313

$S_{mj}/\max_{m'} S_{m'j}$ for higher-is-better metrics and $\min_{m'} S_{m'j}/S_{mj}$ for lower-is-better metrics. The latter applies to log loss, RMSLE, and RMSE; AUC and pass scores are maximised. Aggregates weight the 23 tasks equally. SciCode uses the normalised step score for the Ewald problem, not performance on the full benchmark. Scores within relative tolerance 10^{-6} tie and share the better rank. The normalised measure depends on the included systems and is not invariant to additive changes of a task's reward scale.

Process and configuration analyses. Two cases trace paper methods and peer findings into program development. A 17-task comparison examines literature access, and seven selected continuations examine further search. These analyses characterise system behaviour; they do not isolate the performance contributions of persistence, report exchange, or allocation.

5 Results

5.1 Overall system performance

PAN leads the aggregate, with uneven strengths across task families. Across the reported 23-task summary, PAN has the best average rank (2.00) and normalised score (0.944), with twelve best or tied-best results. Table 1 gives all task scores and Appendix A the aggregate comparison. Performance varies across tasks. PAN leads five of six mathematical and signal-processing tasks, two of five systems tasks, two of six engineering tasks, both planning and scientific-code tasks, and one of four prediction tasks. Its engineering wins occur on sorting networks and adversarial splay, both non-program artifacts. This distribution motivates examining task-level strengths and artifact interfaces alongside the aggregate comparison of complete systems.

The aggregate lead remains after excluding two interface outliers. On adversarial splay the output is exactly 4096 integers. PAN can write and execute a generator that emits the whole sequence, whereas baseline edits operate directly on the artifact. Development traces show CodeEvolve constructing an alternating-extremes sequence that yields little improvement; 13 of 20 inspected ShinkaEvolve patches violated the required length. This large gap includes an action-space advantage and is not evidence for a communication benefit. Sorting networks likewise use a non-program artifact. Removing these two tasks leaves 21 tasks, on which PAN retains both the best average rank (2.10 versus AdaEvolve's 2.52) and the highest normalised score (0.938 versus 0.934). The small remaining normalised-score margin makes individual task outcomes important for understanding the scope of PAN's aggregate advantage.

5.2 Method and information use during search

The denoising paper supplies an executable construction. In signal denoising, full-paper artifacts implement the selected paper's partial-circulant embedding, singular-value decomposition, and low-rank reconstruction (Chen et al., 2019). The construction maps a noisy one-dimensional signal to a matrix and recovers the signal from a low-rank approximation. The identifiable transfer includes both an input representation and a reconstruction procedure, giving the agent concrete operations to implement, test, and adapt to the evaluator. Across three full-paper runs, scores range from 0.7277 to 0.7292; the two no-paper runs range from 0.6006 to 0.6594. The approximately 15.6% increase in mean score accompanies this construction in the artifacts. Selection and information access also differ, so the increase cannot be assigned solely to reading the paper. The artifact content identifies the method used, while the configuration comparison measures the resulting task performance.

Peer information enters a route before it takes the lead. In transaction-scheduling continuation replicate 1, the conflict-based-search agent reports 3703.70 at evaluation 19. After receiving that report, the coupled-task-scheduling agent adds pair-block relocation and scores 3436.43 at evaluation 20. Its transcript identifies the peer report as motivation, without code copying. The receiving route later takes the lead at evaluation 37 and reaches 3968.25 at evaluation 56. The receiving program is initially weaker than the sender's, even though its route has incorporated the reported idea. Judging report use only by whether the receiver immediately outperforms the sender would miss this adoption. The later lead change occurs beyond the main 30-evaluation cap, linking the case to the continuation analysis below. The sequence illustrates why artifact admission and report exchange have separate roles in PAN. Scores select better programs, while a report can inform the continued development of another approach. The trace does not establish how that route would have progressed without the report. Figure 4 and Appendix A.2 map the evidence and supported claims for the denoising and scheduling cases.

5.3 Task-dependent effects of literature access

The aggregate difference between literature configurations is small. Positive denoising case does not translate into a consistent overall advantage from literature access. Table 2 places the denoising case in the broader 17-task comparison. The no-paper configuration keeps agent slots and routing rules but removes paper instructions and retrieval tools. Full-paper scores use the same runs as Table 1. The comparison measures these complete configurations, including the accompanying differences in paper selection and access to retrieval tools.

System-level wins can persist without paper inputs. Sorting networks reach 1.000 in both literature configurations, and the no-paper splay mean of 0.537 exceeds the full-paper mean of 0.527. Strong cross-system results thus coexist with no observed gain from paper access on these tasks. Together with the denoising construction, these comparisons locate where literature provides an identifiable method and where strong outcomes persist without the paper inputs.

The BVH case exposes a mismatch in execution assumptions. Two selected BVH papers target GPU methods, whereas this task measures CPU performance. Full-paper means are lower on transaction scheduling (3718 versus 4108 without papers) and BVH (0.387 versus 0.399); on both tasks, every full-paper run is below every no-paper run. The GPU/CPU difference is a concrete mismatch between source assumptions and the task interface; its contribution to the score gap remains an interpretation that this comparison does not isolate. Method content and execution assumptions help interpret these different outcomes. Denoising provides a construction that appears in candidate

Table 2: **Literature configurations on 17 tasks.** The effect of paper access varies across tasks under the nominal 30-evaluation cap. *No paper* means use two runs; *Full paper* means use the same three runs and scores as Table 1. Bold marks the higher mean, with the tie rule of Table 1. Cloudcast entries are scaled by 10^3 for display.

Task	No paper	Full paper
Mathematical construction		
Heilbronn (convex)	**0.0309**	0.0284
Circle packing (square)	**2.6306**	2.6293
Circle packing (rect.)	2.3580	**2.3622**
Min–max distance	**0.0776**	**0.0776**
Signal processing	0.630	**0.728**
Heilbronn (triangle)	**0.0348**	0.0342
Systems research (ADRS)		
Prism (model placement)	**26.253**	26.239
EPLB (MoE balancing)	0.137	**0.140**
LLM–SQL (prefix cache)	0.709	**0.726**
Transaction scheduling	**4108**	3718
Cloudcast (broadcast)	**1.6037**	1.6037
Engineering optimisation (AutoLab)		
Sorting network	**1.000**	**1.000**
Adversarial splay	**0.537**	0.527
Levenshtein (CPU-timed)	0.456	**0.477**
BVH ray tracing (CPU-timed)	**0.399**	0.387
Adaptive compression	0.341	**0.379**
ResNet bit-flip attack	0.981	**0.983**

code; the BVH example exposes a mismatch between the source platform and the measured objective. These cases identify concrete aspects of compatibility to examine when selecting papers or revising their methods for a task.

5.4 Search progress and additional evaluations

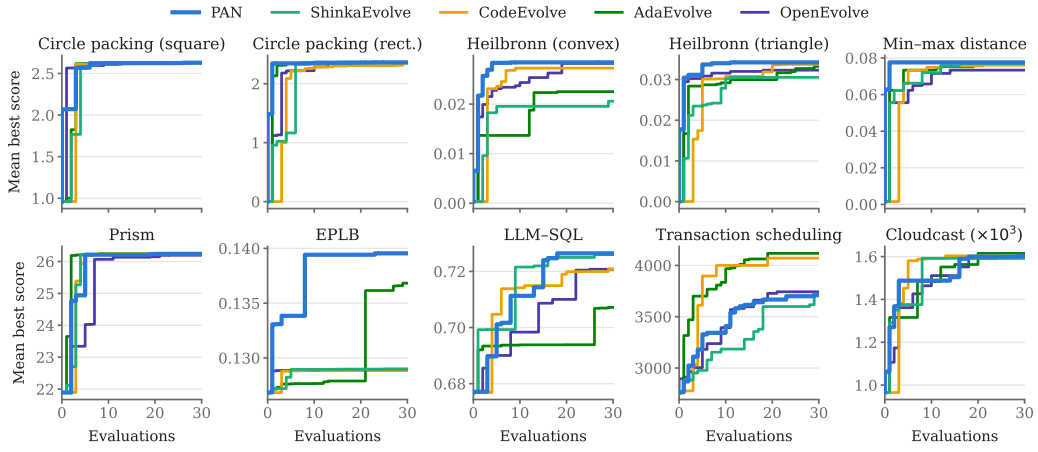

Figure 3: **Search progress over 30 evaluations.** Mean best-so-far scores on ten tasks, averaged over three runs for PAN and two per baseline. Evaluation zero denotes the shared seed score; higher is better in every panel. Each panel uses its own vertical scale, with Cloudcast scores scaled by 10^3.

Additional search produces uneven gains across tasks. Figure 3 complements endpoint scores with search trajectories on ten tasks, showing task-dependent patterns of early improvement and later saturation. The budget extension examines which selected tasks continue improving beyond this cap. Seven selected tasks compare 30 with 60 evaluations (Table 4). PAN gains 6.6% on scheduling, 6.7% on BVH, and 5.5% on splay, versus 0.3% on signal processing and 0.1% on

LLM–SQL. The common cap is therefore a shared checkpoint, while the extensions show different amounts of remaining progress. Signal processing already has a strong score at 30 and gains little thereafter. PAN also improves on splay, where it leads, and on scheduling and BVH, where it trails. Initial comparative rank alone does not describe this observed headroom. Appendix A gives all outcomes and the distinct continuation cohorts.

Final scores and relative gains give different comparisons. At 60 evaluations, PAN has the higher reported mean on four of the seven tasks; CodeEvolve leads scheduling and BVH, and the displayed Cloudcast scores match. On splay, CodeEvolve gains 22.4% but remains below PAN at 60 (0.0104 versus 0.556). Its larger percentage gain starts from 0.0085, while PAN's 5.5% gain starts from 0.527. Relative change describes progress from each system's own starting point, so it must be read alongside the resulting task scores. PAN's scheduling score moves closer to CodeEvolve's, but remains lower at 60 evaluations (3964 versus 4228). On BVH, CodeEvolve also improves, finishing at 0.473 against PAN's 0.413 and widening its lead from 30 to 60 evaluations. The gain from further work therefore does not by itself establish that the initial comparative deficit is resolved by a larger budget. The continuation results thus distinguish further progress within a system from changes in its standing against other systems on the same task.

6 Conclusion

We presented PAN, a network of persistent paper-informed agents that develop and exchange approaches under a common executable objective. It combines implementation histories, evaluated artifacts, report exchange, and adaptive work allocation. The evaluation documents competitive task outcomes and concrete instances of method and information use. Literature access has task-dependent effects, and the scheduling case shows a report entering a route before that route takes the lead. Further discussion appears in Appendix D. These revisable routes offer a concrete way to investigate how research methods enter and evolve within computational problem solving.

Reproducibility statement

Section 3 describes the controller and Section 4 the comparison protocol. Appendix B maps the implementation components and Appendix B.3 records audit exceptions.

AI use statement

Language models generate candidate artifacts and assist paper selection and report structuring in the studied systems, as described in Section 3. An AI coding assistant was also used to inspect the repository, search and organise related literature, draft, restructure, and revise this manuscript, and implement table-generation and document-build scripts from existing experiment records. This manuscript preparation did not generate new experimental runs. The authors take responsibility for the text, references, numerical transcriptions, and interpretation.

Ethics statement

The engineering tasks include adversarial tree-access sequences and a model bit-flip attack; their scores measure performance within the supplied benchmarks and do not demonstrate security of deployed systems. Autonomous coding agents can exploit evaluation interfaces or expose benchmark answers. We document withdrawn artifacts and uncounted feedback to distinguish measured scores from reliable improvements in the intended task. Practical deployment requires controlled execution and evaluation on inputs that were unavailable during optimisation.

REFERENCES

Henrique Assumpção, Diego Ferreira, Leandro Campos, and Fabricio Murai. CodeEvolve: An open source evolutionary coding agent for algorithm discovery and optimization. *arXiv preprint arXiv:2510.14150*, 2025. URL https://arxiv.org/abs/2510.14150.

Mert Cemri, Shubham Agrawal, Akshat Gupta, Shu Liu, Audrey Cheng, Qiuyang Mang, Ashwin Naren, Lutfi Eren Erdogan, Koushik Sen, Matei Zaharia, Alex Dimakis, and Ion Stoica. AdaEvolve: Adaptive LLM driven zeroth-order optimization. *arXiv preprint arXiv:2602.20133*, 2026. URL https://arxiv.org/abs/2602.20133.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. Mlebench: Evaluating machine learning agents on machine learning engineering. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=6s5uXNWGIh.

X. C. Chen, Yu. A. Litvinov, M. Wang, Q. Wang, and Y. H. Zhang. Denoising scheme based on singular-value decomposition for one-dimensional spectra and its application in precision storage-ring mass spectrometry. *Physical Review E*, 99:063320, 2019. doi: 10.1103/PhysRevE.99.063320. URL https://arxiv.org/abs/2010.14508.

Audrey Cheng, Shu Liu, Melissa Pan, Zhifei Li, Bowen Wang, Alex Krentsel, Tian Xia, Mert Cemri, Jongseok Park, Shuo Yang, Jeff Chen, Lakshya Agrawal, Aditya Desai, Jiarong Xing, Koushik Sen, Matei Zaharia, and Ion Stoica. Barbarians at the gate: How AI is upending systems research. *arXiv preprint arXiv:2510.06189*, 2025. URL https://arxiv.org/abs/2510.06189.

Alireza Ghafarollahi and Markus J. Buehler. SciAgents: Automating scientific discovery through multi-agent intelligent graph reasoning. *arXiv preprint arXiv:2409.05556*, 2024. URL https://arxiv.org/abs/2409.05556.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025. URL https://arxiv.org/abs/2509.19349.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024. URL https://arxiv.org/abs/2408.06292.

Yougang Lyu, Xi Zhang, Xinhao Yi, Yuyue Zhao, Shuyu Guo, Wenxiang Hu, Jan Piotrowski, Jakub Kaliski, Jacopo Urbani, Zaiqiao Meng, Lun Zhou, and Xiaohui Yan. EvoScientist: Towards multi-agent evolving AI scientists for end-to-end scientific discovery. *arXiv preprint arXiv:2603.08127*, 2026. URL https://arxiv.org/abs/2603.08127.

Jiacheng Miao, Joe R. Davis, Yaohui Zhang, Jonathan K. Pritchard, and James Zou. Paper2Agent: Reimagining research papers as interactive and reliable AI agents. *arXiv preprint arXiv:2509.06917*, 2025. URL https://arxiv.org/abs/2509.06917.

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. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025. URL https://arxiv.org/abs/2506.13131.

Bernardino Romera-Paredes, Mohammad Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024. doi: 10.1038/s41586-023-06924-6. URL https://www.nature.com/articles/s41586-023-06924-6.

Asankhaya Sharma and contributors. OpenEvolve: Open-source implementation of AlphaEvolve. Software repository, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve. Accessed September 21, 2026.

Michael D. Skarlinski, Sam Cox, Jon M. Laurent, James D. Braza, Michaela Hinks, Michael J. Hammerling, Manvitha Ponnapati, Samuel G. Rodriques, and Andrew D. White. Language agents achieve superhuman synthesis of scientific knowledge. *arXiv preprint arXiv:2409.13740*, 2024. URL https://arxiv.org/abs/2409.13740.

Haoyang Su, Renqi Chen, Shixiang Tang, Zhenfei Yin, Xinzhe Zheng, Jinzhe Li, Biqing Qi, Qi Wu, Hui Li, Wanli Ouyang, Philip Torr, Bowen Zhou, and Nanqing Dong. Many heads are better than one: Improved scientific idea generation by a LLM-based multi-agent system. *arXiv preprint arXiv:2410.09403*, 2024. URL https://arxiv.org/abs/2410.09403.

Minyang Tian, Luyu Gao, Shizhuo Dylan Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, Shengyan Liu, Di Luo, Yutao Ma, Hao Tong, Kha Trinh, Chenyu Tian, Zihan Wang, Bohao Wu, Yanyu Xiong, Shengzhu Yin, Minhui Zhu, Kilian Lieret, Yanxin Lu, Genglin Liu, Yufeng Du, Tianhua Tao, Ofir Press, Jamie Callan, Eliu Huerta, and Hao Peng. SciCode: A research coding benchmark curated by scientists. *arXiv preprint arXiv:2407.13168*, 2024. URL https://arxiv.org/abs/2407.13168.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. TravelPlanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*, 2024. URL https://arxiv.org/abs/2402.01622.

Zhangchen Xu, Junda Chen, Yue Huang, Dongfu Jiang, Jiefeng Chen, Hang Hua, Zijian Wu, Zheyuan Liu, Zexue He, Lichi Li, Shizhe Diao, Jiaxin Pei, Jinsung Yoon, Hao Zhang, Mengdi Wang, Radha Poovendran, Misha Sra, Alex Pentland, and Zichen Chen. AutoLab: Can frontier models solve long-horizon auto research and engineering tasks? *arXiv preprint arXiv:2606.05080*, 2026. URL https://arxiv.org/abs/2606.05080.

Shuo Yan, Ruochen Li, Ziming Luo, Zimu Wang, Daoyang Li, Liqiang Jing, Kaiyu He, Peilin Wu, George Michalopoulos, Yue Zhang, Ziyang Zhang, Mian Zhang, Zhiyu Chen, and Xinya Du. LMR-Bench: Evaluating LLM agent's ability on reproducing language modeling research. *arXiv preprint arXiv:2506.17335*, 2025. URL https://arxiv.org/abs/2506.17335.

Yansen Zhang, Yilu Liu, Tianyu Liu, Jiamin Chen, Xiaokun Zhang, Kai Xie, Xue Liu, Chen Ma, and Yiyan Qi. Budget-aware LLM discovery via cost-calibrated frontier utility. *arXiv preprint arXiv:2607.26828*, 2026. URL https://arxiv.org/abs/2607.26828.

A Additional comparison details

The task-level outcomes in Table 1 place PAN first under three aggregate measures (Table 3). The budget extension examines whether selected tasks continue improving after 30 evaluations (Table 4). These comparisons describe complete-system performance under the conditions in Section 4.

Table 3: **Aggregate system comparison.** PAN leads the three summary measures computed from Table 1, including its six exploratory tasks. Tasks receive equal weight; *Wins* includes ties. *Normalised score* divides a system's score by the best score for maximisation, and the best score by the system's score for minimisation.

System	Average rank ↓	Wins ↑	Normalised score ↑
PAN	2.00	12/23	0.944
ShinkaEvolve	3.43	1/23	0.838
CodeEvolve	3.17	3/23	0.835
AdaEvolve	2.57	7/23	0.891
OpenEvolve	3.57	0/23	0.853

Table 4: **Scores at 30 and 60 evaluations.** Additional search yields task-dependent improvements on seven tasks selected for headroom. Entries are run means; *gain* is their percentage change. PAN uses three continued runs per task. CodeEvolve uses two 60-evaluation runs and their first-30 prefixes, a different sample from Table 1. Bold marks the better score at 60; Cloudcast entries are scaled by 10^3 for display.

	PAN			CodeEvolve		
Task	at 30	at 60	gain	at 30	at 60	gain
Transaction scheduling	3718	3964	+6.6%	4219	**4228**	+0.2%
BVH ray tracing	0.387	0.413	+6.7%	0.436	**0.473**	+8.5%
Adversarial splay	0.527	**0.556**	+5.5%	0.0085	0.0104	+22.4%
Signal processing	0.728	**0.730**	+0.3%	0.672	0.680	+1.2%
LLM–SQL	0.726	**0.727**	+0.1%	0.706	0.725	+2.7%
Circle packing (rect.)	2.3622	**2.3633**	+0.0%	2.3517	2.3599	+0.3%
Cloudcast	1.6037	1.6037	+0.0%	1.5921	**1.6037**	+0.7%
Mean over tasks			+2.45%			+5.14%

A.1 Budget-extension protocol and interpretation

Table 4 compares mean scores at 30 and 60 evaluations on seven tasks selected for headroom. PAN uses three continued runs per task; CodeEvolve uses its two most recent 60-evaluation runs and their first-30 prefixes. Each task's gain is the percentage change between its run-mean scores. CodeEvolve's 30-evaluation sample therefore differs from the main comparison. Within-run pairing controls the starting trajectory but does not remove evaluator noise.

Returns vary by task. PAN improves by 6.6% on transaction scheduling, 6.7% on BVH, and 5.5% on splay, while signal processing and LLM–SQL gain 0.3% and 0.1%. Rectangle packing improves slightly, rounding to 0.0%, and Cloudcast remains flat at the reported precision. CodeEvolve's gains range from 0.2% on transaction scheduling to 22.4% on splay. Splay contributes about 3.2 percentage points to its 5.14% mean gain across tasks. At 60 evaluations, CodeEvolve finishes at 0.0104, compared with 0.556 for PAN on this task.

At 60 evaluations, PAN outperforms CodeEvolve on splay, signal processing, LLM–SQL, and rectangle packing; CodeEvolve remains stronger on transaction scheduling and BVH. These results show continued improvement on some tasks and saturation on others. They do not establish general budget robustness because tasks were selected for headroom, PAN resumes existing runs while CodeEvolve runs through to 60, and only CodeEvolve is compared at the larger budget. The experiment does not isolate proposal diversity from allocation policy or additional feedback access.

A.2 Evidence from the two process cases

Figure 4 summarises the two cases discussed in Section 5.2. It visualises the existing observations from different tasks; the arrows indicate the recorded sequence of method and information use.

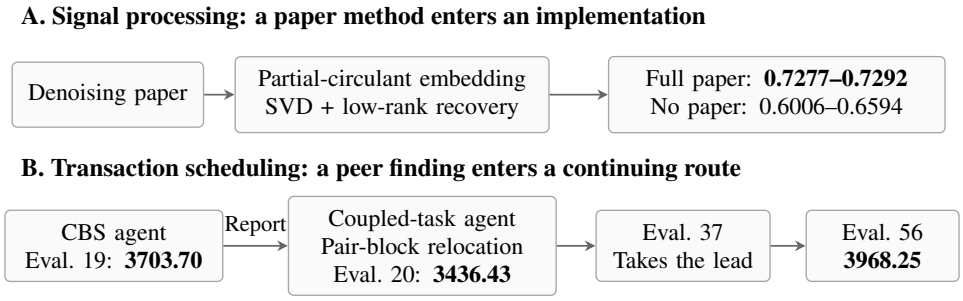

Figure 4: **Complementary evidence from different tasks.** (A) A paper method enters artifacts; ranges cover three full-paper and two no-paper runs. (B) The receiving agent cites the conflict-based-search (CBS) report as motivation for its change. Later progress does not establish a causal benefit from the report.

B EXPERIMENTAL SETTINGS

Table 5 lists the controller defaults. Agents execute synchronously and retain their working context between turns; pairwise discussion is disabled in the reported runs. Opening and later turn limits are $\lceil B_0/K \rceil$ and $\lceil B_0/(2K) \rceil$, respectively, where K is the initial agent count and B_0 the configured turn basis. Continuations can retain this basis when the evaluation cap increases. Utility budget fractions use the remaining token budget. Token limits apply separately to resumed segments and do not establish matched total computation over the complete search runs.

Table 5: **Controller defaults.** These settings govern allocation, independent startup work, and reserve admission. Per-run overrides are unavailable in the accompanying sources.

Parameter	Value
Initial agents / maximum extra agents	3 / 1
Maximum initially selected reserves	3
Initial independent rounds	2
Reports without admission for long plateau	5
Maximum consecutive retasks	3
Minimum significant normalised gain	10^{-3}
Source-family divergence threshold	0.35
Minimum turns before ordinary retirement	4
Newcomer trigger (remaining evaluation fraction)	0.35
Utility smoothing coefficient	0.9
Reference token cost	150,000
Pairwise discussion	Disabled

B.1 PAPER SELECTION AND LITERATURE ACCESS

Full-paper agents share cached paper selections, method indexes, and page text across replicates. Selection favours papers with accessible methods beyond an abstract and uses task-specific retrieval terms with manual adjustments. Indexes guide access but do not verify implementation fidelity. The no-paper configuration retains agent slots and routing rules but removes paper instructions and retrieval tools. Feedback can therefore produce different routing trajectories.

B.2 BASELINE ADAPTERS AND COMPUTE

ShinkaEvolve and CodeEvolve use dedicated adapters; AdaEvolve and OpenEvolve run through SkyDiscover. All wrappers use the task evaluators and enforce a scored-candidate cap, although generation may continue after scoring stops. CodeEvolve omits null sampling parameters and uses temperature 1 for model compatibility. OpenEvolve allows programs up to 60,000 characters to accommodate the task seeds. Token accounting and caching are not matched across systems, so these comparisons do not establish equal-compute or monetary-cost efficiency.

Budget use. In the recorded token audit, 51 full-paper PAN runs average 6.29 million gross input tokens (95% cached), 0.26 million uncached input tokens, and 69,000 output tokens per run. The CodeEvolve audit covers 19 runs up to their evaluation caps, averaging 0.24 million input and 97,000 output tokens per run. Generation after the scoring cap is excluded from these CodeEvolve figures. ShinkaEvolve, AdaEvolve, and OpenEvolve lack usable token accounting in the audited records.

B.3 EVALUATION QUALIFICATIONS

The rectangle-packing evaluator accepts overlaps and perimeter excesses below 10^{-6}, and the score sums the radii, so a program can inflate every radius into that slack. PAN's programs do, by their own comments; we report them re-scored at zero tolerance, which costs 4.2×10^{-6} and changes no ordering. All eight baseline programs are strictly feasible, so the same check leaves them unchanged. The evaluator is unmodified. Repeated evaluation can also change scores through stochastic or timing variation. Signal-processing timeouts can change which inputs contribute to a score; an unreproducible fresh-run result was withdrawn, and the continuations were checked on an idle host. The evaluator permits non-causal denoising of the observed noisy signal.

C Additional method details

C.1 Objects and counters in the search loop

Table 6 separates the persistent state, measured outcomes, and communication decisions used in Section 3. A work turn is a scheduled tool-using session; an evaluation is a call recorded by the scoring wrapper. They are not interchangeable units. The opening and later turn limits in Appendix B use the configured basis B_0, while the task cap B counts evaluations. A separate token allowance supplies the remaining-budget fraction used by the allocation utility.

Table 6: **Search state and controller operations.** Evaluation, progress credit, board admission, and relay serve distinct roles in the development of a persistent route.

Object or event	Operational role
Persistent route	Paper identifier, transcript, working artifacts, and evaluation history survive scheduled turns. The method can change while the identifier remains fixed.
Evaluation	The wrapper produces evaluator output and a frozen artifact digest; this event counts toward the nominal candidate cap.
Progress credit	Local and global gains enter the token-cost-aware utility in Equation 3; utility guides further work.
Board admission	The valid score must clear the current bar by the task threshold δ. This decides artifact retention, independently of whether a report can be relayed.
Queued relay	Startup relays expose the bar only. Later reports include a method summary delivered at a subsequent scheduled turn; stalled routes may also receive reference programs.
Membership	Plateau and remaining-budget triggers can introduce a reserve route, subject to capacity and protection of the leader.

The task-specific admission threshold δ is distinct from the controller's significant normalised-gain threshold in Table 5. The available defaults do not specify every run's B_0, token cap, workspace choice, or configuration override. Implementation snapshots are unavailable for resolving queue overwrite rules or the precise internal order of best-score, bar, and smoothed-utility updates. These are limits on exact reconstruction; Table 6 specifies the roles of the operations.

C.2 Membership, similarity, and optional paths

The controller limits consecutive retasks and uses repeated scores as a stagnation signal. A reserve can occupy a vacancy or the one allowed extra slot; otherwise a weak route is replaced, with protection for the leading route when possible. An empty reserve can be reconsidered from the cached paper pool. Without a suitable reserve, the controller continues an existing route. Source-text similarity is computed after removing seed lines and is used as a practical filter for reference programs and route similarity. It is not a measured distance between proposal distributions. Table 5 lists the thresholds governing work allocation and membership changes.

The implementation also contains optional pairwise dialogue and open-finding-ledger paths. They were not evaluated here. The reported interaction consists of the queued reports and reference artifacts described in Section 3; the case study does not depend on pairwise dialogue.

D Discussion and limitations

Compatibility of papers and tasks. The denoising construction identifies literature use; the near-neutral aggregate and BVH setting mismatch qualify its broader value. A paper-agent system should make both methods and assumptions revisable. PAN provides such a process, while leaving automatic selection of compatible papers unresolved.

Information use during route development. The scheduling account records a report-motivated change whose first implementation is weaker than the sender's. PAN's separate admission and relay paths accommodate information use during route development. Its independent performance benefits from persistence, exchange, and allocation remain unresolved.

Comparison scope and evaluator validity. Unequal tools, feedback, and computation, small replicate counts, and incomplete records limit comparative conclusions (Section 4). Corpus access is partly manual when retrieval services lack full text. Evaluator-guided search can also exploit scoring procedures. Appendix B.3 records packing-tolerance and timing issues, including a withdrawn result. A measured artifact alone does not establish fidelity to the intended task.