OVERCOMING SEMANTIC MISALIGNMENT IN LATENT REASONING AUTO-DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

We envision large language models (LLMs) not only as reasoners, but also as designers of reasoning algorithms. A key step toward this goal is to make reasoning algorithms explicit objects of auto-design. We study this step in training-free latent reasoning, where hand-crafted algorithms can improve reasoning by reorganizing inference-time computation without updating model weights. The central challenge is semantic misalignment between program representations and the the algorithmic semantics of latent reasoning: implementation details obscure algorithm design, also making accumulated experience difficult to translate into targeted revisions. Our key idea is to construct a semantically aligned design space for LLM-driven algorithm evolution. To this end, we introduce LRDSL, a domain-specific language that represents algorithm logic using composable components for observation, stateful control, latent-input construction, and inference-budget management. These components serve as shared semantic units for algorithm design, behavioral measurement, and experience accumulation.Within this space, a proposer LLM designs reasoning algorithms for a target LLM and iteratively refines them using experience distilled from execution feedback. Across three design systems and two backbones, LRDSL improves the macro-average accuracy of the discovered algorithms on five reasoning benchmarks in all six configurations, by up to 5.65 percentage points. The best discovered policies exceed the strongest human-designed baseline by 1.53.

1 INTRODUCTION

Improving the reasoning abilities of large language models (LLMs) has become a central goal of AI research (Huang & Chang, 2023; Zhao et al., 2026). Existing approaches pursue this goal in several complementary directions. Chain-of-thought prompting elicits intermediate reasoning steps (Wei et al., 2022), self-consistency samples and aggregates multiple reasoning paths (Wang et al., 2023), and reinforcement learning can develop reasoning behaviors through training (DeepSeek-AI et al., 2025). These advances show that improving reasoning depends not only on building stronger models, but also on designing effective procedures for using their capabilities. In particular, the way intermediate computation is generated, explored, and organized can substantially affect reasoning performance (Chen et al., 2024; Snell et al., 2025). Designing reasoning algorithms is therefore an important avenue for advancing LLM reasoning.

Recent progress in AI agents creates an opportunity to automate this design process. Coding agents such as Codex (OpenAI, 2025) and Claude Code (Anthropic, 2025) combine frontier LLMs with tools for codebase navigation, code editing, and execution, enabling iterative development guided by execution feedback. LLM-driven evolutionary search also offers a route: systems such as FunSearch (Romera-Paredes et al., 2024) and AlphaEvolve (Novikov et al., 2025) evolve a population of programs through LLM-proposed edits and have found algorithms that surpass the best known solutions in mathematics and computer science. We investigate whether these capabilities translate into effective algorithm discovery in training-free latent reasoning. As an initial test, we task Codex, Claude Code, and OpenEvolve (Novikov et al., 2025), an open-source implementation of AlphaEvolve, with proposing, implementing, evaluating, and iteratively refining candidate algorithms. Under matched search budgets, none of them discovers an algorithm that outperforms the strongest human-designed baseline evaluated in our experiments (Figure 1(a)).

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

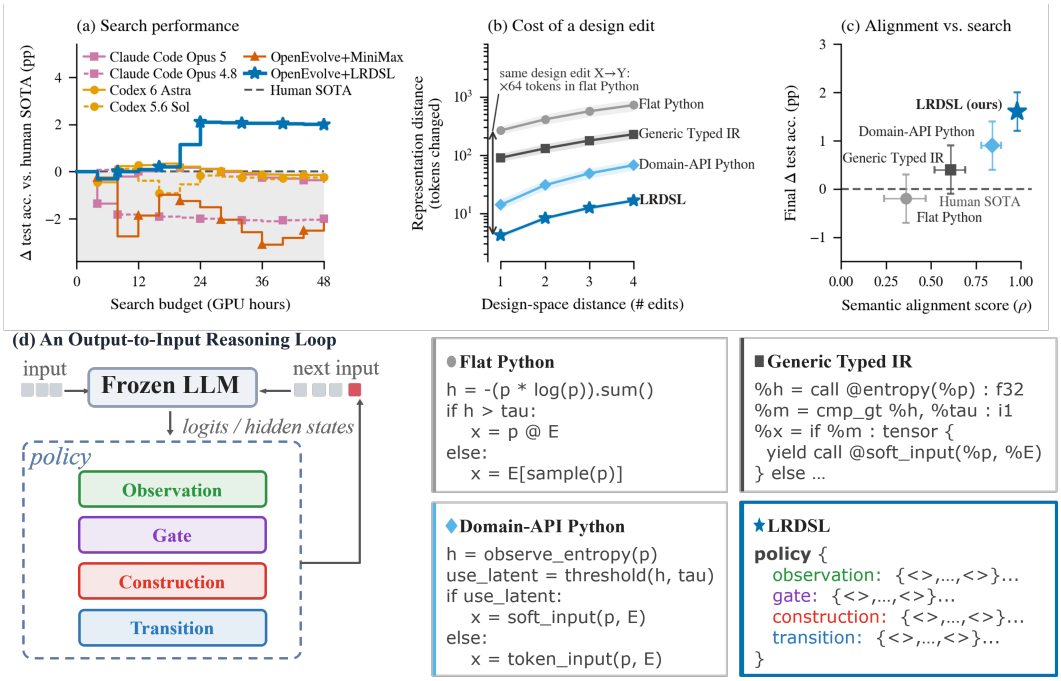

Figure 1: Semantic misalignment limits the search for latent-reasoning algorithms. (a) Test accuracy of the best candidate found so far, relative to the strongest human-designed method (dashed line), on GSM8K with Qwen3-1.7B. Codex, Claude Code and OpenEvolve search over raw code; LRDSL uses our semantic local search. All systems start from SeLaR and share a budget of 48 GPU hours. Candidates are selected on the validation split and evaluated on the test split. (b) Representation distance versus design distance for pairs of algorithms. Design distance counts the decisions (Observation, Gate, Construction, Transition) on which two algorithms differ. Representation distance counts the tokens that the proposer LLM changes when it edits one implementation into the other; only edits whose behavior on probe prompts matches the target design are kept. Lines show medians; bands show interquartile ranges. (c) Semantic alignment score, the Spearman correlation between the two distances in (b), versus final search performance, measured as in (a). Every representation is searched with the same LLM, budget, and starting algorithm. Error bars show 95% bootstrap intervals over base algorithms (horizontal) and the standard deviation over seeds (vertical). (d) The output-to-input reasoning loop and one policy written in four representations. Colors mark the decision each line implements.

We hypothesize that the bottleneck is semantic misalignment: the code that the LLM edits does not reflect the design decisions that define an algorithm. As shown in Figure 1(d), evolutionary search improves an algorithm through many small edits, so it needs a small edit to the program to be a small, predictable change to the algorithm. Raw code breaks this link. A latent-reasoning algorithm is defined by a few decisions distilled from existing methods (Zhang et al., 2025b; Zhuang et al., 2025; Shi et al., 2025; Fu & Luo, 2026; Lv et al., 2026), such as what to observe from the model's outputs and what input to feed back. In Flat Python, each decision is spread across the decoding loop and mixed with state handling. Changing one decision therefore requires a large rewrite, about [64]× more tokens than in LRDSL, and the size of a rewrite says little about how much the algorithm has changed. This misalignment directly hurts search. We quantify it with a semantic alignment score, which measures how closely edit size tracks the number of changed decisions. Across four representations of the same algorithms, search performance falls as alignment falls (Figure 1c). Guaranteeing well-formed programs is not enough: a typed IR with generic operations is also poorly aligned and also searches poorly. To discover better algorithms, the search must operate on a representation whose edits match the design decisions.

To address this bottleneck, we introduce a typed domain special language namde LRDSL, a semantically aligned design space for latent-reasoning policies. Each policy consists of four composable components: Observation, Gate, Construction, and Transition. These components specify what to

observe, which reasoning mode to use, what input to construct, and how to update the execution state.The execution of these components forms an output-to-input reasoning loop: the frozen LLM produces logits and hidden states, the policy observes these outputs, selects a reasoning mode, constructs the next input, and feeds it back to the LLM. (Figure 1d) illustrates this loop. This structure allows the LLM to reason about and modify algorithmic choices rather than low-level implementation details. Typed interfaces and explicit execution semantics define the operational role of each editable decision. This structure allows the LLM to reason about and modify algorithmic choices rather than low-level implementation details. We perform semantic local search within this design space. At each iteration, a proposer LLM selects a parent policy and proposes an edit. Each edit has a declared scope that constrains which algorithmic decisions may be modified. Controlled behavioral diagnostics check whether the intended intervention is realized, while paired end-to-end executions measure its effect on task accuracy and inference cost. We retain each evaluated edit as conditional experience, recording its parent policy, observed effects, and evaluation conditions. The proposer retrieves relevant experience to guide subsequent edits. Policy representations, semantic edits, execution feedback, and accumulated experience thus refer to the same algorithmic decisions. We evaluate both the policies discovered through this process and the role of LRDSL in the search. On Qwen3-1.7B, the best discovered policy achieves a macro-average accuracy of 63.21% across five mathematical and scientific reasoning benchmarks.

Our contributions are summarized as follows:

- We identify semantic misalignment, namely the mismatch between distances in the algorithm design space and distances in the representation edited by the LLM, as a key reason why current agents fail to improve upon human-designed latent-reasoning algorithms. We propose the semantic alignment score to quantify this mismatch and show that it tracks search performance across representations.
- We introduce LRDSL (*Latent Reasoning Domain-Specific Language*), a semantically aligned representation for latent-reasoning policies. LRDSL aligns policies, execution feedback, and accumulated experience within a common search state, enabling semantic local search over algorithmic decisions.
- We show that semantic local search over LRDSL discovers latent-reasoning policies that outperform human-designed methods. On Qwen3-1.7B, the best discovered policy reaches a macro-average accuracy of 63.21% across five mathematical and scientific reasoning benchmarks, 1.5 points above the strongest human-designed method, with a gain of up to 13.34 points on AIME25.

2 Related Work

Auto-design Method. FunSearch and AlphaEvolve combine language-model proposals, executable evaluation, and evolutionary archives to discover programs or algorithms (Romera-Paredes et al., 2024; Novikov et al., 2025). Automated Design of Agentic Systems, Evolution of Heuristics, ReEvo, GEPA, Promptbreeder, and LLaMEA use proposer–evaluator or reflective evolutionary loops to improve agent programs, heuristics, prompts, or metaheuristics (Hu et al., 2025; Liu et al., 2024; Ye et al., 2024; Agrawal et al., 2025; Fernando et al., 2024; van Stein & Bäck, 2025). AFlow and DSPy represent agent workflows as structured computational graphs and optimize them with execution feedback (Zhang et al., 2025a; Khattab et al., 2024); AutoML-Zero, DreamCoder, and ADSL-PDE show how domain-specific primitives or libraries can make algorithm search executable and reusable (Real et al., 2020; Ellis et al., 2021; Kong et al., 2026). Reflexion illustrates how textual feedback can be retained across attempts (Shinn et al., 2023). These systems establish important precedents for automated search, structured representations, and experience reuse, but they target programs, prompts, or agent workflows rather than inference-time latent interventions inside a frozen language model.

Reasoning with explicit and latent representations. Chain-of-thought prompting exposes intermediate reasoning as language, and test-time search methods such as self-consistency and Tree-of-Thoughts explore multiple explicit trajectories (Wei et al., 2022; Wang et al., 2023; Yao et al., 2023). Latent-reasoning methods move part of this computation into continuous representations. COCONUT trains a model to recycle a hidden state as the next input representation (Hao et al., 2025), whereas Soft Thinking constructs a probability-weighted mixture of token embeddings without updating

the target model (Zhang et al., 2025b). Mixture of Inputs instead blends a sampled token with its predictive distribution through a Bayesian posterior construction (Zhuang et al., 2025). Analyses of soft thinking report a tendency to follow the highest-probability token, motivating stochastic alternatives (Wu et al., 2026). SoftCoT uses a fixed assistant model and a trainable projection to provide soft thoughts to a frozen target LLM (Xu et al., 2025a); SoftCoT++ adds specialized initial tokens and contrastive learning to diversify these thoughts (Xu et al., 2025b). SwiReasoning, SeLaR, and DyLaR further control when latent and explicit computation is used, using switching signals, selective activation, or uncertainty-guided residual refinement (Shi et al., 2025; Fu & Luo, 2026; Lv et al., 2026).

3 METHOD

We propose a semantically aligned framework for automatically designing training-free latent-reasoning algorithms. The backbone model remains frozen, while a design LLM searches over reasoning policies. Our Latent Reasoning Domain-Specific Language (LRDSL) represents each policy through four components: Observation, Gate, Construction, and Transition. A shared backend converts these specifications into executable policies, and an iterative proposal–evaluation loop improves them through semantic local search.

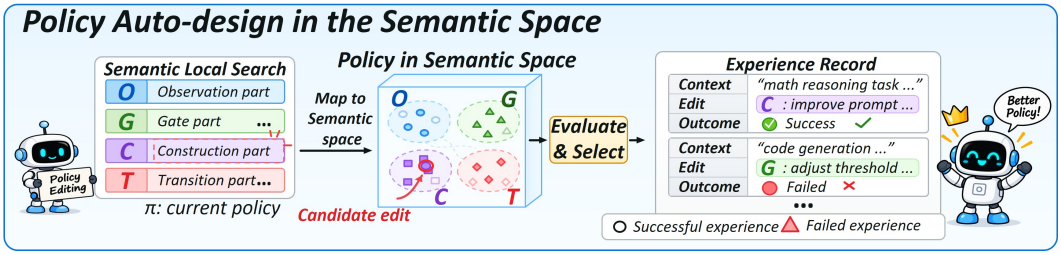

Figure 2: Overview of the semantically aligned policy design process.

3.1 LRDSL POLICY REPRESENTATION

Policy representation. Let f_θ denote the frozen backbone LLM with parameters θ. An LRDSL policy is represented as

$$\alpha = (O, G, C, T) \in \mathcal{A}_{\text{LRDSL}}, \qquad \pi_\alpha = \text{Instantiate}(\alpha; f_\theta), \tag{1}$$

where α is a complete policy specification, $\mathcal{A}_{\text{LRDSL}}$ is the LRDSL policy space, and π_α is the executable policy obtained by instantiating α with f_θ. The four entries of α are Observation O, Gate G, Construction C, and Transition T. The initial policy provides a starting point for search, and later candidates may modify any component.

Policy components. Observation O specifies the information available to the policy. Its primitives can read predictive distributions, hidden representations, and execution history, and derive signals such as uncertainty or computation progress. Gate G selects the reasoning action, such as continuing latent computation, switching to discrete reasoning, or ending the intermediate reasoning stage. Construction C forms a latent input through operations such as selection, mapping, weighted aggregation, and normalization. Transition T applies the selected action, updates the reasoning state, and invokes the backend model interface.

Each component is built from primitives with explicitly specified input and output types. These types distinguish probability distributions, hidden states, input embeddings, and control signals. When two connected operations use different types, LRDSL requires an explicit mapping primitive to convert between them. This constraint prevents type-incompatible compositions while keeping both the mechanism structure and primitive parameters editable during search.

Execution semantics. Let τ index reasoning steps and let s_τ denote the runtime state at step τ, including the current model outputs, context, reasoning mode, and execution history. The executable

policy π_α runs according to

$$o_\tau = O(s_\tau), \qquad a_\tau = G(s_\tau, o_\tau), \qquad z_\tau = C(s_\tau, o_\tau, a_\tau), \qquad s_{\tau+1} = T(s_\tau, o_\tau, a_\tau, z_\tau; f_\theta), \tag{2}$$

where o_τ is the observation, a_τ is the selected action, and z_τ is the latent input. The backend binds LRDSL primitives to predefined implementations and handles model forward passes, cache access, and tensor operations. Candidate policies must pass type, shape, dependency, and short execution checks before task evaluation.

As an example, consider a policy that performs a bounded latent phase before switching to discrete reasoning. At step τ, Observation extracts the predictive distribution p_τ and the number n_τ of completed latent steps. Construction uses the observed distribution to build a latent input:

$$z_\tau = \sum_{v \in \mathcal{V}} p_\tau(v) E_v, \tag{3}$$

where $\mathcal{V}$ is the vocabulary and E_v is the input embedding of token v. Gate uses n_τ to decide whether latent computation should continue, selecting continued latent computation while $n_\tau < N_{\max}$. Transition then feeds z_τ into the next model step. When the latent-step limit $N_{\max}$ is reached, the policy switches to discrete reasoning and preserves the accumulated context.

This decomposition assigns each algorithmic decision to an explicit component. Changing how a latent input is formed is a Construction edit, whereas changing when latent computation stops is a Gate edit, possibly together with an Observation edit that supplies a new signal. The same components also describe the edits and execution outcomes stored in the search history.

3.2 LLM-Guided Policy Design

This section describes how the design LLM, hereafter the proposer, uses LRDSL to propose a new policy. How each proposal is checked, evaluated, and accepted is described in Sec. 3.3.

Proposer inputs. At search round k, the proposer receives the current policy α_k, the LRDSL primitive library $\mathcal{L}$, and the search history $\mathcal{H}_k$. The library contains descriptions of the available primitives, their input–output signatures, and their composition rules. The history records earlier edits together with their measured outcomes and execution diagnostics. The proposer is also given the inference-cost budget B_{infer} that any accepted policy must satisfy and the maximum edit scope r of one round.

Edit-based proposals. Rather than writing a complete policy from scratch, the proposer outputs an LRDSL edit to the current policy:

$$\begin{aligned} \Delta_k &= \text{Propose}_{\text{LLM}}\left(\mathcal{L}, \alpha_k, \mathcal{H}_k; B_{\text{infer}}, r\right), \\ \widetilde{\alpha}_k &= \text{Apply}(\alpha_k, \Delta_k), \end{aligned} \tag{4}$$

where Δ_k is the proposed edit, $\text{Apply}(\alpha, \Delta)$ denotes the policy obtained by applying Δ to α, and $\widetilde{\alpha}_k$ is the candidate policy. An edit consists of operations on the primitives of α_k: replacing a primitive, changing its parameters, or recomposing type-compatible expressions. Together with Δ_k, the proposer returns a brief rationale describing the intended effect of the edit.

An edit may involve several components when they implement the same mechanism. For example, an entropy-based stopping rule requires Observation to compute the predictive entropy and Gate to use this value when deciding whether to continue latent computation. These coordinated changes form a single edit and are evaluated together. Because the current policy, the proposed edit, and the search history are all expressed with the same LRDSL components and primitives, the proposer can connect evaluation feedback to specific, editable reasoning mechanisms.

3.3 Semantic Local Search

A DSL representation does not by itself determine how a policy should be improved. We therefore evolve the policy by semantic local search. Starting from an initial policy α_0, each round proposes one edit to the current policy, evaluates the resulting candidate if it is valid, and keeps it only if it improves on the current policy.

Semantic neighborhoods. The neighborhood of a policy α is

$$\mathcal{N}_r(\alpha) = \{\alpha' = \mathrm{Apply}(\alpha, \Delta) \mid |\Delta| \le r,\ \mathrm{Valid}_{\mathrm{DSL}}(\alpha')\}, \tag{5}$$

where $|\Delta|$ counts the primitive nodes, parameter entries, and dependency connections changed by Δ, and $\mathrm{Valid}_{\mathrm{DSL}}$ denotes the type, shape, and dependency checks of Sec. 3.1. A candidate $\widetilde{\alpha}_k$ is *valid* if $\widetilde{\alpha}_k \in \mathcal{N}_r(\alpha_k)$ and it also passes the short execution checks.

Search update. At round k, the candidate $\widetilde{\alpha}_k$ from Eq. equation 4 is first checked for validity. An invalid candidate receives diagnostic feedback but no task score. A valid candidate is evaluated on the fixed development set $\mathcal{D}_{\mathrm{dev}}$, which returns its accuracy $\widehat{A}_{\mathrm{dev}}(\widetilde{\alpha}_k)$, its inference cost $\widehat{\mathrm{Cost}}_{\mathrm{dev}}(\widetilde{\alpha}_k)$, and execution diagnostics. Inference cost includes latent computation, discrete generation, and additional model calls, and all candidates are evaluated with the same settings. The candidate replaces the current policy if its cost is within B_{infer} and it either achieves higher development accuracy than α_k or matches its accuracy at lower inference cost; in all other cases, $\alpha_{k+1} = \alpha_k$. Every attempt, whether invalid, rejected, or accepted, is added to $\mathcal{H}_{k+1}$.

Search history. Each record in the history contains the parent policy α_k, the edit Δ_k, the proposer's rationale, the measured accuracy and cost if the candidate was evaluated, the execution diagnostics, and whether the candidate was accepted. The rationale is stored separately from the measured outcomes and is not treated as established causal evidence. Because records are expressed with the same LRDSL components and primitives as the policies, later proposals can compare related edits and use both successful and unsuccessful attempts to guide further edits.

Termination and output. The search budget B_{search} is the number of search rounds K, and every round counts toward it, including rounds with invalid candidates. The initial policy α_0 is required to be valid and within B_{infer}. After K rounds, the search returns the current policy $\alpha^\star = \alpha_K$. Let $\mathcal{P}_{\mathrm{eval}}$ denote α_0 together with all valid candidates evaluated during search. Because the current policy is replaced only by a candidate that is within budget and either more accurate or equally accurate and cheaper, α_K is the most accurate policy in $\mathcal{P}_{\mathrm{eval}}$ that satisfies the budget, with ties broken in favor of lower inference cost:

$$\alpha^\star = \alpha_K \in \underset{\substack{\alpha \in \mathcal{P}_{\mathrm{eval}} \\ \widehat{\mathrm{Cost}}_{\mathrm{dev}}(\alpha) \le B_{\mathrm{infer}}}}{\arg\max} \ \widehat{A}_{\mathrm{dev}}(\alpha). \tag{6}$$

The selected policy remains fixed during final testing, and the backbone parameters are never updated.

4 Experiments

We evaluate both the reasoning algorithms discovered with LRDSL and the representation used to search for them. We first compare reasoning accuracy, token usage, and generalization, then examine semantic alignment, search effectiveness, and component contributions.

4.1 Experimental Setup

Benchmarks and evaluation protocol. We evaluate Qwen3-1.7B and Qwen3-8B (Yang et al., 2025) on five reasoning benchmarks: GSM8K (Cobbe et al., 2021), MATH500 (Hendrycks et al., 2021; Lightman et al., 2023), AIME 2024 and AIME 2025 from the American Invitational Mathematics Examination (Mathematical Association of America, n.d.), and GPQA Diamond (Rein et al., 2023). We further evaluate cross-task transfer to HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) on Qwen3-1.7B, applying the algorithm discovered on reasoning tasks without additional search or tuning.

Baselines and metrics. We compare against CoT (Sampling) (Wei et al., 2022), Soft Thinking (Zhang et al., 2025b), SwiReasoning (SwiR) (Shi et al., 2025), Mixture of Inputs (MOI) (Zhuang et al., 2025), and SeLaR (Fu & Luo, 2026). For reasoning, we report final-answer accuracy and its mean across the five benchmarks. For code generation, we report Pass@1 (Chen et al., 2021) on each benchmark and their mean.

Implementation details. DeepSeek-V4.1-Flash (DeepSeek-AI, 2026) serves as the search-time proposer, generating and refining LRDSL programs, while target-model parameters remain frozen throughout search and evaluation. All methods use Hugging Face Transformers (Wolf et al., 2020) on

four NVIDIA RTX PRO 6000 GPUs. Baselines follow their official implementations and reported method-specific hyperparameters. We standardize discrete sampling settings to temperature 0.6, top-$p = 0.95$, top-$k = 20$, and min-$p = 0.0$.

4.2 Performance, Efficiency, and Generalization

Improving automatic algorithm design. LRDSL improves the average reasoning accuracy of the algorithms discovered by all three evaluated design systems. Table 1 compares human-designed algorithms with those discovered by Codex (OpenAI, 2025), Claude Code (Anthropic, 2025), and OpenEvolve (Sharma, 2025) through raw-code search and LRDSL-guided search. On Qwen3-1.7B and Qwen3-8B, adding LRDSL improves Codex's macro-average accuracy by 2.46 and 1.58 percentage points and Claude Code's by 1.45 and 2.45 points, respectively. The gains are largest for OpenEvolve: average accuracy increases from 58.44% to 63.21% on Qwen3-1.7B and from 75.56% to 81.21% on Qwen3-8B, yielding improvements of 4.77 and 5.65 points. The positive gains across all six system–backbone configurations support LRDSL as an effective interface for automatic algorithm design across the evaluated search systems.

Table 1: **Reasoning performance of human-designed and automatically discovered algorithms.** Accuracy (%); Avg. is the unweighted mean across the five benchmarks, computed from unrounded per-benchmark accuracies. Codex, Claude Code, and OpenEvolve denote raw-code search, and each indented + LRDSL row reports the corresponding LRDSL-guided system. Bold indicates the best result per column within each backbone, including ties.

Method	Qwen3-1.7B						Qwen3-8B					
	GSM 8K	MATH 500	GPQA Diamond	AIME 24	AIME 25	Avg.	GSM 8K	MATH 500	GPQA Diamond	AIME 24	AIME 25	Avg.
Human-designed algorithms												
CoT (Sampling)	90.07	92.00	39.39	50.00	33.33	60.96	95.45	**98.00**	61.62	76.67	66.67	79.68
Soft Thinking	89.46	91.00	33.83	36.67	36.67	57.53	94.92	95.80	57.58	70.00	66.67	76.99
MOI	90.60	92.00	38.38	33.33	33.33	57.53	95.75	96.40	57.58	66.67	56.67	74.61
SwiR	89.84	92.00	37.88	46.67	23.33	57.94	95.68	97.00	**62.63**	60.00	66.67	76.40
SeLaR	90.60	**92.60**	35.35	53.33	36.67	61.71	95.38	97.00	60.61	76.67	63.33	78.60
Automatically discovered algorithms												
Codex-6 Astra	89.84	91.80	36.36	43.33	36.67	59.60	95.68	96.60	57.58	73.33	66.67	77.97
+ **LRDSL (Ours)**	90.22	91.20	38.89	**56.67**	33.33	62.06	95.60	97.20	61.62	70.00	73.33	79.55
Claude Code Opus 5	88.70	88.00	30.81	46.67	40.00	58.84	95.30	96.60	57.58	70.00	66.67	77.23
+ **LRDSL (Ours)**	89.54	90.40	34.85	**56.67**	30.00	60.29	**96.06**	96.60	59.09	70.00	**76.67**	79.68
OpenEvolve	89.99	91.80	**40.40**	36.67	33.33	58.44	95.53	97.00	58.59	63.33	63.33	75.56
+ **LRDSL (Ours)**	**90.90**	91.40	**40.40**	46.67	**46.67**	**63.21**	95.30	96.80	60.61	**80.00**	73.33	**81.21**

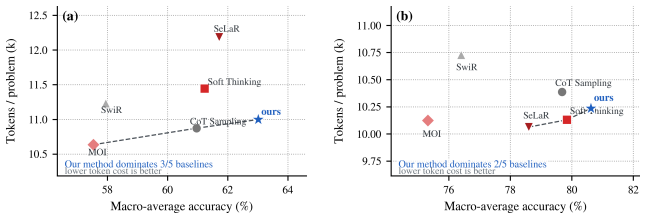

Figure 3: **Accuracy–token trade-off** on (a) Qwen3-1.7B and (b) Qwen3-8B. Ours denotes the strongest discovered policy (OpenEvolve + LRDSL). Lower right is better. Token counts include correct and incorrect answers but exclude search-time proposer tokens.

Table 2: **Code generation (Qwen3-1.7B).** Pass@1 (%); Ours is the strongest discovered policy (OpenEvolve + LRDSL), applied without additional search.

Method	HumanEval	MBPP	Avg.
CoT (Sampling)	79.88	70.04	74.96
CoT (Greedy)	71.95	70.82	71.39
Soft Thinking	77.44	68.48	72.96
SwiR	76.22	73.15	74.69
SeLaR	75.00	66.15	70.58
Ours	**80.49**	**73.93**	**77.21**

Comparison with human-designed algorithms. The improvements also translate into algorithms that exceed strong human-designed baselines. Without LRDSL, neither automatic design system surpasses the strongest evaluated human-designed baseline in macro-average accuracy on either

backbone. With LRDSL, OpenEvolve exceeds SeLaR on Qwen3-1.7B by 1.50 percentage points and CoT (Sampling) on Qwen3-8B by 1.53 points, achieving the highest average accuracy on both backbones. These gains are not uniform across benchmarks: most of OpenEvolve's improvement over raw-code search comes from AIME24 and AIME25. Moreover, although LRDSL improves Codex on both backbones, its Qwen3-8B result remains slightly below CoT (Sampling). Thus, LRDSL improves the observed outcomes of automatic design without guaranteeing superiority on every benchmark or in every configuration.

Code-generation performance. To examine applicability beyond mathematical and scientific reasoning, we evaluate code generation on Qwen3-1.7B. The discovered method achieves 80.49% Pass@1 on HumanEval and 73.93% on MBPP, exceeding all evaluated baselines on both benchmarks. Its average score reaches 77.21%, improving over CoT (Sampling), the strongest baseline by average performance, by 2.25 percentage points. These results extend the evaluation of LRDSL-based algorithm design to code generation.

Token efficiency. We next examine the inference-time generation length of the discovered algorithm alongside its accuracy. Figure 3 presents this trade-off for methods with available token measurements. Token usage is the macro-average generation length across benchmarks, including both correct and incorrect answers. Compared with CoT (Sampling), the discovered method uses slightly more tokens in panel (a), but fewer in panel (b). The token savings therefore depend on the evaluation setting, rather than reflecting a uniform reduction in generation length. This comparison measures generation length, not wall-clock latency, and excludes proposer tokens consumed during algorithm search.

4.3 Understanding Search Progress with LRDSL

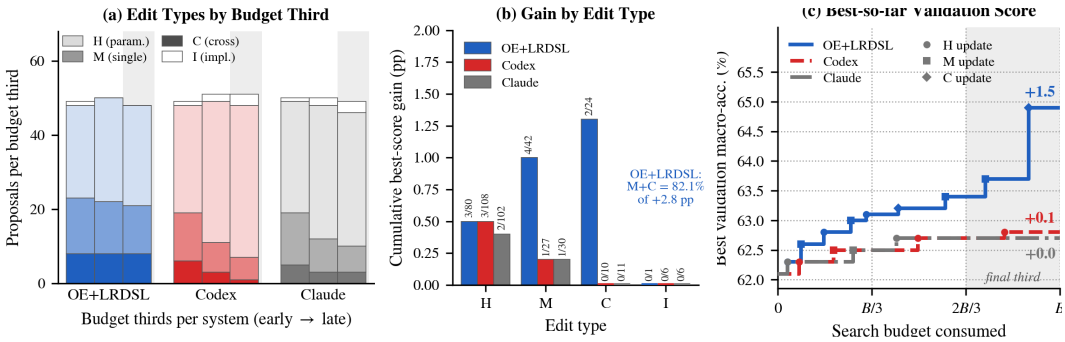

Figure 4: Search-log analysis on Qwen3-1.7B. Edit types: hyperparameter-only (H), single-component (M), cross-component/dependency (C), and implementation-only (I). (a) Proposals by edit type across budget thirds (early to late). (b) Cumulative best-score gain; labels indicate best updates/proposals. (c) Best-so-far validation macro-accuracy; markers indicate update types. Shading denotes the final third; numbers in (c) give its gains.

To understand how performance gains emerge, we compare search traces of OpenEvolve+LRDSL, Codex, and Claude on the frozen Qwen3-1.7B backbone under matched budgets. Scores are the mean validation accuracy across the five reasoning benchmarks.

What does continued search explore? We identify mechanism structures by the rules and dependencies of Observation, Gate, Construction, and Transition, excluding parameter-only and implementation-only differences. Figure 4(a) shows that OpenEvolve+LRDSL explores 50 new structures beyond the initial pool, compared with 29 for Codex and 28 for Claude, despite similar evaluation counts (147–151 per system). The coverage gap widens as search proceeds, suggesting that the difference concerns sustained mechanism exploration rather than the number of evaluated candidates alone.

Which edits accompany improvements? To connect exploration with performance, we label parent–child edits as hyperparameter (H), single-component (M), or cross-component/dependency (C) changes, tracking implementation-only edits separately. Structural labels allow accompanying parameter tuning. We count progress only when a candidate exceeds the previous best validation score. Figure 4(b) shows similar cumulative gains from H-labeled updates across systems (0.4–0.5 percentage points). The larger difference occurs at structural updates: M and C account for 1.0 and

1.3 points under OpenEvolve+LRDSL, together representing 82.1% of its 2.8-point improvement. Each code agent gains only 0.2 points at structural updates. Most structural proposals nevertheless do not improve the search best: additional progress is concentrated in a small number of successful mechanism changes.

How does late-stage search differ? Figure 4(c) distinguishes new structures, new parameter settings within known structures, and previously evaluated configurations, excluding planned reevaluations. In the final third of the budget, new structures account for 31.3% of OpenEvolve+LRDSL proposals, compared with 7.8% for Codex and 10.2% for Claude. The code agents instead predominantly explore new parameter settings within known structures (78.4% and 77.6%), rather than simply repeat identical configurations. Together, these illustrative patterns associate additional progress with sustained mechanism exploration and occasional larger structural updates, rather than parameter refinement alone.

4.4 Ablation Studies

Our ablations show that LRDSL is largely insensitive to the choice of LLM proposer, while the experience component improves search stability. We evaluate three proposers across five reasoning benchmarks and ablate the experience component on GSM8K, using Qwen3-1.7B as the target model throughout.

Table 3: Proposer ablation (Qwen3-1.7B). Avg. Acc. is the mean over the five benchmarks; Δ is the gap to the best proposer (pp).

LLM Proposer	Avg. Acc. (%) ↑	Δ
deepseek-flash-v4.1	**63.21**	—
GPT-5.6-Luna	63.17	−0.04
MiniMax-M3	63.09	−0.12

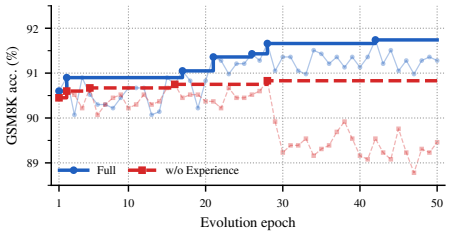

Figure 5: Experience ablation on GSM8K (Qwen3-1.7B). Thick lines: best so far; thin lines: per epoch.

Effect of LLM proposers. LRDSL achieves similar macro-averaged accuracy across the three tested proposers on GSM8K, MATH500, AIME 2024, AIME 2025, and GPQA Diamond (Table 3). Deepseek-flash-v4.1 (DeepSeek-AI, 2026), GPT-5.6-Luna (OpenAI, 2026), and MiniMax-M3 (Lai et al., 2026) achieve 63.21%, 63.17%, and 63.09%, respectively. The gap of only 0.12 percentage points between the best and worst results indicates that LRDSL's performance does not depend strongly on a particular proposer.

Effect of the experience component. The experience component substantially reduces accuracy fluctuations across the 50 evolution epochs on GSM8K (Figure 5). The full framework maintains accuracy between 89.00% and 90.90%, whereas removing experience widens this range to 60.00%–90.00%. The component also improves the best accuracy by 0.90 percentage points (90.90% vs. 90.00%). These results highlight its role in maintaining consistently strong performance during search.

5 Conclusion

We studied how to make reasoning algorithms explicit objects of LLM-driven auto-design, focusing on training-free latent reasoning. We identified semantic misalignment between the representation edited by the proposer and the underlying algorithmic decisions as a bottleneck for existing coding agents and evolutionary frameworks. To address this bottleneck, we proposed LRDSL, a typed domain-specific language, so that policy representations, semantic edits, execution feedback, and accumulated experience refer to the same algorithmic decisions. Across three design systems and two backbones, LRDSL-guided search improved the macro-average accuracy of the discovered algorithms in all six configurations. Search-log analysis further associates these gains with sustained exploration of new mechanism structures and a small number of successful structural updates, rather than parameter refinement alone. Future work includes extending LRDSL to other model families

and reasoning paradigms, allowing the proposer to introduce new primitives into the library, and applying semantically aligned design spaces to other forms of inference-time computation.

AI USE STATEMENT

Generative AI tools assisted with language editing and structural brainstorming. The authors reviewed and verified the manuscript, citations, implementation, and reported results and take responsibility for the final content.

ETHICS STATEMENT

This work uses public benchmark datasets and frozen language-model checkpoints. It does not involve human subjects, private data, or deployment decisions.

REPRODUCIBILITY STATEMENT

We will release the LRDSL grammar, compiler interface, search prompts, experience records, evaluation scripts, and raw metrics upon accepted. The release will also document the discovered policy programs, data splits, decoding settings, candidate budgets, and random seeds needed to reproduce both the search and the final evaluations.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning. *arXiv preprint arXiv:2507.19457*, 2025. doi: 10.48550/arXiv.2507.19457. URL https://arxiv.org/abs/2507.19457.

Anthropic. Claude 3.7 Sonnet and Claude Code. Anthropic Blog, feb 2025. URL https://www.anthropic.com/news/claude-3-7-sonnet. Accessed 2026-09-26.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program Synthesis with Large Language Models. *arXiv preprint arXiv:2108.07732*, 2021. doi: 10.48550/arXiv.2108.07732. URL https://arxiv.org/abs/2108.07732.

Lingjiao Chen, Jared Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. Are More LLM Calls All You Need? Towards the Scaling Properties of Compound AI Systems. In *Advances in Neural Information Processing Systems*, volume 37, pp. 45767–45790, 2024. doi: 10.52202/079017-1455. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/51173cf34c5faac9796a47dc2fdd3a71-Abstract-Conference.html.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*, 2021. doi: 10.48550/arXiv.2107.03374. URL https://arxiv.org/abs/2107.03374.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John

Schulman. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021. doi: 10.48550/arXiv.2110.14168. URL https://arxiv.org/abs/2110.14168.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the Limits of KV Cache Compression. Official model card, Hugging Face, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash. Accessed 2026-09-26.

DeepSeek-AI et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948*, 2025. doi: 10.48550/arXiv.2501.12948. URL https://arxiv.org/abs/2501.12948.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. DreamCoder: Bootstrapping Inductive Program Synthesis with Wake-Sleep Library Learning. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 835–850, 2021. doi: 10.1145/3453483.3454080. URL https://doi.org/10.1145/3453483.3454080.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-Referential Self-Improvement Via Prompt Evolution. In *International Conference on Machine Learning*, 2024. URL https://proceedings.mlr.press/v235/fernando24a.html.

Renyu Fu and Guibo Luo. SeLaR: Selective Latent Reasoning in Large Language Models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7073–7087. Association for Computational Linguistics, jul 2026. doi: 10.18653/v1/2026.acl-long.320. URL https://aclanthology.org/2026.acl-long.320/.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training Large Language Models to Reason in a Continuous Latent Space. In *Second Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2412.06769.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset. *arXiv preprint arXiv:2103.03874*, 2021. doi: 10.48550/arXiv.2103.03874. URL https://arxiv.org/abs/2103.03874.

Shengran Hu, Cong Lu, and Jeff Clune. Automated Design of Agentic Systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=t9U3LW7JVX.

Jie Huang and Kevin Chen-Chuan Chang. Towards reasoning in large language models: A survey. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 1049–1065, 2023. doi: 10.18653/v1/2023.findings-acl.67. URL https://aclanthology.org/2023.findings-acl.67/.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/f1cf02ce09757f57c3b93c0db83181e0-Paper-Conference.pdf.

Shengxin Kong, Liwen Xu, and Jingwen Fu. Improving Auto-Design of Neural PDE Solvers with a Domain-Specific Language. *arXiv preprint arXiv:2608.04384*, 2026. doi: 10.48550/arXiv.2608.04384. URL https://arxiv.org/abs/2608.04384.

Xunhao Lai, Weiqi Xu, Yufeng Yang, Qiaorui Chen, Yang Xu, Lunbin Zeng, Xiaolong Li, Haohai Sun, Haichao Zhu, Vito Zhang, Jinkai Hu, Jiayao Li, Rui Gao, Zekun Li, Songquan Zhu, Jingkai Zhou, and Pengyu Zhao. Minimax sparse attention. *arXiv preprint arXiv:2606.13392*, 2026.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's Verify Step by Step. *arXiv preprint arXiv:2305.20050*, 2023. doi: 10.48550/arXiv.2305.20050. URL https://arxiv.org/abs/2305.20050.

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. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32201–32223. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24bs.html.

Fangrui Lv, Lei Wang, Ruixin Hong, Yong Du, Xiangyu Wu, Tingting Gao, Guorui Zhou, and Changshui Zhang. Beyond Tokens: Dynamic Latent Reasoning via Semantic Residual Refinement. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 32383–32391, 2026. doi: 10.1609/aaai.v40i38.40513. URL https://ojs.aaai.org/index.php/AAAI/article/view/40513.

Mathematical Association of America. MAA Invitational Competitions. Official competition website, n.d. URL https://maa.org/maa-invitational-competitions/. Accessed 2026-09-26.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A Coding Agent for Scientific and Algorithmic Discovery. *arXiv preprint arXiv:2506.13131*, 2025. doi: 10.48550/arXiv.2506.13131. URL https://arxiv.org/abs/2506.13131.

OpenAI. Introducing Codex. OpenAI Blog, may 2025. URL https://openai.com/index/introducing-codex/. Accessed 2026-09-26.

OpenAI. Gpt-5.6: Frontier intelligence that scales with your ambition. https://openai.com/index/gpt-5-6/, July 2026. Accessed: 2026-09-26.

Esteban Real, Chen Liang, David R. So, and Quoc V. Le. AutoML-Zero: Evolving Machine Learning Algorithms From Scratch. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 8007–8019. PMLR, 2020. URL https://proceedings.mlr.press/v119/real20a.html.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. *arXiv preprint arXiv:2311.12022*, 2023. doi: 10.48550/arXiv.2311.12022. URL https://arxiv.org/abs/2311.12022.

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

Asankhaya Sharma. OpenEvolve: An open-source evolutionary coding agent. Open-source software, GitHub, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve. Accessed 2026-09-26.

Dachuan Shi, Abedelkadir Asi, Keying Li, Xiangchi Yuan, Leyan Pan, Wenke Lee, and Wen Xiao. SwiReasoning: Switch-Thinking in Latent and Explicit for Pareto-Superior Reasoning LLMs. *arXiv preprint arXiv:2510.05069*, 2025. doi: 10.48550/arXiv.2510.05069. URL https://arxiv.org/abs/2510.05069.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 10131–10165, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b623663fd9b874366f3ce019fdfdd44-Abstract-Conference.html.

Niki van Stein and Thomas Bäck. LLaMEA: A Large Language Model Evolutionary Algorithm for Automatically Generating Metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29 (2):331–345, 2025. doi: 10.1109/TEVC.2024.3497793. URL https://doi.org/10.1109/TEVC.2024.3497793.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022. URL https://proceedings.neurips.cc/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45. Association for Computational Linguistics, oct 2020. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6/.

Junhong Wu, Jinliang Lu, Zixuan Ren, Gangqiang Hu, Zhi Wu, Dai Dai, and Hua Wu. LLMs are Single-Threaded Reasoners: Demystifying the Working Mechanism of Soft Thinking. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0895de150660b6c2aef9b0f098cb1dcd-Abstract-Conference.html.

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. SoftCoT: Soft Chain-of-Thought for Efficient Reasoning with LLMs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23336–23351. Association for Computational Linguistics, 2025a. doi: 10.18653/v1/2025.acl-long.1137. URL https://aclanthology.org/2025.acl-long.1137/.

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. SoftCoT++: Test-Time Scaling with Soft Chain-of-Thought Reasoning. *arXiv preprint arXiv:2505.11484*, 2025b. doi: 10.48550/arXiv.2505.11484. URL https://arxiv.org/abs/2505.11484.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025. doi: 10.48550/arXiv.2505.09388. URL https://arxiv.org/abs/2505.09388.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Advances in Neural Information Processing Systems*, volume 36,

2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.

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. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1381. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/4ced59d480e07d290b6f29fc8798f195-Abstract-Conference.html.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating Agentic Workflow Generation. In *International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=z5uVAKwmjf.

Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, Shuohang Wang, Yelong Shen, and Xin Eric Wang. Soft Thinking: Unlocking the Reasoning Potential of LLMs in Continuous Concept Space. *arXiv preprint arXiv:2505.15778*, 2025b. doi: 10.48550/arXiv.2505.15778. URL https://arxiv.org/abs/2505.15778.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Zican Dong, Yupeng Hou, Beichen Zhang, Yingqian Min, Junjie Zhang, Peiyu Liu, Xiaolei Wang, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Yiwen Hu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *Frontiers of Computer Science*, 20(12): 2012627, 2026. doi: 10.1007/s11704-026-60308-3. URL https://doi.org/10.1007/s11704-026-60308-3.

Yufan Zhuang, Liyuan Liu, Chandan Singh, Jingbo Shang, and Jianfeng Gao. Mixture of Inputs: Text Generation Beyond Discrete Token Sampling. In *Advances in Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=l6C6Pw30Gl.