RECURSIVE SELF-IMPROVEMENT AGENT FOR ADAPTIVE SYMBOLIC PRUNING STRATEGY DISCOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Post-training pruning is indispensable for deploying large language models (LLMs), and the importance metric that ranks weights largely determines the quality of the sparsified model. Hand-crafted metrics such as Wanda and SparseGPT encode fixed human intuition and do not adapt to new architectures or high-sparsity regimes, while automated symbolic discovery methods such as Pruner-Zero run genetic programming (GP) with frozen hyperparameters and a frozen evaluation pipeline, which leads to premature convergence and a prohibitive GPU cost per candidate. We propose **PrunerAgent**, an agent-based recursive self-improvement (RSI) framework for symbolic pruning-metric discovery. PrunerAgent nests two loops: an inner GP loop evolves symbolic trees over weights, activations, and gradients, and an outer meta-agent monitors population statistics and adapts the GP hyperparameters, compute budget, seeding strategy, grammar constraints, and validation thresholds of the inner loop. The search process itself, not only its output, is therefore improved during the run. Two further components reduce cost: an agent-managed symbolic search space that canonicalizes expressions and prunes theoretically defective subexpressions before evolution, and a six-stage validation pipeline that combines symbolic reasoning with progressively more expensive proxy evaluations so that only promising candidates reach full GPU evaluation. On LLaMA-1/2/3 and Mistral models at 50–80% unstructured and 4:8/2:4 semi-structured sparsity, PrunerAgent consistently outperforms Pruner-Zero and hand-crafted baselines while reducing search cost by 62%.

1 INTRODUCTION

Large language models (LLMs) achieve strong results across natural language tasks (Achiam et al., 2023; Touvron et al., 2023a), but their parameter counts impose heavy compute and memory costs at deployment. Post-training pruning removes weights without retraining (Frantar & Alistarh, 2023; Sun et al., 2023), and the importance metric that ranks weights is the central design decision. Hand-crafted metrics such as Magnitude (Han et al., 2015), Wanda (Sun et al., 2023), and SparseGPT (Frantar & Alistarh, 2023) encode a fixed human hypothesis about which weights matter; they cannot adapt to a new architecture or to extreme sparsity. This motivates automatically discovering metrics tailored to a given model and sparsity target. As Alan Kay put it, "the best way to predict the future is to invent it": rather than designing metrics by hand, we let the search process invent them.

Pruner-Zero (Dong et al., 2024) takes a first step in this direction by using genetic programming (GP) to search for symbolic metrics expressed as algebraic functions of weight W, activation A, and gradient G. However, vanilla GP has three structural limitations. *(i) The search pipeline is frozen.* Mutation rate, population size, and budget are fixed before the run, so the search cannot recover once diversity collapses or fitness plateaus. *(ii) The search space is rigid.* The grammar is static, and many sampled expressions are algebraically redundant, numerically unstable, or inconsistent with what is known about loss perturbation. *(iii) Evaluation is expensive.* Every candidate requires pruning and evaluating an LLM, although most candidates are poor. The central question we address is therefore: *how can we discover high-quality symbolic pruning metrics efficiently and adaptively, such that they generalize across models and sparsity levels?*

We answer this question with PrunerAgent, an agent-based recursive self-improvement (RSI) framework. Most agentic self-refinement methods (Madaan et al., 2023; Shinn et al., 2023) iteratively cri-

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.

tique and rewrite a *solution* while the search algorithm stays fixed. RSI instead optimizes the *search algorithm*: an inner loop solves the task and produces trajectories, and an outer loop uses those trajectories to modify how the inner loop searches (Schmidhuber, 2007). In PrunerAgent, the inner loop is GP over symbolic pruning trees, and the outer loop is a meta-agent that reads population statistics from a replay buffer and adapts GP hyperparameters, compute budget, seeding strategy, grammar constraints, and validation thresholds. Two further components make the search cheap. An agent-managed symbolic search space canonicalizes expressions, enforces grammar constraints, and prunes subexpressions that fail a theoretical screening before they are evolved. A six-stage validation pipeline places static checks, symbolic reasoning with an LLM and SymPy, deduplication, and a lightweight proxy evaluation in front of full perplexity and zero-shot evaluation, so that GPU time is spent only on candidates that survive the cheap stages.

We evaluate PrunerAgent on LLaMA-1 7B/13B, LLaMA-2 7B/13B, LLaMA-3 8B, and Mistral 7B at 50%, 70%, and 80% unstructured sparsity and at 4:8 and 2:4 semi-structured sparsity. PrunerAgent consistently outperforms Pruner-Zero and the hand-crafted baselines Magnitude, SparseGPT, Wanda, and RIA. On LLaMA-2 7B at 50% sparsity it improves LM Harness accuracy over Pruner-Zero by $+1.62$ points; on LLaMA-3 8B it lowers WikiText perplexity from 9.23 to 8.51 (-7.8% relative). At 80% sparsity the gap widens to 44.47% versus 40.17% accuracy. PrunerAgent also cuts GPU search time by 62%: of 350 candidate trials, only 133 reach full evaluation. Ablations show that the RSI meta-agent is the largest contributor to metric quality, while symbolic reasoning and staged evaluation are the main contributors to the cost reduction.

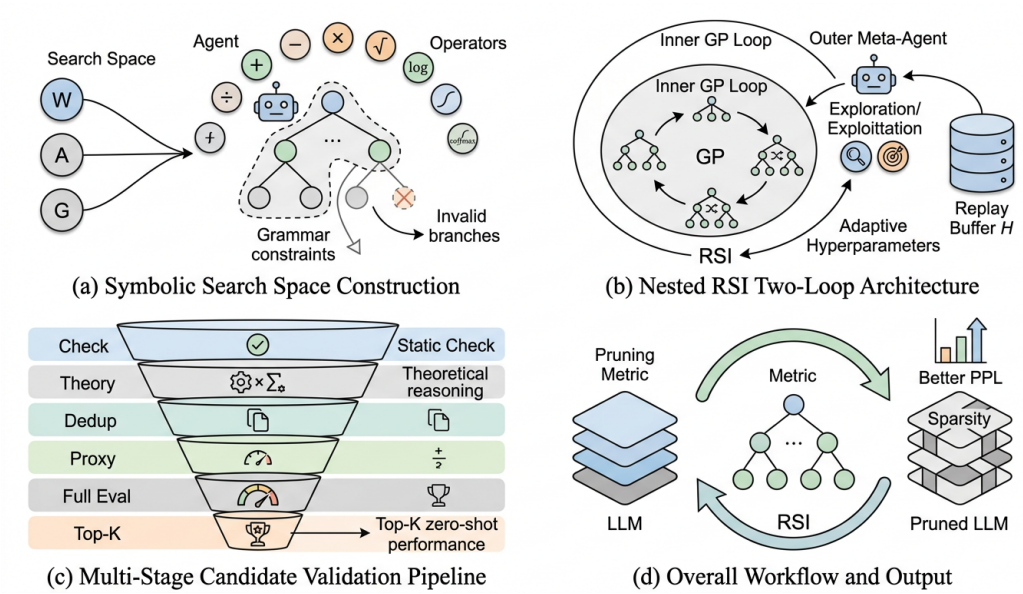
Our contributions are fourfold. (1) We formalize a nested two-loop RSI mechanism for symbolic pruning discovery and show how it differs from self-refinement (Section 3). (2) We design an agent-managed symbolic search space with grammar constraints, canonicalization, and theoretical pre-screening, and we prove that a large class of expressions is mask-equivalent and can be collapsed (Appendix B). (3) We develop a six-stage validation pipeline that filters defective candidates before GPU evaluation. (4) We validate PrunerAgent across model families and sparsity regimes, demonstrating better metrics at lower search cost than Pruner-Zero and recent LLM-driven pruning methods.

2 Related Work

LLM pruning Post-training pruning ranks weights by an importance score. Magnitude (Han et al., 2015), Wanda (Sun et al., 2023), SparseGPT (Frantar & Alistarh, 2023), and RIA (Zhang et al., 2024) are hand-designed and fixed. Layer-wise allocation methods such as OWL (Yin et al., 2024) and DSA (Li et al., 2024) decide *how much* to prune per layer but keep the metric form fixed. All of these metrics can be read as approximations of the loss perturbation caused by removing a weight (Hassibi et al., 1993; Molchanov et al., 2019); we use this view to screen candidates (Appendix B).

Automated discovery of pruning strategies. Pruner-Zero (Dong et al., 2024) discovers symbolic metrics with GP, following the tradition of symbolic algorithm discovery (Koza, 1992; Real et al., 2020; Chen et al., 2023), but its static configuration causes premature convergence. Self-Pruner (Huang et al., 2025) evolves layer-wise pruning rates rather than the metric. LLM-Guider (Hasiura et al., 2026) optimizes symbolic formulas with language guidance through a fixed pipeline. PPF (Ma et al., 2025) predicts performance and selects pruning actions at the policy level. AutoPrune (Kang et al., 2026) lets an LLM design pruning algorithms via graph-driven chain-of-thought, but treats the LLM as a one-shot generator. LLM-driven evolutionary search more broadly (Romera-Paredes et al., 2024; Novikov et al., 2025) uses the LLM as a mutation operator, not as a controller of the search. PrunerAgent differs on three axes: it adapts the search process online through a nested RSI loop, it prunes the search space with symbolic reasoning before GPU evaluation, and it validates candidates through progressively more expensive stages. In short, AutoPrune shows what an LLM can design; PrunerAgent shows how the design process itself can recursively self-improve.

Agents and recursive self-improvement. LLM agents are used for tool use, code generation, and scientific discovery (Wang et al., 2024). Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) refine outputs but leave the search algorithm unchanged. Recursive self-improvement (Schmidhuber, 2007) denotes nested optimization in which an outer loop improves the algorithm run by the inner loop; recent bounded forms apply it to meta-optimization and algorithm discovery (Zhang et al., 2025; Hu et al., 2024). Adaptive parameter control in evolutionary computation (Eiben



(a) Symbolic Search Space Construction (b) Nested RSI Two-Loop Architecture

(c) Multi-Stage Candidate Validation Pipeline (d) Overall Workflow and Output

Figure 1: Overview of PrunerAgent. (a) Symbolic search space: the agent combines terminals $\{W, A, G\}$ with an extended operator set under grammar constraints, canonicalizes expressions, and prunes theoretically defective branches. (b) Nested RSI architecture: the inner GP loop evolves symbolic pruning metrics; the outer meta-agent reads statistics from the replay buffer $\mathcal{H}$ and sends adapted hyperparameters, grammar constraints, and validation thresholds back to the inner loop. (c) Six-stage validation pipeline: static check, symbolic reasoning, deduplication, proxy evaluation, full evaluation, and top-K zero-shot evaluation, with cost increasing from left to right. (d) Workflow: starting from an LLM, PrunerAgent discovers a metric, applies it, and returns a pruned LLM with lower perplexity at the target sparsity.

et al., 1999) is a classical precursor. PrunerAgent instantiates bounded RSI for symbolic pruning discovery, with a fixed meta-policy class but adaptive control of every component of the inner search.

3 Methodology

PrunerAgent addresses the three failure modes of static GP with three coupled components (Figure 1): an agent-managed symbolic search space (Section 3.2) addresses the rigid grammar, a nested two-loop RSI architecture (Section 3.3) addresses the frozen pipeline, and a multi-stage validation pipeline (Section 3.4) addresses the evaluation cost. We first formalize the problem.

3.1 Problem Formulation

We formalize pruning-metric discovery as a symbolic search problem. Consider a linear layer with weights $\mathbf{W} \in \mathbb{R}^{m\times n}$ and, for a calibration set $\mathcal{D} = \{x^{(i)}\}_{i=1}^{N}$, input activations $\mathbf{X} \in \mathbb{R}^{n\times d}$ over d tokens. Let $A_j = \|X_j\|_2$ denote the norm of the j-th input channel and G_{ij} the gradient of the calibration loss with respect to W_{ij}. These three quantities capture complementary evidence about a weight's importance: W measures intrinsic magnitude, A measures how strongly the corresponding input channel is activated, and G measures sensitivity to loss perturbation. A symbolic expression tree T over terminals $\{W, A, G\}$ and an operator set $\mathcal{O}$ defines the importance score

$$S_{ij} = f_T(W_{ij}, A_j, G_{ij}). \tag{1}$$

Given S, we prune neuron-wise by keeping the top-scoring weights in each output row up to the target sparsity. The discovery problem is therefore to find the tree T whose induced mask minimizes the perplexity gap to the dense model.

The search process as the object of optimization. Let $\mathcal{A}_\Theta$ denote a search algorithm with configuration Θ. Self-refinement fixes $\mathcal{A}_\Theta$ and repeatedly rewrites a candidate T, so the algorithm that

produces candidates never changes. Recursive self-improvement instead runs $\mathcal{A}_{\Theta_k}$ to generate candidates and trajectories, then updates $\Theta_k \to \Theta_{k+1}$ from those trajectories, so that the algorithm itself improves as a by-product of its own search. This distinction is what motivates our framework: rather than only refining the metric, we refine the process that discovers it. In PrunerAgent, $\mathcal{A}_\Theta$ is genetic programming and

$$\Theta = \{\mu, N_{\text{pop}}, d_{\max}, B, Q_{\text{th}}, q, r\} \tag{2}$$

collects the mutation rate μ, population size N_{pop}, maximum tree depth $d_{\max}$, per-iteration evaluation budget B, theoretical-screening threshold Q_{th}, proxy keep fraction q, and reseeding fraction r. Every element of Θ is under meta-agent control, which allows the search to adapt its exploration strategy, its evaluation criteria, and its search-space constraints to the observed search state rather than committing to a fixed configuration in advance.

3.2 Agent-Managed Symbolic Search Space

Many GP-sampled expressions are redundant, unstable, or theoretically implausible, and each one that reaches GPU evaluation wastes budget. We therefore let an agent construct and constrain the search space before and during evolution, shifting effort from evaluating bad candidates to generating good ones.

Extended operator set. Pruner-Zero uses 17 primitives (13 unary, 4 binary). We extend this to 28 by adding competitive normalization (Softmax, Log-Sum-Exp), polarity (Sign), adaptive selection (Max, Min), near-zero amplification (Reciprocal), outlier suppression (Exp-Decay), and higher-order nonlinearity (Power-3, Sinh, Sin, Cos). Every original metric remains expressible, so the extension is strictly more expressive. The new operators matter because the original set can only form pointwise functions of $|W|$ and A, while high-sparsity pruning failures—outlier activations, near-zero weights that still carry signal, and layer-dependent competition—require relative or competitive operations. Appendix E lists the full set with rationale.

Grammar constraints. The grammar $\mathcal{G}$ bounds tree depth by $d_{\max}$, requires at least one W-dependent leaf, forbids constant trees, and wraps domain-restricted operators (Logarithm, Square Root, Division) so they are defined on the full input range. A constant tree yields a degenerate mask, a tree without a W leaf cannot rank weights, and an unwrapped domain-restricted operator produces NaNs that corrupt the mask. Enforcing these constraints at generation time is cheaper than detecting them later and keeps the search focused on valid candidates.

Theoretical pre-screening. For a candidate subtree T', the agent computes

$$Q(T') = w_1 \,\text{Mono}(T') + w_2 \,\text{Stab}(T') + w_3 \,\text{Align}(T'), \tag{3}$$

where Mono measures monotonicity of the score in $|W|$, Stab measures numerical stability under scale and outlier stress tests, and Align measures rank agreement with the first-order loss-perturbation score $W_{ij}^2 A_j^2$ (LeCun et al., 1990; Sun et al., 2023). Subtrees with $Q(T') < Q_{\text{th}}$ are excluded from the building-block pool. Appendix B gives formal definitions. The purpose of this stage is to reject expressions with inherent mathematical defects before they consume GPU time; it does not claim to bound generalization, and a candidate that passes screening is still evaluated on the real model. The advantage is that the screening is cheap, runs on CPU, and removes a large class of candidates that would otherwise fail for predictable reasons.

Canonicalization. The agent maps each tree to a canonical algebraic form using SymPy simplification (Meurer et al., 2017) and strips outermost operators that are strictly increasing, since such wrappers do not change the neuron-wise mask (Proposition 1, Appendix B). Trees with the same canonical hash are evaluated once. This is important because GP repeatedly rediscovers algebraically equivalent expressions, and without deduplication the same metric would be evaluated many times. Because the grammar and Q_{th} are part of Θ, the meta-agent loosens them during exploration and tightens them around high-performing subtrees during exploitation, so the search space itself adapts to the stage of the search.

3.3 Nested RSI Two-Loop Architecture

The two loops communicate through a replay buffer $\mathcal{H}$ that stores, for every evaluated candidate, the tree T, its canonical form, proxy and full fitness, the screening scores, structural features (depth,

node count, operator histogram), and per-iteration population statistics. The buffer is what makes recursion possible: the outer loop can only improve the inner loop if it can observe what the inner loop has done, and the inner loop can only benefit from past experience if that experience is stored in a form that supports seeding and diversity control.

Inner loop: GP evolution. Given Θ_k, the inner loop (i) seeds a population from $\mathcal{H}$ with a mixture of elite and diverse historical candidates, (ii) applies tournament selection, crossover, and mutation under grammar $\mathcal{G}$, (iii) passes every offspring through the validation pipeline of Section 3.4, and (iv) writes survivors and statistics back to $\mathcal{H}$:

$$\mathcal{P}_{k+1} = \mathrm{GP}\left(\mathcal{P}_k, \Theta_k, \mathcal{H}\right). \tag{4}$$

Seeding from $\mathcal{H}$ instead of from scratch is a key advantage: it lets the search reuse high-quality trees and maintain diversity without recomputing them, which is especially valuable under a limited budget.

Feedback statistics. After each iteration the meta-agent reads a state $s_k = (D_k, F_k^*, \tau_k, b_k)$: population diversity, best fitness so far, a stagnation counter (generations without improvement), and the fraction of budget consumed. Diversity is the mean normalized tree edit distance (Zhang & Shasha, 1989),

$$D(\mathcal{P}) = \frac{2}{N(N-1)} \sum_{i<j} \frac{\mathrm{TED}(T_i, T_j)}{\max(|T_i|, |T_j|)} \in [0, 1]. \tag{5}$$

These four statistics are deliberately compact. They are sufficient to detect the two failure modes that static GP cannot escape—diversity collapse and fitness stagnation—and they are cheap to compute relative to model evaluation, so the meta-agent does not become a bottleneck.

Meta-improvement. The meta-agent is a hybrid controller $\mathcal{M}$: an LLM proposes an update from a summary of $\mathcal{H}$, and a rule layer projects the proposal onto an admissible box and enforces two regimes (rules in Appendix C). In *exploration mode*, triggered when $\tau_k > \tau_{\text{th}}$ or $D_k < D_{\text{th}}$, it raises μ, enlarges N_{pop}, reseeds part of the population from diverse entries of $\mathcal{H}$, relaxes Q_{th} and the grammar, and raises the proxy keep fraction q so that more novel structures reach evaluation. In *exploitation mode*, triggered when fitness improves steadily and diversity is sufficient, it lowers μ, shrinks N_{pop}, tightens Q_{th}, and concentrates budget on the neighbourhood of high-fitness candidates. The update is

$$\Theta_{k+1} = \mathcal{M}(\Theta_k, s_k, \mathcal{H}_k). \tag{6}$$

Because Θ includes the grammar constraints and validation thresholds, the meta-agent co-designs the search space and the evaluation criteria together with the exploration strategy. This is what makes the loop recursive: data produced by the inner search changes how the inner search is conducted, whereas static GP cannot recover from diversity collapse.

3.4 Multi-Stage Candidate Validation Pipeline

Every candidate passes through six stages of increasing cost, so that most are rejected before any GPU work. This ordering turns a per-candidate GPU cost into a small expected cost and makes the search affordable. **(1) Static check.** No GPU cost: validate syntax and grammar, reject constant or W-independent trees. **(2) Symbolic reasoning.** LLM + SymPy: infer monotonicity, domain issues, scale invariance, and alignment with loss perturbation; reject if $Q(T) < Q_{\text{th}}$. **(3) Canonicalization and deduplication.** Map to canonical form and skip candidates whose hash is already in $\mathcal{H}$. **(4) Proxy evaluation.** Prune with a small calibration subset, score a short held-out slice, and keep the top fraction q. **(5) Full evaluation.** Prune with the full calibration set and evaluate WikiText-2 perplexity; store in $\mathcal{H}$. **(6) Zero-shot evaluation.** Evaluate only the top-K candidates on downstream benchmarks, deferred to the end of the search.

Stages 1–3 cost no GPU time, and Stage 4 is roughly an order of magnitude cheaper than Stage 5. Proposition 3 (Appendix B) gives the expected cost per candidate. The main risk is a false negative at Stage 4; we mitigate it by making the keep fraction q a meta-agent action that is raised in exploration mode and by re-evaluating borderline candidates with a larger proxy set before rejection.

Algorithm 1 PrunerAgent: Nested RSI for Symbolic Pruning Metric Discovery

Input: Model f, calibration data $\mathcal{D}$, target sparsity s, iterations K
Output: Best pruning metric T^*
Initialize replay buffer $\mathcal{H}$, configuration Θ_0, operator set $\mathcal{O}$, grammar $\mathcal{G}$
for $k = 0, \ldots, K-1$ **do**
  $\mathcal{P}_k \leftarrow \textsc{Seed}(\mathcal{H}, \Theta_k)$ ▷ elite + diverse (Alg. 2)
  $\{T_i\} \leftarrow \text{GP}(\mathcal{P}_k, \Theta_k, \mathcal{O}, \mathcal{G})$
  for each candidate T_i **do**
    ok, fit $\leftarrow \textsc{Validate}(T_i, \Theta_k)$ ▷ six stages (Alg. 4)
    if ok **then**
      Store $(T_i, \text{canon}(T_i), \text{fit}, Q(T_i), \text{features})$ in $\mathcal{H}$
    end if
  end for
  $s_k \leftarrow \textsc{Stats}(\mathcal{H})$ ▷ D_k, F_k^*, τ_k, b_k
  $\Theta_{k+1} \leftarrow \mathcal{M}(\Theta_k, s_k, \mathcal{H})$ ▷ meta-agent (Alg. 3)
end for
return $T^* = \arg\min_{T \in \mathcal{H}} \text{PPL}(T)$

3.5 Overall Algorithm

Algorithm 1 summarizes PrunerAgent. It returns the best fully evaluated tree in the append-only buffer $\mathcal{H}$, so the result is never worse than any intermediate configuration (Proposition 2). This elitist property makes aggressive exploration safe: the meta-agent can raise mutation, reseed, or relax screening without risking final quality. Appendix D expands each subroutine.

4 Experiments

4.1 Implementation

We evaluate on LLaMA-1 (Touvron et al., 2023a), LLaMA-2 (Touvron et al., 2023b), LLaMA-3 8B (Meta, 2024), and Mistral 7B (Jiang et al., 2023). We report mean zero-shot accuracy on seven LM Harness tasks (Gao et al., 2023), WikiText-2 perplexity (Merity et al., 2016), GSM8K (Cobbe et al., 2021), and MMLU (Hendrycks et al., 2020). Baselines are Magnitude (Han et al., 2015), SparseGPT (Frantar & Alistarh, 2023), Wanda (Sun et al., 2023), RIA (Zhang et al., 2024), Pruner-Zero (Dong et al., 2024), Self-Pruner (Huang et al., 2025), LLM-Guider (Hasiura et al., 2026), PPF (Ma et al., 2025), and AutoPrune (Kang et al., 2026). We prune neuron-wise at 50%, 70%, and 80% unstructured sparsity and at 4:8 and 2:4 semi-structured sparsity. PrunerAgent uses LiteLLM (LiteLLM Team, 2026) for model routing, GPT-5.4 (OpenAI, 2026) with high reasoning effort for meta-agent reasoning and symbolic property inference, GPT-5.4-mini for lightweight triage in the deduplication and proxy stages, DeepSeek-V4-Flash (DeepSeek, 2026) as an alternative low-cost backend, SymPy for simplification, and SQLite for the replay buffer. Each search issues 350 candidate trials with initial population 64, crossover rate 0.9, initial mutation rate 0.15, tournament size 5, and maximum depth 8; the meta-agent adapts these within the bounds of Appendix C. Further details are in Appendix H.

4.2 Main Results

Table 1 reports results at 50% unstructured sparsity. PrunerAgent attains the best accuracy and perplexity on every model. On LLaMA-2 7B and 13B it improves LM Harness accuracy over Pruner-Zero by $+1.62$ and $+1.15$ points. The gains are largest on LLaMA-3 and Mistral ($+2.13$ and $+1.97$), where static metrics degrade most, which indicates that the discovered metrics capture architecture-specific importance patterns. On LLaMA-3 8B, perplexity drops from 9.23 to 8.51 (-7.8%). On GSM8K and MMLU the improvements are largest on reasoning-intensive settings, suggesting that the discovered metrics better preserve high-level capabilities.

Table 1: Mean zero-shot accuracy (%) on 7 LM Harness tasks, WikiText perplexity, GSM8K, and MMLU accuracy (%) at 50% unstructured sparsity. Best pruned results are **bold**.

Method	Weight Update	Sparsity	LLaMA-1		LLaMA-2		LLaMA-3		Mistral	
			7B	13B	7B	13B	8B	8B-Inst	7B	7B-Inst
LM Harness Accuracy (%)										
Dense	-	0%	59.70	62.58	59.72	63.03	64.21	64.15	60.06	66.69
Magnitude	✗	50%	46.89	47.34	52.40	52.90	44.87	45.31	57.24	63.34
SparseGPT	✓	50%	54.86	58.54	55.90	60.70	53.87	55.89	57.49	62.46
Wanda	✗	50%	54.08	59.18	55.89	60.88	49.66	51.34	54.20	61.04
RIA	✗	50%	55.10	59.45	55.67	61.03	50.76	50.64	54.39	60.48
Pruner-Zero	✗	50%	55.10	59.73	57.47	61.42	55.50	55.94	59.33	63.51
PrunerAgent	✗	50%	**56.73**	**60.88**	**59.09**	**62.57**	**57.63**	**58.07**	**61.30**	**65.12**
WikiText Perplexity										
Dense	-	0%	5.37	4.80	5.04	4.56	5.80	7.91	5.23	4.90
Magnitude	✗	50%	13.27	13.55	11.96	6.16	73.93	5.5e2	7.14	6.59
SparseGPT	✓	50%	6.92	5.87	6.59	5.72	10.89	13.27	6.42	7.02
Wanda	✗	50%	6.90	5.82	6.47	5.64	10.57	16.37	7.24	7.22
RIA	✗	50%	6.81	5.83	6.43	5.63	12.56	15.57	7.27	7.21
Pruner-Zero	✗	50%	6.78	5.74	6.35	5.51	9.23	11.37	6.22	6.55
PrunerAgent	✗	50%	**6.31**	**5.42**	**5.97**	**5.18**	**8.51**	**9.86**	**5.79**	**5.97**
GSM8K Accuracy (%)										
Dense	-	0%	11.07	17.82	14.59	19.86	52.39	74.45	40.11	47.76
Magnitude	✗	50%	1.52	5.99	2.05	6.22	1.97	1.29	15.53	27.37
SparseGPT	✓	50%	8.19	15.60	8.11	13.42	21.46	49.20	25.40	33.97
Wanda	✗	50%	7.96	11.52	7.43	9.10	10.16	32.68	22.74	33.59
Pruner-Zero	✗	50%	8.22	15.62	8.47	15.03	43.07	52.15	25.78	35.14
PrunerAgent	✗	50%	**9.87**	**17.09**	**10.31**	**16.74**	**47.52**	**56.28**	**28.45**	**38.72**
MMLU Accuracy (%)										
Dense	-	0%	35.28	46.98	41.97	51.47	65.23	66.35	58.92	62.54
Magnitude	✗	50%	26.24	30.12	26.04	43.83	4.36	12.03	50.83	49.52
SparseGPT	✓	50%	29.48	38.29	33.03	47.14	49.50	52.27	50.95	52.04
Wanda	✗	50%	29.81	37.84	32.09	48.06	49.05	53.15	53.05	53.62
Pruner-Zero	✗	50%	31.05	39.76	33.06	48.38	51.22	55.60	53.87	54.36
PrunerAgent	✗	50%	**33.12**	**42.08**	**35.17**	**50.41**	**54.35**	**58.72**	**56.29**	**57.13**

4.3 High Sparsity and Semi-Structured Pruning

Table 2 reports LLaMA-2 7B at 70% and 80% sparsity and under 4:8 and 2:4 constraints. At 70% and 80%, PrunerAgent improves LM Harness accuracy over Pruner-Zero by 2.80 and 4.30 points; the widening gap is consistent with the claim that static GP suffers most from premature convergence in the hardest regimes and that the RSI loop mitigates this. Under semi-structured constraints, the discovered metric transfers well: PrunerAgent reaches perplexity 7.41 and GSM8K 5.82% at 4:8 and 9.89 / 4.16% at 2:4, the best in each setting.

Table 2: LLaMA-2 7B under (a) high unstructured sparsity and (b) semi-structured N:M sparsity.

(a) High Sparsity

Method	Sparsity	ACC (%)	PPL
SparseGPT	70%	47.85	9.18
Wanda	70%	47.10	9.45
Pruner-Zero	70%	49.26	8.73
PrunerAgent	70%	**52.06**	**7.95**
SparseGPT	80%	38.82	13.95
Wanda	80%	38.15	14.28
Pruner-Zero	80%	40.17	12.84
PrunerAgent	80%	**44.47**	**11.02**

(b) Semi-Structured Pruning

Method	Sparsity	PPL	GSM8K (%)
SparseGPT	4:8	7.89	4.84
Wanda	4:8	8.01	3.10
Pruner-Zero	4:8	7.72	4.95
PrunerAgent	4:8	**7.41**	**5.82**
SparseGPT	2:4	10.38	3.10
Wanda	2:4	11.31	2.75
Pruner-Zero	2:4	10.34	3.41
PrunerAgent	2:4	**9.89**	**4.16**

4.4 Comparison with Search Algorithms and Automatic Discovery Methods

Table 3 compares PrunerAgent with generic search algorithms and with recent automatic pruning methods on LLaMA-2 7B at 50% sparsity, reporting GPU search time, perplexity, and accuracy. Random Search, TPE, and QMC perform worst, confirming that structured evolutionary search matters. Among automatic methods, AutoPrune is the strongest baseline, but PrunerAgent still improves on it by 1.37 accuracy points and 0.33 perplexity while using 4.8 GPU hours versus 7.9, and versus 12.7 for Pruner-Zero (a 62% reduction). The efficiency gain comes from the validation pipeline: only 133 of 350 trials reach full evaluation.

Table 3: Search algorithms and automatic pruning methods on LLaMA-2 7B (50% sparsity).

Method	GPU Time (h)	WikiText PPL	LM Harness (%)
Search Algorithms			
Random Search (Bergstra & Bengio, 2012)	12.7	6.89	55.32
TPE (Bergstra et al., 2011)	10.4	6.33	55.74
QMC (Bergstra & Bengio, 2012)	11.3	6.39	56.19
Automatic Pruning Methods			
Pruner-Zero (Dong et al., 2024)	12.7	6.35	57.47
Self-Pruner (Huang et al., 2025)	9.8	6.41	57.12
LLM-Guider (Hasiura et al., 2026)	10.2	6.38	57.29
PPF (Ma et al., 2025)	8.6	6.33	57.58
AutoPrune (Kang et al., 2026)	7.9	6.30	57.72
PrunerAgent	**4.8**	**5.97**	**59.09**

4.5 Ablation Studies

Table 4 ablates each component on LLaMA-2 7B at 50% sparsity. Removing the RSI meta-agent (fixed hyperparameters, as in Pruner-Zero) causes the largest drop on every metric, so adaptive control is the main source of quality. Removing symbolic reasoning hurts quality only slightly but raises cost, since more defective candidates reach the GPU. Removing replay-buffer seeding degrades both quality and efficiency. Removing staged evaluation incurs the highest cost and the largest drop on GSM8K and MMLU.

Table 4: Ablation on LLaMA-2 7B (50% sparsity).

Variant	LM Harness (%)	WikiText PPL	GSM8K (%)	MMLU (%)
PrunerAgent (full)	**59.09**	**5.97**	**10.31**	**35.17**
w/o RSI meta-agent	57.47	6.35	8.47	33.06
w/o symbolic reasoning	58.12	6.12	9.45	34.21
w/o replay buffer	57.89	6.21	9.12	33.87
w/o multi-stage eval	58.34	6.08	9.01	33.65

Module configuration. Table 5 varies the configuration within each module. Fixed exploration ($\mu = 0.3$) collapses diversity late in the run, and fixed exploitation ($\mu = 0.05$) explores too little; adaptive control beats both. Syntax-only filtering and no filtering let unstable candidates reach the GPU, raising cost by $1.38\times$ and $1.72\times$. Single-stage evaluation costs $2.63\times$ more because no candidate is rejected early.

4.6 Calibration Data

Table 6 varies the calibration source (C4 or WikiText-2) and size (32–512 sequences) on LLaMA-2 7B at 50% sparsity. Consistent with Williams et al. (2024), performance saturates around 128 sequences. C4 is 0.3–0.8 points better than WikiText-2 at every size, which we attribute to its broader domain coverage. PrunerAgent leads at every size, and the margin over Pruner-Zero is

Table 5: Module configuration analysis on LLaMA-2 7B (50% sparsity).

Module	Configuration	LM Harness (%)	WikiText PPL	GSM8K (%)	Rel. Cost
RSI Meta-Agent	Adaptive (ours)	**59.09**	**5.97**	**10.31**	1.00
	Fixed exploration ($\mu = 0.3$)	57.12	6.42	8.98	1.21
	Fixed exploitation ($\mu = 0.05$)	56.78	6.58	8.65	0.89
Symbolic Reasoning	Full (ours)	**59.09**	**5.97**	**10.31**	1.00
	Syntax-only	58.23	6.18	9.34	1.38
	No filtering	57.45	6.51	9.09	1.72
Validation Pipeline	Multi-stage (ours)	**59.09**	**5.97**	**10.31**	1.00
	Single-stage full eval	58.34	6.14	9.45	2.63

largest with the least data (+1.70 at 32 sequences), indicating that the discovered metric is robust when calibration data is scarce.

Table 6: Zero-shot accuracy (%) on LLaMA-2 7B (50% sparsity) versus calibration source and size.

Method	Calibration	32	64	128	256	512
Pruner-Zero	C4	54.12	55.23	57.47	57.52	57.54
	WikiText-2	53.45	54.89	56.92	57.01	57.05
PrunerAgent	C4	**55.82**	**57.31**	**59.09**	**59.14**	**59.16**
	WikiText-2	**55.12**	**56.89**	**58.76**	**58.83**	**58.87**

4.7 Cross-Architecture Generalization

Table 7 applies the metric discovered on LLaMA-2 7B to other models without re-searching. The entries are therefore slightly below the per-model search results of Table 1, which searches separately for each model. The transferred metric still outperforms SparseGPT and Wanda on LLaMA-1 7B (6.38 vs. 6.92 and 6.90) and Pruner-Zero on Mistral 7B (5.86 vs. 6.22), showing that the discovered metric captures importance patterns that are not specific to one model.

Table 7: Cross-architecture generalization of the metric discovered on LLaMA-2 7B (transferred, without re-searching).

Model	WikiText PPL	GSM8K (%)	LM Harness (%)	MMLU (%)
LLaMA-1 7B	6.38	9.65	56.12	32.87
LLaMA-1 13B	5.49	16.82	60.45	41.76
LLaMA-2 7B	5.97	10.31	59.09	35.17
LLaMA-2 13B	5.24	16.51	62.10	50.12
LLaMA-3 8B	8.62	47.10	57.21	54.02
Mistral 7B	5.86	28.12	60.98	56.01

5 Conclusion

We present PrunerAgent, an agent-driven recursive self-improvement framework for discovering symbolic pruning metrics for large language models. Its nested loop improves the GP search process itself rather than only its outputs, its agent-managed search space canonicalizes and pre-screens expressions, and its six-stage validation pipeline spends GPU time only on promising candidates. Across LLaMA and Mistral models at 50–80% unstructured and 4:8/2:4 semi-structured sparsity, PrunerAgent discovers stronger metrics than Pruner-Zero and recent LLM-driven methods while reducing search cost by 62%. Future work includes learned meta-policies, quality-diversity archives, and cross-task lifelong RSI.

AI USE STATEMENT

We use generative AI tools for language editing and minor implementation support. AI assistants help improve grammar, clarity, and readability of the manuscript and suggest fixes for code debugging and small implementation details. AI tools are not used to generate research ideas, design the methodology, derive theoretical results, run experiments, or write the core technical content. Separately, LLMs are components of the proposed method itself (meta-agent reasoning and symbolic property inference), as described in Section 3 and Appendix C. We review and verify all AI-assisted outputs and take full responsibility for the accuracy and integrity of the manuscript. No AI tool is listed as an author.

ETHICS STATEMENT

We do not identify ethical concerns specific to this work. All datasets, models, and benchmarks are publicly available and used in accordance with their licenses; they contain no personally identifiable information, and no human subjects are involved. PrunerAgent aims to make LLM pruning-metric discovery more efficient, which may support more efficient LLM deployment. We acknowledge that advances in LLM efficiency can be misused, but this work introduces no inherently harmful capability and involves no sensitive application.

REPRODUCIBILITY STATEMENT

Appendix B states and proves the theoretical results; Appendix C details the implementation, replay-buffer schema, meta-agent state and action spaces, decision rules, and prompt templates; Appendix D gives pseudocode for every subroutine; Appendix H lists all hyperparameters. All models, datasets, and benchmarks are public. We release code, prompts, and discovered metrics upon acceptance.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 1

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012. 8

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011. 8

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. Symbolic discovery of optimization algorithms. In *Advances in Neural Information Processing Systems*, volume 36, 2023. 2

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. 6

DeepSeek. DeepSeek-V4-Flash: Efficient 284B MoE language model, 2026. URL https://api-docs.deepseek.com/news/news260424/. Open-weight model, MIT license. 6

Peijie Dong, Lujun Li, Zhenheng Tang, Xiang Liu, Xinglin Pan, Qiang Wang, and Xiaowen Chu. Pruner-zero: Evolving symbolic pruning metric from scratch for large language models. In *Forty-first International Conference on Machine Learning*, 2024. 1, 2, 6, 8, 19

Ágoston E Eiben, Robert Hinterding, and Zbigniew Michalewicz. *Parameter Control in Evolutionary Algorithms*, volume 3. 1999. 2, 16

Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. Deap: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, 2012. 16

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023. 1, 2, 6, 14

Elias Frantar, Sidak Pal Singh, and Dan Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning. *Advances in Neural Information Processing Systems*, 35:4477–4490, 2022. 14

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836. 6, 16

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015. 1, 2, 6

Jedrzej Hasiura, Prashant Shivaram Bhat, Elahe Arani, and Bahram Zonooz. LLM-Guider: A language-guided discovery of symbolic pruning metrics for post-training sparsity in LLMs. *Transactions on Machine Learning Research*, 2026, 2026. URL https://api.semanticscholar.org/CorpusID:290965963. 2, 6, 8

Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pp. 293–299. IEEE, 1993. 2, 14

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2020. 6

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. 2024. 2, 16

Weizhong Huang, Yuxin Zhang, Xiawu Zheng, Fei Chao, and Rongrong Ji. Towards efficient automatic self-pruning of large language models. *arXiv preprint arXiv:2502.14413*, 2025. 2, 6, 8

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*, pp. 240–248, 2016. 16

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023. 6

Haidong Kang, Lihong Lin, Enneng Yang, Hong-Ning Dai, and Hao Wang. Automatic pruning discovery for large language models. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Qay3nI9WRv. 2, 6, 8

John R Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, volume 1. MIT Press, 1992. 2

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, volume 2, pp. 598–605, 1990. 4, 14

Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. volume 19, pp. 189–223, 2011. 22

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. In *International Conference on Learning Representations*, 2017. 16

Lujun Li, Peijie Dong, Zhenheng Tang, Xiang Liu, Qiang Wang, Wenhan Luo, Wei Xue, Qifeng Liu, Xiaowen Chu, and Yike Guo. Discovering sparsity allocation for layer-wise pruning of large language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. 2

LiteLLM Team. LiteLLM: A unified interface for large language model apis, 2026. URL https://github.com/BerriAI/litellm. 6

Zuxin Ma, Yunhe Cui, and Yongbin Qin. Beyond manually designed pruning policies with second-level performance prediction: A pruning framework for LLMs. *arXiv preprint arXiv:2508.02381*, 2025. 2, 6, 8

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2023. 1, 2

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016. 6

Meta. Introducing meta llama 3: The most capable openly available llm to date, 2024. URL https://ai.meta.com/blog/meta-llama-3/. 6

Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. Sympy: Symbolic computing in python. *PeerJ Computer Science*, 3:e103, 2017. 4, 16

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11264–11272, 2019. 2, 14

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. 2015. 22

Alexander Novikov et al. Alphaevolve: A learning framework for algorithm discovery. *arXiv preprint*, 2025. 2

OpenAI. GPT-5.4 system card. Technical report, OpenAI, 2026. 6

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019. 16

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 (ICML)*, pp. 8007–8019, 2020. 2

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, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. 2

Jürgen Schmidhuber. Gödel machines: Fully self-referential optimal universal self-improvers. *Artificial general intelligence*, pp. 199–226, 2007. 2, 16

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2023. 1, 2

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*, 2023. 1, 2, 4, 6

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a. 1, 6

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b. 6

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. 2023. 19

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024. 2

Thomas Williams et al. Calibration data may impact model performance. *arXiv preprint arXiv:2311.09755*, 2024. 8

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, et al. 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, 2020. 16

Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Mykola Pechenizkiy, Yi Liang, Zhangyang Wang, and Shiwei Liu. Outlier weighed layerwise sparsity (OWL): A missing secret sauce for pruning LLMs to high sparsity. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. 2

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, 2025. 2, 16

Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989. 5, 16

Yingtao Zhang, Haoli Bai, Haokun Lin, Jialin Zhao, Lu Hou, and Carlo Vittorio Cannistraci. Plug-and-play: An efficient post-training pruning method for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. 2, 6

Appendix Overview

Appendix B presents the theoretical foundations: the loss-perturbation view of importance metrics, formal definitions of the screening scores, mask-invariance and elitist-monotonicity results, the expected cost of the staged pipeline, and a formal statement of bounded RSI. Appendix C details the framework implementation. Appendix D provides algorithm tables for all subroutines. Appendix E lists the extended operator set. Appendix F describes the PrunerSkill extension and its implementation. Appendix G discusses extended RSI variants. Appendix H gives the experimental setup. Appendix A discusses limitations and positions PrunerAgent against related automatic pruning methods.

A Discussion and Limitations

A.1 Limitations

First, the symbolic reasoning stage is qualitative: it removes candidates with inherent mathematical defects but does not bound the generalization of a pruned model, so full evaluation remains mandatory. Second, the rule layer of the meta-agent uses hand-set thresholds for stagnation and diversity; they transfer across the models we test, but substantially different architectures or sparsity regimes may require retuning, which motivates the learned meta-policy of Appendix G. Third, the RSI loop adds LLM API calls and state collection; this overhead is negligible against GPU pruning cost for 7B–70B models but could matter for very small models or very tight budgets. Fourth, we evaluate LLaMA-family and OPT architectures; mixture-of-experts, state-space, and multimodal models remain open. Finally, the extended operator set still carries the inductive bias of its primitives; discovering new operator categories through LLM-driven program synthesis is a promising direction.

A.2 Positioning Against Automatic Pruning Methods

Table 8 compares PrunerAgent with recent automatic methods along five dimensions. Pruner-Zero uses symbolic GP without adaptive control, theoretical reasoning, progressive validation, or RSI. Self-Pruner and PPF adapt, but over structured rates or policy selection rather than symbolic metric form. LLM-Guider optimizes symbolic formulas through a fixed pipeline. AutoPrune generates pruning algorithms with an LLM but does not iteratively improve the generation process. PrunerAgent is the only method that combines symbolic GP with adaptive meta-control, theoretical reasoning, progressive validation, and a nested RSI loop, which is what allows it to find stronger metrics at 62% lower GPU cost than Pruner-Zero.

Table 8: Comparison of automatic pruning methods.

Method	Search Space	Adaptive Control	Theoretical Reasoning	Progressive Validation	RSI
Pruner-Zero	Symbolic GP	✗	✗	✗	✗
Self-Pruner	Structured rates	✓	✗	✗	✗
LLM-Guider	Symbolic formulas	✗	✗	✗	✗
PPF	Policy selection	✓	✗	✗	✗
AutoPrune	LLM-generated	✗	✗	✗	✗
PrunerAgent	Symbolic GP	✓	✓	✓	✓

B Theoretical Foundations

B.1 Loss-Perturbation View of Importance Metrics

Let $\mathbf{Y} = \mathbf{W}\mathbf{X}$ be the output of a linear layer on calibration inputs $\mathbf{X} \in \mathbb{R}^{n\times d}$. Removing a single entry W_{ij} changes row i of the output by $\Delta Y_i = W_{ij}X_j$, so the single-entry reconstruction error is

$$\|\Delta Y_i\|_2^2 = W_{ij}^2 \, \|X_j\|_2^2 = W_{ij}^2 A_j^2. \tag{7}$$

Wanda's score $|W_{ij}|\, A_j$ is the square root of Eq. (7) and therefore induces the same neuron-wise mask (Proposition 1). Optimal Brain Damage (LeCun et al., 1990) and Optimal Brain Surgeon (Hassibi et al., 1993) give the second-order perturbation $\frac{1}{2}W_{ij}^2/[\mathbf{H}^{-1}]_{jj}$ with $\mathbf{H} = \mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}$, which SparseGPT (Frantar & Alistarh, 2023; Frantar et al., 2022) uses with weight updates. First-order Taylor importance (Molchanov et al., 2019) gives $|G_{ij}W_{ij}|$. All of these are functions of the terminals $\{W, A, G\}$ and are therefore reachable in the search space. We use Eq. (7) as the cheap reference for the alignment score because it requires no matrix inversion.

B.2 Screening Scores

Let Ω be a fixed set of $|\Omega| = 4096$ weight entries sampled from a probe layer (we use the second MLP down-projection), with terminal values (W_{ij}, A_j, G_{ij}) collected once per model. For a candidate tree T we define three scores in $[0, 1]$.

Definition 1 (Monotonicity). *With $\delta = 0.05$,*

$$\textit{Mono}(T) = \frac{1}{|\Omega|} \sum_{(i,j)\in\Omega} \mathbb{I}\Big[f_T\big((1+\delta)|W_{ij}|\,\mathrm{sgn}(W_{ij}), A_j, G_{ij}\big) \geq f_T(W_{ij}, A_j, G_{ij})\Big]. \tag{8}$$

Monotonicity in $|W|$ encodes the prior, shared by all metrics in Appendix B.1, that scaling a weight up never makes it less important when other quantities are held fixed. The LLM reasoning stage additionally attempts a symbolic proof via the sign of $\partial f_T/\partial|W|$ with SymPy; if a proof succeeds, Mono(T) is set to 1 or 0 without numerical probing.

Definition 2 (Stability). *Let Σ be a set of stress transformations applied jointly to (W, A, G): scaling by 10^3 and 10^{-3}, zeroing 10% of entries, and injecting outliers at $50\times$ the row maximum. Then*

$$\textit{Stab}(T) = 1 - \frac{1}{|\Sigma|} \sum_{\sigma\in\Sigma} \mathbb{I}\big[f_T(\sigma(W), \sigma(A), \sigma(G)) \textit{ contains non-finite values or has zero variance on } \Omega\big]. \tag{9}$$

Definition 3 (Alignment). *Let $R_{ij} = W_{ij}^2 A_j^2$ and let ρ_s denote the Spearman rank correlation over Ω. Then Align$(T) = \max\{0, \rho_s(S_T, R)\}$.*

The composite score is $Q(T) = w_1\text{Mono}(T) + w_2\text{Stab}(T) + w_3\text{Align}(T)$ with default weights $(0.3, 0.4, 0.3)$ and threshold $Q_{\text{th}} = 0.5$, which the meta-agent moves within $[0.35, 0.65]$. Alignment is deliberately weighted so that a candidate can pass with low alignment if it is monotone and stable; the screening removes defective candidates rather than enforcing similarity to Wanda.

B.3 Mask Invariance and Canonicalization

Proposition 1 (Mask invariance under row-wise monotone transforms). *Let $S \in \mathbb{R}^{m\times n}$ be a score matrix and, for each row i, let $\phi_i : \mathbb{R} \to \mathbb{R}$ be strictly increasing. Define $S'_{ij} = \phi_i(S_{ij})$. For neuron-wise pruning that keeps the k_i largest scores in row i , the masks coincide: $M(S') = M(S)$, up to tie-breaking. The same holds for N:M semi-structured pruning, where the comparison group is a block of M consecutive entries within a row.*

Proof. A strictly increasing map preserves the strict order of every pair within a row: $S_{ij} > S_{il} \iff \phi_i(S_{ij}) > \phi_i(S_{il})$. The set of the k_i largest entries of a row depends only on this order, so it is unchanged. Blocks of M entries are subsets of a row, so the same argument applies. □

Corollary 1 (Canonicalization). *Let $u \in \mathcal{O}$ be a unary operator that is strictly increasing on the range of f_T (for example Square Root, Logarithm, and Exponent on positive inputs, Tanh, Sigmoid, Log-Sum-Exp, multiplication by a positive constant, and Min-Max Scaling with matrix-wide constants). Then $u \circ f_T$ and f_T induce identical masks, so the outermost such operator can be removed without changing the pruned model.*

Corollary 1 is the basis of Stage 3: after SymPy simplification, the canonicalizer strips outermost increasing wrappers, then hashes the result. Two trees with equal canonical hashes are guaranteed to yield the same mask, so a single evaluation suffices. Note that the same operators are *not* removable when they appear inside a binary operator (for example $\sqrt{|W|}\cdot A$ versus $|W|\cdot A$), so canonicalization only removes wrappers at the root. Appendix D, Algorithm 5 gives the procedure.

B.4 Elitist Monotonicity of the RSI Loop

Proposition 2 (Meta-adaptation cannot degrade the returned metric). *Let $F_k^* = \min_{T\in\mathcal{H}_k} PPL(T)$ be the best fully evaluated fitness after iteration k. Since $\mathcal{H}$ is append-only and Algorithm 1 returns $\arg\min_{T\in\mathcal{H}} PPL(T)$, the sequence F_k^* is non-increasing in k for any sequence of meta-agent actions $\Theta_0, \Theta_1, \ldots$.*

Proof. $\mathcal{H}_k \subseteq \mathcal{H}_{k+1}$ implies $\min_{\mathcal{H}_{k+1}} \le \min_{\mathcal{H}_k}$. □

The practical consequence is that exploration moves (raising μ, reseeding, relaxing Q_{th}) are safe: they may waste part of the budget, but they cannot make the final metric worse than what has already been found. This is why the meta-agent can afford aggressive exploration when stagnation is detected.

B.5 Expected Cost of the Staged Pipeline

Proposition 3 (Expected evaluation cost). *Let stage $t \in \{1, \ldots, 6\}$ have cost c_t and pass rate p_t, with $c_1 \le c_2 \le \cdots \le c_6$. The expected cost of processing one candidate is*

$$\mathbb{E}[C] = c_1 + \sum_{t=2}^{6} c_t \prod_{u<t} p_u, \tag{10}$$

compared with c_5 for single-stage full evaluation (Stage 6 applies only to the final top-K set in both cases).

Proof. A candidate reaches stage t with probability $\prod_{u<t} p_u$ and then incurs cost c_t; linearity of expectation gives Eq. (10). □

In the LLaMA-2 7B run, $c_1 = c_3 = 0$ GPU time, c_2 is an LLM call, $c_4 \approx c_5/10$, and $\prod_{u \le 4} p_u = 133/350 \approx 0.38$, which is consistent with the observed $2.63\times$ cost of single-stage evaluation in Table 5. The risk of the pipeline is a false negative at Stage 4: a candidate with poor proxy perplexity but good full perplexity. We mitigate this in two ways, following the logic of successive halving and multi-fidelity optimization (Jamieson & Talwalkar, 2016; Li et al., 2017): the keep fraction q is a meta-agent action that is raised in exploration mode, and any candidate within a margin of 5% of the proxy threshold is re-evaluated with a larger proxy set before rejection.

B.6 Bounded RSI as Meta-Level Control

We formalize the outer loop as a control problem. The meta-level state is $s_k = (D_k, F_k^*, \tau_k, b_k, s)$, the action is $a_k = \Theta_{k+1} \in \mathcal{A}$, where $\mathcal{A}$ is the compact box of Table 11, and the transition is induced by running the inner loop under Θ_{k+1}. The meta-policy $\mathcal{M}$ maps $(s_k, \mathcal{H}_k)$ to $\mathcal{A}$. This is *bounded* RSI in the following sense: the class of inner algorithms $\{\mathrm{GP}_\Theta : \Theta \in \mathcal{A}\}$ and the code of $\mathcal{M}$ are fixed a priori, and the outer loop changes only Θ. It therefore differs from unbounded self-modification in the Gödel-machine sense (Schmidhuber, 2007), in which the system may rewrite its own meta-level, and is aligned with recent bounded RSI systems (Zhang et al., 2025; Hu et al., 2024). It also differs from self-refinement, which corresponds to the degenerate case $\mathcal{A} = \{\Theta_0\}$. Adaptive parameter control in evolutionary computation (Eiben et al., 1999) is the classical special case in which $\mathcal{M}$ is a fixed deterministic rule over μ alone; PrunerAgent extends the action space to the population, budget, grammar, and validation thresholds and lets an LLM propose actions within the rule-defined box.

B.7 Diversity Measure

Eq. (5) uses the ordered tree edit distance of Zhang & Shasha (1989), normalized by the size of the larger tree so that $D \in [0, 1]$ and thresholds are comparable across population sizes and depths. With $d_{\max} \le 10$ and at most 64 to 128 trees, the $O(N^2)$ pairwise computation takes well under a second per generation and is negligible relative to evaluation.

C Framework Implementation

C.1 System Architecture

PrunerAgent is implemented in Python 3.11 with PyTorch (Paszke et al., 2019) and Hugging Face Transformers (Wolf et al., 2020) for model loading and pruning, a custom GP engine in the style of DEAP (Fortin et al., 2012) for evolution, SymPy (Meurer et al., 2017) for simplification, LiteLLM for LLM routing, SQLite for the replay buffer, and lm-evaluation-harness (Gao et al., 2023) for downstream evaluation. Table 9 lists the modules.

Table 9: Modules of the PrunerAgent implementation.

Module	Responsibility	Key dependencies
`search_space/`	Operator registry, grammar $\mathcal{G}$, canonicalizer, hashing	SymPy
`evolution/`	Population, tournament selection, crossover, mutation, seeding from $\mathcal{H}$	NumPy
`validation/`	Six-stage pipeline, screening scores, proxy and full PPL, top-K zero-shot	PyTorch, lm-eval-harness
`meta_agent/`	State summarizer, LLM proposer, rule projection, action logging	LiteLLM
`replay/`	SQLite schema, insert/query, elite and diverse sampling	sqlite3
`pruning/`	Calibration hooks for A and G, neuron-wise and N:M masking	Transformers
`skills/`	PrunerSkill extraction, scoring, injection (Appendix F)	SymPy

C.2 Pruning Backend

For each linear layer we register forward hooks that accumulate $A_j = \|X_j\|_2$ over the calibration set and run a single backward pass on the calibration language-modelling loss to obtain G_{ij}. A candidate tree is compiled to a vectorized PyTorch function over the broadcast tensors (W, A, G); matrix-level operators (Softmax, Min-Max Scaling, Sum, Mean, Norm-1, Norm-2) are applied per output row so that they are compatible with neuron-wise comparison. Unstructured masks keep the

top $(1-s)n$ scores per row; N:M masks keep the top N scores in each block of M consecutive columns. Layers are pruned sequentially from input to output with activations recomputed on the already-pruned prefix, as in Wanda and SparseGPT.

C.3 Replay Buffer Schema

The replay buffer is a SQLite database with three tables (Table 10). The `candidates` table is append-only. Seeding samples 70% of the population from the top-25% by full fitness (elite) and 30% uniformly from the remaining entries (diverse); when PrunerSkill is enabled, 20% of the elite share is replaced by skill-grafted trees.

Table 10: Replay buffer schema.

Table	Columns	Purpose
`candidates`	`id, iter, gen, tree_json, canon_expr, canon_hash, mono, stab, align, q_score, proxy_ppl, full_ppl, depth, n_nodes, op_hist, parent_ids, seed, timestamp`	Every candidate that passes Stage 3
`iterations`	`iter, diversity, best_ppl, stagnation, budget_used, mode, mu, n_pop, d_max, q_th, keep_frac, reseed_frac, llm_rationale`	Per-iteration statistics and actions
`skills`	`skill_id, canon_hash, expr, arity, contribution, usage, support, source_runs`	PrunerSkill library (Appendix F)

C.4 Meta-Agent: State, Actions, Rules, and Guardrails

State summary. At the end of each iteration, the summarizer produces a compact text state containing D_k, F_k^*, τ_k, b_k, the current Θ_k, the top-5 canonical expressions with fitness, the operator histogram of the top-25% versus the whole population, and the pass rate of each validation stage. The summary is under 1,500 tokens.

Action space. Table 11 lists every controllable quantity with its admissible range. Proposals outside the range are clipped.

Table 11: Meta-agent action space $\mathcal{A}$ and default values.

Symbol	Meaning	Range	Default
μ	Mutation rate	$[0.05, 0.50]$	0.15
N_{pop}	Population size	$[16, 128]$	64
$d_{\max}$	Maximum tree depth	$[4, 10]$	8
B	Full evaluations per iteration	$[4, 32]$	16
Q_{th}	Theoretical screening threshold	$[0.35, 0.65]$	0.50
q	Proxy keep fraction	$[0.30, 0.70]$	0.50
r	Reseed fraction from $\mathcal{H}$	$[0.00, 0.50]$	0.00
$\mathcal{O}_{\text{on}}$	Enabled operator subset	$\mathcal{O}_{\text{base}} \subseteq \mathcal{O}_{\text{on}} \subseteq \mathcal{O}$	$\mathcal{O}$

Decision rules. The rule layer implements the two regimes of Section 3.3 (Table 12). Rules fire before the LLM is consulted and define the mode; the LLM then proposes values within the mode-specific sub-box (for example, in exploration mode μ may only increase). If the LLM response is malformed or violates the sub-box after clipping, the rule action is applied alone.

Table 12: Meta-agent decision rules ($\tau_{\text{th}} = 15$ generations, $D_{\text{th}} = 0.3$).

Condition	Rule action	Mode
Stagnation $\tau_k > 15$	$\mu \leftarrow \min(0.5, 1.5\mu)$; $N_{\text{pop}} \leftarrow \min(128, 1.5N_{\text{pop}})$; $Q_{\text{th}} \leftarrow \max(0.35, Q_{\text{th}} - 0.05)$; $q \leftarrow \min(0.7, q + 0.1)$	Exploration
Diversity $D_k < 0.3$	$r \leftarrow 0.3$ (reseed from diverse entries of $\mathcal{H}$); $d_{\max} \leftarrow \min(10, d_{\max} + 1)$	Exploration
Fitness improves 5 consecutive gens and $D_k \geq 0.3$	$\mu \leftarrow \max(0.05, 0.8\mu)$; $N_{\text{pop}} \leftarrow \max(16, 0.8N_{\text{pop}})$; $Q_{\text{th}} \leftarrow \min(0.65, Q_{\text{th}} + 0.05)$	Exploitation
Budget $b_k > 0.8$	Restrict seeding to top-10 candidates; $r \leftarrow 0$; $q \leftarrow \max(0.3, q - 0.1)$	Exploitation
Otherwise	Keep Θ_k; LLM may adjust within $\pm 10\%$ of each value	Neutral

Prompt template. The LLM meta-agent receives the state summary and the following instruction, and must reply with a JSON object matching the action schema.

```
You control a genetic-programming search for LLM pruning
metrics over terminals W, A, G. Current mode: {MODE}.
Current configuration: {THETA}. Population statistics:
```

```
{STATS}. Top candidates: {TOP5}. Stage pass rates:
{PASS}.
Propose an updated configuration that (i) stays within
the ranges {RANGES}, (ii) respects the mode constraints
{MODE_CONSTRAINTS}, and (iii) is justified by the
statistics. Return JSON with keys mu, n_pop, d_max, budget,
q_th, keep_frac, reseed_frac, disabled_operators, rationale.
```

Symbolic reasoning prompt. For Stage 2, the LLM receives the candidate in canonical infix form and returns JSON with keys `monotone_in_absW` (yes/no/unknown), `domain_issues` (list), `scale_invariant` (yes/no), `alignment_rationale` (string), and `recommend_reject` (boolean). SymPy checks each reported domain issue and attempts the monotonicity proof; the numerical scores of Appendix B.2 are always computed and override the LLM's qualitative judgement when they disagree. The LLM output is thus a prior that accelerates rejection of obviously defective candidates, not the arbiter.

C.5 Validation Stage Configuration

Stage 4 prunes with 16 calibration sequences of 512 tokens and scores perplexity on a disjoint 16-sequence slice; Stage 5 prunes with the full calibration set (128 sequences of 2048 tokens) and scores the full WikiText-2 test set; Stage 6 evaluates the top $K = 5$ candidates on the LM Harness tasks. The proxy-to-full Spearman correlation is logged per iteration and reported to the meta-agent as part of the state.

C.6 Cost Accounting

GPU time in Tables 3 counts Stages 4–6 only, since Stages 1–3 run on CPU. LLM usage per iteration is one GPT-5.4 meta-agent call, one GPT-5.4 symbolic-reasoning call per candidate that passes Stage 1, and GPT-5.4-mini calls for deduplication triage. Wall-clock overhead of the LLM calls is small relative to a single full evaluation of a 7B model and is not included in GPU hours; we report it separately in the released logs.

D Algorithm Tables

Algorithm 2 Inner GP loop: SEED and GP

function SEED($\mathcal{H}, \Theta$)
  $E \leftarrow$ top-25% of $\mathcal{H}$ by `full_ppl`; $U \leftarrow \mathcal{H} \setminus E$
  $\mathcal{P} \leftarrow$ sample $0.7N_{\text{pop}}$ from E and $0.3N_{\text{pop}}$ uniformly from U
  if $r > 0$ **then** replace rN_{pop} members by entries of U maximizing distance to $\mathcal{P}$
  end if
  if skills enabled **then** replace 20% of the elite share by SKILLGRAFT (Alg. 6)
  end if
  return $\mathcal{P}$
end function
function GP($\mathcal{P}, \Theta, \mathcal{O}_{\text{on}}, \mathcal{G}$)
  $\mathcal{C} \leftarrow \emptyset$
  while $|\mathcal{C}| < N_{\text{pop}}$ **do**
    $T_a, T_b \leftarrow$ tournament selection (size 5) from $\mathcal{P}$
    $T \leftarrow$ subtree crossover(T_a, T_b) with prob. 0.9, else copy T_a
    With prob. μ: point mutation, subtree replacement, or skill insertion (if enabled)
    if T satisfies $\mathcal{G}$ with depth $\leq d_{\max}$ and operators $\subseteq \mathcal{O}_{\text{on}}$ **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{T\}$
    end if
  end while
  return $\mathcal{C}$
end function

Algorithm 3 Meta-agent policy $\mathcal{M}$

Require: Θ_k, state $s_k = (D_k, F_k^*, \tau_k, b_k)$, replay buffer $\mathcal{H}$
mode $\leftarrow$ NEUTRAL
if $\tau_k > \tau_{\text{th}}$ **or** $D_k < D_{\text{th}}$ **then** mode $\leftarrow$ EXPLORATION; $\Theta' \leftarrow$ apply exploration rules (Table 12)
else if fitness improved for 5 consecutive generations **or** $b_k > 0.8$ **then** mode $\leftarrow$ EXPLOITATION; $\Theta' \leftarrow$ apply exploitation rules
else $\Theta' \leftarrow \Theta_k$
end if
summary $\leftarrow$ SUMMARIZE$(s_k, \Theta', \mathcal{H})$
proposal $\leftarrow$ LLM(prompt(summary, mode))
if proposal parses and respects mode constraints **then** $\Theta_{k+1} \leftarrow \text{clip}_{\mathcal{A}}$(proposal)
else $\Theta_{k+1} \leftarrow \Theta'$
end if
Log (mode, Θ_{k+1}, rationale) to `iterations`
return Θ_{k+1}

Algorithm 4 Six-stage validation VALIDATE

Require: candidate T, configuration Θ
Stage 1: if T violates $\mathcal{G}$, is constant, or has no W leaf **then return** (false, $\cdot$)
Stage 2: (Mono, Stab, Align) $\leftarrow$ probes on Ω (Appendix B.2), informed by LLM+SymPy report
    $Q \leftarrow w_1$Mono $+ w_2$Stab $+ w_3$Align; **if** $Q < Q_{\text{th}}$ **then return** (false, $\cdot$)
Stage 3: $h \leftarrow$ hash(CANONICALIZE(T)) (Alg. 5); **if** $h \in \mathcal{H}$ **then return** (false, $\cdot$)
Stage 4: $\text{ppl}_{\text{proxy}} \leftarrow$ prune with 16 sequences, score 16-sequence slice
    if $\text{ppl}_{\text{proxy}}$ not within top fraction q of the current generation **then**
        if within 5% of threshold **then** re-evaluate with 32 sequences **else return** (false, $\cdot$)
Stage 5: $\text{ppl}_{\text{full}} \leftarrow$ prune with full calibration set, score WikiText-2
Stage 6: (deferred) evaluate top-K of $\mathcal{H}$ on LM Harness after the final iteration
return (true, ppl_{full})

E EXTENDED OPERATOR SET

Table 13 lists the full operator set. The first 17 are inherited from Pruner-Zero (Dong et al., 2024); the remaining 11 are our extensions. Extensions were chosen to add function classes absent from the original set: competitive normalization (Softmax), smooth maxima (Log-Sum-Exp), polarity (Sign), selection (Max, Min), amplification of small magnitudes (Reciprocal), outlier suppression (Exp-Decay), and higher-order or oscillatory nonlinearities (Power-3, Sinh, Sin, Cos). Operators with restricted domains are wrapped as $\log(|\cdot| + \epsilon)$, $\sqrt{|\cdot|}$, and $a/(|b| + \epsilon)$ with $\epsilon = 10^{-8}$. The meta-agent can disable any extended operator via $\mathcal{O}_{\text{on}}$ but never disables the base set.

F PRUNERSKILL: REUSABLE SYMBOLIC BUILDING BLOCKS

F.1 MOTIVATION AND DEFINITION

Runs on different models and sparsity levels repeatedly rediscover the same subexpressions, for example products of $|W|$ with a normalized activation term or gradient-weighted magnitudes. PrunerSkill extends PrunerAgent with a *skill library* $\mathcal{S}$ that stores such subexpressions and injects them into later searches, analogous to skill libraries in embodied LLM agents (Wang et al., 2023). A skill is a tuple $(\sigma, h, k, c, u, \text{src})$: a subtree σ whose leaves may be typed placeholders, its canonical hash h, its arity k (number of placeholders), a contribution score c, a usage count u, and the set of source runs. Skills are stored in the `skills` table of the replay buffer (Table 10).

F.2 EXTRACTION

After each completed run, EXTRACTSKILLS (Algorithm 6) enumerates all subtrees of depth at least 2 in the top-20% of candidates by full fitness, canonicalizes them with Algorithm 5, and retains those

Algorithm 5 CANONICALIZE: symbolic normalization for deduplication

Require: tree T
1: $e \leftarrow$ SymPy expression of T with W, A, G as positive/real symbols as appropriate
2: $e \leftarrow$ `simplify(powsimp(expand_log(`e`)))`
3: **while** root of e is a strictly increasing unary operator on its input range (Corollary 1) **do** strip the root
4: **end while**
5: Sort arguments of commutative operators ($+$, $\times$, $\max$, $\min$) lexicographically
6: **return** string of e and its SHA-256 hash

Algorithm 6 PrunerSkill: EXTRACTSKILLS, SKILLGRAFT, and skill-insertion mutation

function EXTRACTSKILLS($\mathcal{H}$, support $s_{\min} = 3$, top fraction 0.2)
  $\mathcal{E} \leftarrow$ top-20% of $\mathcal{H}$ by `full_ppl`, across all runs
  Count canonical hashes of all subtrees of depth ≥ 2 in $\mathcal{E}$
  for each subtree σ with count $\geq s_{\min}$ appearing in ≥ 2 runs **do**
    contrib$(\sigma) \leftarrow \overline{\text{fit}}(\{T \in \mathcal{H} : \sigma \not\subseteq T\}) - \overline{\text{fit}}(\{T \in \mathcal{H} : \sigma \subseteq T\})$ ▷ lower PPL is better
    if contrib$(\sigma) > 0$ **then** upsert σ into `skills` with arity (number of leaf placeholders), support, contribution
    end if
  end for
  Decay `contribution` of unused skills by 0.9 per run; keep the top 128 skills
end function
function SKILLGRAFT($\mathcal{S}$)
  Sample skill $\sigma \sim$ softmax(contrib$/\beta$) from library $\mathcal{S}$; fill placeholders with random terminals or depth-≤ 2 subtrees
  Sample a random tree T and replace a random subtree by σ; increment `usage(`σ`)`
  return T
end function
Skill-insertion mutation: with probability $p_{\text{skill}} = 0.2$ among mutation types, replace a random subtree of the parent by a sampled skill (as in SKILLGRAFT).

with support at least 3 that occur in at least two distinct runs; the cross-run requirement prevents run-specific artefacts from entering the library. Leaves that vary across occurrences of otherwise identical subtrees are abstracted into placeholders, yielding parametric skills such as $|\diamond| \cdot \text{Softmax}(A)$.

F.3 SCORING

The contribution $c(\sigma)$ is the difference in mean full perplexity between candidates that do not and do contain σ (positive is better). This marginal-contribution estimate is cheap, requires no additional evaluation, and is recomputed whenever the buffer grows. Contributions of skills not used in the most recent run decay by a factor of 0.9, and the library is capped at 128 skills, so stale skills are eventually evicted.

F.4 INJECTION

Skills enter the search at two points. During seeding, 20% of the elite share is replaced by trees produced by SKILLGRAFT, which samples a skill with probability proportional to softmax(c/β) ($\beta = 0.5$), fills placeholders with random terminals or shallow subtrees, and grafts the result onto a random tree. During evolution, skill insertion is added as a third mutation type with probability $p_{\text{skill}} = 0.2$ among mutation types. All injected trees pass the same six-stage validation as any other candidate, so skills that stop being useful are filtered out rather than trusted.

F.5 RESULTS

Table 14 compares PrunerAgent with and without the skill library on LLaMA-2 7B at 80% sparsity under the same 350-trial budget as the main experiments. Reusing 47 skills from earlier runs

Table 13: Operator set for the symbolic pruning-metric search space.

Category	Operator	Math Form	Rationale (extensions)
Unary (Original)	Square	$(\cdot)^2$	
	Negative	$-(\cdot)$	
	Absolute	$\lvert\cdot\rvert$	
	Logarithm	$\log(\lvert\cdot\rvert+\epsilon)$	
	Exponent	$\exp(\cdot)$	
	Square Root	$\sqrt{\lvert\cdot\rvert}$	
	Tanh	$\tanh(\cdot)$	
	Sigmoid	$\sigma(\cdot)$	
	Norm-1	$\lVert\cdot\rVert_1$ (per row)	
	Norm-2	$\lVert\cdot\rVert_2$ (per row)	
	Min-Max Scaling	$\mathrm{minmax}(\cdot)$	
	Sum	$\sum(\cdot)$ (per row)	
	Mean	$\mathrm{mean}(\cdot)$ (per row)	
Unary (Extended)	Sign	$\mathrm{sign}(\cdot)$	Preserves polarity
	Sinh	$\sinh(\cdot)$	Odd exponential growth
	Cos	$\cos(\cdot)$	Periodicity
	Sin	$\sin(\cdot)$	Phase-shifted periodicity
	Power-3	$(\cdot)^3$	Odd third-order nonlinearity
	Exp-Decay	$\exp(-\lvert\cdot\rvert)$	Suppresses outliers
	Reciprocal	$1/(\lvert\cdot\rvert+\epsilon)$	Amplifies near-zero values
	Softmax	$\mathrm{Softmax}(\cdot)$ (per row)	Competitive normalization
	Log-Sum-Exp	$\log(1+\exp(\cdot))$	Smooth maximum
Binary	Addition	$a+b$	
	Subtraction	$a-b$	
	Multiplication	$a\times b$	
	Division	$a/(\lvert b\rvert+\epsilon)$	
	Max (Extended)	$\max(a,b)$	Adaptive selection
	Min (Extended)	$\min(a,b)$	Conservative estimate

improves perplexity from 11.02 to 10.54 and cuts search time from 4.8 to 3.5 hours, showing that knowledge accumulated across model–task pairs transfers to new searches.

Table 14: PrunerSkill versus PrunerAgent on LLaMA-2 7B at 80% sparsity (350 trials).

Method	PPL	Search Time (h)	Skills Reused
PrunerAgent	11.02	4.8	0
PrunerSkill	**10.54**	**3.5**	47

F.6 Discovered Metrics

The best metric discovered for LLaMA-2 7B at 50% sparsity is

$$S_{ij} = \frac{|W_{ij}| \cdot \|A_j\|_2}{\|W\|_F} \times \left(1 + \log(1 + |G_{ij}|)\right). \tag{11}$$

where σ is the sigmoid. It combines a normalized weight–activation product with a gradient-aware term. The expressions were absent from the initial population and contain subexpressions that the skill library later reused (for example the normalized $|W| \cdot A$ block in Eq. (11)), illustrating that RSI discovers non-trivial and transferable structure rather than re-deriving Wanda.

G Extended RSI Variants

Three extensions follow naturally from the meta-level control view of Appendix B.6. **Learned meta-policy.** Replace the rule layer by a policy trained with reinforcement learning on logged $(s_k, a_k, F^*_{k+1} - F^*_k)$ tuples from many runs; the rule layer remains as a safety projection. **Quality-diversity archive.** Replace the elite/diverse split by a MAP-Elites archive (Mouret & Clune, 2015) over structural descriptors (depth, operator histogram, Q components), which directly optimizes diversity rather than relying on stagnation triggers (Lehman & Stanley, 2011). **Cross-task lifelong RSI.** Persist the skill library and the meta-policy across model–task pairs so that new searches start from a warm state; PrunerSkill (Appendix F) is the first step in this direction.

H Detailed Experimental Setup

All experiments run on NVIDIA RTX A6000 GPUs. Calibration data consists of 128 sequences of 2048 tokens from C4 for general-knowledge evaluation and 100 truncated examples (at most 512 tokens) from GSM8K and MMLU for the respective task-specific evaluations. Table 15 lists all search hyperparameters; the initial GP values match Pruner-Zero so that the ablation "w/o RSI meta-agent" is exactly the static baseline. Each configuration is run with 3 random seeds and we report means. LLM backends: GPT-5.4 (reasoning effort high) for the meta-agent and symbolic property inference, GPT-5.4-mini for deduplication and proxy-stage triage, DeepSeek-V4-Flash as a low-cost alternative backend; all routed through LiteLLM with temperature 0 for reproducibility.

Table 15: Search hyperparameters.

Hyperparameter	Value
Candidate trials per search	350
Initial population size / crossover rate / mutation rate	64 / 0.9 / 0.15
Tournament size / initial max depth	5 / 8
Stagnation threshold τ_{th} / diversity threshold D_{th}	15 generations / 0.3
Screening weights (w_1, w_2, w_3) / initial Q_{th}	$(0.3, 0.4, 0.3)$ / 0.5
Probe set size $\lvert\Omega\rvert$ / stress transformations $\lvert\Sigma\rvert$	4096 / 4
Proxy calibration / proxy scoring sequences	16×512 tokens / 16×512 tokens
Full calibration sequences	128×2048 tokens (C4)
Top-K for zero-shot evaluation	5
Elite / diverse seeding split	70% / 30%
PrunerSkill: $s_{\min}$, p_{skill}, β, library cap	3, 0.2, 0.5, 128
Seeds	3