PSI-EVOLVE: AUTONOMOUSLY ILLUMINATING THE SPACE OF PROBLEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Combining modern Large Language Models (LLMs) and large amounts of resources has enabled AI systems to solve increasingly complex existing scientific problems. However, to achieve truly autonomous scientific discovery, AI systems must also be able to propose diverse problems that go beyond readily available human knowledge. This is inherently challenging for LLMs, which have been trained on human sources and verifiable objectives, and may lack the experience and intuition to find interesting new problems with potential downstream utility. In this work, we introduce Psi-Evolve (Ψ-Evolve), a new framework leveraging evolutionary computation to bridge this gap. Ψ-Evolve combines carefully designed task descriptors, mutations, and objectives to find diverse problems whose solutions are inherently generalizable. Empirically, we show that Ψ-Evolve autonomously discovers archives of new problems that span well beyond the space covered by readily available human archives and whose solutions can zero-shot transfer across a given scientific domain. Furthermore, we also show that Ψ-Evolve's archives can be used in conjunction with prior AI systems, accelerating and cutting the cost of LLM-driven solution discovery itself.

1 INTRODUCTION

In recent years, scaling LLMs test-time compute has enabled AI to tackle increasingly complex scientific problems. In mathematics, these advances have advanced the field from traditional reasoning benchmarks (Hendrycks et al., 2021; Shao et al., 2025) to formal proofs of complex theorems or conjectures (Hubert et al., 2026; Ren et al., 2025) A central ambition of this line of research is to move beyond solving isolated problems and toward partially automating scientific research and accelerating the discovery of new knowledge (Boiko et al., 2023; Lu et al., 2024).Still, most of these methods rely on humans to handcraft and select the problems to solve. However, scientific research goes well beyond the solving or hill-climbing of specific problems: it is an open-ended process in which new artifacts are constantly produced (Sigaud et al., 2023; Hughes et al., 2024) and reused in order to expand the field of knowledge. In the longer term, as LLMs solve an increasing share of human-made problems, it could be increasingly relevant to endow them with problem generation abilities, even simply to ensure the production of new knowledge. That would assist researchers by offering the ability to constitute a library of information over the domain they seek to investigate. Moreover, for any new problem LLMs come up with, the same information could provide themselves with new knowledge to efficiently reach future solutions or gain domain understanding.

We propose **Psi-Evolve (Ψ-Evolve)**, a framework for ***P***roblem ***S***pace ***I***llumination that autonomously explores a domain to build a diverse library of problems whose solutions exhibit reusable algorithmic structure. Problem-space exploration faces two key challenges: the relevant directions of variation are not known in advance, and there is no obvious notion of what makes one generated problem more valuable than another. Ψ-Evolve aims to address both by maintaining an evolving semantic representation that guides generation toward underexplored or newly discovered directions, while evaluating each problem through the transferability of the solver it induces. To preserve a compact and diverse collection, Ψ-Evolve borrows the archive-selection principle of Quality Diversity (QD), while replacing fixed behavioral descriptors and solution-level fitness with adaptive semantic exploration and transfer-defined problem quality. The resulting framework jointly discovers *where to explore* and *which problems are worth retaining*. This paper makes three contributions.

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.

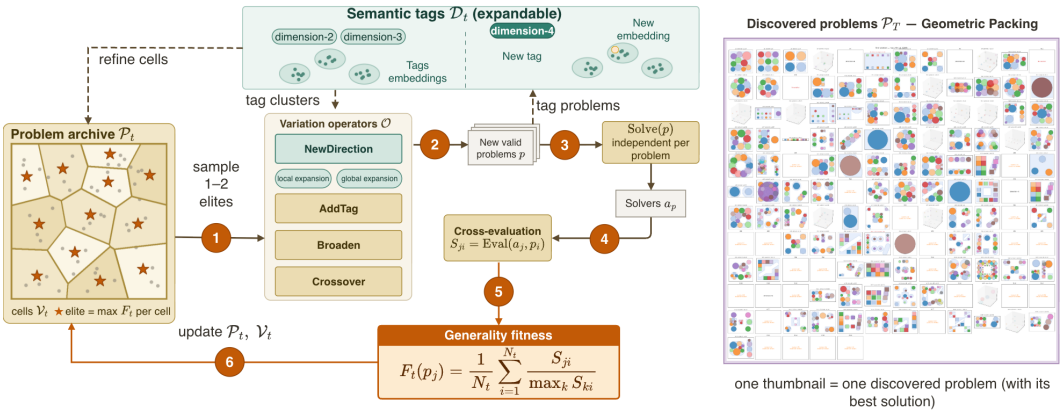


Figure 1: **Overview of Ψ-Evolve.** *Left*: starting from validity constraints C, the search repeatedly (1) samples elite problems from a Voronoi-tessellated archive, (2) creates new valid problems with four mutations (Crossover, Broaden, AddTag, NewDirection), (3) optimizes a solver on new problems, (4) cross-evaluates new and existing solvers to update a transfer matrix S, (5) recomputes a fitness score F_t as the average normalized transfer of a problem's induced solver, and (6) updates the archive, keeping the most general problem in each cell. In parallel, an expandable vocabulary of semantic tags D_t is embedded and clustered for AddTag and NewDirection. *Right*: example set of all problems discovered by one run on the Circle Packing task.

1. We introduce Ψ-Evolve, a framework for problem-space illumination that does not require a fixed, researcher-defined parameterization of the domain: it jointly generates new problems, expands a semantic descriptor space to discover relevant dimensions of variation, and defines problem quality through the cross-problem transferability of the problem solver.
2. Empirically, over three mathematical domains, Ψ-Evolve produces problems whose induced solvers achieve strong zero-shot and few-shot performance against from-scratch solvers on literature-derived evaluation problems. We show that Ψ-Evolve generates a wider diversity of problems inducing a greater diversity of solutions compared to strong baselines such as CVT-MAP-Elites and POET.
3. Finally, we construct three diverse datasets of literature-derived problems with permissive licenses crawled from the web, providing external reference sets for evaluating the coverage and relevance of generated problems.

2 Ψ-Evolve: Automated Problem Space Illumination

We consider the problem of *illuminating* a computational problem domain $\mathcal{X}$ specified through validity constraints, rather than a known parameterization or fixed behavioral descriptors. Our goal is to construct a compact and diverse collection of problems that exposes distinct structure in the domain, following the principles of unsupervised quality-diversity (Grillotti & Cully, 2022). Ψ-Evolve combines three components: a fixed problem-embedding space that organizes the archive, an evolving semantic vocabulary that guides problem generation, and a transfer-based fitness that defines problem quality. It jointly maintains valid problems, independently optimized solvers, and semantic problem features. A problem is high quality when its induced solver transfers well to other discovered problems. Thus, the embedding space determines *where problems are stored*, the semantic representation determines *where to explore*, and solver transfer determines *which problems are retained* (Appendix B.2). Given a validity specification $\mathcal{C}$, the method progressively constructs a QD archive without requiring an explicit enumeration of valid problems. For each valid problem p, a solver-development procedure produces $a_p = \mathrm{Solve}(p)$, whose performance on any problem is measured by $\mathrm{Eval}(a, p)$. Over T iterations, the method builds a discovered set $\mathcal{P}_T = \{p_1, \ldots, p_{N_T}\} \subset \mathcal{P}_\mathcal{C}$ covering heterogeneous and non-redundant regions of the domain.

At each iteration, the method samples one or more elite problems from the current archive corresponding to the highest-fitness problem in one archive cell. A sampled elite is mutated to produce a new valid candidate using crossover, broadening, feature addition, or semantic new-direction dis-

covery (Section 2.2). Each new problem is independently solved and then cross-evaluated against the discovered problems. Its quality is measured by a fitness function F_t that captures how well its induced solver transfers to other problems at time step t (Section 2.3). After evaluation, transfer scores and fitness values are updated, new problems are inserted into the archive, and archive cells are refined when needed. In parallel, newly discovered semantic tags expand the representation used by variation operators. Repeating this process jointly expands the set of discovered problems, the semantic vocabulary, and the empirical map of algorithmic transfer across the domain. We refer to Appendix B.6 for the complete algorithm.

2.1 Structure of the archive

Archive Representation and Partition The search starts without knowing which problem properties define meaningful directions of variation in $\mathcal{P}_\mathcal{C}$. Unlike standard MAP-Elites, we therefore cannot rely on a fixed, researcher-defined behavior descriptor $b(p) \in \mathbb{R}^d$. Instead, each problem is mapped through a fixed text-embedding function $\mathcal{E}(p) \in \mathbb{R}^{d_e}$, which provides a generic geometric space in which to organize the archive. The archive forms a recursive partition of this embedding space. As problems are discovered, sufficiently populated regions are clustered and the resulting cluster centroids define Voronoi cells (Aurenhammer, 1991). At time t, the archive therefore consists of a collection of leaf cells $\mathcal{V}_t = \{v_1, \ldots, v_{n_t}\}$. Each discovered problem belongs to one leaf cell according to its embedding, and each cell retains one elite: the member with the highest current fitness F_t. Importantly, the embedding function $\mathcal{E}$ is fixed throughout the run. What evolves is the partition induced by the discovered problems, not the embedding representation itself.

Dual Descriptor Space The fixed problem-embedding space provides a generic geometry for archive construction, but it does not explicitly describe the semantic properties along which problems differ. Ψ-Evolve therefore maintains a second, evolving representation that is used to characterize problems and guide variation. Let $\mathcal{D}_t = \{\tau_1, \ldots, \tau_{m_t}\}$ denotes the vocabulary of semantic features, or *tags*, discovered by iteration t. Each problem is associated with a set of applicable tags $T_t(p) \subseteq \mathcal{D}_t$. As new tags are introduced, $\mathcal{D}_t \subseteq \mathcal{D}_{t+1}$ and $m_{t+1} \geq m_t$. Previously discovered problems may be retroactively checked against newly introduced tags, reducing dependence on the order in which semantic concepts are discovered. Each tag is embedded as $e_\tau = \phi(\tau) \in \mathbb{R}^{d_\tau}$, and these embeddings are clustered to group semantically related tags. The resulting clusters provide a local approximation to the conceptual directions currently represented in the domain. They are used by the semantic variation operators to refine existing families of features or to identify directions that are not yet represented. The two representations have distinct roles. Problem embeddings define the persistent archive geometry and determine which problems compete within the same cell. Semantic tags and their clusters do *not* define archive cells; they are used to describe discovered problems and to guide the generation of new ones. Finally, the transfer-based fitness F_t determines the elite within each archive cell but does not determine the geometry of the partition.

We provide the full construction and update procedure for the recursive Voronoi archive and the semantic tag clustering in Appendix B.3.

2.2 Problem-Space Variation and Ontology Expansion

Given an archive p_b, Ψ-Evolve generates candidate problems using four problem-space variation operators: $\mathcal{O} = \{\text{CROSSOVER}, \text{BROADEN}, \text{ADDTAG}, \text{NEWDIRECTION}\}$ with complementary roles: the first three explore combinations or modifications of previously discovered semantic structure, whereas NEWDIRECTION can expand the representation itself. At each problem generation step, the algorithm uniformly selects up to two problems among the cluster's elites and an operator:

CROSSOVER. Two distinct archive representatives p_b and $p_{b'}$ are selected, and a new problem is generated by synthesizing structural elements of both parents. Crossover explores combinations of semantic properties that have separately produced high-generality elites.

BROADEN. Starting from p_b, the generator constructs a related problem designed to modify the parent problem by making it more general (explicitly trying to maximize F_t). This operator provides local exploration around successful regions without requiring a new semantic dimension. In addi-

tion to the parent problem, the generator is conditioned on problems for which the parent solution achieves a high score and problems for which the parent solution is a non-zero low score.

ADDTAG. The search selects a known tag $\tau \notin T_t(p_b)$ and asks the generator to modify p_b such that the new problem additionally expresses τ. Tags are sampled with an inverse-frequency bias, $w_t(\tau) = \frac{1}{n_t(\tau)+1}$, where $n_t(\tau)$ is the number of discovered problems currently carrying tag τ. Consequently, underrepresented semantic features receive greater probability, providing an explicit pressure toward coverage of the known descriptor space.

NEWDIRECTION. This operator expands the semantic vocabulary by proposing a new feature $\tau_{\text{new}} \notin \mathcal{D}_t$ with two complementary modes: The *local ontology expansion* mode, in which a cluster of semantically related tag embeddings is selected, and the generator proposes a sibling feature lying on the same conceptual direction but not already represented in the vocabulary. A high-quality problem associated with that semantic region provides an example from which a problem expressing the new feature is generated. In *global ontology expansion* mode, the current tag clusters are summarized, and the generator is explicitly asked to identify a semantically divergent feature not represented by the existing clusters. A new problem is then generated from this feature without conditioning on a parent example. The goal of local expansion is to efficiently explore different regions of the semantic space, while global expansion aims to discover entirely new regions.

2.3 FITNESS FUNCTION

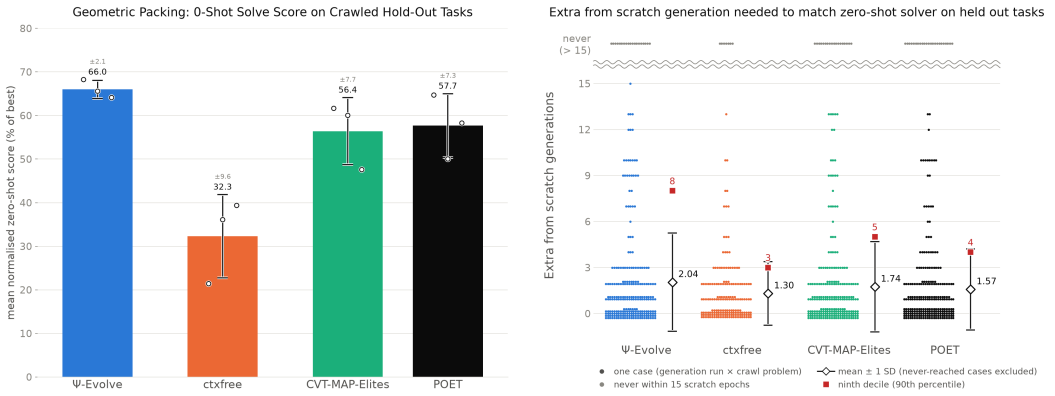
There is no inherent fitness function related to the objective of exploring the large and complex space of problems. A natural goal for this function would be to allow our algorithm to avoid getting stuck in very small niches where problems differ by meaningless, very specific variables. Instead, we would like our algorithm to bring up problems in which the solutions would also solve a large number of different problems. We baked this generality heuristic into a generality fitness function by defining the quality of a discovered problem through the cross-problem generality of the solver that it induces. For each new problem p_j, associated with a solver $a_j = \text{Solve}(p_j)$, we evaluate it on every compatible discovered problem and maintain the cross-evaluation matrix $S^{(t)} \in \mathbb{R}_{\geq 0}^{N_t \times N_t}$, $S_{ji}^{(t)} = \text{Eval}(a_j, p_i)$. Invalid executions, errors, or evaluations exceeding the prescribed evaluation budget receive a score of zero. For readability, we omit the superscript (t) when the search step is clear from context. We then define the normalized performance of solver a_j on target p_i as

$$R_{ji} = \begin{cases} \dfrac{S_{ji}}{b_i}, & b_i > 0 \\ 0, & b_i = 0 \end{cases} \qquad b_i = \max_{k \in \{1,\dots,N_t\}} S_{ki} \tag{1}$$

where b_i is the maximum score observed on p_i, to satisfy $R_{ji} \in [0, 1]$. Then, the fitness of a problem p_j at time step t is the average normalized transfer performance of its solver across all of the discovered domains: $F_t(p_j) = \frac{1}{N_t - 1} \sum_{i \neq j}^{N_t} R_{ji}$. Thus, a problem receives high quality if the solver obtained by optimizing on that problem approaches the best-known performance across many other discovered problems, and each target problem contributes according to the fraction of its best-known performance achieved by a_j/b_i, preventing problems with numerically large evaluation scales from dominating the objective.

3 EXPERIMENTS

Domains and datasets construction We study our method on three different and structurally complementary domains: geometric packing, discrete spectral design, and geometry of numbers. These domains were selected to combine three properties that make domain exploration particularly challenging. First, each domain contains a very large and heterogeneous space of valid problems, with variation arising from several interacting factors. Second, there is no unique natural choice of axes on which to organize the problems: two instances may look similar syntactically while requiring very different solution strategies. Third, within each domain we can define a common solver interface, so that a solver developed for one problem can be evaluated on other problems from the same domain. This makes it possible to study not only whether the method generates diverse problems, but also whether the discovered diversity corresponds to meaningful differences



(a) (b)

Figure 2: **(a) Ψ-Evolve gets highest normalized score.** On geometric packing, Ψ-Evolve beats by 9 points POET, the best baseline. **(b) Zero-shot archive solutions saves multiple search steps from optimization from scratch.** For each held-out human-authored problem, we report the number of ShinkaEvolve optimization steps required from scratch to match the score of the best zero-shot solver in each generated archive. Ψ-Evolve corresponds to 2.04 optimization steps on average and reaches 8 steps at the ninth decile, compared with 5 for the strongest baseline.

in solver behavior. In order to get some reference points in the problem space of each domain, we crawled human-generated problems from the web. More details on the dataset building process are available in Appendix A.

Problem Generator and Solver For most of our main experiments, we used GPT-5.6 Luna (OpenAI, 2026) together with the ShinkaEvolve solver (Lange et al., 2026), choices we later ablate in Section 3.3. Unless otherwise specified, in all experiments, the method or baselines generate 100 problems, and all the experiments are repeated and across three seeds, with final results averaged. We provide all exact prompts used in Appendix F and tune the distribution over the mutation operators through a hyperparameter search detailed in Appendix D. All problems in the evaluation set across Sections 3.1- 3.2 were held out from method-development decisions to avoid biasing our results: they were not used for prompt design, mutation-operator selection, or stopping decisions.

Baselines We compare Ψ-Evolve against three baselines: context-free generation, CVT-MAP-Elites (Vassiliades et al., 2018), and POET (Wang et al., 2019), with the latter two adapted to our setting. The context-free baseline generates each problem independently, conditioned only on domain descriptions, therefore measuring the model's intrinsic ability to propose problems without search history. The CVT-MAP-Elites baseline follows the standard select-vary-evaluate-describe-archive loop. Because QD descriptors are intended to capture behavior, we represent each generated problem by the behavior it induces: the source code of its independently optimized solver. Solver embeddings therefore define the behavioral space, so problems are considered similar when yielding similar solutions. An ablation using problem-space embeddings instead is reported in Appendix D.5. As in Ψ-Evolve, the embedding space is partitioned into niches and problems are generated through crossover, broadening, and descriptor-guided new-direction mutations, while using the same cross-problem generality objective for selection. This baseline tests whether diversity in the latent space of induced solvers is sufficient for broad problem-space coverage. Finally, the POET-inspired baseline co-evolves problem–solver pairs, with each solver continuing to optimize on its associated problem. Unlike Ψ-Evolve, it maintains a growing population without explicit semantic niches or top-K pruning. It therefore tests whether open-ended problem–solver co-evolution and solver transfer alone are sufficient to explore the problem domain. Full implementation details are given in Appendix C.

3.1 Results for Geometric Packing

In order to better grasp the characteristics of Ψ-Evolve, we present a deep dive into the results in the geometric packing domain. In the following section, we present aggregated results averaged over the three domains. Geometric packing studies arrangements of objects within bounded regions.

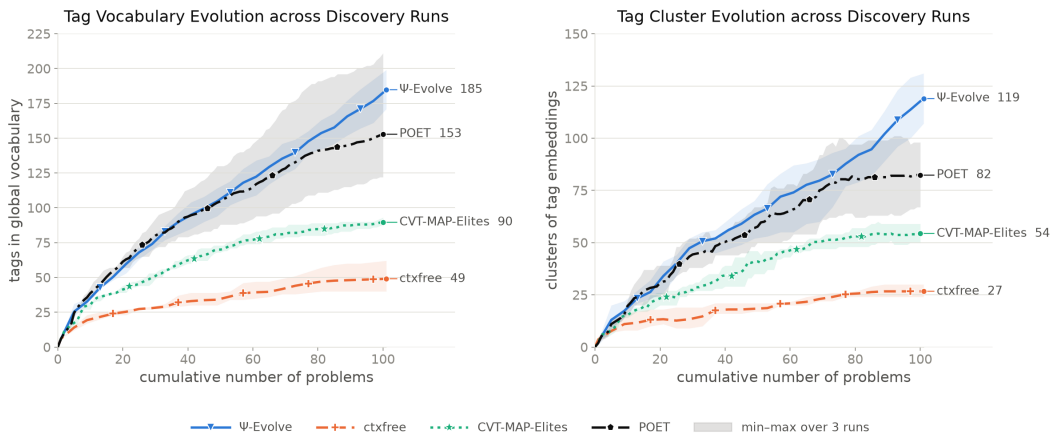


Figure 3: **Ψ-Evolve sampled problems exhibit a higher diversity.** Evolution of the number of tags (left) and tag clusters (right) for the geometric packing domain. Ψ-Evolve does not saturate.

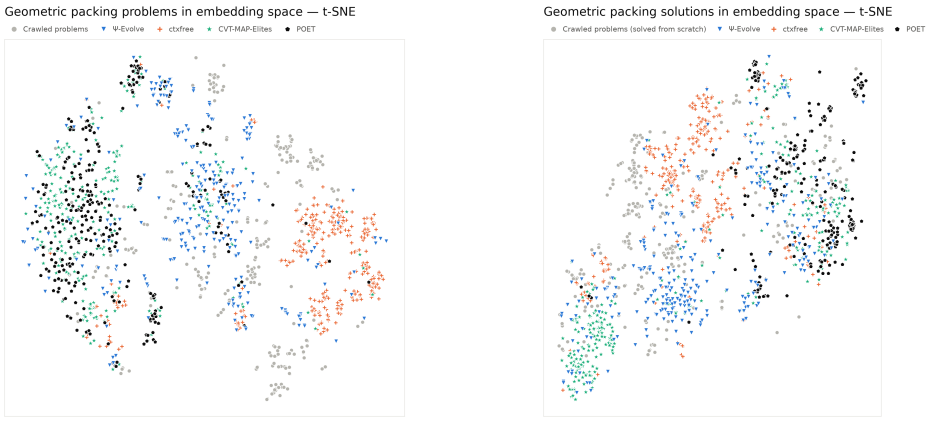
Problems vary in object and container geometry, allowed transformations, constraints, and objectives such as maximizing packed volume or separation. These factors create a heterogeneous domain in which small structural changes can substantially alter the optimization landscape. We present more details on this domain in the Appendix A.1.

Performance in zero-shot against crawled problems. In this experiment, each method samples 100 problems, and we measured the normalized performance of their respective solver on the 261 problems generated by humans. In Figure 2a we plot the average score of the best zero-shot solver; Ψ-Evolve outperforms context-free generation with a normalized score of 66.0 against 32.3, also beating CVT-MAP-Elites and POET baselines scoring 56.4 and 57.7.Although our main evaluation reports oracle selection over the full archive, we find that retrieval based only on the target problem representation recovers most of this utility, more results in Appendix D.7.

Does internal generality predict external transfer? Although F_t is computed only from transfer among generated problems, it strongly predicts utility on held-out literature problems. Across the three geometric-packing seeds, internal generality correlates with the external performance of each problem's induced solver at $r = 0.893$ (Pearson). Thus, problems favored by F_t tend to induce solvers that transfer beyond the generated distribution, suggesting that the fitness captures external solver utility rather than merely within-search transfer. More details are given in Appendix D.8

Equivalent from-scratch optimization budget. Beyond zero-shot performance, we quantify the computational value of the generated archive by asking how many optimization steps a from-scratch solver requires to recover the performance already provided by an archive solution. For each held-out problem, we run ShinkaEvolve from scratch and record the first optimization step at which it matches the score of the best zero-shot solver from the generated archive. We restrict the analysis to non-trivial problems for which the from-scratch run requires at least one optimization step to reach its best score, and exclude cases where the corresponding archive solver does not produce a valid solution. As shown in Figure 2b, the best zero-shot solver produced by Ψ-Evolve corresponds, on average, to approximately 2.04 steps of from-scratch ShinkaEvolve optimization. The distribution is concentrated near zero for all methods, but Ψ-Evolve exhibits a heavier tail: at the ninth decile, its archive solutions correspond to 8 steps of from-scratch optimization, compared with 5 for CVT-MAP-Elites, 4 for POET, and 3 for context-free generation. Moreover, for approximately 20 evaluation problems, the from-scratch run does not recover the score of the best Ψ-Evolve archive solution within the allocated budget of 15 steps. These results characterize the *search-equivalent value* of the discovered solutions: from-scratch optimization is required to reproduce their zero-shot performance, rather than measuring acceleration from warm-starting the solver itself.

Evolution of the dual descriptor space To characterize how the internal semantic representation evolves during search, Figure 3 reports the growth of the tag vocabulary and its semantic clusters. Ψ-Evolve discovers the largest vocabulary, reaching 185 tags, compared with 153 for POET, 90 for CVT-MAP-Elites, and 49 for context-free generation. Its tag-cluster count also continues to



(a) (b)

Figure 4: **Ψ-Evolve allows for a better coverage of the problem and solution space.** t-SNE projections of the embeddings of the problems (a) and solutions (b). Ψ-Evolve provides a greater diversity of solutions compared to baselines.

increase throughout the generation budget, whereas the baselines largely saturate after roughly 80 problems. Thus, through ADDTAG and NEWDIRECTION, Ψ-Evolve continues to introduce new semantic distinctions instead of repeatedly exploiting a fixed vocabulary.

Structure of the problem space and solution space To visualize the space spanned by different sets of we problem, in Figure 4, we plot the embeddings of generated problems and their induced solutions. Quantitatively, comparing the cosine distance between points, Ψ-Evolve visibly exhibits the largest median problem-space radius beyond context-free LLM generation and both our CVT-MAP-Elites and POET baselines. This broader spread is corroborated by the independent 5NN analysis in Appendix D.6, where Ψ-Evolve also shows substantially lower local redundancy ($D_5 = 0.180$ versus 0.131, 0.056, and 0.062, respectively). The effect extends to solution space: Ψ-Evolve achieves the largest median radius (0.119), compared with 0.117, 0.084, and 0.073 for the baselines. Together with the literature-distance analysis in Appendix D.6, these results indicate that Ψ-Evolve explores a broader problem distribution while producing a correspondingly diverse set of solutions.

Blinded expert evaluation. To validate whether our discovered problems are coherent, meaningful, and potentially useful beyond our automatic metrics, we conducted a user study with six math and physics experts (Appendix E). The experts were tasked with blindly rating problems from either Ψ-Evolve, context-free LLM generation, or the actual literature. Ψ-Evolve problems not only received similar coherence and meaningfulness ratings to real ones, but were also rated as more non-trivial and more likely to yield reusable algorithmic insight beyond statistical significance. We also separately asked experts to evaluate parent-child problem variations and again recorded a significant rating increase, showing all four Ψ-Evolve operators being more meaningful and structurally consequential than a superficial-mutation control - further validating our algorithm's specific design.

3.2 GENERALIZATION TO OTHER SPACES

After the careful analysis of our method against baselines. We next report the average results in three domains to understand how Ψ-Evolve behaves in a variety of settings.

Discrete spectral design. Discrete spectral design concerns the construction of finite discrete signals or sequences with desirable frequency-domain properties. Intuitively, the domain asks how to arrange discrete symbols so that the resulting sequence has particular global regularities or cancellation properties. Problems vary in the size and dimensionality of the underlying group, the alphabet, structural restrictions on the signal, spectral constraints, and the quantity being optimized. These factors interact non-locally: changing a small number of symbols can affect the spectrum of the entire signal, making the relation between problem description and algorithmic difficulty difficult to anticipate. The crawled domain dataset is composed of 321 problems (Appendix A.2).

Geometry of numbers. Geometry of numbers studies discrete lattices embedded in continuous Euclidean space. A lattice can be viewed as a regularly repeating set of points generated by a small number of basis vectors. Problems in this domain ask for one or more lattices $L \subset \mathbb{R}^n$ satisfying structural constraints while optimizing a geometric quantity relative to a body K. Classical examples include finding lattices that pack spheres densely, cover space efficiently, or have many nearest neighbors. Problem difficulty can depend simultaneously on dimension, normalization, lattice class, symmetry, and the optimized invariant.We gather 93 literature-derived problems for this domain, further explanations on the domain are given in the Appendix A.3.

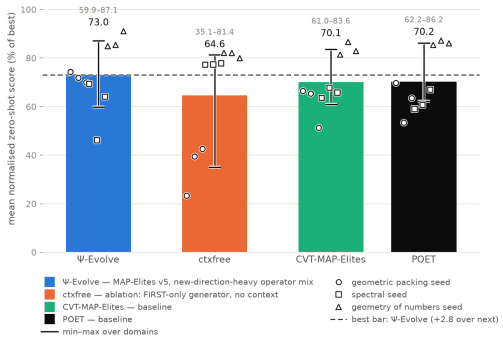
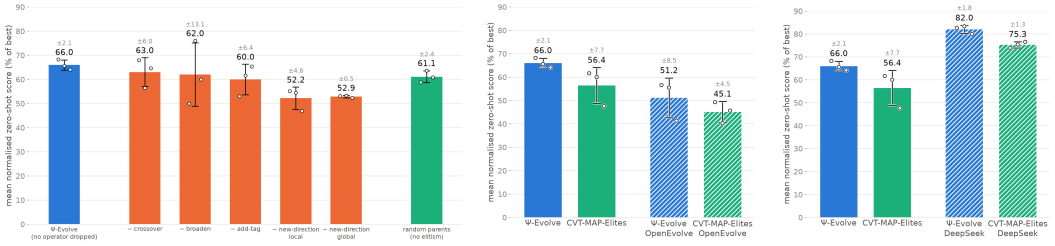


Figure 5: **0-shot results over three domains.** Ψ-Evolve improved on the best baseline by 4%

Results over three domains Figure 5 reports zero-shot transfer on held-out human-authored problems across the three domains. Averaged across domains, Ψ-Evolve achieves the highest normalized score (73), compared with 65–70 for the baselines. At the same time, its generated problems are the most distant from the literature set in embedding space, 0.651 on average, versus 0.617 for context-free generation and ∼ 0.390 for CVT-MAP-Elites and POET(see Appendix D.6). In geometric packing and geometry of numbers, Ψ-Evolve combines high literature distance with the strongest transfer performance, while the spectral domain exhibits a different trade-off, with context-free generation achieving higher transfer despite exploring less distant problems. Overall, Ψ-Evolve tends to explore farther from existing human-authored examples while preserving strong external utility (see Figure 10). In addition, Appendix D.9 shows that, across domains, Ψ-Evolve maintains high validity (99.0%) and the lowest problem duplication rate (9.0%). Its broader exploration is not obtained through malformed or redundant proposals.

Computational cost. Ψ-Evolve does not obtain its gains by using substantially more compute. Although semantic expansion requires additional short LLM calls, its solver-development budget remains comparable to the baselines. Averaged across domains, Ψ-Evolve uses 43.2M tokens and 11.4k cross-evaluations per run, versus 37.7M/10.4k for context-free generation and 59.0M/20.2k for CVT-MAP-Elites; POET is markedly more expensive at 117.6M tokens and 119.2k cross-evaluations (Appendix G).

3.3 Ablations



(a) **Component ablations.** (b) **Solver.** (c) **Model.**

Figure 6: **Ablations and robustness of Ψ-Evolve. Left:** removing individual variation operators or generality-based selection. **Middle:** replacing ShinkaEvolve with OpenEvolve. **Right:** replacing GPT-5.6 Luna with DeepSeek-V4.1-Flash. Error bars show variation across seeds.

Component ablations. To identify which components drive the performance of Ψ-Evolve, we ablate each problem-space variation mechanism as well as the generality-based selection pressure (Figure 6a). Removing any individual operator decreases held-out zero-shot performance relative to the complete method (66). The largest degradation occurs when disabling either mode of NewDirection, reducing the score to 52.2 without local expansion and 52.9 without global expansion.

Removing ADDTAG also produces a substantial drop to 60, whereas removing CROSSOVER or BROADEN has a smaller effect (63 and 62, respectively). This pattern supports the role of adaptive semantic exploration: operators that expand the representation or deliberately explore underrepresented semantic directions contribute most strongly to downstream transfer. Finally, replacing fitness-based elite selection with random parent selection decreases performance to 61.1, showing that the gains do not arise from semantic variation alone, but also from preferentially expanding problems whose induced solvers transfer across the discovered domain (Appendix D.2).

Robustness to the choice of solver and models In Figure 6b we plot the mean normalized score of Ψ-Evolve in the Geometric Packing domain when replacing ShinkaEvolve with OpenEvolve, an open-source re-implementation of AlphaEvolve (Novikov et al., 2025), and also compare to the CVT-MAP-Elites baseline using OpenEvolve as well. Remarkably, while switching the solver led to a performance decrease for both methods, Ψ-Evolve still managed to obtain a performance of 51.2, beating CVT-MAP-Elites using either solver together with all previously evaluated baselines. Replacing GPT-5.6 Luna with DeepSeek-V4.1-Flash (DeepSeek-AI, 2026) on geometric packing(Figure 6c) substantially improves absolute performance for both methods. Nonetheless, Ψ-Evolve still significantly outperformed CVT-MAP-Elites under both models by a similar margin (9.6 and 6.7 points, respectively), reaching a performance of 82. While the choice of underlying model expectedly affects the absolute quality of the generated problem–solution archive, these results suggest that the relative benefits of Ψ-Evolve are consistent across solvers and LLMs.

4 RELATED WORK

Quality-Diversity and open-ended search. QD methods aim to discover collections that are both high-performing and behaviorally diverse (Pugh et al., 2016). MAP-Elites (Mouret & Clune, 2015) relies on predefined behavioral descriptors, while CVT-MAP-Elites (Vassiliades et al., 2018) manages high-dimensional descriptor space, and methods such as TAXONS (Paolo et al., 2020) and AURORA (Grillotti & Cully, 2022) learn behavioral representations online. Ψ-Evolve differs in that the semantic representation of the problem space can itself expand, allowing new problem properties and directions of variation to emerge during search.

LLM-driven exploration. LLMs are increasingly used as evolutionary search operators but mostly rely on simple mutations or fixed descriptors (Lehman et al., 2022; Bradley et al., 2024; Pourcel et al., 2024), in contrast to Ψ-Evolve. Separately, DreamCoder (Ellis et al., 2021), FunSearch (Romera-Paredes et al., 2024), AlphaEvolve (Novikov et al., 2025), and ShinkaEvolve (Lange et al., 2026) demonstrate automated solution discovery for fixed problems (Appendix B.1, Table 1). Evolutionary search has also been applied directly to problem generation. QD methods generate diverse TSP and Knapsack instances in predefined feature spaces (Bossek & Neumann, 2022; Marrero et al., 2023), while POET (Wang et al., 2020), PAIRED (Dennis et al., 2020), and ACCEL (Parker-Holder et al., 2022) adapt generation through transfer, regret, or curriculum objectives. These methods typically assume a parameterized problem space, while our Ψ-Evolve can target domains with arbitrary dimensions of variation without requiring prior specification.

5 CONCLUSION

We introduced Ψ-EVOLVE, a quality-diversity framework for autonomous problem-space exploration that jointly discovers *where to explore* and *what to retain*. Rather than relying on predefined behavioral descriptors, Ψ-EVOLVE expands a semantic representation of the domain while defining problem quality through the transferability of the solver induced by each problem. Across three mathematical domains, this combination yields diverse archives whose solvers achieve stronger zero-shot transfer to held-out literature-derived problems than the considered baselines, including in regions distant from existing human-authored examples. These results suggest that problem generation can serve not only to create new tasks, but also to uncover reusable algorithmic structure. While transfer captures only one notion of scientific value and requires a shared executable interface, Ψ-EVOLVE provides a concrete step toward systems that ask not only which problems they can solve, but which problems are worth solving because their principles and solutions may uncover new knowledge and even remain useful to accelerate future LLM-driven discovery itself.

AI USE STATEMENT

(This section is **required** and does not count toward the page limit.)

In this work, we used generative AI tools for structuring the code, reformulating some prompt used by our method, editing, and checking the grammar spelling in the paper. We have not used generative AI tools to generate the idea for the paper. We reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies the automated generation and exploration of mathematical optimization problems. The experiments are conducted entirely in mathematical domains, and we do not identify direct harmful applications arising from the problems or solutions considered in this study. More broadly, methods for automated problem generation could potentially be applied to domains with different ethical or dual-use considerations; such applications are outside the scope of this work.

The paper also includes a blinded expert evaluation in which researchers rated mathematical problem statements and problem variations. Participants were not exposed to harmful or sensitive content, and the analysis reports ratings only in aggregate without relying on sensitive personal attributes. The evaluation was designed solely to assess properties of the generated mathematical problems, including coherence, meaningfulness, non-triviality, and potential for reusable algorithmic insight.

REPRODUCIBILITY STATEMENT

We provide the complete implementation of Ψ-Evolve and the experimental pipeline in an accompanying GitHub repository. The supplementary material documents all hyperparameters used for Ψ-Evolve and the baselines, the problem-generation and semantic-descriptor prompts, the solver-development procedure, evaluation protocols, and computational budgets. We also report the random seeds and model configurations used in the experiments. We release the literature-derived datasets used for evaluation in all three domains, together with the procedure used to crawl, process, validate, and deduplicate these problems, allowing the datasets to be reproduced or extended to new sources. The domain specifications, validity checks, solver interfaces, and evaluation functions required to generate and evaluate new problems are included with the code. Finally, the supplementary material provides detailed descriptions of the baselines, ablations, archive construction, and cross-problem evaluation procedure. Together, these resources are intended to support reproduction of the reported experiments as well as application of Ψ-Evolve to new problem domains.

REFERENCES

Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: Ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pp. 49–60. ACM, 1999. doi: 10.1145/304182.304187.

Franz Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991. doi: 10.1145/116873.116880.

Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023. doi: 10.1038/s41586-023-06792-0.

Jakob Bossek and Frank Neumann. Exploring the feature space of TSP instances using quality diversity. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, pp. 186–194. Association for Computing Machinery, 2022. doi: 10.1145/3512290.3528851.

Herbie Bradley, Andrew Dai, Hannah Teufel, Jenny Zhang, Koen Oostermeijer, Marco Bellagente, Jeff Clune, Kenneth O. Stanley, Grégory Schott, and Joel Lehman. Quality-diversity through AI feedback. In *The Twelfth International Conference on Learning Representations*. OpenReview.net, 2024. URL https://openreview.net/forum?id=owokKCrGYr.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. *arXiv preprint arXiv:2609.19969*, 2026. URL https://arxiv.org/abs/2609.19969.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre M. Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc., 2020.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. DreamCoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI '21, pp. 835–850. Association for Computing Machinery, 2021. doi: 10.1145/3453483.3454080.

Luca Grillotti and Antoine Cully. Unsupervised behavior discovery with quality-diversity optimization. *IEEE Transactions on Evolutionary Computation*, 26(6):1539–1552, 2022. doi: 10.1109/TEVC.2022.3159855.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z. Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651:607–613, 2026. doi: 10.1038/s41586-025-09833-y.

Edward Hughes, Michael Dennis, Jack Parker-Holder, Feryal Behbahani, Aditi Mavalankar, Yuge Shi, Tom Schaul, and Tim Rocktäschel. Open-endedness is essential for artificial superhuman intelligence. *arXiv preprint arXiv:2406.04268*, 2024. doi: 10.48550/arXiv.2406.04268. URL https://arxiv.org/abs/2406.04268.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=lKEdGCoDNC.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O. Stanley. Evolution through large models. *arXiv preprint arXiv:2206.08896*, 2022. URL https://arxiv.org/abs/2206.08896.

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.

Alejandro Marrero, Eduardo Segredo, Emma Hart, Jakob Bossek, and Aneta Neumann. Generating diverse and discriminatory knapsack instances by searching for novelty in variable dimensions of feature-space. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '23, pp. 312–320. Association for Computing Machinery, 2023. doi: 10.1145/3583131.3590504.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015. URL https://arxiv.org/abs/1504.04909.

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.

OpenAI. GPT-5.6 System Card. Technical report, OpenAI, July 2026. URL https://deploymentsafety.openai.com/gpt-5-6.

Giuseppe Paolo, Alban Laflaquière, Alexandre Coninx, and Stéphane Doncieux. Unsupervised learning and exploration of reachable outcome space. In *2020 IEEE International Conference on Robotics and Automation*, ICRA, pp. 2379–2385. IEEE, 2020. doi: 10.1109/ICRA40945.2020.9196819.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 17473–17498. PMLR, 2022. URL https://proceedings.mlr.press/v162/parker-holder22a.html.

Julien Pourcel, Cédric Colas, Gaia Molinaro, Pierre-Yves Oudeyer, and Laetitia Teodorescu. ACES: Generating a diversity of challenging programming puzzles with autotelic generative models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-2160. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/7d0c6ff18f16797b92e77d7cc95b3c53-Abstract-Conference.html.

Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016. doi: 10.3389/frobt.2016.00040.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025. URL https://arxiv.org/abs/2504.21801.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Zhihong Shao, Yuxiang Luo, Chengda Lu, Z. Z. Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. DeepSeekMath-V2: Towards self-verifiable mathematical reasoning. *arXiv preprint arXiv:2511.22570*, 2025. URL https://arxiv.org/abs/2511.22570.

Olivier Sigaud, Gianluca Baldassarre, Cedric Colas, Stephane Doncieux, Richard Duro, Pierre-Yves Oudeyer, Nicolas Perrin-Gilbert, and Vieri Giuliano Santucci. A definition of open-ended learning problems for goal-conditioned agents. *arXiv preprint arXiv:2311.00344*, 2023. doi: 10.48550/arXiv.2311.00344. URL https://arxiv.org/abs/2311.00344.

Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation*, 22(4):623–630, 2018. doi: 10.1109/TEVC.2017.2735550.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (POET): Endlessly generating increasingly complex and diverse learning environments and their solutions. *CoRR*, abs/1901.01753, 2019. URL https://arxiv.org/abs/1901.01753.

Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeffrey Clune, and Kenneth O. Stanley. Enhanced POET: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9940–9951. PMLR, 2020. URL https://proceedings.mlr.press/v119/wang20l.html.

APPENDIX TABLE OF CONTENTS

A Dataset collection **14**

A.1 Geometric Packing 14

A.2 Discrete Spectral Design 15

A.3 Geometry of Numbers 17

B Ψ-Evolve complementary details **19**

B.1 Comparison to literature baselines 19

B.2 Dual representation of the problem space 20

B.3 Adaptive Archive Partition and Tag Clustering 21

B.4 Constraint-Preserving Problem Generation 21

B.5 Sampling Mutation Operator 22

B.6 Ψ-Evolve Algorithm 22

C Baselines **24**

D Additional Experiments **28**

D.1 Hyperparameter search 28

D.2 Component Ablations 28

D.3 Effect of the online descriptor expansion 29

D.4 Effect of the broadening operator 30

D.5 Effect of the CVT-MAP-Elites descriptor space. 30

D.6 Distance between generated and crawled problems 31

D.7 Retrieval-based use of the solver archive. 32

D.8 Relationship Between Generality Fitness and External Transfer 33

D.9 Validity of generated problems 34

E Blinded expert evaluation of generated problems **35**

E.1 Study design 35

E.2 Results 37

E.3 Takeaways 38

F Prompt Templates and LLM Roles **38**

F.1 Problem designer 38

F.2 NEW-DIRECTION tag inventors 43

G Generation cost and scaling **44**

H Hyperparameters **45**

A Dataset collection

A.1 Geometric Packing

Geometric packing provides a broad family of optimization problems combining continuous geometry, combinatorial decisions, and heterogeneous constraints under a common solver interface. A problem specifies a finite set of rigid bodies that must be arranged inside a bounded region of $\mathbb{R}^d$. A placement may determine the selected body type, its center, its orientation, and its scale. Problems differ in the geometry of the objects and container, the admissible transformations, the feasibility constraints, and the objective being optimized. This common representation allows us to treat problems that are traditionally studied as separate packing variants as members of a single computational domain.

Problem representation. Each problem is characterized by five principal components: the *container*, the set of *objects*, the allowed geometric *structure*, the problem-specific *constraints*, and the *objective*. Containers include, among others, boxes, balls, polytopes, ellipsoids, convex level sets, non-convex regions obtained by removing obstacles, and regions specified through an implicit membership function. Objects include balls, boxes, ellipsoids, convex polytopes, capsules, non-convex polygons, polyomino-like shapes, unions of convex components, and arbitrary shapes represented through a membership test. The ambient dimension d is itself problem dependent.

A candidate placement of object k can be written as

$$x_k = (i_k,\, c_k,\, \theta_k,\, s_k)\,, \tag{2}$$

where i_k indexes the selected object type, $c_k \in \mathbb{R}^d$ is its center, $\theta_k \in \mathbb{R}^{d(d-1)/2}$ parameterizes rotations in the coordinate planes, and $s_k > 0$ is a scale factor. Individual problems may restrict any of these degrees of freedom, for example by enforcing congruent objects, forbidding rotation, fixing the scale, imposing non-overlap, or requiring minimum separation.

Uniform solver contract. All geometric-packing problems expose the same solver interface,

$$\mathrm{Solve}(\texttt{inside}, \texttt{shapes}, \texttt{bounds}) \to \{x_1, \ldots, x_n\}, \tag{3}$$

where `inside` evaluates containment of a placement, `shapes` defines the available bodies, and `bounds` gives an axis-aligned bounding box for the search region. Problem-specific feasibility conditions and objectives are implemented outside the solver through a common evaluation contract. Consequently, a solver produced for one problem can be executed without modification on any other problem using the same contract. This property is essential for constructing the cross-problem performance matrix used by our method.

The contract is intentionally permissive by default and each problem specializes it through explicit constraints. As a result, classical congruent-body packing, rotated polygon nesting, three-dimensional container loading, max–min dispersion, and finite covering problems can be represented as different configurations of the same interface.

Objectives and constraints. The benchmark supports heterogeneous optimization objectives while enforcing a common convention that larger scores correspond to better solutions. Examples include maximizing packing density or packed volume, maximizing minimum separation, minimizing the size of an enclosing container through an equivalent maximization score, and satisfying finite covering objectives. Feasibility conditions can constrain object count, orientation, scale, overlap, containment, separation, or coverage. Importantly, conceptually similar textual formulations are not automatically treated as distinct problems: differences in the declared constraints and objective determine the actual optimization problem.

Geometric verification. The evaluator operates in two verification regimes. Whenever possible, it uses an algebraic geometric representation for which feasibility checks can be computed exactly. Examples include closed-form volumes for standard bodies, support-function-based containment

in boxes and polytopes, exact overlap tests for several primitive pairs, and GJK-based intersection testing for general convex bodies. Some geometric computations are conservative rather than exact; in these cases the checker may reject a feasible placement but is designed not to accept an infeasible one.

For geometries that cannot be represented by this algebraic vocabulary, the contract provides an implicit representation through membership functions. Containment and overlap are then checked using a deterministic finite witness set sampled from the body. We refer to this as the *sampled* verification regime. Because violations may theoretically occur between witness points, sampled verification is explicitly recorded in the evaluation metadata rather than being presented as an exact geometric certificate. This distinction lets us retain broad geometric expressivity without conflating approximate and exact verification.

Validity checks. Generated problems are subjected to a common collection of guardrails before entering the search. These checks verify that the instance has the expected structure, that the objective is finite and follows the common score convention, and that basic perturbations of candidate solutions behave consistently with the declared constraints. Probe solutions include the provided initial solution, perturbed placements, reduced arrangements, randomized placements, and degenerate cases. Local modifications include translation, rotation, resizing, adding, removing, and relocating objects. These checks are intended to detect malformed generated objectives or constraints before solver optimization and cross-evaluation.

Scope and limitations. The contract is deliberately broad but does not claim to represent every problem commonly described as geometric packing. In particular, its containers are bounded, so genuinely infinite lattice packing and half-infinite strip packing are outside the current representation. Likewise, deformable objects, physical stability constraints, and some specialized combinatorial objectives require additional machinery. Such cases are recorded as unsupported rather than approximated by a different problem. Cross-problem solver evaluation is also performed only among problems sharing the same solver contract, since transfer scores between incompatible interfaces would not be meaningful.

Overall, this domain is useful for our study because large semantic variation can arise while the computational interface remains fixed. Two problems can differ in dimension, topology, object geometry, admissible transformations, constraints, and optimization objective while still admitting direct cross-evaluation of their solvers. Geometric packing therefore provides a controlled setting in which to test whether our method can progressively discover meaningful axes of variation, preserve diverse problems, and identify problems whose induced solvers generalize across structurally different regions of the domain.

A.2 Discrete Spectral Design

Discrete spectral design provides a broad class of optimization problems in which a finite signal is constructed to satisfy prescribed spectral, correlation, or structural properties. Such problems arise in several sequence-design settings and can differ substantially in their mathematical formulation while retaining a common computational interface. This combination of heterogeneous objectives and shared representation makes spectral design a useful test domain for studying problem-space exploration and solver transfer.

Problem representation. Let

$$G = \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_d} \tag{4}$$

be a finite Abelian group and let $\mathcal{A} \subseteq \mathbb{C}$ denote an alphabet of admissible symbol values. A candidate solution is a signal

$$s : G \to \mathcal{A}. \tag{5}$$

The search space therefore contains $|\mathcal{A}|^{|G|}$ signals when the alphabet is finite, before accounting for additional structural constraints. The representation also permits alphabets that are not simply binary, allowing problems to differ in both combinatorial structure and signal geometry.

A problem may additionally restrict the admissible signals through a problem-specific predicate

$$\mathrm{Adm}(s) \in \{0, 1\}. \tag{6}$$

Such restrictions define the structural component of the problem and can impose properties on the symbols or on the signal as a whole.

Each problem in the domain is characterized through five components: the *group*, the *alphabet*, the *structure*, the *constraints*, and the *objective*. The group determines the size, dimension, and factorization of the signal domain. The alphabet specifies the allowed signal values, including binary, ternary, polyphase, unimodular, integer-valued, or continuous choices. Structural restrictions determine the admissible subset of signals, while additional hard constraints impose requirements on their spectral or correlation properties. Finally, each problem defines a single scalar objective to be maximized.

Spectral and correlation objectives. For a signal s, let $\widehat{s}$ denote its discrete Fourier transform on G. A spectral-design problem optimizes a functional

$$\Psi(\widehat{s}) \tag{7}$$

subject to the problem's alphabet and admissibility constraints. By the Wiener–Khinchin relation, the same class of objectives can often be expressed through the periodic autocorrelation coefficients

$$c_k = \sum_{j \in G} \overline{s_j}\, s_{j+k}, \qquad k \in G. \tag{8}$$

This common representation accommodates objectives that penalize correlation sidelobes, reward prescribed correlation structure, or constrain spectral energy over selected regions.

An important practical issue is that closely related objectives are often described using different terminology across the literature. For example, integrated sidelobe level and its common aliases refer to the same underlying quantity, while merit factor is a monotone transformation of sidelobe energy. In contrast, weighted sidelobe objectives or objectives imposing particular correlation zones define genuinely different optimization problems. We therefore distinguish aliases and monotone reformulations from refinements that change the feasible set or objective itself.

Uniform solver contract. All problems expose the same solver interface,

$$\mathrm{Solve}(\texttt{admissible}, \texttt{alphabet}, \texttt{group}) \to (s_g)_{g \in G}. \tag{9}$$

Here, `group` specifies the finite Abelian group, `alphabet` specifies the admissible symbol values, and `admissible` provides the structural feasibility condition. The problem-specific spectral constraints and objective are evaluated through a shared, non-evolvable contract.

This common interface is essential for our experiments. A solver obtained by optimizing one spectral-design problem can be executed without modification on another problem in the same domain. We can therefore construct the cross-problem performance matrix used by Ψ-Evolve and measure whether the algorithmic structure learned from one problem transfers to another.

Validity and evaluation. To make scores comparable across automatically generated problems, every problem follows the convention that valid solutions receive finite, non-negative scores and that larger values always indicate better performance. Failed evaluations receive score zero. Objectives whose natural form is a minimization are therefore transformed into an equivalent maximization score when necessary.

Before a generated problem is admitted to the experimental domain, it is subjected to a collection of executable guardrails. These checks verify that the group and instance have a valid and tractable representation, that the problem specification defines both constraints and an objective, and that repeated construction of the same instance is deterministic. The initial signal supplied with the problem must satisfy the declared constraints and obtain a strictly positive finite score. The raw objective is additionally evaluated on multiple feasible probes to ensure that it remains finite and non-negative and that it is not constant over the feasible set.

We also record softer diagnostics concerning optimization headroom and solver runtime. In particular, a short local search is used to test whether the provided initial solution can be improved, and the initial solver is checked against the same execution-time regime later used during cross-evaluation. These tests are intended to detect malformed or degenerate generated problems before expensive solver optimization.

Redundancy control. Spectral-design problems can appear textually different while defining nearly the same optimization task. Conversely, very similar descriptions may differ by a hard constraint that substantially changes which signals are feasible. For this reason, semantic embedding similarity is used only to identify candidate duplicates, rather than to decide equivalence directly. Highly similar pairs are additionally compared using their problem statements, declared constraints, and objectives, and are removed only when their difference is judged marginal and a solver for one would necessarily solve the other. This distinction is important for correlation-design problems in which a seemingly small additional constraint can define a substantially different feasible set.

Role in our evaluation. Discrete spectral design is particularly challenging for domain illumination because meaningful axes of variation are not reducible to a single parameter such as sequence length. Problems may vary simultaneously in group structure, alphabet, admissible signal family, spectral constraints, and objective, while different formulations can still encode equivalent optimization criteria. At the same time, their shared solver contract permits direct empirical comparison through cross-evaluation. The domain therefore tests both aspects of our method: whether it can progressively discover useful semantic coordinates for an initially unstructured problem space, and whether its generality objective identifies problems that induce solvers transferable across distinct regions of that space.

A.3 Geometry of Numbers

Geometry of numbers studies the interaction between discrete lattices and continuous geometry. We use it as a broad family of optimization problems in which the solution objects have a common algebraic representation while the objectives, admissible lattice classes, dimensions, and underlying geometries can vary substantially.

Problem representation. A problem asks for a lattice

$$L \subset \mathbb{R}^n \tag{10}$$

of rank $m \le n$, represented by a generator matrix

$$B = \begin{bmatrix} b_1^\top \\ \vdots \\ b_m^\top \end{bmatrix} \in \mathbb{R}^{m \times n}, \tag{11}$$

whose linearly independent rows form a basis of L. Given an admissible lattice class $\mathcal{C}$, a gauge body K, and a geometric invariant Φ, a typical optimization problem can be written abstractly as

$$\max_{L} \quad \Phi(L; K) \qquad \text{s.t.} \qquad L \in \mathcal{C}, \quad c_r(L) = 1, \;\; r = 1, \dots, R, \tag{12}$$

where c_r denotes problem-specific structural or normalization constraints.

We organize the domain according to five components: the *invariant* being optimized, the structural lattice class, the *gauge* defining the underlying geometry, additional hard *constraints*, and the type of *task*. These components are not treated as fixed behavioral axes by our exploration method; rather, they describe the expressivity of the benchmark contract.

The invariant may correspond, for example, to the Hermite invariant, successive minima, lattice packing or covering density, kissing number, packing–covering trade-offs, theta-series quantities, Epstein zeta quantities, Rankin or Bergé–Martinet invariants, quantization objectives, Voronoi structure, automorphism groups, or products of linear forms. Admissible classes include integral, even, unimodular, modular, isodual, perfect, eutactic, or symmetry-restricted lattices. The gauge K may be a Euclidean ball, an ℓ_p ball, a cube, a cross-polytope, an ellipsoid, a symmetric polytope, or another symmetric gauge.

Uniform solver contract. Every problem uses the same solver interface,

$$\text{Solve}(\texttt{admissible}, \texttt{gauge}, \texttt{context}) \longrightarrow \{B_1, \ldots, B_q\}, \tag{13}$$

where `admissible` determines whether a candidate basis satisfies the problem-specific lattice class, `gauge` provides the ambient dimension and geometric norm, and `context` contains immutable auxiliary data such as a reference lattice or a prescribed symmetry group.

A solution is formally a *family* of lattices rather than necessarily a single lattice. Most optimization problems require $q = 1$, but this representation also supports enumeration and classification-style tasks, in which the solver attempts to return several pairwise inequivalent lattices. This common representation is important for our experiments because any solver produced for one problem can be executed on any other problem sharing the same contract.

Constraints and normalization. The base contract is permissive: generator matrices may have arbitrary real entries, determinant, and rank within the supported range unless the problem specifies otherwise. Individual problems restrict this space through explicit conditions, including integrality, evenness, unimodularity, determinant or minimum normalization, prescribed rank, lattice index, prescribed vectors, or symmetry requirements.

A common normalization is particularly important for lattice optimization. Without one, many geometric invariants can be changed trivially by uniformly rescaling a basis. The benchmark therefore expresses determinant, covolume, or minimum constraints explicitly rather than relying on an implicit convention.

All evaluation objectives follow the global convention used by our framework: valid solutions receive finite non-negative scores and larger scores correspond to better solutions. Quantities that are naturally minimized, such as covering radius or quantization error, are therefore mapped to equivalent monotone maximization scores.

Optimization and enumeration tasks. Most problems ask the solver to optimize an invariant over an admissible class of lattices. The contract additionally supports problems whose goal is to discover multiple inequivalent lattices satisfying a property. In such cases, the solution family is constrained to contain pairwise inequivalent candidates and the objective can reward the number of valid candidates found.

This distinction allows the same interface to represent both continuous optimization over lattice bases and discrete searches over distinguished lattice classes. We do not, however, claim that such enumeration proves completeness: the evaluator can verify the lattices returned by a solver but cannot in general certify that no additional lattices exist.

Exact and approximate evaluation. The evaluator distinguishes between quantities that can be computed exactly under the contract and quantities requiring a deterministic approximation. Basic lattice operations—including Gram matrices, covolume, dual lattices, and lattice reduction—are computed directly. Short-vector enumeration is used to obtain minimal vectors and related quantities, which in turn supports the computation of successive minima, kissing numbers, Hermite-type invariants, and packing quantities within the supported regime.

Structural properties such as integrality, evenness, unimodularity, and determinant constraints are checked directly. The contract also includes verification procedures for more specialized properties such as perfection, eutaxy, isometry, modularity, isoduality, and prescribed automorphisms. Some of these procedures may return an undecided result when a bounded search cannot complete, rather than treating an incomplete computation as a certificate.

For several more expensive quantities, evaluation is restricted to a declared computational approximation. For example, high-dimensional or non-Euclidean covering radii are estimated using a deterministic set of candidate points in the fundamental cell, while normalized second moments are estimated by a fixed quasi-Monte-Carlo sample. Infinite quantities such as the Epstein zeta function or theta series at a real argument are evaluated only after introducing an explicit truncation radius. These approximations are part of the problem definition and are applied identically to every solver.

Each problem records whether its evaluation belongs to the exact or sampled regime, allowing experiments to analyze the two groups separately.

Method	Problem generation	Semantic descriptors	Online descriptor discovery	Solver-transfer quality
CVT-MAP-Elites	✓	✗	✗	✗
TAXONS / AURORA	✗	✗	✓	✗
POET	✓	✗	✗	✓
QDAIF	✓	✓	✗	✗
ACES	✓	✓	✗	✗
Ψ-Evolve	✓	✓	✓	✓

Table 1: Positioning of Ψ-Evolve relative to closely related quality-diversity and open-ended search methods. "Online descriptor discovery" denotes methods that can introduce previously unspecified semantic dimensions during the search itself. "Solver-transfer quality" denotes evaluating a generated problem through the cross-problem transferability of the solver obtained by optimizing on that problem.

Verification against known lattices. The contract is additionally checked against classical reference lattices and known values rather than relying solely on internal consistency. Reference cases include integer lattices $\mathbb{Z}^n$, root lattices such as A_n and D_n, and exceptional lattices such as E_6, E_7, and E_8. These provide known values for quantities such as Hermite invariants, kissing numbers, packing densities, theta-series coefficients, covering quantities, and automorphism groups.

Such reference cases are also useful when constructing diagnostic candidate solutions. Random real matrices alone provide poor probes for this domain, since many lattice problems impose measure-zero algebraic constraints such as integrality, unit determinant, or prescribed symmetry. The evaluation guardrails therefore include both perturbations of the initial lattice and structured reference lattices satisfying representative constraints.

Scope and limitations. The contract deliberately focuses on finite-dimensional lattice problems for which candidate lattices and their objective values can be evaluated computationally. Our experiments restrict the ambient dimension to a bounded range because several exact procedures depend on short-vector enumeration, whose cost grows rapidly with dimension.

Several important topics from the geometry-of-numbers literature consequently remain outside the present benchmark. These include asymptotic statements, bounds whose optimization variables are functions rather than lattices, general periodic packings, random-lattice statements, some constructions such as laminated or ideal lattices, and classification results whose completeness cannot be verified from a finite candidate family. The current gauge representation also focuses on symmetric bodies. Unsupported concepts are treated as gaps in the domain contract rather than being silently replaced by an easier surrogate problem.

Why this domain is useful for problem-space exploration. Geometry of numbers provides an especially challenging test of pur method because problems that share an identical low-level solution representation can encode substantially different mathematical structure. Changing the dimension, normalization, gauge, lattice class, symmetry group, or optimized invariant can radically alter which algorithms are effective, while the common generator-matrix interface still permits direct cross-evaluation.

B Ψ-EVOLVE COMPLEMENTARY DETAILS

B.1 COMPARISON TO LITERATURE BASELINES

Comparison with related approaches. Table 1 summarizes the main design differences between Ψ-Evolve and closely related quality-diversity and open-ended search methods. The comparison highlights two aspects that are central to our setting: how diversity is represented during search, and how the quality of a generated problem is evaluated. Classical QD methods such as CVT-MAP-Elites rely on a predefined behavioral representation, while methods such as TAXONS and AURORA learn behavioral representations from experience. However, these methods do not explic-

Component	Representation / update	Role
Problem embedding	$E(p) \in \mathbb{R}^d$; fixed map	Archive geometry
Semantic vocabulary	$D_t = \{\tau_1, \ldots, \tau_{m_t}\}$; expands	Semantic descriptors
Problem tags	$T_t(p) \subseteq D_t$; updated	Problem characterization
Tag clusters	$\{\phi(\tau)\}$ clustered online	Semantic directions
Transfer matrix	$S_{ji}^{(t)} = \mathrm{Eval}(a_j, p_i)$; expands	Solver transfer
Generality fitness	$F_t(p_j)$; recomputed	Elite selection

Table 2: Internal representations maintained by Ψ-Evolve. A fixed problem-embedding space organizes the archive, while the semantic vocabulary expands during search and guides semantic variation. Cross-problem solver transfer provides the quality signal.

itly construct an interpretable semantic vocabulary of problem properties. Open-ended environment generation methods such as POET instead co-evolve problems and their corresponding solutions, but operate within a predefined environment representation and do not autonomously discover new semantic dimensions along which the problem space can be organized.

LLM-based QD approaches such as QDAIF and ACES move closer to our setting by using semantic information during generation and evaluation. In particular, ACES associates generated problems with semantic descriptors using an LLM, but the descriptor vocabulary itself is specified by the researcher before search. The LLM therefore determines how a problem should be characterized with respect to an existing set of semantic dimensions, rather than deciding which semantic dimensions should exist. In contrast, Ψ-Evolve treats the semantic representation itself as part of the search process: new descriptors can be proposed autonomously when the current vocabulary does not capture a promising direction of variation, and the descriptor set can therefore expand throughout the run.

A second distinction concerns the notion of quality used to guide exploration. Ψ-Evolve does not evaluate a generated problem solely according to its intrinsic difficulty, novelty, or the performance obtained on that problem. Instead, the quality of a source problem is defined through the cross-problem transferability of the solver independently optimized on it. Consequently, diversity and quality play complementary roles: the evolving semantic vocabulary encourages exploration of distinct problem properties, while the transfer objective preferentially retains source problems whose induced solvers capture algorithmic structure that is reusable elsewhere in the domain. The combination of *online semantic descriptor discovery* and *solver-transfer-based problem quality* is the main distinction of Ψ-Evolve represented in Table 1.

B.2 Dual representation of the problem space

Table 2 clarifies the different representations maintained by Ψ-Evolve. The method does not treat each semantic descriptor as an independent coordinate of a conventional MAP-Elites grid. Instead, it maintains two complementary representations of the discovered problem space.

First, each problem p is mapped through a fixed text-embedding function $E(p)$ into a continuous latent space. This representation provides a generic geometric substrate for partitioning the discovered problems and maintaining the archive. The embedding function itself is fixed throughout a run; what changes is the set of discovered problems occupying this space and, consequently, the archive constructed over them.

Second, Ψ-Evolve maintains an explicit semantic vocabulary $D_t = \{\tau_1, \ldots, \tau_{m_t}\}$. Unlike the continuous embedding space, this representation is open-ended: new descriptors can be introduced during search when the current vocabulary does not capture a proposed direction of variation. Each problem is associated with a set of descriptors $T_t(p) \subseteq D_t$, and previously discovered problems can be retroactively checked when the vocabulary expands. Embeddings of the semantic descriptors are additionally clustered into K_t, providing a structured view of the semantic directions currently represented in the domain. These clusters are used by the semantic variation mechanisms, particularly AddTag and NewDirection, to target underrepresented regions or propose previously unrepresented concepts.

The remaining representations determine problem quality rather than diversity. The cross-evaluation matrix $S^{(t)}$ records how the solver induced by each discovered problem performs on the other compatible problems. From this matrix, Ψ-Evolve recomputes the generality fitness F_t, which determines which problems are retained as high-quality archive representatives. Thus, archive geometry, semantic exploration, and transfer-based quality are distinct but interacting components: the problem embedding organizes the discovered set, the expandable semantic vocabulary directs exploration, and cross-problem solver transfer determines which problems are preferentially retained.

B.3 Adaptive Archive Partition and Tag Clustering

Problem archive. The problem archive is organized as a recursive partition of the problem-embedding space. Each problem statement is mapped to a fixed embedding $E(p)$, and cosine distance is used throughout the archive. The archive initially consists of a single cell containing all discovered problems. Each leaf cell stores one elite, defined as the problem with the highest current generality fitness among the problems assigned to that cell.

The archive is updated once per search turn, after the complete batch of new problems has been solved and cross-evaluated. Each new problem is first assigned to an existing leaf. Starting from the root, it follows the child whose centroid is closest to its embedding until reaching a leaf. This assignment does not modify the existing partition.

A leaf is considered for splitting when it contains more than $n_{\text{crit}} = 5$ problems. Its members are clustered with OPTICS (Ankerst et al., 1999) using cosine distance, with `min_samples`$= 2$, $\xi = 0.05$, and `max_eps` equal to the largest pairwise distance within the cell. If OPTICS identifies at least two clusters, we compute one centroid per cluster as the normalized mean of its member embeddings. These centroids define the children of the current cell. All problems in the parent cell, including points marked as noise by OPTICS, are then assigned to their nearest centroid. This corresponds to a Voronoi partition induced by the cluster centroids.

Splitting is applied recursively until no eligible leaf can be divided further. If OPTICS finds fewer than two clusters, the cell is left unchanged. Once a cell is split, both the split and its centroids remain fixed. Future problems are routed through the resulting tree, and only leaf cells may subsequently be split. The archive therefore grows incrementally rather than being reconstructed from scratch at every turn.

After the partition is updated, the representative of every leaf is recomputed using the current fitness values. Fitness determines which problem is retained as the elite of each cell, but does not affect the geometry of the partition.

Semantic tag clusters. The tag clusters used by NewDirection are maintained separately from the problem archive. Whenever this operator is selected, the embeddings of all current semantic tags are clustered from scratch with OPTICS using the same cosine distance settings. These clusters group related semantic features and are used only to guide problem generation; they do not define the archive cells.

In the local mode of NewDirection, one tag cluster is selected and the model proposes a new feature along the same conceptual direction. In the global mode, the current clusters are summarized and the model proposes a feature that is semantically distinct from the existing groups. The problem archive thus provides a persistent partition for diversity preservation, while the tag clustering provides a dynamic representation for proposing new directions of exploration.

B.4 Constraint-Preserving Problem Generation

Semantic diversity is useful only if generated candidates remain members of the target domain. Every candidate problem $\hat{p}$ produced by a variation operator is therefore subjected to the domain constraints before it can enter the evaluation pipeline:

$$\hat{p} \xrightarrow{\text{Validate}_{\mathcal{C}}} \begin{cases} p, & \mathcal{C}(p) = 1, \\ \text{repair or reject}, & \text{otherwise.} \end{cases} \tag{14}$$

When a violated constraint admits a well-defined repair, the candidate may be regenerated or repaired and validated again. Candidates that cannot be made valid are discarded and do not enter the archive.

This separation between *generation* and *validation* is important because the generator is not required to provide a complete symbolic model of $\mathcal{P}_{\mathcal{C}}$. It only needs to propose candidate problems, while the validity interface defines membership in the domain. Whenever a constraint can be tested programmatically, the programmatic checker is treated as authoritative rather than relying on the generator's self-assessment.

After validation, a naive solver seed associated with the new problem is constructed and independently optimized:

$$a_p = \mathrm{Solve}\left(p; \theta_p^{(0)}\right), \tag{15}$$

where $\theta_p^{(0)}$ depends only on the new problem and not on an evolved parent solver. This independent initialization ensures that the resulting transfer profile measures what the solver-development process can discover *from p itself*.

All solvers are evaluated under the same evaluation protocol and resource limit. Because the archive quality in Equation 1 is determined entirely by cross-evaluation, consistent evaluation budgets are necessary: an evaluation terminated for resource reasons is assigned score zero and can otherwise alter both the best-known target score b_i and the relative ordering of elites.

B.5 Sampling Mutation Operator

Let

$$\boldsymbol{\pi} = (\pi_{\mathrm{cross}}, \pi_{\mathrm{broad}}, \pi_{\mathrm{tag}}, \pi_{\mathrm{new}}) \tag{16}$$

denote the nominal operator probabilities. Not every operator is always available. In particular, CROSSOVER requires at least two archive representatives, while ADDTAG requires the existence of at least one known tag not already assigned to the base problem. Let $\mathcal{O}_t(p_b) \subseteq \mathcal{O}$ be the currently available set. We sample

$$\Pr(o \mid p_b, t) = \frac{\pi_o}{\sum_{o' \in \mathcal{O}_t(p_b)} \pi_{o'}}, \qquad o \in \mathcal{O}_t(p_b). \tag{17}$$

If all available operators have zero nominal mass, we fall back to a uniform distribution over $\mathcal{O}_t(p_b)$.

Let

$$\boldsymbol{\rho} = (\rho_{\mathrm{local}}, \rho_{\mathrm{global}}) \tag{18}$$

denote the probability distribution over the two NEWDIRECTION modes. Both $\boldsymbol{\pi}$ and $\boldsymbol{\rho}$ are treated as search hyperparameters.

Together, these operators create two interacting search processes:

$$\underbrace{\text{search over problems within the current representation}}_{\text{CROSSOVER, BROADEN, ADDTAG}} \quad \text{and} \quad \underbrace{\text{search over the representation itself}}_{\text{NEWDIRECTION}}. \tag{19}$$

B.6 Ψ-EVOLVE Algorithm

Algorithm 1 summarizes the complete procedure. The algorithm begins with one valid problem generated only from the domain constraints. It then alternates between problem-space variation, independent solver optimization, cross-evaluation, semantic representation updates, and quality-diversity archive refinement.

The implementation supports batched search turns. Problems in a batch are generated from the archive state at the beginning of the turn and their solver-development runs can be executed concurrently. After all new solvers have been obtained, a single cross-evaluation stage updates the required entries of S. The best-known target scores and the generality of *all* problems are then recomputed before the archive is updated.

The output of Ψ-Evolve is therefore richer than a conventional set of optimized solutions. It comprises: (i) a quality-diversity archive of representative problems, (ii) an interpretable semantic vocabulary and its discovered structure, (iii) a set of independently optimized problem-specific solvers, and (iv) the cross-evaluation matrix S, which provides an empirical map of algorithmic relationships across the explored domain.

The method encourages domain illumination through the interaction between generality-based quality and semantic diversity. Explicitly estimating the marginal information supplied by a new problem, for example from redundancy between columns of S, is complementary to the present formulation and is discussed separately as a possible extension.

The code and supplementary resources are available in our anonymous repository.

C BASELINES

CVT-MAP-ELITES

We construct a quality-diversity baseline inspired by CVT-MAP-Elites. The baseline retains the high-level CVT-MAP-Elites life-cycle:select individuals, generate variations, evaluate them, compute behavioral descriptors, and update a bounded container; however, the baseline adapts the descriptor mechanism to our problem-space exploration setting.

An individual is a pair

$$x_j = (p_j, a_j), \tag{20}$$

where p_j is a generated problem and

$$a_j = \mathrm{Solve}(p_j) \tag{21}$$

is the best solver obtained from an independent solver-optimization run on that problem.

Behavioral descriptor. CVT-MAP-Elites was originally designed to infer behavioral descriptors from observations generated by an agent. Such trajectories are not naturally available in our setting. Instead, we describe each individual through the program produced by solving its associated problem. If $\mathrm{src}(a_j)$ denotes the Python source code of solver a_j, its descriptor is

$$z_j = \phi(\mathrm{src}(a_j)), \tag{22}$$

where ϕ is the configured text-embedding model.

Thus, behavioral similarity is measured in the space of *induced solvers*, rather than in the space of problem statements. Two problems are close according to this baseline if independently solving them yields similar programs, even when their natural-language descriptions differ.

Initialization. The baseline begins by independently generating N_{init} valid problems. For each problem p_j, we construct its authored initial solver and run the same solver-development procedure used in our main experiments:

$$a_j = \mathrm{Solve}\left(p_j; \theta^{(0)}_{p_j}\right). \tag{23}$$

Solver optimization always starts from the new problem's own seed; no solver is inherited from another problem. After optimization, the source code of a_j is embedded according to Equation 22.

The resulting solvers are cross-evaluated on all initial problems. Until the active container reaches its maximum capacity, all valid individuals remain active.

Problem generation. At each subsequent round, the baseline generates a batch of B new problems from the current active population. By default, half of the batch is produced through crossover and half through mutation.

For CROSSOVER, two distinct active individuals are sampled uniformly and their problem descriptions are provided to the problem generator, which constructs a new problem combining properties of both parents.

For mutation, one active parent is sampled uniformly and either BROADEN or NEWDIRECTION is selected. BROADEN constructs a related problem that extends or relaxes the parent's formulation.

The NEWDIRECTION operator uses the solver-code embedding as a behavioral signal. Given a base individual x_b, we compute cosine distances

$$d_{bj} = 1 - \frac{z_b^\top z_j}{\|z_b\|_2 \|z_j\|_2} \tag{24}$$

to the other active individuals. The three nearest solver descriptors are identified and their corresponding *problem statements* are supplied to the problem generator. The generator is instructed to construct a problem that departs structurally from this local neighborhood. Importantly, neighbor selection is determined exclusively by similarity between the induced solvers, not by similarity between the problem statements themselves.

Every valid child receives a fresh solver-development run from its own authored seed. There is no solver transfer from the parent to the child. Furthermore, the baseline does not apply a minimal-criterion rejection based on the child's difficulty or novelty: once a generated problem passes the domain-validity checks and its solver-development run completes, it becomes part of the historical set.

Cross-problem evaluation and fitness. After each round, the candidate population consists of the previous active individuals together with the newly generated children. We use the same fitness as the one defined in Ψ-Evolve. Using the same fitness is deliberate: differences between this baseline and our method should primarily reflect how the problem space is represented and explored rather than a different definition of problem quality.

Relation to our method. The principal difference between this baseline and our approach is the role of the descriptor space. The CVT-MAP-Elites-inspired baseline represents each individual by a point in a fixed continuous embedding space derived from its solver code. This representation can identify local similarity between induced solution programs, but it does not construct explicit semantic axes, does not grow an interpretable descriptor vocabulary, and does not adapt the archive geometry around newly discovered concepts.

Our method instead maintains an explicit and expandable semantic representation of the problem domain and uses that representation to organize quality-diversity search. Comparing the two therefore tests whether operating directly in a generic latent representation of solver behavior is sufficient, or whether explicit online discovery and organization of problem-space features provides additional exploration capacity.

POET

Our second baseline adapts the Paired Open-Ended Trailblazer (POET) paradigm to our problem-generation setting. We preserve the core POET life cycle: maintain a population of environment–agent pairs, mutate environments from existing ones, admit new environments only when they satisfy a minimal criterion, continue optimizing each agent on its own environment, and transfer agents across environments whenever cross-evaluation identifies a better match. The adaptation is limited to the representation of the two components: an environment is an LLM-authored optimization problem with an executable evaluator, and an agent is a Python program rather than a policy network.

At round t, each individual is a pair

$$x_j^{(t)} = \left(p_j, a_j^{(t)}\right), \tag{25}$$

where p_j is a generated problem, fixed after acceptance, and $a_j^{(t)}$ is the program currently associated with it. Unlike the parameterized environments typically considered in POET, p_j is represented by a natural-language problem statement together with its generated evaluator. Environment mutation is therefore performed by the same LLM-based problem generator used elsewhere in our framework, rather than by perturbing a fixed parameter vector.

Continuing agent optimization. POET repeatedly applies a short optimization block to the agent associated with each active environment. We instantiate this step with the same solver-development system used in our main experiments. Given a problem p and its current program a, one block runs the solver-development procedure for k generations:

$$a' = \mathrm{Solve}_k(p, a). \tag{26}$$

Each block is an independent solver-optimization run initialized from the current program. Optimizer state is not carried across blocks; only the program itself is reused. This is the key distinction from our method and from the CVT-MAP-Elites baseline in Appendix C, where each problem receives a single solver-development run initialized from its own authored seed. In the POET baseline, agents can instead accumulate improvements over multiple rounds and can subsequently transfer across environments.

All programs are evaluated with the same harness used throughout the paper:

$$\sigma(a, p) = \mathrm{Eval}(a, p). \tag{27}$$

The minimal criterion, agent replacement, and transfer decisions therefore use the same problem-specific scores as the main evaluation pipeline.

Initialization. The run begins with N_{init} independently generated problems, authored with the same canonical-problem prompt used by the other methods. Each candidate is validated by executing its generated evaluator on its authored seed program under the corresponding domain contract. If an initial environment is invalid, the run terminates rather than silently replacing it.

For each valid initial environment, we optimize its authored seed with a longer initialization budget:

$$a_j^{(0)} = \text{Solve}_{K_{\text{init}}}\left(p_j; \theta_{p_j}^{(0)}\right), \tag{28}$$

where $K_{\text{init}} \geq k$ because the initial agents have no parent program to inherit. Initialization concludes with the cross-evaluation and transfer stage described below, so transfer is available from the first round onward.

Environment mutation and agent inheritance. At each round, we sample M distinct parents uniformly without replacement from the full population. Parent selection is deliberately uniform: unlike our method, the POET baseline has no fitness-based parent selection or archive ranking. The population is frozen during the round, so children accepted at round t become eligible for selection only at round $t + 1$.

Each selected parent is mutated using one operator sampled from a fixed distribution over $\{\text{BROADEN}, \text{NEWDIRECTION}\}$. BROADEN asks the generator to produce a more general version of the parent problem. NEWDIRECTION asks the generator to move away from the parent's local neighborhood. For a parent p_b, we compute cosine distances between problem-statement embeddings,

$$d_{bj} = 1 - \frac{\phi(p_b)^\top \phi(p_j)}{\|\phi(p_b)\|_2 \|\phi(p_j)\|_2}, \tag{29}$$

identify the three nearest problems, and provide them to the generator as local context to move beyond. Thus, this neighborhood is defined in problem-statement embedding space. By contrast, the CVT-MAP-Elites baseline defines neighborhoods in the embedding space of induced solver programs. Neither baseline maintains an explicit, expandable descriptor vocabulary.

A generated child inherits the current program of its parent:

$$a_{\text{child}}^{(t)} \leftarrow a_{\text{parent}}^{(t)}. \tag{30}$$

The child therefore starts from the parent's current program rather than from its own authored seed.

Minimal criterion. A generated child is admitted only if its inherited agent can make measurable progress on the child environment. Let a denote the inherited agent and p' the child problem. We first evaluate

$$s_{\text{before}} = \sigma(a, p'), \tag{31}$$

then optimize the inherited agent for one short block,

$$a'' = \text{Solve}_k(p', a), \qquad s_{\text{after}} = \sigma(a'', p'). \tag{32}$$

The child is accepted if and only if

$$s_{\text{after}} > s_{\text{before}}. \tag{33}$$

This criterion requires the inherited agent to exhibit immediate learning progress on the new environment.

Children that fail Equation 33 are discarded completely: the problem does not enter the population and no solver state is retained. We also reject children whose authoring call fails or whose generated evaluator does not execute correctly on its authored seed. Each rejection is logged by cause.

Problem authoring is performed at the full batch width in every round, including near the end of the run; only the number of accepted children is capped by the number of remaining population slots. This avoids a pathological slowdown near termination, where small proposal batches would frequently yield no accepted child while still incurring a full round of optimization and cross-evaluation.

Continuing optimization of the population. After the mutation stage, each active environment that did not just receive an acceptance block receives one additional short optimization block. For each active pair, we compute

$$\tilde{a}_j^{(t+1)} = \mathrm{Solve}_k\left(p_j, a_j^{(t)}\right), \tag{34}$$

and retain the candidate only if it improves the score on its own environment:

$$a_j^{(t+1)} = \begin{cases} \tilde{a}_j^{(t+1)}, & \text{if } \sigma\left(\tilde{a}_j^{(t+1)}, p_j\right) > \sigma\left(a_j^{(t)}, p_j\right), \\ a_j^{(t)}, & \text{otherwise.} \end{cases} \tag{35}$$

Thus, an environment never replaces its current agent with a worse one. These optimization blocks are independent across environments and can be executed concurrently. Because the population only grows, the cost of this stage grows linearly with the number of accepted environments.

Cross-evaluation and transfer. Each round ends with a full cross-evaluation of every current agent on every current environment:

$$S_{ji} = \mathrm{Eval}\left(a_j, p_i\right). \tag{36}$$

Scores are recomputed over the full population, without reusing values from earlier rounds. For each environment i, we identify the best donor

$$j^\star(i) = \arg\max_j S_{ji}, \tag{37}$$

breaking ties in favor of the earliest-created individual. The donor's agent is transferred to environment i only if it strictly outperforms the incumbent:

$$S_{j^\star(i),\,i} > S_{ii}. \tag{38}$$

All transfer decisions are computed from the same pre-transfer score matrix and applied simultaneously, preventing transfer chains within a single round. A receiving environment adopts the donor program and the corresponding donor row of scores. Transfers are logged individually.

Population growth and termination. The population is never pruned. Every accepted environment remains active, selectable as a parent, and included in all subsequent cross-evaluations. The run terminates once $N_{\max}$ environments have been accepted. Consequently, the per-round optimization cost grows linearly with population size, while full cross-evaluation grows quadratically.

Reported quality. For comparability with our method and the CVT-MAP-Elites baseline, we compute the same transfer-based generality statistic that our method after each transfer stage.

Solver budget. Because agents continue to be optimized for as long as their environments remain active, the per-environment solver budget is neither uniform nor matched to the other methods. In a completed run, the median environment receives 14–30 optimization blocks, while the oldest environments can receive up to 53 blocks. This corresponds to approximately 1.3–2.8×10^3 optimization blocks per run; the newest environments may receive only one. We report this asymmetry explicitly because it is intrinsic to the POET mechanism: repeatedly optimizing and transferring agents across a growing population is precisely the behavior the baseline is intended to test.

Relation to our method. POET and our method both exploit the idea that the usefulness of a problem can be assessed through how solvers behave on it, but they operationalize this idea differently. POET uses a local criterion: a child environment is admitted when an inherited agent can improve on it, and diversity emerges implicitly through lineage expansion and transfer. The population has no explicit descriptor space, no coverage objective, and no mechanism that compares a candidate problem with the collection as a whole. Consequently, two near-duplicate children can both remain active if each satisfies the minimal criterion.

Our method instead maintains an explicit, expandable semantic representation of the problem domain and uses that representation to organize search and archive retention. The comparison therefore tests whether local progress-based admission and agent transfer are sufficient to sustain diverse problem exploration, or whether explicitly representing what the current collection covers provides an additional advantage.

Table 3: POET run statistics for three seeds per domain, with each run terminating at 100 accepted environments. *Rounds* counts mutation rounds after initialization. *MC* counts children rejected by Equation 33; *Invalid* counts children whose generated evaluator failed its validity check; and *Failed* counts authoring calls that raised an exception. Failed-authoring events in geometric packing and spectral design were partly caused by a concurrency defect in the shared repair budget, which was fixed before the geometry-of-numbers runs. A failed or invalid child consumes a proposal slot in its round but never enters the population.

Domain	Seed	Rounds	MC	Invalid	Failed	Transfers
Geometric packing	0	39	78	13	113	115
Geometric packing	1	37	44	16	95	218
Geometric packing	2	44	59	20	134	51
Spectral	0	39	59	7	83	44
Spectral	1	43	54	5	159	86
Spectral	2	52	79	6	176	54
Geometry of numbers	0	28	91	31	0	145
Geometry of numbers	1	23	60	19	0	83
Geometry of numbers	2	29	88	36	3	102

D Additional Experiments

We provide additional experiments to better isolate the design choices of Ψ-Evolve and to characterize the archives it produces. We first select the mutation-operator distribution on a separate development set (Appendix D.1), then ablate the main components of the method to identify which mechanisms drive performance (Appendix D.2). We further test the role of the semantic representation by comparing online descriptor expansion with fixed pre-generated vocabularies (Appendix D.3), analyzing the specific effect of BROADEN (Appendix D.4), and controlling for the choice of descriptor space in the CVT-MAP-Elites baseline (Appendix D.5). We then study the resulting archives more directly by measuring their distance and redundancy relative to the literature-derived reference set (Appendix D.6), evaluating whether useful solvers can be retrieved without exhaustive archive evaluation (Appendix D.7), and checking whether the internal generality fitness predicts transfer to held-out problems (Appendix D.8). Finally, we verify that the broader exploration of Ψ-Evolve is not obtained at the cost of problem validity or increased redundancy (Appendix D.9).

D.1 Hyperparameter search

In order to determine the best distribution over the mutation operators we conduct an hyperparameter search in the Geometric Packing on set of different problem than the one used in the main paper (generated from Wikipedia only) with several distributions over $\{$CROSSOVER, BROADEN, ADDTAG, NEWDIRECTION$\}$. As NEWDIRECTION is composed of two mutation operators we always give them the same weight. We tried two kind of distribution uniformly sampling each operators and sampling one operator with a 0.75 probability with the others uniformly on the disposable 0.25 probability. It appears clearly in Figure 7 that NEWDIRECTION75, meaning with a 0.75 probability NEWDIRECTION75 is sample and the rest of the mutation operators are uniformly sampled, bring the best results with a minimal variance. Thus we choose this combinaison of hyperparameter for Ψ-Evolve.

D.2 Component Ablations

Figure 6a decomposes the contribution of the variation operators and of generality-based selection. The complete Ψ-Evolve configuration achieves a mean normalized zero-shot score of 66 ± 2.1. Removing CROSSOVER reduces the mean to 63.0 ± 6, while removing BROADEN gives 62 ± 13.1. These smaller average reductions suggest that recombination and generality-oriented local variation improve search, but that neither mechanism alone accounts for the performance of the full system.

A clearer effect appears for operators acting directly on the semantic representation. Removing ADDTAG, which encourages exploration of underrepresented directions already present in the de-

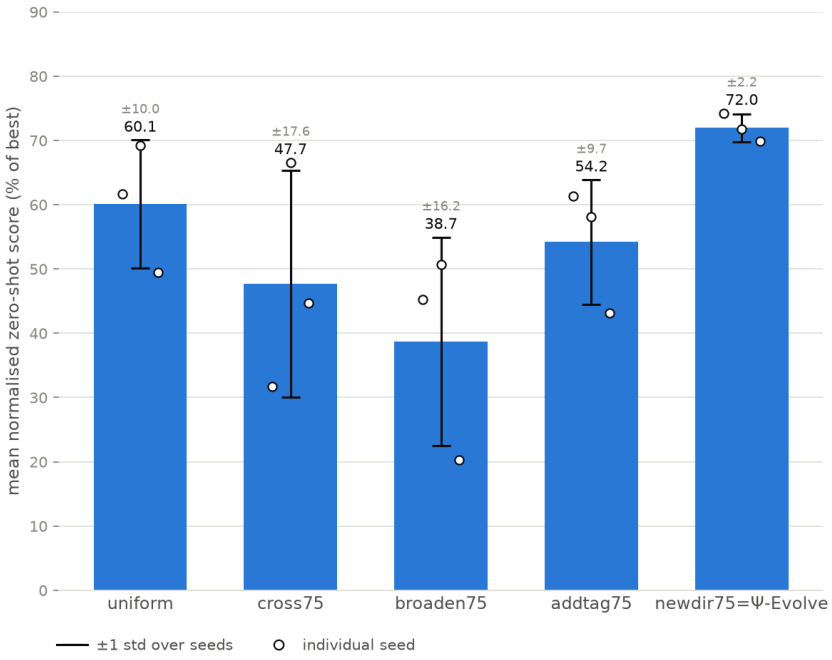


Figure 7: Normalized score for different choice of distribution in Ψ-Evolve on the geometric packing domain.

scriptor vocabulary, lowers the score to 60.0 ± 6.4. The largest reductions arise when disabling semantic expansion itself: removing local NEWDIRECTION gives 52.2 ± 4.6, while removing global NEWDIRECTION gives 52.9 ± 0.5, corresponding to approximately 15 and 14 normalized-score points below the complete method. Thus, exploiting a fixed set of discovered semantic properties is not sufficient in these experiments; continued discovery of new directions appears important for constructing problem archives whose solvers transfer to held-out problems. Interestingly, both local refinement of the current ontology and global exploration of semantically distant directions contribute, suggesting that useful coverage requires both extending known conceptual regions and opening new ones.

We additionally remove the quality component of the QD loop by replacing elite-based parent selection with random parent selection. Performance decreases to 61.1 ± 2.4. This control is important because the semantic operators could in principle obtain broad coverage even without a meaningful notion of local quality. Instead, the reduction shows that diversity-oriented problem generation and transfer-based selection play complementary roles: semantic operators determine which regions are explored, while the generality objective biases continued search toward regions whose induced solvers exhibit reusable algorithmic structure. Taken together, the ablations support the two-part design of Ψ-Evolve as an interaction between adaptive semantic diversity and transfer-defined quality rather than either mechanism operating independently.

D.3 EFFECT OF THE ONLINE DESCRIPTOR EXPANSION

To separate the effect of an expanding semantic representation from simply providing the search with a sufficiently large descriptor vocabulary, we compare in Figure 8 Ψ-Evolve with variants whose tag vocabularies are generated once before search and then frozen. We consider vocabularies of 10, 50, and 100 tags, while keeping the remaining search procedure unchanged.

Increasing the size of the frozen vocabulary progressively recovers held-out transfer performance: the 10-tag variant performs substantially worse, while the 100-tag variant approaches the zero-shot performance of the full method. However, even with 100 predefined tags, the generated problems remain substantially closer to the literature-derived reference set than those generated by Ψ-Evolve. Thus, a large fixed vocabulary can capture many useful directions of variation, but online semantic

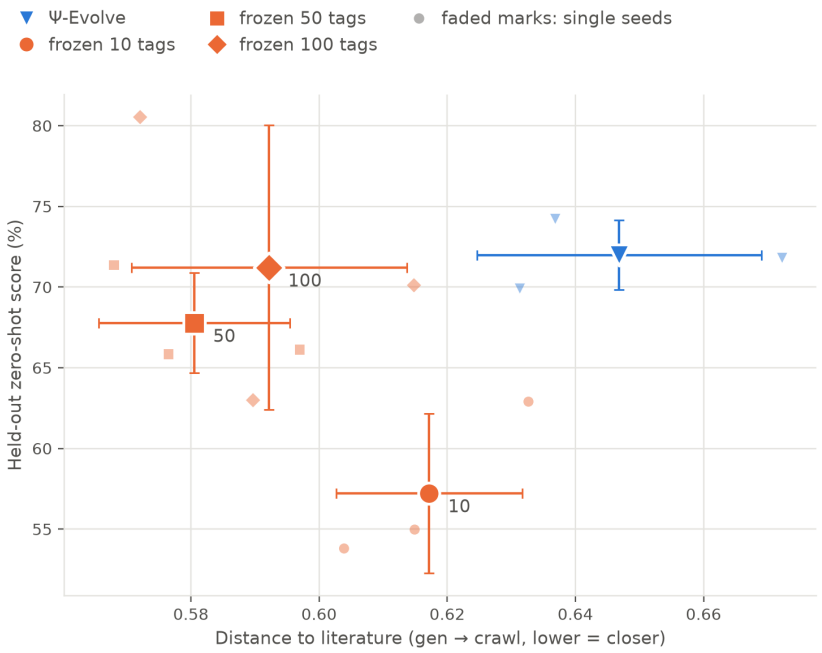


Figure 8: **Online semantic expansion enables broader exploration while preserving transfer.** Held-out zero-shot performance versus distance to the literature-derived reference set in geometric packing. Frozen-vocabulary variants use 10, 50, or 100 tags generated before search and do not introduce new tags during the run. Increasing the frozen vocabulary recovers much of the transfer performance, but the full Ψ-Evolve method explores substantially farther from the literature distribution while maintaining high transfer. Large markers show means over three seeds; faded markers show individual seeds and error bars show ± 1 standard deviation.

expansion allows the search to continue opening new regions of problem space while preserving strong external transfer.

D.4 Effect of the broadening operator

The goal of BROADEN is to generate problems whose induced solvers remain useful beyond the current archive. Measuring this effect only at the child's creation is difficult because parents are already cell elites selected for high generality, making newly generated children systematically disadvantaged. We therefore also measure how well each child transfers to problems discovered *after* its creation, which provides a cleaner test of whether the mutation produces a solver with broader future coverage. Under the uniform operator mix, broadened children initially retain $0.36\times$ their parent's generality, essentially identical to AddTag children ($0.37\times$, p=0.91). The difference emerges on subsequently generated problems: broadened solvers match their parents' coverage (3.35 vs. 3.31 problems, measured in units of best-known score), whereas AddTag solvers reach only about half of their parents' coverage (1.63 vs. 3.32). This corresponds to a $+2.05$-problem advantage for BROADEN over AddTag (95% CI $[0.42, 3.73]$, p=0.02), with the effect positive in all three seeds. Consistently, final-archive retention rises to $0.48\times$ for BROADEN, compared with $0.36\times$ for AddTag.

D.5 Effect of the CVT-MAP-Elites descriptor space.

To determine whether the advantage of Ψ-Evolve arises simply from organizing the archive in problem space rather than solution space, we additionally evaluate a variant of CVT-MAP-Elites in which the behavioral descriptor is the embedding of the generated problem instead of the embedding of its induced solver. In Figure 9, the two variants achieve similar zero-shot performance: CVT-MAP-Elites using solver embeddings reaches 56.2 ± 7.7, while the problem-embedding variant reaches

Table 4: Distance between each method's generated problems and the held-out literature crawl, averaged over the three domains (geometric packing, spectral, geometry of numbers; equal weight per domain). Problem-statement embeddings (`text-embedding-3-small`, 1536-d, unit norm), Euclidean nearest-neighbour distances. *Novelty* is the mean distance from each generated problem to its nearest crawled one; *coverage* is the mean distance from each crawled problem to its nearest generated one. $\pm$ is the spread across the three domains.

Method	Novelty (gen $\rightarrow$ crawl)	Coverage (crawl $\rightarrow$ gen)
Ψ-Evolve	**0.651** $\pm$ 0.108	0.707 $\pm$ 0.115
ctxfree	0.617 $\pm$ 0.069	**0.692** $\pm$ 0.100
CVT-MAP-Elites	0.389 $\pm$ 0.067	0.721 $\pm$ 0.105
POET	0.395 $\pm$ 0.048	0.722 $\pm$ 0.107

60.0 ± 3.7. The larger variance of the solver-embedding variant is mainly driven by one seed with noticeably lower performance, whereas the other runs are close to those obtained with problem embeddings. Overall, changing the CVT representation from induced-solver space to problem space yields only a modest improvement and does not close the gap to Ψ-Evolve. This suggests that the gains of Ψ-Evolve cannot be explained solely by organizing QD search in a problem-embedding space, and instead supports the contribution of its adaptive semantic exploration mechanism.

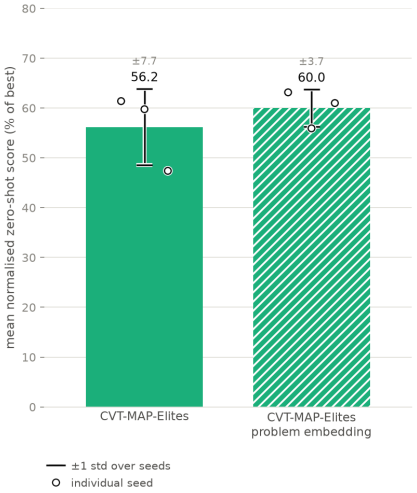


Figure 9: Zero-shot performance of CVT-MAP-Elites using either induced-solver embeddings or problem embeddings as the descriptor space.

D.6 Distance between generated and crawled problems

Relation to the literature distribution. Tables 4 and 5 characterize how the generated problem distributions relate to the held-out literature crawl. Ψ-Evolve achieves the largest average novelty, with a generated-to-crawl distance of 0.651, compared with 0.617 for context-free generation and approximately 0.39 for CVT-MAP-Elites and POET. This effect is particularly pronounced in the spectral domain (0.761), while in geometric packing Ψ-Evolve and context-free generation reach comparable novelty. Importantly, the larger distance of Ψ-Evolve does not correspond to substantially poorer coverage of the literature distribution: its crawl-to-generation distance is 0.707, close to the best value of 0.692 obtained by context-free generation and lower than those of CVT-MAP-Elites (0.721) and POET (0.722). In contrast, CVT-MAP-Elites and POET generate problems substantially closer to existing literature examples without achieving better coverage of the literature set, suggesting that their samples concentrate around narrower regions rather than broadly matching the reference distribution. These results indicate that Ψ-Evolve explores farther from known problems while retaining comparable coverage of the human-authored problem space. Since nearest-neighbour distance to the literature does not reveal whether this spread arises from a few isolated problems or from reduced redundancy throughout the generated set, we next analyze local kNN dispersion.

Local problem-space dispersion. Table 6 complements the global embedding-space analysis by measuring local redundancy among generated problems. Ψ-Evolve achieves the largest 5NN dispersion among the generation methods, with $D_5 = 0.1798 \pm 0.0132$, compared with 0.1310 ± 0.0100 for context-free generation, 0.0563 ± 0.0075 for CVT-MAP-Elites, and 0.0621 ± 0.0082 for POET. Thus, the nearest semantic neighbours of Ψ-Evolve problems are substantially farther apart, indicating that its broader embedding-space spread is not explained only by a small number of distant outliers. Consistently, none of the Ψ-Evolve problems have an average distance below 0.05 to their five nearest neighbours, whereas this occurs for 66–78% of CVT-MAP-Elites and 65–74% of POET problems, revealing substantial local concentration in these baselines. The result is also stable across

Table 5: Novelty per domain: mean Euclidean distance from each generated problem to its nearest problem in that domain's held-out literature crawl, on problem-statement embeddings. Per-domain values are means over three seeds (± seed standard deviation); the last column averages the three domains with equal weight (± spread across domains).

Method	Geometric packing	Spectral	Geometry of numbers	Average
Ψ-Evolve	0.647 ± 0.022	**0.761** ± 0.010	**0.546** ± 0.040	**0.651** ± 0.108
Context-free	**0.656** ± 0.012	0.658 ± 0.009	0.538 ± 0.009	0.617 ± 0.069
CVT-MAP-Elites	0.395 ± 0.018	0.452 ± 0.020	0.320 ± 0.013	0.389 ± 0.067
POET	0.400 ± 0.017	0.441 ± 0.019	0.345 ± 0.007	0.395 ± 0.048

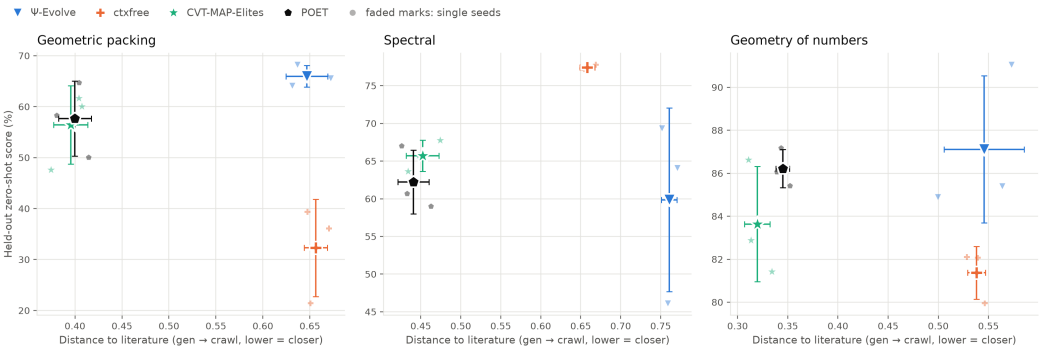


Figure 10: Normalized score for different choice of distribution in Ψ-Evolve on the geometric packing domain.

the three independent seeds. For reference, size-matched subsets of the literature crawl remain more dispersed ($D_5 = 0.2587 \pm 0.0074$), showing that Ψ-Evolve does not exhaust the diversity of the human-authored problem distribution. Overall, the kNN analysis provides independent evidence that Ψ-Evolve generates a less locally redundant set of problems, complementing the global radius and literature-distance measurements.

Transfer utility beyond the literature distribution. To jointly assess how far generated problems depart from existing literature and whether this exploration remains algorithmically useful, Figure 10 plots the distance of generated problems to the held-out literature set against the zero-shot transfer performance of their induced solvers. Ψ-Evolve consistently explores regions that are distant from the literature distribution while retaining substantial transfer utility. In geometric packing, Ψ-Evolve achieves markedly higher transfer than the other methods while remaining substantially farther from the literature than CVT-MAP-Elites and POET; notably, context-free generation reaches a comparable distance but with much lower transfer performance. In geometry of numbers, Ψ-Evolve simultaneously attains the largest literature distance and the highest held-out transfer score. The spectral domain exhibits a different trade-off: Ψ-Evolve explores the most distant region, although context-free generation achieves higher transfer. Overall, these results show that the gains of Ψ-Evolve cannot be explained by remaining close to known human-authored problems. Instead, the method tends to combine exploration away from the literature distribution with solvers that retain strong external utility, illustrating the intended interaction between semantic diversity and transfer-based quality.

D.7 Retrieval-based use of the solver archive.

The main experiments report an oracle archive score obtained by evaluating all 100 archive solvers on each held-out target and selecting the best-performing one. To assess whether useful solvers can be identified without exhaustively evaluating the entire archive, we additionally consider retrieval-based solver selection. For each held-out problem, archive problems are ranked either by similarity of their problem-statement embeddings or by Jaccard similarity between their semantic tag sets, and

Table 6: kNN dispersion of the generated problems in geometric packing ($k = 5$). Each problem p_i is embedded as z_i (problem-statement embeddings, `text-embedding-3-small`); $D_k(p_i) = \frac{1}{k}\sum_{j\in\mathrm{kNN}(i)} d(z_i, z_j)$ with d the cosine distance and neighbours taken among the other problems of the same run; $D_k = \frac{1}{N}\sum_i D_k(p_i)$. Higher means the problems are more spread out. The last column is the share of problems whose five nearest neighbours lie within cosine distance 0.05 on average. The held-out literature crawl is given for reference, on random subsets of the archives' size (100 problems, 2000 draws), since kNN distances shrink as a set grows.

Method	Seed	N	D_5	Median $D_5(p_i)$	$D_5(p_i) < 0.05$
Ψ-Evolve	0	100	0.1671	0.1759	0.0%
	1	100	0.1935	0.2008	0.0%
	2	100	0.1789	0.1857	0.0%
	mean		0.1798 ± 0.0132		
ctxfree	0	100	0.1233	0.1236	3.0%
	1	100	0.1274	0.1202	0.0%
	2	100	0.1424	0.1424	7.9%
	mean		0.1310 ± 0.0100		
CVT-MAP-Elites	0	100	0.0581	0.0303	69.0%
	1	100	0.0628	0.0315	66.0%
	2	100	0.0481	0.0288	78.0%
	mean		0.0563 ± 0.0075		
POET	0	100	0.0657	0.0388	65.0%
	1	100	0.0680	0.0372	71.0%
	2	100	0.0528	0.0359	74.0%
	mean		0.0621 ± 0.0082		
Crawl (reference)	subsets	100	0.2587 ± 0.0074	–	–
	all	261	0.2237	0.2314	0.0%

only the top-k associated solvers are evaluated on the target. The target tags are inferred from the problem specification using the descriptor vocabulary discovered during the corresponding run; no target-side solver optimization or performance information is used for retrieval.

Table 7 shows that semantic tag retrieval consistently outperforms embedding-based retrieval at matched evaluation budgets. At $k = 10$, tag-based retrieval reaches a normalized score of 41.6, compared with 32.2 for embedding retrieval. At $k = 50$, tag retrieval reaches 63.1, recovering 95.6% of the full-archive oracle score of 66.0 while evaluating only half of the archive. Embedding retrieval at the same budget reaches 59.8. These results indicate that the semantic representation discovered during search is useful not only for directing problem-space exploration, but also for indexing the resulting archive and retrieving transferable solvers for previously unseen problems.

D.8 Relationship Between Generality Fitness and External Transfer

External utility. To measure whether the generality fitness learned from generated problems predicts usefulness outside the search distribution, we define the external utility of a generated problem p_j through the performance of its induced solver a_j on the held-out literature-derived problem set $\mathcal{H}$. Let

$$S_{ji}^{\mathrm{ext}} = \mathrm{Eval}(a_j, h_i), \qquad h_i \in \mathcal{H},$$

and let $\widetilde{S}_{ji}^{\mathrm{ext}}$ denote the corresponding normalized score under the same normalization used for our zero-shot evaluation. We define

$$U(p_j) = \frac{1}{|\mathcal{H}|}\sum_{h_i\in\mathcal{H}} \widetilde{S}_{ji}^{\mathrm{ext}}.$$

Importantly, the held-out problems in $\mathcal{H}$ are never used to compute the generality fitness $F(p_j)$ or to guide the search. Thus, $U(p_j)$ provides an external measure of the transferability of the solver induced by p_j.

Retrieval method	k	Normalized score	Oracle retained
Problem embedding	5	21.1 ± 7.6	32.0%
Problem embedding	10	32.2 ± 7.9	48.8%
Problem embedding	50	59.8 ± 4.4	90.6%
Tag Jaccard	5	29.5 ± 7.4	44.7%
Tag Jaccard	10	41.6 ± 6.7	63.0%
Tag Jaccard	50	63.1 ± 0.8	95.6%
Full archive (oracle)	100	66.0 ± 2.1	100%

Table 7: Retrieval-based solver selection from a 100-problem Ψ-Evolve archive. For each held-out target, we rank archive problems using either problem-statement embedding similarity or Jaccard similarity between semantic tag sets, then evaluate only the top-k associated solvers. The full-archive condition evaluates all 100 solvers and provides an oracle upper bound for archive selection. Values are means ± standard deviation over three seeds. "Oracle retained" reports the fraction of the full-archive normalized score recovered at each retrieval budget.

Consistency of the generality–utility relationship across seeds. To verify that the strong pooled correlation between internal generality and external transfer is not driven by differences between independent runs, we compute the relationship separately for each seed in the geometric packing domain. Despite noticeable variation in the absolute levels of both external utility U and generality fitness F across runs, the association between the two quantities is remarkably stable. We obtain Pearson correlations of $r = 0.881$, $r = 0.904$, and $r = 0.906$ for seeds 0, 1, and 2, respectively, with all correlations highly significant ($p < 10^{-33}$). In particular, seed 2 exhibits lower mean utility (0.045) and mean fitness (0.029) than the other runs, yet retains essentially the same generality–utility relationship. This consistency shows that the strong pooled correlation is not explained by between-seed shifts in average performance. Instead, within each independently evolved problem distribution, problems assigned higher generality by Ψ-Evolve systematically induce solvers with stronger transfer to the held-out literature-derived problems. These results provide additional evidence that the internally computed fitness F captures a robust signal of external solver utility rather than a seed-specific property of the generated archive.

D.9 Validity of generated problems

Validity and redundancy of generated problems. Table 8 shows that the broader exploration of Ψ-Evolve is not obtained at the cost of unreliable problem generation. Across domains, 99.0% of its proposals are valid, only 20.2% require repair, and the duplicate rate is 9.0%. Context-free generation achieves similarly high validity (99.8%) but produces more than twice as many duplicates (19.9%). CVT-MAP-Elites and POET exhibit substantially higher repair (87.6% and 87.4%) and duplicate rates (77.7% and 70.8%, respectively); POET additionally reaches only 68.7% validity overall, driven by geometric packing (60.8%) and spectral design (55.0%). Solver executability remains comparable overall for Ψ-Evolve, CVT-MAP-Elites, and POET (90.2–90.9%), although geometry of numbers is more challenging for Ψ-Evolve. Overall, these results indicate that Ψ-Evolve combines semantic exploration with high validity and markedly lower redundancy, rather than obtaining diversity through malformed or repeatedly repaired proposals.

Validity across mutation operators. Table 9 further decomposes these statistics by Ψ-Evolve mutation operator. Notably, the two NewDirection operators, which are responsible for expanding the semantic search space, do not incur a reliability penalty: local and global NewDirection achieve solver-execution rates of 93.3% and 91.8%, respectively, while global NewDirection has the lowest duplicate rate of all operators (5.5%). AddTag and Broaden exhibit similarly moderate repair and duplicate rates, whereas Crossover is the least reliable operator, with a lower solver-execution rate (78.4%) and the highest duplicate rate (16.2%). This suggests that the semantic expansion mechanisms can introduce new directions without sacrificing problem validity or executability.

Domain	Method	Valid (%)	Repaired (%)	Duplicate (%)	Solver exec. (%)
Geometric packing	Ψ-Evolve	99.3	17.2	3.7	97.4
	Context-free	100.0	10.2	6.7	84.5
	CVT-MAP-Elites	100.0	81.7	71.7	96.2
	POET	60.8	85.7	68.0	90.8
Spectral	Ψ-Evolve	98.4	19.5	3.3	95.0
	Context-free	100.0	12.5	18.7	99.0
	CVT-MAP-Elites	86.1	88.7	71.0	93.2
	POET	55.0	91.0	62.0	96.5
Geometry of numbers	Ψ-Evolve	99.3	24.1	20.3	78.2
	Context-free	99.3	19.5	34.3	98.0
	CVT-MAP-Elites	99.4	92.3	90.2	84.3
	POET	99.5	85.7	82.5	86.4
All domains	Ψ-Evolve	99.0	20.2	9.0	90.2
	Context-free	99.8	14.1	19.9	93.8
	CVT-MAP-Elites	94.8	87.6	77.7	90.9
	POET	68.7	87.4	70.8	90.9

Table 8: Problem-generation statistics per domain and method (3 seeds pooled).

Operator	n	Repaired (%)	Duplicate (%)	Solver executable (%)
Crossover	74	10.8	16.2	78.4
Broaden	89	23.6	11.4	85.4
AddTag	67	20.9	9.0	85.1
NewDirection$_{\text{local}}$	342	19.6	10.2	93.3
NewDirection$_{\text{global}}$	328	22.3	5.5	91.8

Table 9: Ψ-Evolve problem-generation statistics per operator (3 domains, 3 seeds pooled).

E BLINDED EXPERT EVALUATION OF GENERATED PROBLEMS

The automatic metrics in the main paper quantify whether generated problems are useful for solver transfer and whether they cover diverse regions of the problem space. They do not directly measure whether experts regard the generated problems as coherent and mathematically meaningful. We therefore conducted a blinded expert study in the geometric-packing domain. Experts evaluated both standalone problems and parent–child variations without being told whether the problems came from the literature, Ψ-Evolve, or a baseline.

E.1 STUDY DESIGN

Standalone problems. We sampled 66 problems: 22 literature-derived problems from the held-out crawl, 22 problems from the final Ψ-Evolve archives, and 22 problems from the context-free baseline. Sampling was uniform within each source and did not depend on quality, fitness, or archive status. The Ψ-Evolve sample therefore reflects the distribution of problems produced by the search rather than a hand-selected subset.

For each problem, participants gave five-point ratings for: (i) *coherence*, whether the problem defines a clear and sufficiently specified task; (ii) *non-triviality*, whether its structure is non-trivial beyond simply increasing instance size; (iii) *conceptual meaningfulness*, whether its objective, constraints, and mathematical structure form a natural problem rather than an arbitrary collection of restrictions; and (iv) *reusable insight*, whether a strong solution would plausibly contain ideas useful for related problems. Participants also indicated whether they had seen the same or a nearly identical problem before.

Measure	What participants evaluated
Standalone problems	
Coherence	Is the problem clear and sufficiently well specified?
Non-triviality	Does it contain non-trivial structure beyond instance size?
Meaningfulness	Do the objective, constraints, and structure form a natural mathematical question?
Reusable insight	Would solving it plausibly yield ideas useful for related problems?
Problem variations	
Meaningful variation	Is the candidate a substantive variation rather than an arbitrary complication?
Structural change	Does it change the underlying problem structure rather than only numerical parameters?
Algorithmic consequence	Is it likely to require a modified, generalized, or different solution strategy?
Ad hoc complication	Does it add restrictions or special cases without a clear conceptual motivation?

Table 10: Summary of the expert-rating dimensions. All measures use five-point scales.

Problem variations. We separately evaluated 48 parent–child examples, with eight examples for each of six conditions: CROSSOVER, BROADEN, ADDTAG, local NEWDIRECTION, a human–human reference condition, and a superficial-mutation control (SHAM). Operator examples were sampled from all accepted children, not only from those that later became archive representatives, and were balanced across the three seeds.

For the human–human condition, the reference and candidate were two literature-derived problems restated from the same source paper. The SHAM condition used the same model, problem generator, templates, validation procedure, and repair loop as Ψ-Evolve, but explicitly instructed the generator to make only a superficial numerical, parameter, or additional-constraint modification while preserving the underlying problem and intended solution strategy. This provides a control for generic LLM rewriting without the semantic variation operators.

Participants rated each candidate on four five-point scales: *meaningful variation*, *structural change*, *algorithmic consequence*, and *ad hoc complication*. They also selected a categorical description of the variation (e.g., parameter change, broadening, new direction, combination, or arbitrary complication). CROSSOVER items displayed two reference problems; all other conditions displayed one.

Blinding. Participants saw only the domain name and the problem text. Source labels, operator names, seeds, fitness values, tags, URLs, file names, and generation metadata were hidden. All texts went through the same normalization procedure, which removed template boilerplate and explicit provenance markers such as tags, source lines, URLs, and paper identifiers without paraphrasing the mathematical content. Items were shuffled independently for each participant.

After the main ratings, participants could optionally classify previously rated standalone problems as *human-authored*, *AI-generated*, or *unable to tell*. This check was presented last so that source attribution would not influence the main evaluation.

Participants and assignment. Six researchers with backgrounds in mathematics or theoretical physics participated. Three completed both modules and the source-identification check, while one completed 20 standalone ratings only. Two additional participants completed 17 and 13 standalone ratings, respectively, together with the source-identification check, but did not complete the problem-variation module. Items were distributed across participants so that each problem received ratings from multiple researchers.

Analysis. We first averaged ratings across participants for each item. Source and condition comparisons were then computed from these item-level means, with 95% percentile bootstrap confidence intervals over items using 4,000 resamples. A confidence interval containing zero is treated as inconclusive rather than as evidence of equivalence.

Source	Coherence	Non-triviality	Meaningfulness	Reusable insight
HUMAN	4.12 (0.86)	3.22 (1.19)	3.34 (1.00)	3.21 (1.09)
Ψ-Evolve	4.02 (0.83)	**3.74** (0.68)	3.16 (0.86)	3.61 (0.78)
CONTEXT-FREE	4.20 (0.56)	3.45 (0.72)	3.51 (0.77)	**3.65** (0.75)

Table 11: Standalone expert ratings (mean and standard deviation over ratings; 22 problems per source). Higher is better for all four measures.

Condition	Meaningful variation	Structural change	Algorithmic consequence	Ad hoc ↓
CROSSOVER	3.06 (0.94)	3.06 (1.06)	3.22 (1.22)	2.89 (1.13)
BROADEN	3.50 (1.15)	3.83 (1.15)	3.78 (1.17)	2.78 (1.06)
ADDTAG	3.39 (0.85)	3.61 (1.14)	3.78 (1.06)	2.50 (0.86)
NEWDIRECTION (local)	**3.56** (0.92)	3.50 (0.71)	3.72 (0.83)	2.61 (0.70)
HUMAN–HUMAN	**3.56** (0.78)	**4.00** (1.03)	**3.89** (1.13)	**2.50** (1.10)
SHAM	1.50 (0.51)	1.33 (0.59)	1.33 (0.59)	3.39 (1.65)

Table 12: Expert ratings of problem variations (mean and standard deviation over ratings; eight items per condition). Higher is better for the first three measures; lower is better for *ad hoc complication*.

E.2 RESULTS

Standalone problem quality. Table 11 summarizes the expert ratings. Ψ-Evolve problems receive coherence ratings close to the literature-derived set (3.83 vs. 4.00), while being rated more non-trivial (3.64 vs. 3.07) and more likely to yield reusable algorithmic insight (3.56 vs. 3.13). Conceptual meaningfulness is also similar between the two sets (3.12 vs. 3.18).

Relative to the literature-derived problems, the difference of item-level mean ratings (22 problems per source; 95 % percentile bootstrap intervals over items) is $+0.61$ [0.21, 1.00] on non-triviality and $+0.49$ [0.13, 0.86] on reusable insight, both intervals excluding zero. The corresponding differences in coherence (-0.15 [-0.48, 0.20]) and meaningfulness ($+0.01$ [-0.37, 0.38]) are small and their intervals cover zero. Both positive effects persist when ratings of problems a rater marked as familiar are excluded ($+0.53$ [0.12, 0.97] and $+0.43$ [0.08, 0.80]) and when the ratings of the participating author are excluded ($+0.76$ [0.24, 1.31] and $+0.73$ [0.26, 1.20]).

The comparison with context-free generation is more mixed. Ψ-Evolve is rated somewhat more non-trivial (3.64 vs. 3.41; $+0.27$ [0.04, 0.47]), but lower on coherence (3.83 vs. 4.17; -0.33 [-0.59, -0.06]) and conceptual meaningfulness (3.12 vs. 3.50; -0.32 [-0.62, -0.05]). Reusable-insight ratings are similar (3.56 vs. 3.66; -0.09 [-0.37, 0.20]). This pattern suggests that Ψ-Evolve tends to produce structurally richer problems, while context-free generation can produce problems that appear more immediately coherent or conventional to experts.

Quality of the variation operators. Table 12 shows that all four Ψ-Evolve operators produce substantially more meaningful and structurally consequential variations than the superficial-mutation control. On meaningful variation, the improvement over SHAM ranges from $+1.54$ to $+2.04$ points; on structural change, from $+1.79$ to $+2.58$; and on algorithmic consequence, from $+1.96$ to $+2.54$. For all three measures and all four operators, the lower end of the 95% confidence interval is above one point.

The categorical labels tell the same story. The SHAM examples were overwhelmingly classified as restatements or numerical parameter changes, whereas the actual operators were more often described as meaningful combinations, broadenings, or new directions. The operator scores are also close to the human–human reference condition. We do not interpret this as evidence of equivalence, but the current sample shows no clear separation between the operator-generated and human–human variations on the three primary mutation-quality measures.

Source-identification check. Participants did not reliably recover provenance from the normalized problem statements. Across 168 guesses, overall source-identification accuracy was 0.51 when

"unable to tell" was counted as incorrect, against 0.50 for a uniformly random binary guess. Human and Ψ-Evolve problems were labeled "human authored" at similar observed rates (24 of 57 and 19 of 57 guesses, respectively; 24 of 54 for context-free problems), and human problems were labeled "AI-generated" more often than not (32 of 57). We use this only as a check that obvious source cues were limited, not as evidence that human and generated problems are perceptually equivalent.

Generality fitness and expert judgments. In this sample, expert ratings of reusable insight are only weakly and not significantly related to generality fitness. Across the 22 Ψ-Evolve problems, the Spearman correlation between the item-level mean rating and generality fitness is $\rho = 0.19$, and $\rho = -0.08$ with the number of held-out problems the problem's solver transfers to; for the 22 context-free problems the values are 0.09 and 0.24. Given the limited number of problems and ratings per item, we do not draw a strong conclusion from this analysis; the paper's primary evidence for generality fitness remains the direct transfer measurements reported in the main experiments.

E.3 Takeaways

The expert study complements the automatic diversity and transfer metrics in two ways. First, Ψ-Evolve problems receive average coherence and meaningfulness scores close to literature-derived problems, while receiving higher ratings for structural non-triviality and reusable algorithmic insight. The comparison with context-free generation is more mixed: Ψ-Evolve problems are rated more non-trivial, but somewhat lower on coherence and conceptual meaningfulness, with similar reusable-insight ratings. Second, the semantic variation operators produce changes that experts perceive as substantially more meaningful, structural, and algorithmically consequential than matched superficial mutations.

F Prompt Templates and LLM Roles

This appendix documents the complete language-model interface used by Ψ-Evolve. Table 13 summarizes the model roles, when each role is invoked, and where its prompt is specified. The system uses one *problem designer*, three *descriptor* roles, two *new-direction tag inventors*, and an inner ShinkaEvolve *solver*. The embedding model receives raw text directly and is therefore not associated with a prompt.

> ***Reproducibility note.*** *All prompt blocks below reproduce the strings used by the implementation. Only typography, line wrapping, and sectioning have been changed for presentation; the prompt content itself is left unchanged.*

The prompt templates are shared with the previous version of the method by construction. The generator subclass used here differs from its parent only in the accounting label under which token usage is recorded. Comparisons between versions therefore isolate changes to the search algorithm—generality-only fitness, the tunable operator mixture, and the removal of solution inheritance—from changes in prompt wording.

Role	Accounting label	Trigger	Prompt
Problem designer	`problem_generator_v7`	Every generation, including repairs	§F.1
Sibling tag inventor	`new_direction_refine_tag`	NEW-DIRECTION, cluster mode	§F.2
Divergent tag inventor	`new_direction_divergent_tag`	NEW-DIRECTION, divergent mode	§F.2

Table 13: Language-model roles used in one run. Accounting labels correspond to the keys in `usage_log.json` under which calls, tokens, and cost are recorded.

F.1 Problem designer

F.1.1 Operator directives

One directive per generation operator. All of them end with the same generality clause, which is the objective the archive is searched under.

Directive: first Used for the very first problem of a run (filiation `first`).

```
This is the FIRST problem. Design the most GENERAL and REPRESENTATIVE
    problem of the targeted class — a canonical packing problem capturing
     the essence of the domain. Design the problem so that a solver which
     solves it well could also solve a great number of OTHER packing
    problems.
```

Directive: crossover Used for operator CROSSOVER: base + a second cell leader.

```
CROSSOVER: synthesize the two example problems below into ONE new problem
     that combines their key structural features (container, objects,
    objective). Design the problem so that a solver which solves it well
    could also solve a great number of OTHER packing problems.
```

Directive: broaden (example-conditioned) Used for operator BROADEN when the base's solver already scores on other problems.

```
MUTATION (broaden — more general AND more powerful): the BASE problem A
    below already has a solver that transfers to OTHER problems. Some it
    SOLVES BUT DOES NOT BEAT (a stronger solver exists for them); some it
     already BEATS (its solver is their best known). Study these examples
    , then design a NEW problem whose ideal solution would be STRICTLY
    MORE GENERAL and MORE POWERFUL than A's — capturing what makes A
    transfer, while demanding a solver that would dominate on even more
    problems. Design the problem so that a solver which solves it well
    could also solve a great number of OTHER packing problems.
```

Directive: broaden (fallback) Used for operator BROADEN when the base's solver transfers to nothing yet.

```
MUTATION (broaden): take the BASE problem below and mutate it into a new
    problem that is strictly more general / broader, so its solution
    would also solve many of the existing problems. Design the problem so
     that a solver which solves it well could also solve a great number
    of OTHER packing problems.
```

Directive: ADDTAG Used for operator ADD-TAG; the tag is drawn from the vocabulary the base lacks, with weight $1/(\mathrm{count}(t)+1)$.

```
MUTATION (add concept): take the BASE problem below and design a new
    problem that additionally incorporates the concept of the tag '{
    new_tag}', which the base problem does not currently cover, while
    staying a valid packing problem. Design the problem so that a solver
    which solves it well could also solve a great number of OTHER packing
     problems.
```

Directive: NEW-DIRECTION (cluster / sibling) Used for operator NEW-DIRECTION, cluster sub-mode; the new tag is the one invented in §F.2.

```
MUTATION (new direction — sibling on the same axis): take the EXAMPLE
    problem below and mutate it so the result NO LONGER involves the
    concepts [{cluster_tag_1}, {cluster_tag_2}, ...] but INSTEAD embodies
     the new tag '{new_tag}', a SIBLING VARIATION on the SAME conceptual
    axis — a different value in the same family, one step away (NOT a
```

```
    narrower sub-case, and NOT an unrelated concept). Keep it a valid,
    self-contained packing problem that clearly instantiates '{new_tag}'.
     Design the problem so that a solver which solves it well could also
    solve a great number of OTHER packing problems.
```

Directive: NEW-DIRECTION (divergent) Used for operator NEW-DIRECTION, divergent sub-mode: no example problem.

```
MUTATION (new direction — divergent): design a BRAND-NEW packing problem
     in the domain, with NO base example, that embodies the concept of
    the tag '{new_tag}'. This tag was chosen to push the exploration in
    an original, divergent direction not yet covered. Make the problem a
    clean, canonical instantiation of '{new_tag}'. Design the problem so
    that a solver which solves it well could also solve a great number of
     OTHER packing problems.
```

Directive: NEW-DIRECTION (no-tag fallback) Used for operator NEW-DIRECTION before any tag exists (inherited form).

```
MUTATION (new direction): take the BASE problem below in a NEW, never-
    seen direction — via generalization or by introducing a new
    mathematical notion NOT present in the existing tags below — so the
    problem space is expanded.
EXISTING TAGS: {existing_tag_1}, {existing_tag_2}, ...
Design the problem so that a solver which solves it well could also solve
     a great number of OTHER packing problems.
```

F.1.2 CONTEXT BLOCKS

The example problems a directive refers to. A problem is always shown as the module docstring of its `evaluate.py` – the statement its author wrote, which is also what is embedded to place it in the archive and what the tagger reads. Every operator sees the current cell leaders, except the example-conditioned BROADEN, whose context is its own examples.

Context: crossover

```
[PARENT 1]
{base_problem}

[PARENT 2]
{partner_problem}


CELL LEADERS (the top problem of each current cell — for context on what
     is already covered):

[LEADER 1]
{cell_leader_1}

[LEADER 2]
{cell_leader_2}

[LEADER 3]
...
```

Context: broaden (example-conditioned)

```
[BASE PROBLEM A 1]
{base_problem}


Problems A SOLVES BUT DOES NOT BEAT (stronger solver exists):

[A-SOLVES-NOT-BEATEN 1]
{solved_example}


Problems A BEATS (A's solver is their best known):

[A-BEATS 1]
{beaten_example}
```

Context: single-base operators (broaden fallback, add-tag, no-tag new-direction)

```
[BASE PROBLEM 1]
{base_problem}


CELL LEADERS (the top problem of each current cell — for context on what
     is already covered):

[LEADER 1]
{cell_leader_1}

[LEADER 2]
{cell_leader_2}

[LEADER 3]
...
```

Context: NEW-DIRECTION, cluster sub-mode

```
[EXAMPLE PROBLEM 1]
{example_problem}


CELL LEADERS (top problem of each current cell — for context on what is
    already covered):

[LEADER 1]
{cell_leader_1}

[LEADER 2]
{cell_leader_2}

[LEADER 3]
...
```

Context: no-base operators (first, NEW-DIRECTION divergent)

```
CELL LEADERS (the top problem of each current cell — for context on what
     is already covered):

[LEADER 1]
{cell_leader_1}
```

```
[LEADER 2]
{cell_leader_2}

[LEADER 3]
...
```

F.1.3 SYSTEM MESSAGE (ALL FOUR TAG ROLES)

```
You analyse optimization problems and describe them with short,
reusable TAGS. A good tag names ONE component or property of the problem:
     the
object type (e.g. circle-packing), the container/shape, the objective
(e.g. max-radius-sum), structural properties (symmetric / asymmetric),
    the
dimensionality (dimension-2 / dimension-3 / any-dimension), constraints,
    etc.
Tags are short lowercase hyphenated phrases. Be consistent and reuse
    existing
tags whenever they apply.
```

F.1.4 INITIAL DESCRIPTORS (FIRST PROBLEM OF A RUN)

```
PROBLEM (evaluate.py docstring):
{problem_statement}

Propose EXACTLY 6 tags describing this packing problem's distinct
    components/properties (object type, container/shape, objective,
    symmetry, dimensionality, constraints, ...). Respond with one block:
<tags>
- tag-one
- tag-two
</tags>
```

F.1.5 DESCRIPTOR ASSIGNMENT AND VOCABULARY EXTENSION (EVERY NEW PROBLEM)

```
EXISTING TAG SET:
{existing_tag_1}, {existing_tag_2}, ...

PROBLEM (evaluate.py docstring):
{problem_statement}

1) From the EXISTING set, list every tag that applies.
2) List any NEW tags needed for important components the set lacks.
Respond with two blocks (either may be empty):
<applicable>
- tag
</applicable>
<new_tags>
- tag
</new_tags>
```

F.1.6 RETROACTIVE DESCRIPTOR CHECK (ONE NEW TAG AGAINST ONE OLD PROBLEM)

```
TAG: {tag}

PROBLEM (evaluate.py docstring):
```

```
{problem_statement}

Does the tag '{tag}' accurately describe a component/property of this
    problem? Respond <answer>yes</answer> or <answer>no</answer>.
```

F.2 NEW-DIRECTION TAG INVENTORS

The NEW-DIRECTION operator first asks for a *name* for where to go, then asks the designer for a problem that embodies it. The cluster sub-mode samples one OPTICS cluster of the tag embeddings and asks for a sibling value on the same conceptual axis; the divergent sub-mode shows every cluster and asks for a new axis.

F.2.1 SIBLING TAG (CLUSTER SUB-MODE)

```
CLUSTER OF RELATED TAGS:
{cluster_tag_1}, {cluster_tag_2}, ...

These tags lie on a shared conceptual AXIS — an underlying dimension
    along which they vary (e.g. dimensionality, object type, container
    shape, a numeric parameter, or the objective). Propose EXACTLY ONE
    new tag that stays ON THAT SAME AXIS but takes a DIFFERENT value NOT
    already present — a sibling/variation that explores a new region of
    the SAME family, one step away from the cluster (NOT a narrower sub-
    case, and NOT an unrelated or orthogonal concept).

The tag must be DISTINCT from every tag already listed above, stay close
    to the cluster (same family), and be meaningful in the domain of
    packing problems. Respond with one block:
<tags>
- tag
</tags>
```

F.2.2 DIVERGENT TAG (DIVERGENT SUB-MODE)

```
EXISTING TAG CLUSTERS:
[Cluster 1] {cluster_1_tag_1}, ...
[Cluster 2] {cluster_2_tag_1}, ...

First, briefly summarize the theme of EACH cluster. Then THINK about what
     is NOT yet covered, and propose EXACTLY ONE new tag that goes in an
    original, DIVERGENT direction from all existing clusters (a genuinely
     new axis) while staying within the domain of packing problems. End
    with one block:
<tags>
- tag
</tags>
```

Domain	Method	LLM calls: Prob. gen.	LLM calls: Aux.	LLM calls: Solver dev.	Tokens (M)	Cross-evals (k)	CPU (h)	API cost ($)	Wall-clock (h)
Geometric packing	Ψ-Evolve	358	4926	1430	42.3	10.2	10.8	15.6	34.0
	Context-free	401	1627	1470	42.3	10.2	6.9	15.6	23.6
	CVT-MAP-Elites†	684	0	1409	56.8	18.5	16.5	20.6	31.1
	POET	1783	0	3135	134.9	133.2	147.0	49.7	97.8
Spectral	Ψ-Evolve	372	8909	1427	42.3	10.2	11.7	15.7	65.3
	Context-free	375	982	1426	35.9	10.2	17.8	13.5	39.2
	CVT-MAP-Elites	786	0	1683	64.5	21.1	10.5	22.7	15.6
	POET‡	1688	0	4511	160.4	128.6	46.3	56.7	61.5
Geometry of numbers	Ψ-Evolve	429	6229	1418	45.1	13.7	5.9	15.9	25.6
	Context-free	388	1173	1420	34.9	10.8	4.7	11.1	9.5
	CVT-MAP-Elites	676	0	1409	55.0	20.3	8.8	20.0	14.6
	POET	1283	0	1630	85.9	102.0	35.9	30.6	25.9
All domains	Ψ-Evolve	386	6688	1425	43.2	11.4	9.5	15.7	41.6
	Context-free	388	1260	1439	37.7	10.4	9.8	13.4	24.1
	CVT-MAP-Elites	720	0	1500	59.0	20.2	11.4	21.2	20.4
	POET	1555	0	3092	117.6	119.2	85.0	42.5	61.7

Table 14: Compute per generation run of approximately 100 discovered problems (mean over 3 seeds). The "All domains" rows report the unweighted mean across domains. No local GPU time is used, as all LLM inference is performed through hosted APIs. †/‡ indicate that usage-log quantities (problem-generation and auxiliary calls, tokens, cross-evaluations, CPU time, and API-equivalent cost) are averaged over 2/1 seeds, respectively, because resumed runs truncated the remaining usage logs; solver-development counts and wall-clock times remain available for all seeds.

Method	Cumulative cross-evals: N=25	N=50	N=100	Cross-evals / problem: N=25	N=100	Wall-clock (h): N=25	N=50	N=100	α
Ψ-Evolve	625	2401	10236	46	286	5.9	16.5	39.8	1.38
Context-free	625	2592	9600	46	198	5.3	11.2	23.4	1.08
CVT-MAP-Elites	1556	7520	20163	190	494	4.0	8.5	20.4	1.19
POET	1959	17655	119200	207	7262	8.5	23.3	61.7	1.44

Table 15: Empirical scaling with the number N of discovered problems. Values pool the three domains and three seeds; cross-evaluation statistics include only runs with complete usage logs. The wall-clock exponent α is the slope of $\log(\text{wall-clock})$ against $\log N$ for $N \in \{25, 50, 75, 100\}$ and characterizes the measured regime rather than the asymptotic complexity of the algorithms.

G Generation cost and scaling

Computational cost and scaling. Tables 14 and 15 report the computational cost of problem-space illumination. Although Ψ-Evolve introduces substantially more auxiliary LLM interactions than the baselines due to semantic tagging and ontology expansion, these calls are comparatively short. Averaged across domains, Ψ-Evolve consumes 43.2M tokens, 14.6% more than context-free generation (37.7M), but 26.8% fewer than CVT-MAP-Elites (59.0M) and 63.3% fewer than POET (117.6M). The corresponding API-equivalent cost is $15.7 per generation run, compared with $13.4 for context-free generation, $21.2 for CVT-MAP-Elites, and $42.5 for POET. In particular, Ψ-Evolve and context-free generation use nearly identical solver-development budgets (1425 and 1439 calls on average), indicating that the additional cost of Ψ-Evolve does not arise from allocating a larger solver-search budget. In contrast, the POET adaptation performs substantially more solver-development calls (3092 on average), as its problem-solver pairs continue to co-evolve throughout the run.

Cross-evaluation is the main component whose cost grows with the discovered problem set. At approximately $N = 100$, Ψ-Evolve performs 11.4k cross-evaluations on average, only 9.6% more

than context-free generation, but 43.6% fewer than CVT-MAP-Elites and 90.4% fewer than POET. Correspondingly, Ψ-Evolve requires 9.5 CPU hours on average, comparable to context-free generation (9.8 h) and CVT-MAP-Elites (11.4 h), while POET requires 85.0 h. Thus, despite maintaining both an adaptive semantic representation and an empirical transfer matrix, the additional semantic machinery of Ψ-Evolve does not result in a large increase in local compute relative to the non-co-evolutionary baselines.

The main overhead of Ψ-Evolve relative to context-free generation and CVT-MAP-Elites is instead wall-clock time. A complete run takes 41.6 h on average, compared with 24.1 h for context-free generation and 20.4 h for CVT-MAP-Elites. This increase is not mirrored by CPU usage and is consistent with latency and synchronization overhead from the larger number of short hosted-API interactions required for semantic inference and descriptor expansion. Nevertheless, Ψ-Evolve remains substantially less expensive than POET under the present implementation: POET requires 61.7 h of wall-clock time, 117.6M tokens, and an API-equivalent cost of $42.5, compared with 41.6 h, 43.2M tokens, and $15.7 for Ψ-Evolve. This difference illustrates an important computational distinction between Ψ-Evolve and continuous problem-solver co-evolution: Ψ-Evolve independently optimizes a solver when a problem is introduced and subsequently reuses it for transfer evaluation, whereas POET repeatedly updates solvers during the evolutionary process.

As expected from explicitly maintaining solver-problem transfer information, the cumulative number of cross-evaluations increases rapidly with the size of the discovered set. For Ψ-Evolve, it grows from 625 evaluations near $N = 25$ to 10,236 near $N = 100$, while the number of evaluations associated with each newly added problem increases from 46 to 286. This growth does not translate directly into quadratic end-to-end runtime over the regime studied here: the empirical wall-clock exponent over $N \in \{25, 50, 75, 100\}$ is 1.38 for Ψ-Evolve, compared with 1.08 for context-free generation, 1.19 for CVT-MAP-Elites, and 1.44 for POET. The measured wall-clock scaling reflects batching and parallel execution as well as hosted-LLM latency, and should therefore not be interpreted as the asymptotic complexity of the algorithms. In particular, the all-pairs transfer computation remains a scaling limitation of the current Ψ-Evolve implementation. Scaling to substantially larger archives will likely require sparse or approximate transfer evaluation, for example by evaluating new solvers on representative subsets of the archive or by learning a predictor of cross-problem transfer.

Some usage logs were truncated when experimental runs were resumed. The affected entries are marked in Table 14, and the corresponding cross-evaluation statistics in Table 15 are computed only from runs with complete usage logs. We therefore use these measurements primarily to characterize the order and sources of computational cost rather than to make fine-grained claims from small differences between methods.

H HYPERPARAMETERS

Table 16: Ψ-Evolve hyperparameters.

Component	Hyperparameter	Value
Budget	Turns × problems per turn	100 problems
	Concurrent generation / solve workers	4 / 4
Operators	P(crossover, broaden, add-tag, new-direction)	$(\frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{3}{4})$
	P(new-direction: cluster, divergent)	$(0.5, 0.5)$
	Unavailable operator	removed, rest renormalised
	Parent (base) selection	uniform over cell leaders
	Broaden examples	2
	Example-selection softmax temperature	1.0
Archive	Descriptor	problem-statement embedding
	Embedding model / metric	`text-embedding-3-small` / cosine
	Cell split threshold n_{crit}	5
	OPTICS `min_samples` (cells / tag clusters)	2 / 2
	Initial tags	6
Fitness	Fitness	generality: $\frac{1}{N-1}\sum_{j\neq i} S_{ji} / \max_k S_{ki}$
	Solver inheritance	none (every problem starts from its own seed)
Solver	ShinkaEvolve generations per problem	15
	Cross-evaluation timeout	90 s
LLMs	Problem generator	GPT-5.6 Luna (high effort)
	Tagger	GPT-5.6 Luna (low effort)
	ShinkaEvolve proposals	GPT-5.6 Luna (low / medium effort)
	Generation retries	3

Table 17: CVT-MAP-Elites hyperparameters.

Component	Hyperparameter	Value
Budget	Total problems	100
	Initial problems	4
	Children per round	4 (2 crossover + 2 mutation)
Operators	P(broaden, new-direction) within mutation	$(0.5, 0.5)$
	Parent selection	uniform over active archive
	New-direction examples	3 nearest active problems (by descriptor)
Archive	Descriptor	solver-code embedding
	Embedding model / metric	`text-embedding-3-small` / cosine
	Active archive size	25 (rest retired, not deleted)
	Admission	top-25 by fitness; no novelty threshold
Fitness	Fitness	generality over the round's candidate set
	Solver inheritance	none (every problem starts from its own seed)
	Minimal criterion	none
Solver	ShinkaEvolve generations per problem	15
	Cross-evaluation timeout	90 s
LLMs	Problem generator	GPT-5.6 Luna (high effort)
	ShinkaEvolve proposals	GPT-5.6 Luna (low / medium effort)

Algorithm 1 Ψ-*Evolve*: open-ended illumination of a problem domain

Require: validity constraints $\mathcal{C}$, solver procedure Solve, evaluator Eval, number of turns T, batch size B, operator probabilities $\boldsymbol{\pi}$, new-direction probabilities $\boldsymbol{\rho}$

$\{\mathcal{P}, \mathcal{D}, \mathcal{A}\} \leftarrow \{\varnothing, \varnothing, \varnothing\}$
$S \leftarrow [\,]$
$p_1 \leftarrow \textsc{GenerateInitial}(\mathcal{C})$
$p_1 \leftarrow \textsc{ValidateOrRepair}(p_1, \mathcal{C})$
$T(p_1) \leftarrow \textsc{InferTags}(p_1)$
$\mathcal{D} \leftarrow \mathcal{D} \cup T(p_1)$
$a_1 \leftarrow \mathrm{Solve}(p_1; \theta_{p_1}^{(0)})$
$\mathcal{P} \leftarrow \{p_1\}$
$S_{11} \leftarrow \mathrm{Eval}(a_1, p_1)$
$\mathcal{A} \leftarrow \textsc{BuildArchive}(\mathcal{P}, \mathcal{D}, S)$
for $t = 2, \ldots, T$ **do**
  $L, \mathcal{B} \leftarrow \textsc{Representatives}(\mathcal{A}), \varnothing$
  for $b = 1, \ldots, B$ **do**
    $p_{\text{base}} \sim \mathrm{Uniform}(L)$
    $\mathcal{O}_{\text{avail}} \leftarrow \textsc{AvailableOperators}(p_{\text{base}}, L, \mathcal{D})$
    $o \sim \textsc{Renormalize}(\boldsymbol{\pi}, \mathcal{O}_{\text{avail}})$
    if $o = \textsc{Crossover}$ **then**
      $p_{\text{other}} \sim \mathrm{Uniform}(L \setminus \{p_{\text{base}}\})$
      $\hat{p} \leftarrow \textsc{Crossover}(p_{\text{base}}, p_{\text{other}})$
    else if $o = \textsc{Broaden}$ **then**
      $\hat{p} \leftarrow \textsc{Broaden}(p_{\text{base}})$
    else if $o = \textsc{AddTag}$ **then**
      $\tau \sim \Pr(\tau) \propto 1/(n(\tau)+1),\ \tau \notin T(p_{\text{base}})$
      $\hat{p} \leftarrow \textsc{AddTag}(p_{\text{base}}, \tau)$
    else
      $m \sim \boldsymbol{\rho}$
      if $m = \text{local}$ **then**
        $\tau_{\text{new}} \leftarrow \textsc{RefineDirection}(\mathcal{D})$
        $\hat{p} \leftarrow \textsc{GenerateFromNewTag}(\tau_{\text{new}}, p_{\text{base}})$
      else
        $\tau_{\text{new}} \leftarrow \textsc{DivergentDirection}(\mathcal{D})$
        $\hat{p} \leftarrow \textsc{GenerateFromNewTag}(\tau_{\text{new}}, \varnothing)$
      end if
    end if
    $p \leftarrow \textsc{ValidateOrRepair}(\hat{p}, \mathcal{C})$
    $T(p), \mathcal{B} \leftarrow \textsc{InferTags}(p), \mathcal{B} \cup \{p\}$
  end for
  for all $p \in \mathcal{B}$ **in parallel do**
    $a_p \leftarrow \mathrm{Solve}(p; \theta_p^{(0)})$ ▷ Independent solver optimization for the new batch
  end for
  $\mathcal{P}, \mathcal{D} \leftarrow \mathcal{P} \cup \mathcal{B}, \mathcal{D} \cup \bigcup_{p \in \mathcal{B}} T(p)$
  for all newly required pairs (a_j, p_i) **do**
    $S_{ji} \leftarrow \mathrm{Eval}(a_j, p_i)$ ▷ Complete all newly required cross-evaluations
  end for
  $\{e_\tau : \tau \in \mathcal{D}\} \leftarrow \textsc{EmbedTags}(\mathcal{D})$ ▷ Update semantic representation
  $\mathcal{K} \leftarrow \mathrm{OPTICS}(\{e_\tau : \tau \in \mathcal{D}\})$
  $\textsc{RetroactiveTagCheck}(\mathcal{P}, \mathcal{D})$
  for all $p_i \in \mathcal{P}$ **do**
    $b_i \leftarrow \max_k S_{ki}$ ▷ Recompute global quality under the new state of knowledge
  end for
  for all $p_j \in \mathcal{P}$ **do**
    $F(p_j) \leftarrow \frac{1}{|\mathcal{P}|-1} \sum_{j \neq i} \textsc{Normalize}(S_{ji}, b_i)$
  end for
  $\mathcal{A} \leftarrow \textsc{UpdateAdaptiveArchive}(\mathcal{A}, \mathcal{P}, \mathcal{D}, \mathcal{K}, F)$ ▷ Refine adaptive archive and retain highest-generality elites
end for
return $\mathcal{A}, \mathcal{P}, \mathcal{D}, S$

Table 18: POET hyperparameters.

Component	Hyperparameter	Value
Budget	Total problems (population cap)	100
	Initial problems	4
	Children per round	8
Operators	P(broaden, new-direction)	$(0.5, 0.5)$
	Parent selection	uniform over population
	New-direction examples	3 nearest problems (by embedding)
Population	Pruning	none (population grows to the cap)
	Minimal criterion	strict improvement after a short ShinkaEvolve run
Solvers	Solver inheritance	child inherits parent's solver
	Transfer	solver replaced by the best strictly better cross-evaluated solver
	ShinkaEvolve generations per run	15 (round 0), then 2
	Cross-evaluation timeout	90 s
LLMs	Problem generator	GPT-5.6 Luna (high effort)
	ShinkaEvolve proposals	GPT-5.6 Luna (low / medium effort)