SMCEVOLVE: PRINCIPLED SCIENTIFIC DISCOVERY VIA SEQUENTIAL MONTE CARLO EVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-driven program evolution has emerged as a powerful tool for automated scientific discovery, yet existing frameworks offer no principled guide for designing their individual components and provide no guarantee that the search converges. We introduce SMCEVOLVE, which recasts program search as sampling from a reward-tilted target distribution and approximates it with a Sequential Monte Carlo (SMC) sampler. From this view, three core mechanisms emerge as principled components: adaptive parent resampling, mixture of mutation with acceptance, and automatic convergence control. We further prove finite-sample guarantees for the deployed sampler, both for the reward-tilted distribution it tracks and for its best program. Across math, algorithm efficiency, symbolic regression, and end-to-end ML research benchmarks, SMCEVOLVE surpasses state-of-the-art evolving systems while using fewer LLM calls under self-determined termination.

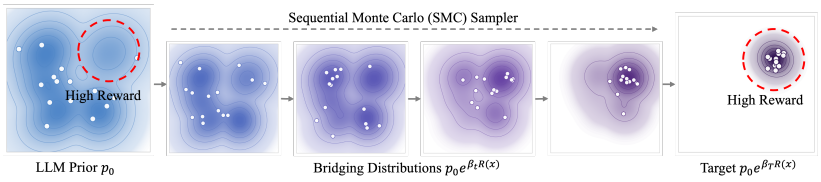


Figure 1: **LLM-driven program evolution targeting the reward-tilted distribution via Sequential Monte Carlo (SMC).** The contour depicts the distribution over the program space; each white dot is a particle (program) in the evolving population, and the dashed circle marks the high-reward region. Starting from particles drawn from the LLM prior p_0 (*left*), where the high-reward region is rarely covered, the population is driven by the LLM toward new programs that form a sequence of bridging distributions $p_0\, e^{\beta_t R(x)}$ with increasing reward intensity β_t, and ultimately concentrates on the target $p_0\, e^{\beta_T R(x)}$ (*right*). This yields a principled evolutionary framework SMCEVOLVE with convergence guarantees, in contrast to ad hoc pipelines.

1 INTRODUCTION

LLM-driven agents have enabled automated scientific discovery at scale: given the problem description and evaluator, the agent can autonomously search for programs that optimize a target objective, iteratively refining candidates based on evaluation feedback (Ye et al., 2024; Liu et al., 2024; Romera-Paredes et al., 2024; Novikov et al., 2025; Lange et al., 2025). This paradigm has rapidly expanded across diverse domains, including mathematical discovery (Georgiev et al., 2025), symbolic regression (Shojaee et al., 2024), materials design (Abhyankar et al., 2026), machine learning (Chan et al., 2024; Imajuku et al., 2025), and end-to-end research automation (Yamada et al., 2025).

Despite these successes, current agent design remains largely *empirical*, and the area lacks a unifying *theoretical* framework that guides the design. A typical pipeline, exemplified by AlphaEvolve (Novikov et al., 2025), ShinkaEvolve (Lange et al., 2025), and others (Ye et al., 2024; Liu et al., 2024; Jiang et al., 2026), maintains a population of candidate programs, prompts an LLM to mutate a selected parent based on context, and adds the evaluated offspring back into the population. The absence of theoretical guidance leads to several limitations. First, there is no principled guide

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.

for designing each component within the pipeline: components such as parent selection, context construction, and population management are hand-crafted in isolation and validated through extensive ablations rather than derived from a coherent principle that connects them. Second, the convergence of the pipeline is not guaranteed, leading to substantial randomness and high variance across runs; moreover, existing systems cannot distinguish whether the population has converged to a high-reward region or merely stalled at a local mode, forcing the number of iterations to be fixed in advance—either too small to reach a good solution, or too large, wasting compute. These observations motivate the central question of this work:

Can we ground LLM-driven program evolution in a rigorous probabilistic framework that unifies its components and provides convergence guarantees?

We address this question from a simple observation: searching for high-reward programs with an LLM can be cast as *sampling from a reward-tilted target distribution* $p^*(x \mid q) \propto p_0(x \mid q)\, e^{\beta R(x)}$, which reweights the LLM prior p_0 over programs x for a given problem q toward those with high evaluation reward R. Since directly sampling from p^* is intractable, we turn to *Sequential Monte Carlo (SMC) samplers* (Del Moral et al., 2006; Dai et al., 2022), which progressively transport particles from the prior p_0 toward the target p^* through a sequence of bridging distributions with increasing reward weight. Treating each candidate program as a particle, we derive SMCEVOLVE, a complete evolutionary framework whose three core mechanisms emerge directly from SMC theory:

- **Adaptive parent resampling.** Parents are resampled by importance weights derived from the current bridging distribution, where the influence of reward on selection strengthens as the search progresses—from near-uniform sampling that encourages exploration in early iterations, to sharply reward-focused sampling that drives exploitation in later iterations. This stands in contrast to existing systems, which rely on a fixed selection rule throughout the search.
- **Mixture of mutation with acceptance.** Following SMC, the LLM generates multiple proposals at each step, each accepted or rejected with a probability related to the current bridging distribution. To accelerate convergence, we further design a mixture of LLM-driven kernels along two axes—edit scope (local diff vs. full rewrite) and information source (single-parent vs. cross-program inspiration)—and adaptively select among them via Thompson sampling (Russo et al., 2018), which favors kernels that have recently produced high-reward proposals. In contrast, existing systems accept the proposal unconditionally and rely on either a single mutation strategy or a fixed-weight combination of strategies.
- **Automatic convergence control.** The number of iterations is determined adaptively from the state of the population rather than fixed in advance: the search proceeds in small steps when the population is still spread across diverse candidates and in larger steps once it concentrates on high-reward programs, and terminates automatically once it reaches the target distribution. In contrast, existing systems run for a fixed number of iterations specified in advance, without a signal to detect convergence or stalling.

Together, these three mechanisms instantiate a complete SMC sampler for LLM-driven program search, and the resulting framework admits formal convergence analysis with finite-sample complexity guarantees. Our main contributions are as follows:

- **A principled framework for evolutionary program search.** We introduce SMCEVOLVE, the first evolving agent whose components—parent resampling, mutation, and convergence control—are derived from SMC theory rather than hand-crafted in isolation.
- **Finite-sample convergence analysis.** We show that SMCEVOLVE, including its reward-aware acceptance rule, approximates its reward-tilted limit to accuracy ϵ with $\mathcal{O}(\epsilon^{-2})$ particles, and that its best program provably comes close to high rewards that the LLM alone rarely achieves (Theorem 4.1); with exact Metropolis–Hastings kernels, this limit is p^* and the LLM-call budget is bounded in terms of particle budget, mutation mixing, and reward intensity.
- **Empirical validation across diverse domains.** We evaluate SMCEVOLVE on multiple domains spanning math (Novikov et al., 2025), algorithm efficiency (Press et al., 2025), symbolic regression (Shojaee et al., 2025), and LLM pretraining (Karpathy, 2026), where it surpasses state-of-the-art evolving systems on the majority of tasks while consuming fewer LLM calls under automatic convergence control.

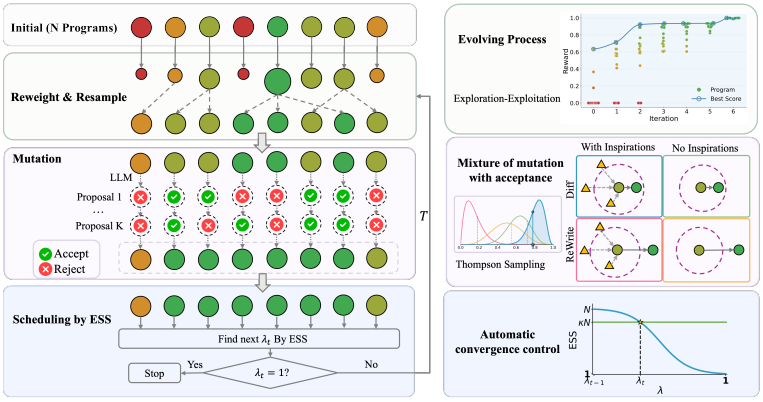


Figure 2: **Overview of SMCEVOLVE.** *Left:* the main loop. Starting from N initial programs, each of T iterations *(i)* reweights and resamples particles, *(ii)* mutates each parent through a K-step MH chain of LLM proposals and accept/reject updates, and *(iii)* schedules the next λ_t via ESS, terminating when $\lambda_t = 1$. *Right, top to bottom:* the evolving process on circle packing (best-so-far reward across iterations); the mixture of four mutation kernels selected by Thompson sampling; and the automatic convergence control finding λ_t at the intersection of $\mathrm{ESS}(\lambda)$ with the threshold κN.

2 Related Work

LLM-driven program evolution. FunSearch (Romera-Paredes et al., 2024), AlphaEvolve (Novikov et al., 2025), ShinkaEvolve (Lange et al., 2025), ReEvo (Ye et al., 2024), and Evolution of Heuristics (Liu et al., 2024) evolve programs with LLM mutations and evaluator feedback, but design parent selection, acceptance, and stopping as separate heuristics; SMCEVOLVE derives these components jointly from SMC. AlphaEvolve and ShinkaEvolve arise as special cases in which these components are fixed (Section 3.3).

SMC with language models. SMC has been used to steer LLM generation or reasoning within a single episode (Lew et al., 2023; Zhao et al., 2024; Feng et al., 2025). There, particles are partial or complete responses, reweighted by constraints or by learned scores of partial sequences. Closest to our setting, ModelSMC (Wahl et al., 2026) treats candidate mechanistic models as particles weighted by likelihood-based criteria, whereas SMCEVOLVE targets a reward-tilted distribution defined by an arbitrary evaluator.

SMC theory. Our analysis builds on SMC samplers and Feynman–Kac particle theory (Del Moral et al., 2006; Del Moral, 2004; Dai et al., 2022) and on finite-sample SMC bounds (Marion et al., 2023). Feynman–Kac error bounds hold for general mutation kernels, which is why Theorem 4.1 needs no invariance. Our reward-tempered path is the geometric path of annealed importance sampling (Neal, 2001). Section A gives an extended discussion.

3 Method: Program Evolution as Sequential Monte Carlo

We formulate program search as sampling from a reward-tilted target distribution that balances reward maximization against the LLM prior (Section 3.1), and approximate this intractable target with a Sequential Monte Carlo sampler whose components—intermediate distributions, importance weights, mutation kernel, and adaptive scheduling—are derived from SMC theory (Section 3.2).

Notation. Throughout, let $\mathcal{X}$ denote the countable program space for a task q. All distributions over $\mathcal{X}$ are probability mass functions, and integrals $\int_{\mathcal{X}} f(x)\,dx$ denote sums over programs. Given an LLM, let $p_{\mathrm{LLM}}(\cdot \mid \mathcal{C})$ denote the distribution induced by sampling from the LLM under context $\mathcal{C}$. We define $p_0(\cdot \mid q) = p_{\mathrm{LLM}}(\cdot \mid q)$ as the LLM prior for task q, and $Q_t(\cdot \mid x, \mathcal{C}_t) = p_{\mathrm{LLM}}(\cdot \mid x, \mathcal{C}_t)$ as the stage-conditional LLM proposal distribution used for mutation, conditioned on the current program x and context $\mathcal{C}_t$. Let $R : \mathcal{X} \to \mathbb{R}$ be the reward function, assumed bounded on $\mathrm{supp}(p_0)$. Write $R_- = \inf_{\mathrm{supp}(p_0)} R$, $R_+ = \sup_{\mathrm{supp}(p_0)} R$, and $\Delta_R = R_+ - R_- < \infty$.

3.1 Reward-Tilted Target Distribution

Across various domains including mathematical discovery (Georgiev et al., 2025), symbolic regression (Shojaee et al., 2024), and materials design (Abhyankar et al., 2026), scientific discovery shares a common structure: a search problem in program space $\mathcal{X}$ guided by an evaluator, where the goal is to find the program that maximizes the reward $R(x)$. However, the program space is discrete and large, making direct maximization intractable. We therefore leverage an LLM prior p_0 to guide the search toward plausible programs. To balance reward maximization with adherence to the prior, we formulate LLM-driven program search as the following KL-regularized variational optimization problem:

$$\max_{p} \; \mathbb{E}_{x\sim p}\big[R(x)\big] - \frac{1}{\beta}\, D_{\mathrm{KL}}\big(p \,\|\, p_0\big), \tag{1}$$

where the maximization is over all probability mass functions on $\mathcal{X}$ that are absolutely continuous with respect to p_0. The following result characterizes the optimal solution.

Lemma 3.1 (Reward-tilted target distribution)**.** *The unique maximizer of* (1) *is*

$$p^*(x \mid q) \;\triangleq\; \frac{1}{Z(q)}\, p_0(x \mid q)\, e^{\beta R(x)}, \tag{2}$$

where $Z(q) = \int p_0(x' \mid q)\, e^{\beta R(x')}\, dx'$ is the partition function.

The proof follows from standard calculus of variations: setting the functional derivative of the Lagrangian (with a normalization constraint) to zero yields (2) directly; see Appendix C for details. The bounded-oscillation assumption ensures that $e^{\beta R_-} \le Z(q) \le e^{\beta R_+} < \infty$, so (2) is a well-defined probability mass function on $\mathcal{X}$ that is absolutely continuous with respect to p_0.

The parameter β interpolates between the LLM prior ($\beta \to 0$) and pure reward maximization ($\beta \to \infty$). Sampling from p^* is intractable, as the normalization term $Z(q)$ requires summing over all programs in an extremely large program space.

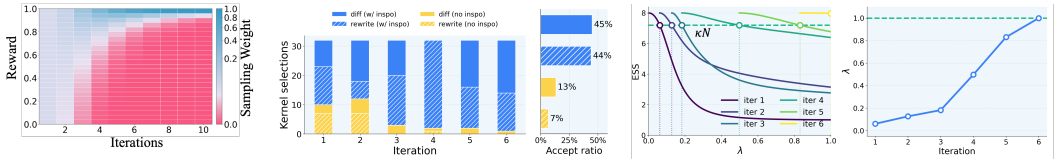
3.2 Sequential Monte Carlo Samplers

To sample from the intractable target $p^*(\cdot \mid q)$, we turn to the Sequential Monte Carlo (SMC) method (Del Moral et al., 2006). It bridges p_0 to p^* through a sequence of *bridging distributions* $p_0, p_1, \ldots, p_T = p^*$, with p_t written as $p_t(x) = \gamma_t(x)/Z_t$ for an unnormalized density γ_t and normalizing constant Z_t. The sampler maintains N particles $\{x_{t-1}^{(n)}\}_{n=1}^N$ approximating p_{t-1}, and advances them as samples from p_t by repeating three operations: *reweighting* each particle by an importance weight, *resampling* them to discard low-weight particles, and *mutating* each particle to inject diversity. The reweighting step assigns the importance weight as (Dai et al., 2022; Del Moral et al., 2006)

$$w_t(x_{t-1}, x_t) \;=\; \frac{\gamma_t(x_t)\, L_{t-1}(x_t, x_{t-1})}{\gamma_{t-1}(x_{t-1})\, M_t^K(x_{t-1}, x_t)}, \tag{3}$$

where $(x_{t-1}, x_t) = (x_{t-1}^{(n)}, x_t^{(n)})$ for each particle n, $M_t^K(x_{t-1}, \cdot)$ is the *stage forward kernel*—the K-fold composition of a one-step kernel M_t used to mutate each particle from stage $t-1$ to stage t—and $L_{t-1}(x_t, \cdot)$ is an auxiliary *backward kernel* introduced by the SMC framework to make (3) tractable. Crucially, the effectiveness of SMC hinges on how these components are *jointly designed for the target problem*: the bridging distributions $\{p_t\}$ determine the annealing path, M_t controls exploration and proposal quality, and the backward kernel L_{t-1} stabilizes importance weighting.

Kernel properties in classical SMC. Classical SMC analyses ask the forward kernel M_t to behave well in two cases: starting from p_t, applying M_t should keep the distribution at p_t (p_t*-invariance*); starting away from p_t, iterating M_t should drive the distribution toward p_t (*ergodicity*). Invariance ensures that the move step, applied to a particle already distributed as p_t, does not drift the distribution away from p_t. But invariance alone is not enough for finite-budget search. The identity kernel, for instance, is perfectly p_t-invariant but simply copies particles and provides no exploration after resampling. Uniform ergodicity rules out such ineffective kernels by giving a quantitative total-variation bound after K mutation steps; smaller ρ corresponds to faster mixing. Our main result (Theorem 4.1) does not require either property; invariance only identifies the population limit with p_t.



(a) Parent resampling (b) Kernel selection & acceptance (c) ESS curves & λ_t schedule

Figure 3: **Case study of SMCEVOLVE on circle packing**, illustrating the three mechanisms. **(a)** Resampling probability of each particle (ranked by reward) across iterations. **(b)** Per-iteration selection counts and cumulative MH acceptance rates of the four mutation kernels $M^{(1)}, \ldots, M^{(4)}$. **(c)** The ESS curves $\mathrm{ESS}(\lambda)$ at each iteration (left) and the resulting schedule λ_t (right); each λ_t is set where $\mathrm{ESS}(\lambda)$ crosses the threshold κN (green dashed).

Definition 3.2 (p_t-invariance)**.** *A Markov kernel M_t on $\mathcal{X}$ is* p_t-invariant *if applying M_t to a state distributed as p_t leaves the distribution unchanged:*

$$\int_{\mathcal{X}} p_t(x)\, M_t(x, y)\, dx \;=\; p_t(y), \qquad \forall\, y \in \mathcal{X}. \tag{4}$$

Definition 3.3 (Uniform ergodicity; Meyn & Tweedie, 2012, Ch. 16)**.** *A Markov kernel M_t with invariant distribution p_t is* uniformly ergodic *if there exist constants $C \geq 1$ and $\rho \in (0, 1)$ such that, for every $x \in \mathcal{X}$ and every $K \geq 1$,*

$$\left\| M_t^K(x, \cdot) - p_t \right\|_{\mathrm{TV}} \;\leq\; C \cdot \rho^K, \tag{5}$$

where $\|\mu - \nu\|_{\mathrm{TV}} = \sup_{A \subseteq \mathcal{X}} |\mu(A) - \nu(A)|$ is the total variation distance.

3.3 SMCEVOLVE: LLM-DRIVEN PROGRAM EVOLUTION VIA SMC

Building on this perspective, we instantiate SMC for program evolution by designing problem-specific components tailored to LLM-based search. Our method, SMCEVOLVE, integrates (i) *adaptive parent resampling* aligned with reward shaping, (ii) a *mixture of mutation kernels with acceptance correction* to balance exploration and validity, and (iii) *automatic convergence control* via adaptive annealing. These designs are summarized in Figure 2, with a case study on circle packing shown in Figure 3.

Adaptive parent resampling. We construct the bridging distributions $\{p_t\}_{t=0}^T$ as a geometric annealing path between the LLM prior $p_0(\cdot \mid q)$ and the target $p^*(\cdot \mid q)$, with unnormalized densities

$$\gamma_t(x) \;=\; p_0(x \mid q)\, \exp\big(\beta_t R(x)\big), \qquad \beta_t \;=\; \lambda_t\, \beta, \tag{6}$$

where the temperature schedule $0 = \lambda_0 < \lambda_1 < \cdots < \lambda_T = 1$ interpolates the reward intensity from zero to its target value β. The endpoints recover $p_0(\cdot \mid q)$ at $t = 0$ and $p^*(\cdot \mid q)$ at $t = T$ from (2).

Next, we specialize the backward kernel L_{t-1} as the time reversal (Dai et al., 2022) of M_t^K under p_t,

$$L_{t-1}(x_t, x_{t-1}) \;=\; \frac{p_t(x_{t-1})\, M_t^K(x_{t-1}, x_t)}{p_t(x_t)}. \tag{7}$$

Substituting (7) into (3) cancels the forward and backward kernels, leaving a weight that is independent of x_t and depends only on the parent's reward (full derivation in Appendix E):

$$w_t(x_{t-1}, x_t) \;=\; \frac{\gamma_t(x_{t-1})}{\gamma_{t-1}(x_{t-1})} \;=\; \exp\big((\beta_t - \beta_{t-1})\, R(x_{t-1})\big). \tag{8}$$

Here (7) is a proper backward kernel only if M_t^K preserves p_t; otherwise, (8) is still the Feynman–Kac potential analyzed in Section 4. Normalized across the N particles, the resampling probability of particle n is the temperature-based softmax over particle rewards,

$$W_t^{(n)} \;=\; \frac{\exp\big((\beta_t - \beta_{t-1})\, R(x_{t-1}^{(n)})\big)}{\sum_{m=1}^N \exp\big((\beta_t - \beta_{t-1})\, R(x_{t-1}^{(m)})\big)}, \tag{9}$$

where $\beta_t - \beta_{t-1} > 0$ acts as the inverse temperature: small increments flatten the softmax toward uniform sampling, while large increments sharpen it onto top-reward particles.

Case study. Figure 3a illustrates this transition on circle packing. In early iterations the increment $\beta_t - \beta_{t-1}$ is small, so reward exerts little influence on the resampling weight and sampling probabilities are nearly uniform across particles; in later iterations the increment grows, and the weight concentrates resampling on high-reward parents.

Design of the forward kernel. Leveraging the in-context learning capability of LLMs, we design the one-step forward/mutation kernel M_t around an LLM-based proposal $x' \sim Q_t(\cdot \mid x, \mathcal{C}_t)$. However, raw LLM proposals do not depend on the current temperature β_t, and a single prompt explores only one way of editing a program. To address this, we augment the proposal mechanism with a mixture of mutation kernels for exploration and a reward-aware Metropolis–Hastings (MH) accept/reject step applied to multiple candidate samples. Specifically, each parent particle is propagated by repeatedly applying a one-step mutation kernel for K iterations, yielding the resulting K-step mutation block M_t^K.

Reward-aware MH acceptance. Each LLM-proposed candidate x' is accepted with probability

$$\alpha_t(x, x') = \min\{1,\ e^{\beta_t(R(x')-R(x))}\}, \tag{10}$$

and otherwise the parent x is kept. Iterating this proposal-and-acceptance step K times per particle yields a K-step MH chain, whose final state is taken as the mutated particle. This Metropolis rule applies the reward tilt $e^{\beta_t R}$ of the current bridge to the LLM's proposals: it always accepts reward-improving proposals and accepts reward-decreasing ones with probability that decays exponentially in the reward gap, so mutation explores freely at small β_t and becomes reward-greedy at large β_t. It requires only rewards, so it runs with black-box LLM APIs, and Theorem 4.1 covers it directly. When LLM likelihoods are available, the full MH ratio (24) additionally makes each step p_t-invariant (Appendices D and F), in which case $\eta_T = p^*$.

LLM proposal with context for ergodicity. Ergodicity governs how fast newly generated programs spread across the program space, and we accelerate it through the design of the LLM context $\mathcal{C}_t$. Following the mixture-of-kernels construction (Del Moral et al., 2006, Sec. 3.2.2), we instantiate Q_t as a mixture of four LLM proposal modes; composed with MH accept/reject, these modes induce the four one-step mutation kernels shown below.

	With inspiration	**No inspiration**
Diff	$M^{(1)}$: edit parent using peer programs	$M^{(2)}$: edit parent alone
ReWrite	$M^{(3)}$: rewrite parent from scratch using peer programs	$M^{(4)}$: rewrite parent from scratch

Diff kernels produce small `SEARCH/REPLACE` edits, while rewrite kernels generate the entire program from scratch. Inspiration programs include the top-k programs by reward and m maximally diverse ones, selected by picking programs whose text embeddings are farthest from the parent across the full evaluation history. Crucially, this history contains *every* program ever proposed by the LLM, including those rejected by the MH step, so that information from rejected proposals is not discarded but reused as inspiration for future mutations.

As the most useful kernel changes over the course of the search, we adaptively select among the four kernels via Thompson sampling (Russo et al., 2018): each kernel maintains a Beta posterior over its success rate, and at each step we sample from every posterior and pick the kernel with the highest draw, favoring recently successful kernels. Together, these four kernels cover both local and global moves and both single-parent and cross-program information, broadening the reachable set from any given particle and accelerating mixing.

Case study. Figure 3b confirms this behavior on circle packing. The inspiration-based kernels $M^{(1)}$ and $M^{(3)}$ attain substantially higher MH acceptance under (10) than their no-inspiration counterparts $M^{(2)}$ and $M^{(4)}$, and Thompson sampling progressively selects them more often.

Automatic convergence control. Rather than pre-specifying the inverse temperature sequence $(\lambda_t)_{t=0}^T$, we determine λ_t adaptively using the effective sample size (ESS; Del Moral et al., 2006) defined as

$$\mathrm{ESS}(\lambda) = \frac{\left(\sum_{n=1}^N u^{(n)}(\lambda)\right)^2}{\sum_{n=1}^N \left(u^{(n)}(\lambda)\right)^2}, \qquad u^{(n)}(\lambda) = e^{(\lambda-\lambda_{t-1})\beta R(x_{t-1}^{(n)})}. \tag{11}$$

Intuitively, $\mathrm{ESS}(\lambda)$ measures how many particles effectively contribute after reweighting—close to N when weights are balanced, and dropping toward 1 as they concentrate on a few high-reward particles. Since increasing λ sharpens the reward tilt and decreases $\mathrm{ESS}(\lambda)$, we set λ_t to be the largest λ whose reweighting still keeps a target fraction $\kappa \in (0, 1)$ of particles effective. This can be computed via bisection on $\mathrm{ESS}(\lambda_t) = \kappa N$ starting from λ_{t-1}; we iterate until $\lambda_t = 1$ (set directly when $\mathrm{ESS}(1) \geq \kappa N$). The total number of iterations T is therefore determined automatically and adapts to the difficulty of the task.

To illustrate, Figure 3c shows how the schedule is built on circle packing. At each iteration we compute the curve $\mathrm{ESS}(\lambda)$ as a function of λ (left panel) and get the next λ_t at its intersection with the threshold κN (green dashed line). The resulting schedule (right panel) grows slowly at first—small steps because the early ESS curves decay sharply—and accelerates as the curves flatten in later iterations, reaching $\lambda_t = 1$ at automatic termination.

The complete algorithm. Algorithm 1 in Appendix B assembles the three designs into a single loop: at each iteration t, we (i) reweight and resample particles via the temperature-based softmax (9), (ii) mutate each parent through a K-step MH chain with the Thompson-sampled mixture of LLM proposal modes, and (iii) advance λ_t by ESS-based bisection (11), terminating once $\lambda_t = 1$. The total cost is $N \cdot K \cdot T$ LLM calls, where T is determined adaptively rather than fixed in advance. More implementation details, including island-based parallelism, are in Appendix I.

A unified exploration–exploitation transition. The three core designs of SMCEVOLVE are not independent: they are coupled through the single temperature parameter β_t, which governs the exploration–exploitation trade-off across all of them. At small β_t, adaptive parent resampling (9) draws nearly uniformly across particles, MH acceptance (10) admits almost any LLM proposal, and ESS-driven scheduling (11) takes small steps in β_t—together driving exploration. At large β_t, parent resampling concentrates on high-reward particles, MH acceptance admits only reward-improving proposals, and ESS-driven scheduling takes larger steps—together driving exploitation. Through this single parameter, the three designs are no longer independent choices but coordinated mechanisms of one exploration–exploitation schedule that emerges from the SMC framework itself.

Existing frameworks as special cases. Existing agents such as AlphaEvolve (Novikov et al., 2025) and ShinkaEvolve (Lange et al., 2025) arise as special cases of our framework when each component reduces to a specific choice: adaptive parent resampling reduces to a fixed inverse temperature $\beta_t \equiv \beta$, mixture of mutation with acceptance reduces to a single LLM proposal ($K = 1$, $\alpha_t \equiv 1$) with unconditional acceptance, and automatic convergence control reduces to a pre-fixed iteration count T. This explains why existing frameworks work, while SMCEVOLVE provides a more general and unified principle for designing evolving agents.

4 CONVERGENCE ANALYSIS

Direct sampling from the reward-tilted target p^* is intractable. The SMC formulation of SMCEVOLVE instead constructs an interacting particle approximation through annealed reweighting, resampling, and mutation. We measure this approximation in the standard SMC sense: for bounded test functions f, the terminal empirical measure $\eta_T^N = N^{-1} \sum_n \delta_{x_T^{(n)}}$ should satisfy $\eta_T^N(f) \approx \eta_T(f)$, where η_T is the distribution that the population converges to as $N \to \infty$.

Let $\mathcal{B} := NTK$ denote the total number of LLM calls, accounting for N particles, T annealing stages, and K applications of the one-step mutation kernel per particle. Let $\eta_0 = p_0(\cdot \mid q)$ and

$$\eta_t(y) \;\propto\; \sum_{x \in \mathcal{X}} \eta_{t-1}(x)\, e^{\Delta\beta_t R(x)}\, M_t^K(x, y), \qquad \Delta\beta_t = \beta_t - \beta_{t-1}, \quad t = 1, \ldots, T, \tag{12}$$

i.e., η_t applies exactly the reweighting (8) and mutation steps of Algorithm 1 to an infinite population. The following theorem states the resulting finite-sample complexity bound.

Theorem 4.1 (Finite-sample complexity of SMCEVOLVE)**.** *Suppose the reward has finite oscillation Δ_R on the programs reachable by the LLM. Let the annealing schedule $\{\beta_t\}_{t=0}^T$ be given, and let each one-step mutation kernel M_t be an LLM proposal followed by an accept/reject step, such as the reward-aware rule* (10)*; no invariance or ergodicity is assumed. Then there is a constant $C_T < \infty$,*

Table 1: Math domain (Novikov et al., 2025).

Task	REEVO	OPENEVOLVE	SHINKAEVOLVE	SMCEVOLVE	
				Best	Calls (↓)
Circle Packing in Rect. N=21 (↑)	0.9085	0.9514	0.9514	**0.9993**	192
Hexagon Packing N=11 (↑)	0.7790	0.9398	0.9069	**0.9821**	176
Heilbronn Convex N=13 (↑)	0.6933	0.7178	0.7098	**0.9163**	176
Heilbronn Triangle N=11 (↑)	0.6862	0.8736	**0.9046**	0.8966	192
Min-Max-Min Dist. (n=16, d=2) (↑)	0.8634	0.9915	0.9842	**1.0000**	160
Kissing Number (d=11) (↑)	0.1754	0.2530	0.7386	**0.9005**	144
1st Autocorrelation Inequality (↑)	0.9912	0.9893	0.9971	**0.9974**	176
2nd Autocorrelation Inequality (↑)	0.9990	1.0079	1.0130	**1.0220**	128
3rd Autocorrelation Inequality (↑)	0.9636	0.0028	0.9942	**0.9981**	128
Erdős Minimum Overlap (↑)	0.9901	0.9973	0.9990	**0.9998**	192

Table 2: Algorithm efficiency domain (Press et al., 2025).

Task	REEVO	OPENEVOLVE	SHINKAEVOLVE	SMCEVOLVE	
				Best	Calls (↓)
`affine_transform_2d` (↑)	1.0572	0.9957	1.1032	**4.7160**	400
`convolve2d_full_fill` (↑)	0.9963	1.0886	1.2682	**11.6000**	480
`eigenvectors_complex` (↑)	1.0688	1.0767	1.26	**1.8350**	288
`fft_cmplx_scipy_fftpack` (↑)	0.9785	1.3590	1.1001	**9.2280**	464
`fft_convolution` (↑)	1.0756	1.6226	1.9817	**19.9000**	320
`lu_factorization` (↑)	1.2854	1.8120	1.4119	**6.1600**	368
`polynomial_real` (↑)	1.4333	1.6908	**33.8776**	2.4000	336
`psd_cone_projection` (↑)	1.0501	0.9753	34.5129	**45.8200**	416

depending only on T and $\{\Delta\beta_t\Delta_R\}_{t\le T}$, such that for any $\epsilon \in (0,1]$, $N = \lceil 16C_T^2\epsilon^{-2}\rceil$ particles, i.e., an LLM-call budget $\mathcal{B} = NTK = \mathcal{O}(C_T^2\,TK\,\epsilon^{-2})$, guarantee the following.

1. Distributional approximation. *For any fixed bounded test function $f : \mathcal{X} \to \mathbb{R}$ with $\|f\|_\infty \le 1$,*

$$\mathbb{P}\big(\big|\eta_T^N(f) - \eta_T(f)\big| \le \epsilon\big) \;\ge\; \tfrac{3}{4}. \tag{13}$$

2. Best-so-far reward. *Let R_T^{best} be the highest reward among all programs evaluated by Algorithm 1. For a reward level r, let P_r be the probability that a single-particle run of Algorithm 1, in which the LLM mutates one program without any resampling, produces programs $x_0, \dots, x_{T-1}$ whose weighted average reward $\frac{1}{\beta_T}\sum_{t=1}^{T}\Delta\beta_t R(x_{t-1})$ is at least r. Then, with probability at least $3/4$,*

$$R_T^{\mathrm{best}} \;\ge\; \max_r \Big\{ r - \tfrac{1}{\beta_T}\big[\log(1/P_r) + \log 2\big]\Big\}. \tag{14}$$

Part 1 provides the standard SMC guarantee for the terminal empirical measure: for any fixed bounded statistic f, the final particle population yields a high-accuracy estimate of its expectation under η_T; e.g., for the normalized reward $f = (R - R_-)/\Delta_R$, the mean reward of the final population is close to the mean reward under η_T. Intuitively, η_T favors programs that the LLM reaches through high-reward intermediate programs, trading off reward against staying close to the LLM's own behavior, as in (1); see Section G. Part 2 concerns the objective reported in our experiments, the best program found. A small P_r means that the LLM, improving a single program on its own, rarely sustains reward level r. Even so, by (14), SMCEVOLVE reaches level r up to a gap of $[\log(1/P_r) + \log 2]/\beta_T$: the rarity of good programs enters only through its logarithm, whereas repeating single-particle runs would need about $1/P_r$ attempts.

Relation to p^*. In general, $\|\eta_T - p^*\|_{\mathrm{TV}} \le e^{\beta\Delta_R}\sum_t \delta_t$, where δ_t measures how far one mutation block moves the bridge p_t; for the reward-aware rule, $\delta_t = 0$ when the LLM proposal is in detailed balance with the prior (Proposition G.1). When LLM likelihoods are available, the full MH ratio (24) gives $\delta_t = 0$ for every t. Then $\eta_T = p^*$, so Part 1 approximates the reward-tilted target itself, and (14) holds with P_r replaced by $p_0(R \ge r \mid q)$, the probability that the LLM prior directly produces a program with reward at least r. Under uniform ergodicity and the ESS bridge condition, the budget also sharpens to $\widetilde{\mathcal{O}}\big((\epsilon^{-2} \vee \kappa^{-1})(1-\rho)^{-1}\beta\Delta_R\big)$ (Corollary H.1). All proofs are in Section G.

5 EXPERIMENTS

5.1 MAIN RESULTS

We evaluate SMCEVOLVE on four domains: *Math*—combinatorial geometry problems from the AlphaEvolve benchmark (Novikov et al., 2025; Table 1); *AlgoTune*—runtime-optimization tasks (Press et al., 2025) spanning numerical computation, signal processing, and linear algebra (Table 2); *Symbolic Regression*—problems from LLM-SRBench (Shojaee et al., 2025) across physical oscillators, bacterial population growth, chemical reactions, and materials science (Table 3); and *AutoResearch* (Karpathy, 2026)—end-to-end GPT pretraining on TINYSTORIES (Figure 4), where reward is $\max(0,\ 2.0 - \text{val_bpb})$ on a held-out set; each candidate trains for 60 s on a single A5000. We compare against REEVO (Ye et al., 2024), OPENEVOLVE, and SHINKAEVOLVE (Lange et al., 2025). All methods share an identical `gpt-5-mini+gemini-3-flash` ensemble, and each cell

Table 3: Symbolic regression domain (Shojaee et al., 2025).

Split	Task	ReEvo	OpenEvolve	ShinkaEvolve	SMCEvolve	
					Best	Calls (↓)
bio_pop_growth	BPG0	4.9052	5.0918	6.4700	**7.1021**	384
	BPG1	3.4338	2.8472	5.8589	**7.1159**	336
	BPG2	−0.0962	5.9800	4.2176	**6.4054**	128
	BPG3	8.9428	8.9602	8.8261	**8.9725**	304
chem_react	CRK0	8.9952	8.9789	8.9846	**8.9985**	256
	CRK1	8.9537	8.6241	8.9830	**8.9972**	288
	CRK2	8.9990	8.5082	8.9906	**9.0000**	304
	CRK3	6.8599	**8.1226**	7.5276	7.8772	288
matsci	MatSci0	3.8771	4.5670	4.2790	**5.5511**	304
	MatSci1	0.1518	0.1520	0.1521	**0.1522**	144
	MatSci2	4.4763	6.9586	6.0224	**8.2500**	208
	MatSci3	1.4764	3.0537	3.1858	**4.4687**	304
phys_osc	PO0	4.1917	3.2642	**5.6612**	4.9567	256
	PO1	3.5738	6.5793	6.6001	**6.6998**	336
	PO2	4.9201	7.4520	**7.4928**	7.4296	368
	PO3	8.9929	8.9278	8.9486	**8.9985**	288

Figure 4: AutoResearch domain (Karpathy, 2026). SMCEvolve terminated automatically under ESS control.

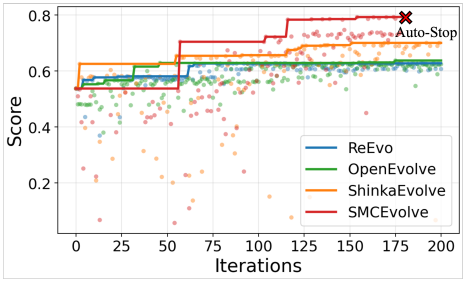


Table 4: Ablations on Circle Packing N=21. Best reward over 3 seeds; baseline in **bold**.

	Default	**Parent resampling**		**Mutation kernel choice**					**Breadth/depth**	
	N=8, K=2 mixture	uniform	greedy (argmax)	diff no insp.	diff w/ insp.	rewrite no insp.	rewrite w/ insp.	uniform 4-mix	N=4 K=4	N=16 K=1
Best	**0.9993**	0.9514	0.9760	0.9379	0.9868	0.9379	0.9514	0.9929	0.9379	0.9379

reports the best score over 3 seeds. Baselines run to a fixed LLM-call budget (200 for Math and AutoResearch, 400 for Symbolic Regression, 500 for AlgoTune), while SMCEvolve terminates automatically via ESS-driven stopping. SMCEvolve attains the best score on the majority of tasks across all four domains and the highest final reward on AutoResearch. Notably, these gains come with strictly fewer LLM calls—the mean call count under ESS-driven termination is below the fixed baseline budget on every domain (reported in the **Calls** column).

5.2 Ablation Study

We ablate the key designs—adaptive parent resampling and the mixture of mutation kernels in Section 3—as well as the hyperparameters N (particle count) and K (MH chain length). Each variant changes a single choice while holding the LLM-call budget fixed (Table 4).

Parent resampling. The adaptive resampling interpolates between uniform and greedy selection, transitioning from exploration to exploitation. To verify that this adaptivity is necessary, we replace it with two fixed alternatives at the extremes: `uniform` ignores the reward signal and `greedy` (argmax) always selects the best. Both degrade, confirming that our adaptive choice is essential.

Mutation kernel choice. The mixture targets ergodicity, and Thompson sampling adaptively selects among kernels. We ablate each: replacing the mixture with single kernels, and Thompson with a uniform mixture. Every single kernel or uniform mixture underperforms the baseline (≤ 0.9929), showing that both kernel diversity and adaptive selection are necessary.

Breadth vs. depth. Under a fixed budget of $N \times K = 16$ LLM calls per iteration, the budget can favor more particles or longer MH chains. Both extremes collapse to 0.9379: N=4, K=4 starves resampling of diversity; N=16, K=1 leaves chains too short to mix from the parent. The balanced N=8, K=2 split lets resampling explore broadly while each particle refines locally.

6 Conclusion

In this paper, we propose SMCEvolve, recasting LLM-driven program evolution as Sequential Monte Carlo sampling from a reward-tilted target distribution. Three coupled mechanisms emerge from this single principle, yielding a finite-sample bound for evolutionary program search and state-of-the-art performance across domains. Future directions include tightening the bound and extending to multi-objective or reward-free settings.

AI USE STATEMENT

In this paper, the mutation kernel used in our experiments calls an LLM API. We also used LLMs to polish the manuscript and check grammar. The research problem, methodological design, experimental design, interpretation of results, and scientific conclusions were determined by the authors.

ETHICS STATEMENT

This work does not involve human subjects, personally identifiable information, or sensitive data. All experiments use publicly available benchmarks (Section J), and potential societal impacts are discussed in Section M.

REPRODUCIBILITY STATEMENT

Algorithm 1 states the complete algorithm; Sections I and L give implementation details and all hyperparameters; Section N lists the prompts of the four mutation kernels; and Section J specifies every benchmark task and its reward. All assumptions of our theoretical results are stated in Section 4, and complete proofs are given in Sections C to H. We will release the code, configurations, and reproduction instructions; reproducing the experiments requires access to the LLM APIs listed in Section L.

REFERENCES

Nikhil Abhyankar, Sanchit Kabra, Saaketh Desai, and Chandan K Reddy. LLEMA: Evolutionary search with LLMs for multi-objective materials discovery. In *International Conference on Learning Representations*, 2026.

Alexandros Beskos, Ajay Jasra, Nikolas Kantas, and Alexandre Thiery. On the convergence of adaptive sequential Monte Carlo methods. *The Annals of Applied Probability*, 26(2):1111–1146, 2016.

James Carpenter, Peter Clifford, and Paul Fearnhead. Improved particle filter for nonlinear problems. *IEE Proceedings – Radar, Sonar and Navigation*, 146(1):2–7, 1999. doi: 10.1049/ip-rsn:19990255.

Frédéric Cérou, Pierre Del Moral, and Arnaud Guyader. A nonasymptotic theorem for unnormalized Feynman–Kac particle models. *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, 47(3):629–649, 2011. doi: 10.1214/10-AIHP358.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. MLE-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

Nicolas Chopin and Omiros Papaspiliopoulos. *An Introduction to Sequential Monte Carlo*. Springer Series in Statistics. Springer, Cham, 2020. ISBN 978-3-030-47844-5. doi: 10.1007/978-3-030-47845-2.

Chenguang Dai, Jeremy Heng, Pierre E Jacob, and Nick Whiteley. An invitation to sequential Monte Carlo samplers. *Journal of the American Statistical Association*, 117(539):1587–1600, 2022.

Pierre Del Moral. *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*. Probability and Its Applications. Springer, New York, 2004. ISBN 978-0-387-20268-6. doi: 10.1007/978-1-4684-9393-1.

Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential Monte Carlo samplers. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 68(3):411–436, 2006.

Randal Douc, Olivier Cappé, and Eric Moulines. Comparison of resampling schemes for particle filtering. In *Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis (ISPA 2005)*, pp. 64–69. IEEE, 2005. doi: 10.1109/ISPA.2005.195385.

Shengyu Feng, Xiang Kong, Shuang Ma, Aonan Zhang, Dong Yin, Chong Wang, Ruoming Pang, and Yiming Yang. Step-by-step reasoning for math problems via twisted sequential Monte Carlo. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Ze4aPP0tIn.

Bogdan Georgiev, Javier Gómez-Serrano, Terence Tao, and Adam Zsolt Wagner. Mathematical exploration and discovery at scale. *arXiv preprint arXiv:2511.02864*, 2025.

Yuki Imajuku, Kohki Horie, Yoichi Iwata, Kensho Aoki, Naohiro Takahashi, and Takuya Akiba. ALE-Bench: A benchmark for long-horizon objective-driven algorithm engineering. *arXiv preprint arXiv:2506.09050*, 2025.

Jiachen Jiang, Tianyu Ding, and Zhihui Zhu. DeltaEvolve: Accelerating scientific discovery through momentum-driven evolution. *arXiv preprint arXiv:2602.02919*, 2026.

Andrej Karpathy. autoresearch. https://github.com/karpathy/autoresearch, 2026. GitHub repository.

Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi: 10.1126/science.220.4598.671.

Genshiro Kitagawa. Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 5(1):1–25, 1996.

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

Alexander K. Lew, Zhi-Xuan Tan, Gabriel Grand, and Vikash K. Mansinghka. Sequential Monte Carlo steering of large language models using probabilistic programs. *arXiv preprint arXiv:2306.03081*, 2023. doi: 10.48550/arXiv.2306.03081. URL https://arxiv.org/abs/2306.03081.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *arXiv preprint arXiv:2401.02051*, 2024.

Joe Marion, Joseph Mathews, and Scott C Schmidler. Finite-sample complexity of sequential Monte Carlo estimators. *The Annals of Statistics*, 51(3):1357–1375, 2023.

Sean P Meyn and Richard L Tweedie. *Markov chains and stochastic stability*. Springer Science & Business Media, 2012.

Radford M. Neal. Annealed importance sampling. *Statistics and Computing*, 11(2):125–139, 2001. doi: 10.1023/A:1008923215028.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Ori Press, Brandon Amos, Haoyu Zhao, Yikai Wu, Samuel K. Ainsworth, Dominik Krupke, Patrick Kidger, Touqir Sajed, Bartolomeo Stellato, Jisun Park, Nathanael Bosch, Eli Meril, Albert Steppi, Arman Zharmagambetov, Fangzhao Zhang, David Perez-Pineiro, Alberto Mercurio, Ni Zhan, Talor Abramovich, Kilian Lieret, Hanlin Zhang, Shirley Huang, Matthias Bethge, and Ofir Press. AlgoTune: Can language models speed up general-purpose numerical programs? *arXiv preprint arXiv:2507.15887*, 2025. doi: 10.48550/arXiv.2507.15887. URL https://arxiv.org/abs/2507.15887.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Reuven Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1(2):127–190, 1999. doi: 10.1023/A:1010091220143.

Daniel J. Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on Thompson sampling. *Foundations and Trends in Machine Learning*, 11(1):1–96, 2018. doi: 10.1561/2200000070.

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K Reddy. LLM-SR: Scientific equation discovery via programming with large language models. *arXiv preprint arXiv:2404.18400*, 2024.

Parshin Shojaee, Ngoc-Hieu Nguyen, Kazem Meidani, Amir Barati Farimani, Khoa D Doan, and Chandan K Reddy. LLM-SRBench: A new benchmark for scientific equation discovery with large language models. *arXiv preprint arXiv:2504.10415*, 2025.

Stefan Wahl, Raphaela Schenk, Ali Farnoud, Jakob H. Macke, and Daniel Gedon. A probabilistic framework for LLM-based model discovery. *arXiv preprint arXiv:2602.18266*, 2026.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large language models as hyper-heuristics with reflective evolution. *Advances in neural information processing systems*, 37:43571–43608, 2024.

Stephen Zhao, Rob Brekelmans, Alireza Makhzani, and Roger Baker Grosse. Probabilistic inference in language models via twisted sequential Monte Carlo. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 60704–60748. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhao24c.html.

This appendix provides supplementary material for SMCEVOLVE. We first position the method relative to prior work in Appendix A, then state the complete algorithm in Appendix B. Next, we give the derivations and proofs supporting the SMC formulation (Appendices C, D, E, F, G, and H), followed by implementation details (Appendix I), benchmark/task specifications (Appendix J), hyperparameters (Appendix L), prompts (Appendix N), and a visualization of a representative run (Appendix O).

A EXTENDED RELATED WORK

LLM-driven program evolution and scientific discovery. LLMs have become effective generators inside evaluator-guided discovery loops. FunSearch repeatedly proposes and evaluates programs to obtain mathematical constructions (Romera-Paredes et al., 2024); AlphaEvolve scales this idea to a general coding agent for scientific and algorithmic discovery (Novikov et al., 2025); and ShinkaEvolve, ReEvo, and Evolution of Heuristics improve sample efficiency through open-ended evolution, reflection, and heuristic design (Lange et al., 2025; Ye et al., 2024; Liu et al., 2024). Related agent benchmarks and systems study automated scientific workflows, machine-learning engineering, symbolic equation discovery, materials discovery, and numerical-program optimization (Yamada et al., 2025; Chan et al., 2024; Imajuku et al., 2025; Shojaee et al., 2024; 2025; Abhyankar et al., 2026; Press et al., 2025). These works motivate evaluator-guided program search, but their parent selection, population management, mutation schedules, and stopping rules are typically engineered as separate heuristics. SMCEVOLVE focuses on this population-search core and interprets it as SMC sampling from a reward-tilted program distribution.

SMC for language-model inference and reasoning. Recent work also applies SMC directly to language-model generation. Lew et al. (2023) use SMC steering for constrained generation specified as posterior inference in probabilistic sequence models. Zhao et al. (2024) develop twisted SMC for sampling from sequence-level unnormalized targets in LLM inference and safety tasks, and Feng et al. (2025) use twisted SMC to guide multi-step mathematical reasoning. These methods treat prefixes, complete sequences, or reasoning traces as particles within a single generation episode. In contrast, SMCEVOLVE treats executable programs as particles, obtains rewards from an external evaluator, and uses SMC to structure population-based scientific program evolution rather than controlled decoding. Closest to our setting, Wahl et al. (2026) also treat candidate models as particles: their ModelSMC casts LLM-based model discovery as probabilistic inference over mechanistic models that explain observational data, refining particles with LLM proposals and weighting them by likelihood-based criteria. SMCEVOLVE instead targets a reward-tilted distribution defined by an arbitrary external evaluator, which requires no likelihood and covers optimization tasks such as mathematical constructions and runtime optimization.

SMC, annealing, and adaptive sampling foundations. Our method builds on SMC samplers, which move particles through intermediate distributions via weighting, resampling, and mutation (Del Moral et al., 2006; Dai et al., 2022; Del Moral, 2004; Chopin & Papaspiliopoulos, 2020). The reward-tempered path is related to annealed importance sampling, simulated annealing, and cross-entropy methods for stochastic optimization (Neal, 2001; Kirkpatrick et al., 1983; Rubinstein, 1999). We import finite-sample SMC complexity theory (Marion et al., 2023) rather than proving a new general theorem. The contribution is the specialization to LLM-driven program evolution: the reward-tilted target, bridge path, resampling weights, and reference-kernel requirements jointly explain the design levers and limits of the evolutionary coding agent.

B SMCEVOLVE ALGORITHM

The detailed SMCEVOLVE algorithm is shown below in Algorithm 1.

C DERIVATION OF THE REWARD-TILTED TARGET DISTRIBUTION

We provide the full derivation of the reward-tilted target distribution in Lemma 3.1. Starting from the regularized variational objective in (1), the Lagrangian with the normalization constraint $\int p(x)\,dx =$

Algorithm 1 SMCEVOLVE

Require: N particles, K sequential LLM proposals per particle, ESS threshold $\kappa \in (0,1)$, prior $p_0(\cdot \mid q)$, LLM proposal $Q_t(\cdot \mid x, \mathcal{C}_t)$, reward $R(\cdot)$, target inverse temperature β
Ensure: Particles approximating η_T (Theorem 4.1), which equals $p^*(\cdot \mid q)$ for invariant kernels

for $n = 1, \ldots, N$ **do**
  $x_0^{(n)} \sim p_0(\cdot \mid q)$; $R^{(n)} \leftarrow R(x_0^{(n)})$ # Initialize from LLM prior
end for
$\lambda_0 \leftarrow 0, t \leftarrow 0$
repeat
  $t \leftarrow t + 1$
  Determine λ_t via ESS bisection (11); $\Delta\beta_t \leftarrow (\lambda_t - \lambda_{t-1})\beta$ # Adaptive temperature
  Compute weights $W_t^{(n)}$ via (9); resample ancestor indices $\{A_t^{(n)}\}_{n=1}^N$ # Parent resampling
  for $n = 1, \ldots, N$ **do**
    $z \leftarrow x_{t-1}^{(A_t^{(n)})}$ # Resampled parent (current state of MH chain)
    for $k = 1, \ldots, K$ **do**
      $x' \sim Q_t(\cdot \mid z, \mathcal{C}_t)$ # LLM proposes from current state
      With probability $\alpha_t(z, x')$ via (10): $z \leftarrow x'$ # MH accept/reject
    end for
    $x_t^{(n)} \leftarrow z$; $R^{(n)} \leftarrow R(x_t^{(n)})$
  end for
until $\lambda_t = 1$ # Automatic termination

1 is

$$\mathcal{L}[p] = \int p(x)\, R(x)\, dx - \frac{1}{\beta} \int p(x) \log \frac{p(x)}{p_0(x \mid q)}\, dx - \mu \left(\int p(x)\, dx - 1 \right). \tag{15}$$

Setting the functional derivative $\delta\mathcal{L}/\delta p(x) = 0$ gives

$$R(x) - \frac{1}{\beta} \left(\log \frac{p(x)}{p_0(x \mid q)} + 1 \right) - \mu = 0. \tag{16}$$

Solving for $\log p(x)$ yields

$$\log p(x) = \beta R(x) + \log p_0(x \mid q) - (1 + \mu\beta). \tag{17}$$

Exponentiating and absorbing the x-independent term $e^{-(1+\mu\beta)}$ into the partition function $Z(q)$ gives the target distribution $p^*(x \mid q)$ in (2).

D MH CONSTRUCTION YIELDS A p_t-INVARIANT KERNEL

Condition on the stage t and context $\mathcal{C}_t$, and write $Q_t(\cdot \mid x, \mathcal{C}_t)$ for the proposal kernel.

Theorem D.1 (MH construction yields a p_t-invariant kernel)**.** *The MH kernel M_t defined by* (24) *is p_t-invariant, i.e., satisfies* (4)*.*

Proof. We show that M_t satisfies the stronger property of *detailed balance*,

$$p_t(x)\, M_t(x, y) \;=\; p_t(y)\, M_t(y, x), \qquad \forall\, x, y \in \mathcal{X}, \tag{18}$$

which immediately implies p_t-invariance: integrating both sides of (18) over x gives

$$\int p_t(x)\, M_t(x, y)\, dx \;=\; p_t(y) \int M_t(y, x)\, dx \;=\; p_t(y),$$

which is (4). Intuitively, (18) says that at equilibrium p_t, the probability flux from x to y equals the flux from y to x.

It remains to verify (18) for the MH kernel. For $x = y$, both sides are equal, so the identity holds trivially. For $x \neq y$, we need to show

$$p_t(x)\, Q_t(y \mid x, \mathcal{C}_t)\, \alpha_t(x, y) \;=\; p_t(y)\, Q_t(x \mid y, \mathcal{C}_t)\, \alpha_t(y, x).$$

Without loss of generality, assume $p_t(x)\, Q_t(y \mid x, \mathcal{C}_t) \leq p_t(y)\, Q_t(x \mid y, \mathcal{C}_t)$ (if the opposite inequality holds, swap the roles of x and y in the argument below). Under this assumption:

- The ratio in (24) satisfies $\dfrac{p_t(y)\,Q_t(x \mid y, \mathcal{C}_t)}{p_t(x)\,Q_t(y \mid x, \mathcal{C}_t)} \geq 1$, so $\alpha_t(x, y) = 1$.
- The reverse ratio is $\dfrac{p_t(x)\,Q_t(y \mid x, \mathcal{C}_t)}{p_t(y)\,Q_t(x \mid y, \mathcal{C}_t)} \leq 1$, so $\alpha_t(y, x) = \dfrac{p_t(x)\,Q_t(y \mid x, \mathcal{C}_t)}{p_t(y)\,Q_t(x \mid y, \mathcal{C}_t)}$.

Now compute both sides:

$$\begin{aligned}
\text{Left side:} \quad & p_t(x)\,Q_t(y \mid x, \mathcal{C}_t)\,\alpha_t(x, y) \;=\; p_t(x)\,Q_t(y \mid x, \mathcal{C}_t) \cdot 1 \;=\; p_t(x)\,Q_t(y \mid x, \mathcal{C}_t). \\
\text{Right side:} \quad & p_t(y)\,Q_t(x \mid y, \mathcal{C}_t)\,\alpha_t(y, x) \;=\; p_t(y)\,Q_t(x \mid y, \mathcal{C}_t) \cdot \frac{p_t(x)\,Q_t(y \mid x, \mathcal{C}_t)}{p_t(y)\,Q_t(x \mid y, \mathcal{C}_t)} \\
& \qquad = \; p_t(x)\,Q_t(y \mid x, \mathcal{C}_t).
\end{aligned}$$

Both sides agree, so detailed balance holds, and hence p_t-invariance. □

E Weight Simplification under Time-Reversal Backward Kernel

Starting from the general SMC weight:

$$w_t(x_{t-1}, x_t) = \frac{\gamma_t(x_t)\,L_{t-1}(x_t, x_{t-1})}{\gamma_{t-1}(x_{t-1})\,M_t^K(x_{t-1}, x_t)}. \tag{19}$$

Substituting the time-reversal backward kernel $L_{t-1}(x_t, x_{t-1}) = p_t(x_{t-1})\,M_t^K(x_{t-1}, x_t)/p_t(x_t)$:

$$w_t(x_{t-1}, x_t) = \frac{\gamma_t(x_t)}{\gamma_{t-1}(x_{t-1})} \cdot \frac{p_t(x_{t-1})\,M_t^K(x_{t-1}, x_t)}{p_t(x_t)\,M_t^K(x_{t-1}, x_t)} \tag{20}$$

$$= \frac{\gamma_t(x_t)}{\gamma_{t-1}(x_{t-1})} \cdot \frac{\gamma_t(x_{t-1})/Z_t}{\gamma_t(x_t)/Z_t} \tag{21}$$

$$= \frac{\gamma_t(x_{t-1})}{\gamma_{t-1}(x_{t-1})} = e^{\Delta\beta_t \cdot R(x_{t-1})}. \tag{22}$$

Both M_t^K and L_{t-1} cancel completely, confirming (8). This derivation requires (7) to be a Markov kernel, i.e., $p_t M_t^K = p_t$. Without invariance, (8) is no longer the SMC-sampler weight for p_t, but it is still the potential G_t of the Feynman–Kac flow (12), whose particle approximation Theorem 4.1 analyzes; Proposition G.1 bounds the resulting gap between η_T and p^*.

F Realizing the Forward Kernel with LLM Proposals

Identifying the limit η_T of Algorithm 1 with p^* requires a p_t-invariant kernel M_t at each step; Theorem 4.1 itself does not. Here Q_t is the LLM proposal distribution, while one application of M_t is the proposal plus MH accept/reject transition; Algorithm 1 applies this transition K times per particle. We now detail how this kernel is realized when the only available sampler is a black-box LLM.

LLM as proposal distribution. At step t, given a parent program x selected by resampling, the LLM proposal kernel generates a candidate program x' by sampling

$$x' \sim Q_t(\cdot \mid x, \mathcal{C}_t), \tag{23}$$

where $\mathcal{C}_t$ denotes the *context* at step t: a structured collection of information provided to the LLM alongside the parent program x and task q. This context may include the history of previously evaluated programs with their rewards, high-performing programs from other particles (inspirations), and instructions specifying the edit granularity. The notation $Q_t(\cdot \mid x, \mathcal{C}_t)$ emphasizes that the proposal is implemented by the *same LLM* that defines the prior $p_0(\cdot \mid q)$, but with additional conditioning on the parent and context. When $\mathcal{C}_t = \varnothing$ and no parent is provided, the proposal reduces to the prior $p_0(\cdot \mid q)$.

The invariance requirement. To ensure that M_t leaves p_t invariant, the standard approach is Metropolis–Hastings (MH): accept x' with probability

$$\alpha_t(x, x') = \min\left\{1,\ \frac{p_t(x')\, Q_t(x \mid x', \mathcal{C}_t)}{p_t(x)\, Q_t(x' \mid x, \mathcal{C}_t)}\right\}. \tag{24}$$

Expanding $p_t(x) \propto p_0(x \mid q)\, e^{\beta_t R(x)}$, the acceptance ratio becomes

$$\alpha_t(x, x') = \min\Bigg\{1,\ e^{\beta_t(R(x')-R(x))} \cdot \underbrace{\frac{p_0(x' \mid q)}{p_0(x \mid q)} \cdot \frac{Q_t(x \mid x', \mathcal{C}_t)}{Q_t(x' \mid x, \mathcal{C}_t)}}_{\sigma_t(x, x')}\Bigg\}. \tag{25}$$

The composite ratio $\sigma_t(x, x')$ captures the asymmetry between forward and reverse transitions.

From the full MH ratio to a tractable approximation. Computing $\sigma_t(x, x')$ requires evaluating the LLM prior ratio and the proposal density ratio, both of which are unavailable through black-box LLM APIs. To obtain a tractable acceptance rule, we adopt the reward-aware criterion in (10) of the main text, which corresponds to substituting $\sigma_t(x, x') = 1$ in (25). This substitution is exact, i.e., M_t is p_t-invariant, whenever the LLM proposal is in detailed balance with the prior, $p_0(x \mid q)\, Q_t(x' \mid x, \mathcal{C}_t) = p_0(x' \mid q)\, Q_t(x \mid x', \mathcal{C}_t)$. In general it is not, and neither small edits nor high acceptance rates imply otherwise: a proposal that always favors one of two equally rewarded programs is always accepted, yet the resulting population does not follow p_t. We therefore do not rely on $\sigma_t \approx 1$. Theorem 4.1 holds for the kernel as deployed, and Proposition G.1 bounds the distance between its limit η_T and p^* by how far the proposal is from detailed balance with the prior.

G Proof of Theorem 4.1

Notation and Feynman–Kac representation. Write $G_t(x) \triangleq e^{\Delta\beta_t R(x)}$ for the incremental weight (8) and, for a distribution μ on $\mathcal{X}$, let

$$\Psi_G(\mu)(x) \triangleq \frac{G(x)\,\mu(x)}{\mu(G)}, \qquad \mu(G) = \sum_{x \in \mathcal{X}} G(x)\,\mu(x),$$

denote the reweighting (Boltzmann–Gibbs) map. The recursion (12) then reads

$$\eta_0 = p_0(\cdot \mid q), \qquad \eta_t = \Psi_{G_t}(\eta_{t-1})\, M_t^K, \qquad t = 1, \ldots, T. \tag{26}$$

With multinomial resampling, conditionally on the particles at stage $t-1$ with empirical measure η_{t-1}^N, the N particles at stage t are iid draws from

$$\Psi_{G_t}\big(\eta_{t-1}^N\big)\, M_t^K, \tag{27}$$

because each particle picks an ancestor with probability (9) and is then mutated by M_t^K. Hence Algorithm 1 is the standard mean-field particle approximation of the Feynman–Kac flow (26) (Del Moral, 2004). Let P be the distribution of the programs $X_0, \ldots, X_T$ produced by a single-particle run of Algorithm 1, which involves no resampling: $X_0 \sim p_0(\cdot \mid q)$ and $X_t \mid X_{t-1} \sim M_t^K(X_{t-1}, \cdot)$. Let

$$S \triangleq \sum_{t=1}^{T} \Delta\beta_t\, R(X_{t-1}).$$

The Feynman–Kac formula expresses the flow and its normalizing constant through P:

$$\eta_T(f) = \frac{\mathbb{E}_P\left[f(X_T)\, e^S\right]}{\mathbb{E}_P\left[e^S\right]}, \qquad Z_T \triangleq \mathbb{E}_P\left[e^S\right] = \prod_{t=1}^{T} \eta_{t-1}(G_t). \tag{28}$$

Step 1: distributional approximation (Part 1). Since R has oscillation at most Δ_R on the programs reachable by the LLM, every potential is positive and satisfies, uniformly in x and independently of the kernels,

$$\frac{\sup_x G_t(x)}{\inf_x G_t(x)} \le e^{\Delta\beta_t \Delta_R}, \qquad t = 1, \ldots, T. \tag{29}$$

For Feynman–Kac models with such potentials, the non-asymptotic L^1 bound for the mean-field particle approximation holds for arbitrary Markov kernels M_t (Del Moral, 2004):

$$\mathbb{E}\,\big|\eta_T^N(f) - \eta_T(f)\big| \;\le\; \frac{C_T\,\|f\|_\infty}{\sqrt{N}}, \tag{30}$$

where C_T depends only on T and the ratios in (29). By Markov's inequality, for $\|f\|_\infty \le 1$,

$$\mathbb{P}\big(|\eta_T^N(f) - \eta_T(f)| > \epsilon\big) \;\le\; \frac{C_T}{\epsilon\sqrt{N}} \;\le\; \frac{1}{4} \qquad \text{whenever } N \ge 16\,C_T^2\,\epsilon^{-2},$$

which is (13). Each of the T stages uses NK LLM calls, so $\mathcal{B} = NTK = \mathcal{O}(C_T^2\,TK\,\epsilon^{-2})$.

Step 2: best-so-far reward (Part 2). Consider the normalizing-constant estimate built from the resampling weights,

$$\hat{Z}_T \;\triangleq\; \prod_{t=1}^{T} \eta_{t-1}^N(G_t) \;=\; \prod_{t=1}^{T} \frac{1}{N} \sum_{n=1}^{N} e^{\Delta\beta_t R(x_{t-1}^{(n)})}. \tag{31}$$

A pathwise bound. Every $x_{t-1}^{(n)}$ is an evaluated program, so each factor in (31) is at most $e^{\Delta\beta_t R_T^{\text{best}}}$. Since $\sum_t \Delta\beta_t = \beta_T$, on every run

$$\hat{Z}_T \;\le\; \exp\Big(\sum_{t=1}^{T} \Delta\beta_t\, R_T^{\text{best}}\Big) = e^{\beta_T R_T^{\text{best}}}, \qquad \text{i.e.,} \qquad R_T^{\text{best}} \;\ge\; \frac{1}{\beta_T} \log \hat{Z}_T. \tag{32}$$

Concentration of $\hat{Z}_T$. With multinomial resampling, $\hat{Z}_T$ is an unbiased estimator of Z_T (Del Moral, 2004), and its relative variance is of order $1/N$ (Cérou et al., 2011):

$$\mathbb{E}\big[\hat{Z}_T\big] = Z_T, \qquad \frac{\mathrm{Var}(\hat{Z}_T)}{Z_T^2} \;\le\; \frac{c_T}{N}, \tag{33}$$

where c_T depends only on T and the ratios in (29). Choosing $C_T \ge \max(1, c_T)$ ensures $N \ge 16C_T^2\epsilon^{-2} \ge 16c_T$, and Chebyshev's inequality gives

$$\mathbb{P}\big(\hat{Z}_T < Z_T/2\big) \;\le\; \frac{4\,\mathrm{Var}(\hat{Z}_T)}{Z_T^2} \;\le\; \frac{4c_T}{N} \;\le\; \frac{1}{4}.$$

A lower bound on Z_T. For every level r, since $P_r = P(S \ge \beta_T r)$,

$$Z_T = \mathbb{E}_P\left[e^S\right] \;\ge\; \mathbb{E}_P\left[e^S\,\mathbf{1}\{S \ge \beta_T r\}\right] \;\ge\; e^{\beta_T r}\,P_r. \tag{34}$$

Combining. On the event $\{\hat{Z}_T \ge Z_T/2\}$, which has probability at least $3/4$, (32) and (34) give, simultaneously for all r,

$$R_T^{\text{best}} \;\ge\; \frac{1}{\beta_T}\big(\log Z_T - \log 2\big) \;\ge\; r - \frac{1}{\beta_T}\Big[\log\frac{1}{P_r} + \log 2\Big],$$

which is (14).

Step 3: the invariant case. Suppose each M_t is p_t-invariant; then so is M_t^K. We show $\eta_t = p_t$ by induction. For $t = 0$, $\eta_0 = p_0$. If $\eta_{t-1} = p_{t-1}$, then

$$\Psi_{G_t}(p_{t-1})(x) \;\propto\; p_0(x \mid q)\,e^{\beta_{t-1}R(x)}\,e^{\Delta\beta_t R(x)} \;=\; \gamma_t(x),$$

so $\Psi_{G_t}(p_{t-1}) = p_t$ and $\eta_t = p_t M_t^K = p_t$; hence $\eta_T = p_T = p^*$. Writing $Z_b \triangleq \mathbb{E}_{p_0}[e^{bR}]$, the normalizing constant (28) then telescopes:

$$Z_T = \prod_{t=1}^{T} \mathbb{E}_{p_{t-1}}\left[e^{\Delta\beta_t R}\right] = \prod_{t=1}^{T} \frac{Z_{\beta_t}}{Z_{\beta_{t-1}}} = \mathbb{E}_{p_0}\left[e^{\beta_T R}\right] \;\ge\; e^{\beta_T r}\,p_0(R \ge r \mid q),$$

so (14) holds with P_r replaced by $p_0(R \ge r \mid q)$.

Step 4: variational characterization of η_T. By (28), η_T is the time-T marginal of the path measure

$$Q^\star(dx_{0:T}) \triangleq \frac{e^{S(x_{0:T})}}{Z_T} P(dx_{0:T}).$$

Since S is bounded, $Q^\star$ and P are mutually absolutely continuous, and every $Q \ll P$ satisfies $Q \ll Q^\star$ and

$$\mathbb{E}_Q[S] - \mathrm{KL}(Q \,\|\, P) = \mathbb{E}_Q\left[S - \log \frac{dQ}{dQ^\star} - \log \frac{dQ^\star}{dP}\right] = \log Z_T - \mathrm{KL}(Q \,\|\, Q^\star).$$

Hence $Q^\star$ is the unique maximizer of $\mathbb{E}_Q[S] - \mathrm{KL}(Q \,\|\, P)$, the analogue of (1) for whole sequences of programs. □

Distance between η_T and p^*. The next result quantifies how far the limit η_T of Theorem 4.1 can be from p^* when the kernels are not p_t-invariant.

Proposition G.1 (Distance to the reward-tilted target)**.** *Under the assumptions of Theorem 4.1, let $\delta_t \triangleq \|p_t M_t^K - p_t\|_{\mathrm{TV}}$ be the amount by which one mutation block moves the bridge p_t. Then*

$$\|\eta_T - p^*\|_{\mathrm{TV}} \le \sum_{t=1}^{T} e^{(\beta_T - \beta_t)\Delta_R}\, \delta_t. \tag{35}$$

If M_t is the reward-aware rule (10) *with proposal Q_t, then $\delta_t \le K e^{\beta_t \Delta_R} \varepsilon_t$, where*

$$\varepsilon_t \triangleq \tfrac{1}{2} \sum_{x \neq y} \left| p_0(x \mid q)\, Q_t(y \mid x) - p_0(y \mid q)\, Q_t(x \mid y) \right| \tag{36}$$

measures how far the proposal is from detailed balance with the prior. Hence $\|\eta_T - p^*\|_{\mathrm{TV}} \le K e^{\beta_T \Delta_R} \sum_{t=1}^{T} \varepsilon_t$, and $\eta_T = p^*$ whenever every $\varepsilon_t = 0$.

Proof. Stability of one update. Let μ, ν be distributions on $\mathcal{X}$ and let $0 < g_- \le G \le g_+$. For any $A \subseteq \mathcal{X}$,

$$\Psi_G(\mu)(A) - \Psi_G(\nu)(A) = \frac{(\mu - \nu)\big(G\,(\mathbf{1}_A - \Psi_G(\nu)(A))\big)}{\mu(G)},$$

and the function inside has oscillation at most g_+, so $\|\Psi_G(\mu) - \Psi_G(\nu)\|_{\mathrm{TV}} \le (g_+/g_-)\, \|\mu - \nu\|_{\mathrm{TV}}$. A Markov kernel does not increase the total variation distance, and $g_+/g_- \le e^{\Delta\beta_t \Delta_R}$ for $G = G_t$. Hence the update $\Phi_t(\mu) \triangleq \Psi_{G_t}(\mu) M_t^K$ of (26) is $e^{\Delta\beta_t \Delta_R}$-Lipschitz in total variation, and $\Phi_{t,T} \triangleq \Phi_T \circ \cdots \circ \Phi_{t+1}$ (the identity for $t = T$) is $e^{(\beta_T - \beta_t)\Delta_R}$-Lipschitz.

Telescoping. Since $\Psi_{G_t}(p_{t-1}) = p_t$ (Step 3), $\Phi_t(p_{t-1}) = p_t M_t^K$. As $\eta_T = \Phi_{0,T}(p_0)$ and $p_T = p^*$,

$$\eta_T - p^* = \sum_{t=1}^{T} \Big[\Phi_{t,T}\big(p_t M_t^K\big) - \Phi_{t,T}(p_t)\Big],$$

and (35) follows from the Lipschitz bound.

Reward-aware rule. Let M be one step of (10) with proposal Q, and write $m_t(x,y) \triangleq \min\{e^{\beta_t R(x)}, e^{\beta_t R(y)}\}$, which is symmetric. Then $p_t(x)\, \alpha_t(x,y) = p_0(x \mid q)\, m_t(x,y)/Z_{\beta_t}$, so for every y,

$$(p_t M)(y) - p_t(y) = \frac{1}{Z_{\beta_t}} \sum_{x \neq y} m_t(x,y) \big[p_0(x \mid q)\, Q(y \mid x) - p_0(y \mid q)\, Q(x \mid y)\big].$$

Since $m_t \le e^{\beta_t R_+}$ and $Z_{\beta_t} \ge e^{\beta_t R_-}$, this gives $\|p_t M - p_t\|_{\mathrm{TV}} \le e^{\beta_t \Delta_R}\, \varepsilon_t$. Finally, $\|p_t M^K - p_t\|_{\mathrm{TV}} \le \sum_{k=0}^{K-1} \|(p_t M - p_t) M^k\|_{\mathrm{TV}} \le K\, \|p_t M - p_t\|_{\mathrm{TV}}$. □

Remarks on the implementation. The argument uses multinomial resampling and a schedule and kernel mixture fixed in advance. Our implementation uses systematic resampling, a standard lower-variance variant, and the ESS-adaptive schedule, whose consistency is established by Beskos et al. (2016).

H SHARPER BUDGET FOR INVARIANT KERNELS

When the mutation kernels are p_t-invariant, Theorem 4.1 identifies the limit as $\eta_T = p^*$, and the finite-sample theory of Marion et al. (2023) gives an explicit budget.

Corollary H.1 (Sharper budget for invariant kernels)**.** *Suppose the reward has finite positive oscillation Δ_R on the support of $p_0(\cdot \mid q)$. Let the schedule $\{\lambda_t\}_{t=0}^T$ be produced by the ESS bisection rule* (11) *with threshold $\kappa \in (0,1)$, and assume the resulting adjacent bridges satisfy $\sup_{1\le t\le T} \|p_t/p_{t-1}\|^2_{L^2(p_{t-1})} \le \kappa^{-1}$. Assume also that for each t, the one-step mutation kernel M_t is p_t-invariant, e.g., the exact MH kernel* (24)*, and uniformly ergodic with rate ρ_t, and let $\rho = \max_{1\le t\le T} \rho_t \in (0,1)$. Then for any fixed bounded test function $f : \mathcal{X} \to \mathbb{R}$ with $\|f\|_\infty \le 1$ and any $\epsilon > 0$, there exists a choice of (N, T, K) such that the empirical measure η_T^N satisfies*[1]

$$\mathbb{P}\big(\big|\eta_T^N(f) - p^*(f)\big| \le \epsilon\big) \;\ge\; \tfrac{3}{4}. \tag{37}$$

The corresponding LLM-call budget $\mathcal{B} = NTK$ satisfies

$$\mathcal{B} \;=\; \widetilde{\mathcal{O}}\left(\frac{\epsilon^{-2} \vee \kappa^{-1}}{1-\rho} \cdot \beta\Delta_R\right), \tag{38}$$

where $\widetilde{\mathcal{O}}$ suppresses polylogarithmic factors.

Proof of Corollary H.1. We state the imported finite-sample SMC bound of Marion et al. (2023, Theorem 1 & Sec. 5.1) in its mixing-time form (which does not require kernel reversibility) and then derive the three substitutions (S1), (S2), (S3) that map its quantities to the SMCEVOLVE variables.

Warm-start mixing time. For a Markov kernel M with invariant distribution μ, an ω-warm starting set $P_\omega(\mu) \triangleq \{\nu : \sup_B \nu(B)/\mu(B) \le \omega\}$, and an accuracy $\eta \in (0,1)$, the warm-start mixing time of M is

$$\tau_M(\eta,\omega) \;\triangleq\; \min\left\{K \ge 1 : \sup_{\nu\in P_\omega(\mu)} \|\nu M^K - \mu\|_{\mathrm{TV}} \;\le\; \eta\right\}. \tag{39}$$

This is the definition in Sec. 2.1 of Marion et al. (2023).

Imported bound (Marion et al., 2023, Theorem 1 & Sec. 5.1). Consider an SMC sampler over a geometric-mixture path $\mu_0, \dots, \mu_T = \pi$ with $T = \lceil \log \Gamma \rceil$ and $\beta_t = t/T$, initialized with iid samples from μ_0 and applying K transitions of a μ_t-invariant Markov kernel M_t at each stage t. Suppose

1. the bridge weights satisfy $\sup_x w_t(x) \le W$ and $z_{t-1}/z_t \le Z$ for every t, with $\gamma \triangleq W \cdot Z$ (this is AS1 of Marion et al., 2023);
2. there exists $\tau < \infty$ such that $\tau_{M_t}(\eta_\star, \omega_\star) \le \tau$ for every t, at the internal precision/warmness level chosen by the analysis (Theorem 1 of Marion et al., 2023 uses $\omega_\star = 2$ and $\eta_\star = 1/(8NT)$);
3. $\Gamma \triangleq \sup_x d\pi/d\mu_0(x) < \infty$.

Then for every $\|f\|_\infty \le 1$ and $\epsilon > 0$, there exists a choice of (N, T, K) such that the SMC empirical measure η_T^N satisfies $\mathbb{P}(|\eta_T^N(f) - \pi(f)| \le \epsilon) \ge 3/4$, and the total kernel-evaluation budget is bounded by

$$\mathcal{B} \;=\; NTK \;=\; \mathcal{O}^*\big((\epsilon^{-2} \vee \gamma^2) \cdot \tau \cdot \log\Gamma \, \log^2\!\log\Gamma\big). \tag{40}$$

The reversible-kernel form (Marion et al., 2023, Cor. 6.1) is recovered by the spectral-gap-to-mixing-time bound $\tau_{M_t}(\eta,\omega) \le (1/\rho^{\mathrm{gap}})\big[\log(2/\eta) + \log(\omega - 1)\big]$, which gives $\tau = \mathcal{O}(1/\rho^{\mathrm{gap}})$ up to log factors absorbed into $\mathcal{O}^*$.

Setup for SMCEVOLVE. The bridges $p_t(x) \propto p_0(x \mid q) e^{\beta_t R(x)}$ with $0 = \beta_0 < \cdots < \beta_T = \beta$ form a geometric-mixture path from $p_0(\cdot \mid q)$ to $p^*(\cdot \mid q)$. The initialization condition holds because particles are iid from $p_0(\cdot \mid q)$ (Algorithm 1, line 2), and the kernel correctness/mixing condition holds under the kernel hypothesis of Corollary H.1. It remains to identify $(\gamma^2, \tau, \log\Gamma)$ in (40) with SMCEVOLVE quantities. We do so in three steps.

[1] The constant $3/4$ is the high-probability guarantee in Theorem 1 of Marion et al. (2023); it can be amplified to $1-\delta$ at a $\log(1/\delta)$ multiplicative cost by running the sampler $\mathcal{O}(\log(1/\delta))$ times and taking the median of the estimates.

(S1) BRIDGE REGULARITY $\gamma^2 \le \kappa^{-1}$

The imported bound (40) requires a deterministic upper bound on the adjacent L^2-bridge ratio, which we adopt as part of the hypothesis of Corollary H.1:

$$\left\| \frac{p_t}{p_{t-1}} \right\|^2_{L^2(p_{t-1})} \le \kappa^{-1}, \qquad t = 1, \dots, T. \tag{41}$$

The parameterization in terms of κ is justified by two considerations.

Worst-case schedule-dependent bound. The geometric reward-tilted bridge (6) satisfies

$$\left\| \frac{p_t}{p_{t-1}} \right\|_\infty \le \frac{e^{\Delta\beta_t R_+}}{p_{t-1}(e^{\Delta\beta_t R})} \le e^{\Delta\beta_t (R_+ - R_-)} = e^{\Delta\beta_t \Delta_R}, \tag{42}$$

where the second inequality uses $p_{t-1}(e^{\Delta\beta_t R}) \ge e^{\Delta\beta_t R_-}$. Since the L^2 norm is dominated by the L^∞ norm, this gives the deterministic certificate $\kappa^{-1} \le e^{2\Delta\beta_{\max}\Delta_R}$, with $\Delta\beta_{\max} \triangleq \max_t \Delta\beta_t$. Short bridges (small $\Delta\beta_t$) imply small bridge ratios.

ESS bisection as a finite-sample mechanism that targets (41)**.** The empirical ESS

$$\widehat{\mathrm{ESS}}_t = \frac{\left(\sum_{n=1}^N w_t^{(n)}\right)^2}{\sum_{n=1}^N (w_t^{(n)})^2}, \qquad w_t^{(n)} = \frac{p_t(x_{t-1}^{(n)})}{p_{t-1}(x_{t-1}^{(n)})} \tag{43}$$

is the finite-sample analogue of $\|p_t/p_{t-1}\|^{-2}_{L^2(p_{t-1})}$: when $\{x_{t-1}^{(n)}\}_{n=1}^N$ is an iid sample from p_{t-1}, a law of large numbers gives $\widehat{\mathrm{ESS}}_t/N \to \|p_t/p_{t-1}\|^{-2}_{L^2(p_{t-1})}$ as $N \to \infty$. The ESS bisection rule (11) is the algorithmic mechanism that targets (41) by enforcing its empirical analogue $\widehat{\mathrm{ESS}}_t/N \ge \kappa$. We do not turn this targeting into a formal derivation, since the SMC particles at stage $t-1$ are not iid from p_{t-1} and the schedule $\{\lambda_t\}_{t=0}^T$ is data-dependent; (41) is therefore treated as an explicit assumption (with worst-case certificate $e^{2\Delta\beta_{\max}\Delta_R}$) rather than a consequence of the ESS rule.

(S2) MIXING-TIME BOUND $\tau \le 1/(1-\rho)$ VIA UNIFORM ERGODICITY

The kernel hypothesis of Corollary H.1 provides uniform ergodicity $\sup_x \|M_t^K(x,\cdot) - p_t\|_{\mathrm{TV}} \le C\rho^K$ with rate $\rho \in (0,1)$. We show that this directly yields a warm-start mixing-time bound that feeds into (40) without any reversibility hypothesis. The argument is in four steps.

(i) TV convexity in the first argument. For any starting distribution ν and any measurable A, since ν is a probability measure,

$$\nu M_t^K(A) - p_t(A) = \int [M_t^K(x,A) - p_t(A)]\, \nu(dx).$$

Taking absolute values, then $\sup_A$, and exchanging $\sup_A \int \le \int \sup_A$ gives

$$\|\nu M_t^K - p_t\|_{\mathrm{TV}} \le \int \|\delta_x M_t^K - p_t\|_{\mathrm{TV}}\, \nu(dx) \le \sup_x \|\delta_x M_t^K - p_t\|_{\mathrm{TV}} \le C\rho^K.$$

In particular this holds for every ω-warm $\nu \in P_\omega(p_t)$.

(ii) Inverting. Setting $C\rho^K \le \eta$ yields $K \ge \log(C/\eta)/\log(1/\rho)$, so the warm-start mixing time of M_t is bounded by

$$\tau_{M_t}(\eta, \omega) \le \frac{\log(C/\eta)}{\log(1/\rho)}, \qquad \forall\, \omega \ge 1,\ \eta \in (0,1). \tag{44}$$

(iii) Elementary inequality. For $\rho \in (0,1)$, $\log(1/\rho) \ge 1-\rho$: the function $f(\rho) = -\log\rho - (1-\rho)$ has $f(1) = 0$ and $f'(\rho) = -1/\rho + 1 < 0$ on $(0,1)$, so $f \ge 0$ on $(0,1)$. Combining with (44) gives

$$\tau_{M_t}(\eta, \omega) \le \frac{\log(C/\eta)}{1-\rho}. \tag{45}$$

(iv) Substitution into (40)**.** Theorem 1 of Marion et al. (2023) sets $(\eta_\star, \omega_\star) = (1/(8NT), 2)$, so the mixing-time factor in (40) satisfies

$$\tau \;=\; \max_t \tau_{M_t}(1/(8NT), 2) \;\le\; \frac{\log(8NTC)}{1-\rho} \;=\; \frac{\mathcal{O}(\log(NT))}{1-\rho}.$$

The $\mathcal{O}(\log(NT))$ factor is polylogarithmic and is absorbed into $\widetilde{\mathcal{O}}$ in the SMCEVOLVE rate (38).

(S3) $\log \Gamma \le \beta\Delta_R$ VIA REWARD OSCILLATION

Lemma H.2 (Path length for reward tilting)**.** *Under the kernel hypothesis of Corollary H.1, suppose R is bounded on the support of $p_0(\cdot \mid q)$ with oscillation Δ_R. Then $\log \Gamma \le \beta\Delta_R$.*

Proof. Since $p^*(x \mid q) = Z(q)^{-1} p_0(x \mid q) e^{\beta R(x)}$,

$$\Gamma = \sup_{x: p_0(x|q)>0} \frac{p^*(x \mid q)}{p_0(x \mid q)} = \sup_{x: p_0(x|q)>0} \frac{e^{\beta R(x)}}{Z(q)} \le \frac{e^{\beta R_+}}{Z(q)}.$$

Moreover,

$$Z(q) = \mathbb{E}_{p_0}[e^{\beta R}] \;\ge\; e^{\beta R_-}$$

because $R(x) \ge R_-$ on the support of p_0. Hence $\Gamma \le e^{\beta(R_+ - R_-)} = e^{\beta\Delta_R}$, which gives $\log \Gamma \le \beta\Delta_R$. □

Combining (S1)–(S3). Substituting (41), (45), and Lemma H.2 into the imported bound (40):

$$\mathcal{B} \;=\; \mathcal{O}^*\!\left(\frac{\epsilon^{-2} \vee \kappa^{-1}}{1-\rho}\, \log\Gamma\, \log^2\!\log\Gamma\right) \;\subseteq\; \widetilde{\mathcal{O}}\!\left(\frac{\epsilon^{-2} \vee \kappa^{-1}}{1-\rho}\, \beta\Delta_R\right),$$

where the iterated-logarithm factors are polylogarithmic in $\beta\Delta_R$ and absorbed into $\widetilde{\mathcal{O}}$. The concentration statement $\mathbb{P}(|\eta_T^N(f) - p^*(f)| \le \epsilon) \ge 3/4$ is inherited directly from the imported bound. This is exactly (37)–(38) in Corollary H.1. □

H.1 PER-FACTOR REQUIREMENTS AND DEGENERATE REGIMES

Corollary H.1 bounds the joint LLM-call budget $\mathcal{B} = NTK$, but each factor must individually be large enough for the SMC sampler to function. Concretely, the corollary requires the particle count to satisfy $N \gtrsim \kappa^{-1} \vee \epsilon^{-2}$, so the ESS constraint controls the bridge-ratio variance and the Monte Carlo error is at most ϵ, and the mutation count to satisfy $K \gtrsim \log(1/\epsilon)/(1-\rho)$, so the uniform-ergodicity TV error $C\rho^K$ falls below ϵ at every bridge; the annealing length $T = \widetilde{\mathcal{O}}(\beta\Delta_R)$ is set adaptively by the ESS bisection rule and is not a free hyperparameter. The extreme cases $N = 1$ (no resampling diversity) and $K = 1$ (a single MH step per particle, insufficient mixing whenever ρ is close to 1) sit outside this regime, and the formal guarantee no longer applies.

I IMPLEMENTATION DETAILS

I.1 ISLAND-BASED PARALLELISM

SMCEVOLVE runs I independent SMC chains (*islands*), each with N particles and its own temperature schedule. Every τ epochs, islands exchange top-performing particles via a migration step: each island sends its best m particles to a randomly assigned neighbor and retains the top N after merging. This maintains diversity across subpopulations while enabling parallel execution.

I.2 RESAMPLING AND NUMERICAL STABILITY

We use *systematic resampling* (Carpenter et al., 1999), which requires only a single uniform random number and is computationally convenient in practice; see Douc et al. (2005) for a comparison with the stratified variant of Kitagawa (1996) and a variance analysis. For numerical stability, the log-weights $\log w_t^{(n)} = \Delta\beta_t \cdot R(x_{t-1}^{(n)})$ are shifted by their maximum before exponentiation (the log-sum-exp trick), preventing overflow without affecting the normalized weights (9).

I.3 SUMMARY OF THEORETICAL GUARANTEES

Table 5 summarizes how each component of SMCEVOLVE maps to the SMC framework and which guarantees are preserved.

Table 5: Mapping between the SMC framework and the SMCEVOLVE implementation. **Exact**: the theoretical guarantee is preserved. **Approx.**: a practical relaxation is introduced.

Component	Implementation	Status
Target distribution $p^* \propto p_0\, e^{\beta R}$	Same	Exact
Annealing path $p_t \propto p_0\, e^{\beta_t R}$	Same	Exact
Importance weights $w_t = e^{\Delta\beta_t R(x_{t-1})}$	Log-sum-exp stabilized	Exact
Temperature schedule	ESS bisection	Adaptive extension
Resampling	Systematic	Exact
Forward kernel M_t	MH kernel with LLM proposal mixture	Exact†
Kernel selection	Thompson sampling mixture	Approx. (via $\widehat{M}_t^{H_t}$)
Parallelism	Islands + migration	Approx.

†Covered by Theorem 4.1, whose limit η_T equals p^* for invariant kernels (Section F).

The target distribution, annealing path, and importance weights remain exact. Mutation heuristics, adaptive kernel selection, adaptive temperature schedules, and island-based parallel extensions enter the theory only through additional approximation terms or extra assumptions beyond those of the main theorem.

J TASK DETAILS

This appendix collects natural-language descriptions of every benchmark task evaluated in Tables 1–3 and Figure 4. Formal interfaces (input/output signatures, evaluators) are documented in the per-task `task.md` files released with our code.

J.1 MATH DOMAIN (TABLE 1)

Ten open problems from analytic combinatorics and discrete geometry, ported from the public release accompanying DeepMind's AlphaEvolve. Each task asks the program to *construct* a concrete mathematical object (a packing, a point set, or a step function) that pushes a known numerical constant in the direction predicted by theory; a score of 1.0 ties the 2024 AlphaEvolve benchmark, and scores above 1.0 strictly improve on it.

J.2 SYMBOLIC REGRESSION DOMAIN (TABLE 3)

Sixteen scientific-equation discovery tasks (4 splits $\times$ 4 tasks) drawn from LLM-SRBench (Shojaee et al., 2025). Each task hides a ground-truth analytical expression behind a small numeric dataset; the program must propose a closed-form parametric expression (with up to ten free coefficients) whose mean-squared error against the held-out training data is as small as possible. The reward $-\log_{10}(\mathrm{MSE})$ rewards exponentially better fits, so a one-decade improvement in MSE counts as $+1.0$ in score.

J.3 ALGOTUNE DOMAIN (TABLE 2)

Eight numerical, signal-processing, and linear-algebra tasks from the public AlgoTune benchmark (Press et al., 2025). Each task ships with a known *correct* reference implementation (typically a thin wrapper around SciPy or LAPACK); the program's job is to produce a drop-in replacement that is measurably faster on the benchmark's input distribution while still passing AlgoTune's correctness verifier. The reward is the wall-clock speedup over the reference (so 1.0 = no speedup, 10.0 = ten times faster); any incorrect output scores 0.

Table 6: Math domain—AlphaEvolve combinatorial geometry tasks.

Task	Description
Circle Packing in Rect. (N=21)	Place 21 disjoint circles in an axis-aligned rectangle of perimeter ≤ 4 (width + height ≤ 2); maximize the sum of radii. AlphaEvolve App. B.13.
Hexagon Packing (N=11)	Pack 11 disjoint unit regular hexagons inside a larger regular hexagon; minimize the outer side length (equivalently, maximize its reciprocal). AlphaEvolve App. B.7.
Heilbronn Convex (N=13)	Inside a convex region of unit area, place 13 points so that the smallest triangle spanned by any three is as large as possible. AlphaEvolve App. B.10.
Heilbronn Triangle (N=11)	Same min-triangle-area objective restricted to the unit equilateral triangle with vertices $(0,0),(1,0),(\frac{1}{2},\frac{\sqrt{3}}{2})$. AlphaEvolve App. B.9.
Min-Max-Min Dist. (n=16, d=2)	Place 16 points in the plane; minimize the ratio of the largest to smallest pairwise Euclidean distance (a "close-to-uniform" configuration). AlphaEvolve App. B.8.
Kissing Number (d=11)	Construct an integer lattice cloud in $\mathbb{R}^{11}$ whose minimum pairwise squared distance is at least the maximum squared norm of any point; maximize cardinality (a lower bound on the 11D kissing number). AlphaEvolve App. B.11.
1st Autocorrelation Inequality	Construct a non-negative step function $f:\mathbb{R}\to\mathbb{R}_{\geq 0}$ whose self-convolution improves the upper bound on the constant C_1. AlphaEvolve App. B.1.
2nd Autocorrelation Inequality	Construct a non-negative step function f that pushes the lower bound on the related constant C_2 (defined via the autoconvolution norm) upward. AlphaEvolve App. B.2.
3rd Autocorrelation Inequality	Construct a (possibly signed) step function f that improves the upper bound on C_3, governing $\lvert f\star f\rvert$. AlphaEvolve App. B.3.
Erdős Minimum Overlap	Construct a non-negative step function h satisfying the Minimum Overlap regularity constraints, lowering the best known upper bound on its asymptotic constant. AlphaEvolve App. B.5.

Table 7: Symbolic Regression domain—four LLM-SRBench splits, four tasks per split.

Split	Description
`bio_pop_growth`	Population-growth ODEs for $P(t)$ (logistic, Gompertz, Allee-effect, and similar). Given samples (t,P), the program must express the instantaneous growth rate $\mathrm{d}P/\mathrm{d}t$ in closed form. Tasks BPG0–BPG3 cover four ODEs of varying complexity.
`chem_react`	Chemical reaction-rate ODEs for the concentration $A(t)$ of a species (first-order decay, autocatalytic, Michaelis–Menten-like, etc.). Given (t,A) samples, the program must recover $\mathrm{d}A/\mathrm{d}t$. Tasks CRK0–CRK3 cover four mechanisms.
`matsci`	Materials-science constitutive laws $\sigma(\varepsilon,T)$ from continuum mechanics (linear elasticity with thermal expansion, power-law hardening, Arrhenius softening, etc.). Tasks MatSci0–MatSci3 cover four laws.
`phys_osc`	Second-order oscillator ODEs for the velocity $v(t)$ of a particle (harmonic, damped, driven, anharmonic). The program expresses the acceleration $\mathrm{d}v/\mathrm{d}t$ from a task-dependent subset of (x,t,v): PO0/PO2 use all three; PO1 uses (x,t); PO3 uses (t,v).

J.4 AutoResearch Domain (Figure 4)

A single high-stakes systems task ported from `karpathy/autoresearch` (Karpathy, 2026). The program under evolution is a *complete* single-file GPT pretraining script: model architecture, tokenizer setup, optimizer, learning-rate schedule, batching, training loop, and final evaluation. Each candidate

Table 8: AlgoTune domain—runtime-optimization tasks.

Task	Description
`affine_transform_2d`	Apply a 2D affine warp (encoded as a 2×3 matrix) to an $n\times n$ image with cubic-spline interpolation and zero padding. Reference: `scipy.ndimage.affine_transform`.
`convolve2d_full_fill`	Compute the full 2D convolution of two real matrices of size $(30n, 30n)$ and $(8n, 8n)$ with zero-padded boundaries—the dense direct case where blocking, FFT, or specialized kernels yield large speedups.
`eigenvectors_complex`	Full eigendecomposition of a real square matrix that may have complex-conjugate eigenvalue pairs; return eigenvalues sorted by descending real then imaginary part with unit-norm eigenvectors. Reference: `numpy.linalg.eig`.
`fft_cmplx_scipy_fftpack`	N-dimensional FFT of a complex $n\times n$ matrix matching the `scipy.fftpack` reference output.
`fft_convolution`	Discrete convolution $z[n] = \sum_k x[k]\, y[n-k]$ of two 1D signals via FFT, supporting the standard `full`/`same`/`valid` output modes.
`lu_factorization`	LU factorization $A = PLU$ of a square matrix, returning P, unit-lower-triangular L, and upper-triangular U. Reference: `scipy.linalg.lu`.
`polynomial_real`	Given coefficients of a degree-n polynomial with all real roots, return the roots in descending order, accurate to at least three decimal places.
`psd_cone_projection`	Project a symmetric matrix A onto the cone of symmetric PSD matrices (zero out negative eigenvalues). The closed form is short, but a naive implementation leaves a several-times speedup on the table.

trains for 60 s of wall-clock time on a single RTX A5000 (24 GB) on the climbmix corpus, after which the script must report `val_bpb` (bits-per-byte on a held-out shard). Lower `val_bpb` means a better language model; the reward $\max(0,\ 2.0 - \texttt{val_bpb})$ is anchored so that a random baseline ($\texttt{val_bpb} \approx 2.6$) scores 0 and a well-trained small GPT ($\texttt{val_bpb} \approx 1.0$) scores about 1.0.

Table 9: AutoResearch domain—single-GPU GPT pretraining task.

Field	Specification
Goal	Evolve a single-file GPT pretraining script that minimizes `val_bpb` on a held-out shard within a fixed compute budget.
Reward	$\max(0,\ 2.0 - \texttt{val_bpb})$. Random baseline ≈ 0; well-trained small GPT ≈ 1.0.
Hardware / time	Single RTX A5000 (24 GB), 60 s wall-clock training per candidate ($\approx$90 s end-to-end including evaluation).
Hard constraints	VRAM $\leq$ 24 GB (OOM scores 0); single GPU (no distributed training); no extra Python packages may be installed.
Seed program	6-layer GPT, embedding dim 384, 3 attention heads, full causal attention via PyTorch `scaled_dot_product_attention`; MuonAdamW optimizer (Muon for 2D weight matrices, AdamW elsewhere).
Search space	Free to modify model depth/width, optimizer, attention backend, batch size, learning-rate schedule, initialization, etc. The evaluation harness (tokenizer, dataloader, `evaluate_bpb` from the fixed `prepare.py`) must be left untouched, and the last stdout line must be the literal `val_bpb:` `<float>` produced by it.

K Existing Assets and Licenses

We use existing assets only as benchmark and evaluation resources. The math tasks are ported from the public AlphaEvolve benchmark release; the symbolic-regression tasks are drawn from LLM-SRBench; the numerical optimization tasks are drawn from AlgoTune; and the AutoResearch task uses the TinyStories/climbmix pretraining setup from `karpathy/autoresearch`. We credit the original sources in the main text and task descriptions. We do not redistribute third-party datasets, pretrained models, or benchmark code as new assets; our code release will include attribution, links, and the applicable license or terms-of-use information for each external resource. Commercial LLM APIs are accessed under the providers' terms, and users reproducing the experiments must supply their own API credentials.

L Hyperparameters

Table 10 lists the hyperparameters that materially affect SMCEvolve's behavior, together with the values used in our experiments. The compute-budget group sets the LLM-call ceiling $\leq n_{\text{islands}} \times N \times K \times T_{\max}$; the SMC-annealing group controls the adaptive temperature schedule; the remaining groups govern mixing, mutation, and LLM behavior.

Table 10: SMCEvolve hyperparameters.

Parameter	Value	Meaning
Compute budget		
`n_islands`	2	Number of parallel islands.
`particles_per_island`	8	Particles N per island.
`n_proposals`	2	K sequential LLM proposals per particle per step, used as a K-step MH chain.
`min_iterations`	3	Lower bound on SMC steps; also caps $\Delta\lambda \leq 1/$`min_iterations`.
`max_iterations`	15	Hard upper bound $T_{\max}$ on SMC steps.
SMC annealing		
`beta`	20	Target inverse temperature β (annealing endpoint).
`kappa`	0.9	ESS threshold used by bisection to pick λ_t.
Island migration		
`migration_interval`	3	SMC steps between successive migration events.
`migration_size`	1	Particles exchanged per migration event.
Mutation kernel		
`kernel_selection`	adaptive	Thompson-sampling mixture over the 2×2 kernel grid (`diff`/`rewrite` $\times$ with/without inspirations).
`top_k_inspiration`	2	Highest-reward exemplars from the same island.
`diverse_inspirations`	2	MAP-Elites-style diversity exemplars (embedding-distance).
LLM backend		
`temperature`	1.0	LLM sampling temperature.
`max_tokens`	4096	Maximum tokens per LLM response.
`models`	50/50	Ensemble of `gpt-5-mini` and `gemini-3-flash` (`gpt-5.4` + `gemini-3-pro` for AutoResearch).

M Broader Impact

SMCEvolve is intended as a research framework for LLM-driven program search. Its main potential benefit is to make automated scientific discovery and program optimization more sample-efficient and easier to monitor through explicit resampling and stopping criteria. Our experiments are conducted on benchmark tasks and do not involve human subjects or sensitive data. As with any automated

code-generation system, generated candidates should be treated as suggestions and validated by task-specific tests before use, especially outside controlled benchmarks. The method also relies on access to external LLM APIs, which may affect cost and reproducibility across users.

N MUTATION KERNEL PROMPTS

SMCEVOLVE mutates particles through a 2×2 grid of LLM prompt templates along two axes:

Axis	Options
Edit granularity	`diff` (local SEARCH/REPLACE patches) vs. `rewrite` (full-program rewrite)
Information source	`no_inspo` (parent only) vs. `with_inspo` (parent + top-k and diversity exemplars from the same island)

At each call, the system prompt fixes the kernel's role and the expected output format; the user prompt injects the parent program (rendered with the language tag `{{language}}` into `{{code_content}}`), its reward (`{{performance_metrics}}`), and—for `with_inspo` kernels—a block of reference programs (`{{inspiration_section}}`). When no reference programs are available (e.g., at initialization), `with_inspo` kernels fall back to `no_inspo`.

N.1 `diff_no_inspo`: SINGLE-PARTICLE LOCAL KERNEL

System Prompt

```
You are an expert programmer tasked with making targeted improvements to an existing
    program.
Focus on small, precise edits that improve performance -- fix inefficiencies, tune
    parameters, optimize hot paths, or refine logic.

You MUST respond using an edit name, description, and the exact SEARCH/REPLACE diff format
    :

<NAME>
A shortened name summarizing the edit. Lowercase, no spaces, underscores allowed.
</NAME>

<DESCRIPTION>
Explain the targeted change you are making and why it should improve performance.
</DESCRIPTION>

<DIFF>
<<<<<<< SEARCH
# Original code to find and replace (must match exactly including indentation)
=======
# New replacement code
>>>>>>> REPLACE
</DIFF>

* Every SEARCH section must be copied verbatim from the current file, including
    indentation, whitespace, and comments -- matching is byte-for-byte.
* Every SEARCH section must match EXACTLY ONE location in the current file. If the snippet
     you want to change appears more than once, add enough surrounding lines of context
    to make the match unique; ambiguous edits are rejected.
* SEARCH must differ from REPLACE (no-op edits are rejected).
* You can propose multiple independent edits. SEARCH/REPLACE blocks follow one after
    another with no other text between them; they are applied in order, so later blocks
    see the result of earlier ones.
* Make sure the file still runs after your changes.
* IMPORTANT: Do not rewrite the entire program -- focus on targeted, surgical improvements
    .
```

User Prompt

```
# Current Program

```{{language}}
{{code_content}}
```

Performance metrics:
{{performance_metrics}}

Task

Suggest targeted edits to improve the program's performance.
```

```
Focus on the most impactful small changes -- parameter tuning, algorithmic micro-
    optimizations, or bug fixes.
Describe each change with a SEARCH/REPLACE block.
```

N.2 `diff_with_inspo`: INTERACTING LOCAL KERNEL

System Prompt

```
You are an expert programmer tasked with making targeted improvements to an existing
    program.
You will be shown the current program AND one or more high-performing reference programs
    that solve the same problem.
Study the reference programs to identify specific techniques, parameter choices, or code
    patterns that could be surgically transplanted into the current program to improve it.


[Output format identical to diff_no_inspo: <NAME>, <DESCRIPTION>, and <DIFF> with SEARCH/
    REPLACE blocks.]

* IMPORTANT: Do not rewrite the entire program -- borrow and adapt specific ideas from the
     references via targeted edits.
```

User Prompt

```
# Current Program

```{{language}}
{{code_content}}
```

Performance metrics:
{{performance_metrics}}

Reference Programs

The following high-performing programs solve the same problem. Study their techniques and
    borrow specific ideas to improve the current program via targeted edits.

{{inspiration_section}}

Task

Identify the most useful techniques from the reference programs and transplant them into
    the current program via small, targeted SEARCH/REPLACE edits.
Focus on adapting specific patterns -- do not rewrite the entire program.
```

N.3 `rewrite_no_inspo`: SINGLE-PARTICLE GLOBAL KERNEL

System Prompt

```
You are an expert algorithm designer. Given a program and its performance, design a
    fundamentally improved or completely different algorithm to solve the same problem.
Do not just tweak the existing code -- rethink the approach from scratch. Consider
    different data structures, algorithmic paradigms, mathematical formulations, or
    optimization strategies.

You MUST respond using a short summary name, description, and the full code:

<NAME>
A shortened name summarizing the code you are proposing. Lowercase, no spaces, underscores
     allowed.
</NAME>

<DESCRIPTION>
Explain the new algorithmic approach you are taking, how it differs from the current one,
    and why it should perform better.
</DESCRIPTION>

<CODE>
```{{language}}
# The new program here.
```
```

```
</CODE>

* Your program must maintain the same inputs and outputs as the original.
* Think broadly -- consider entirely different algorithms, not just parameter changes.
* Make sure the file still runs after your changes.
```

User Prompt

```
# Current Program

```{{language}}
{{code_content}}
```

Performance metrics:
{{performance_metrics}}

Task

Design a new algorithm to replace the current implementation.
Do not make incremental changes -- propose a fundamentally different or significantly
     improved approach.
Your new program must maintain the same inputs and outputs.
```

N.4 `rewrite_with_inspo`: INTERACTING GLOBAL KERNEL (CROSSOVER)

System Prompt

```
You are an expert algorithm designer. You will be shown the current program AND one or
     more high-performing reference programs that solve the same problem.
Your task is to synthesize a new program that combines the best ideas from all provided
     programs, or uses them as inspiration to design something even better.
Think of this as intelligent crossover -- not just copy-paste, but creative recombination
     of algorithmic insights.

[Output format identical to rewrite_no_inspo: <NAME>, <DESCRIPTION>, and full <CODE>.]

* Combine strengths from multiple programs -- don't just pick one and copy it.
```

User Prompt

```
# Current Program

```{{language}}
{{code_content}}
```

Performance metrics:
{{performance_metrics}}

Reference Programs

The following high-performing programs solve the same problem. Use them as inspiration to
     design a new, superior program that combines the best ideas.

{{inspiration_section}}

Task

Synthesize a new program that combines the best elements from the current program and the
     reference programs.
Go beyond simple merging -- creatively recombine algorithmic insights to produce something
      better than any individual program.
Your new program must maintain the same inputs and outputs.
```

O VISUALIZING AN SMCEVOLVE RUN

Figure 5 traces a real run of SMCEVOLVE on the *Circle Packing in a Rectangle* (N=21) task with I=2 islands, N=8 particles per island, and K=2 MH proposals per particle. Each panel shows one SMC iteration on a single island, organized into the three sub-steps of the iteration: *reweight*

(top), *resample* (middle), and *mutate* (bottom). In the reweight row, node area is proportional to the normalized importance weight $w_i / \max_j w_j$, and a thick colored ring marks particles that arrived via inter-island migration in the previous epoch. Resample arrows depict the replication map: arrows fanning out from a single parent indicate replication of a high-weight ancestor, while parents with no outgoing arrow are discarded. The mutate block stacks one row per MH proposal, with vertical dashed lines linking the successive proposals of the same particle.

Node *fill* color encodes reward through a global red–yellow–green colormap normalized over the full run. On mutate nodes we additionally encode three pieces of kernel metadata: *stroke color* distinguishes the edit operator (blue = local diff edit, orange = full-program rewrite); *stroke width* marks whether the proposal was conditioned on inspiration programs (thick) or not (thin); and *line style with fill opacity* encodes the MH outcome (solid/opaque = accepted, dashed/translucent = rejected with the fill reverting to the parent reward, dotted/faded = skipped).

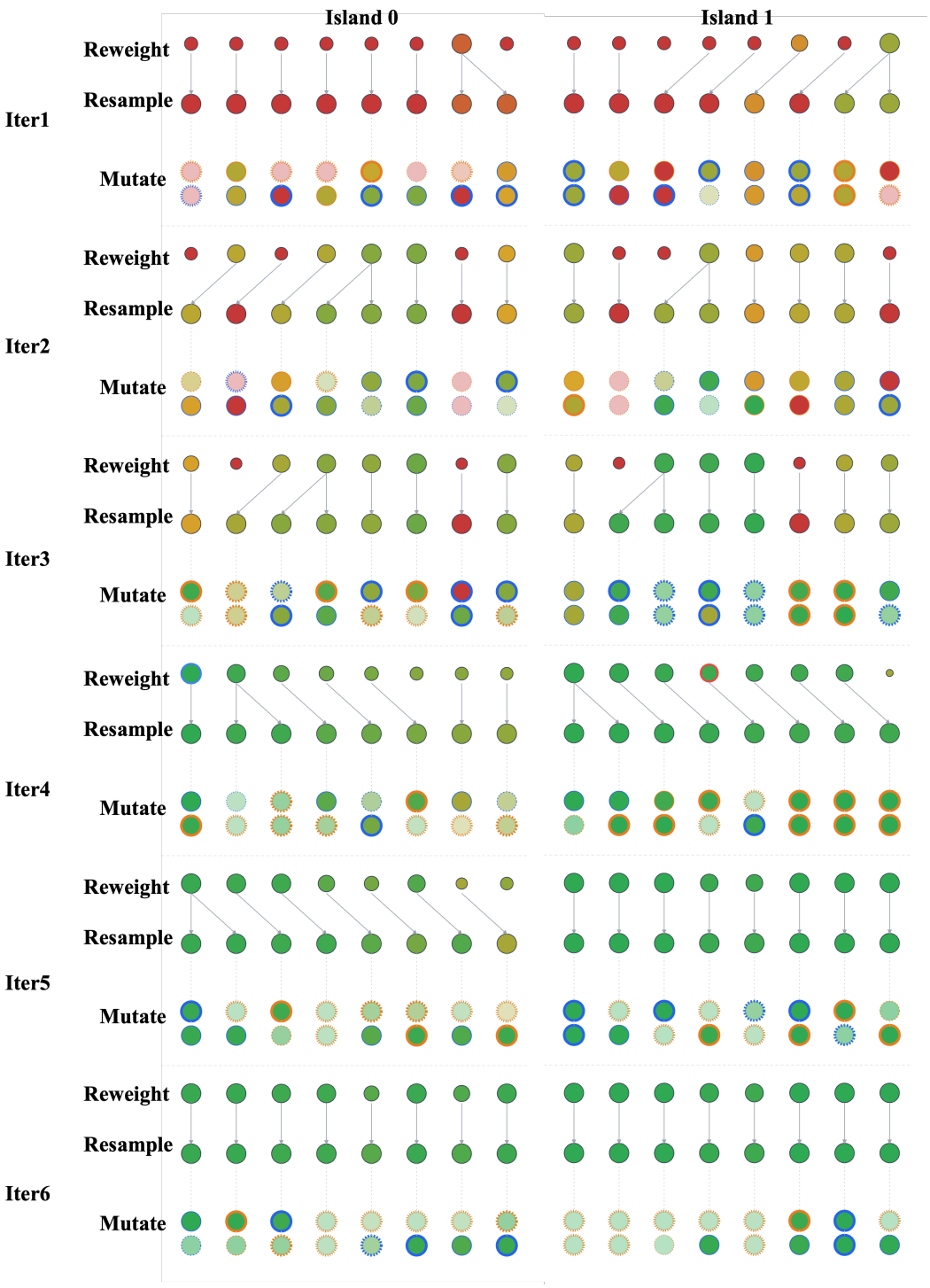


Figure 5: **Per-iteration SMC flow on Circle Packing** N=21 (I=2 islands, N=8 particles per island, K=2 MH proposals). Rows of each panel correspond to *reweight*, *resample*, and *mutate*; node fill encodes reward (red–yellow–green); thick rings in the reweight row mark inter-island migrants; resample arrows show the replication map; on mutate nodes, stroke color distinguishes the edit operator (blue = local diff, orange = full rewrite), stroke width flags inspiration-conditioned proposals, and line style/opacity encodes the MH outcome (solid = accepted, dashed = rejected, dotted = skipped).