AUTO-LTOPT: AUTOMATED OPTIMIZER EVOLUTION FOR EXTREME CLASS IMBALANCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Practical learning tasks frequently exhibit imbalanced long-tailed data distributions, posing significant challenges for standard optimizers to generalize to tail classes. Existing long-tailed optimizers rely heavily on hand-crafted rules and limited gradient adjustment strategies, resulting in poor flexibility and generalization. In this paper, we propose Auto-LTOpt, a novel automated framework that uses large language models to automatically design tailored optimizers for long-tailed learning without human intervention. By exploring a rich search space of gradient manipulation operations through program evolution, Auto-LTOpt can autonomously discover effective optimization strategies to balance head and tail classes. Extensive experiments on standard long-tailed benchmarks show that Auto-LTOpt outperforms state-of-the-art handcrafted optimizers, while achieving strong and stable performance across diverse imbalanced settings. Our code is available at **Supplement**.

1 INTRODUCTION

In realistic environments, data inherently emerges in a long-tailed distribution, characterized by a staggering abundance of a few head classes and an extreme scarcity of numerous tail classes. This intrinsic imbalance is pervasive across ubiquitous and safety-critical applications, ranging from recognizing rare diseases in medical imaging [35], and identifying edge-case hazards in autonomous driving [10], to fine-grained species classification in the wild [30]. While modern deep neural networks have achieved unprecedented success, their triumph is largely predicated on idealized, artificially balanced datasets [25]. When exposed to the stark reality of long-tailed distributions, these models inevitably exhibit a severe bias, performing exceptionally well on common categories but failing drastically on the rare, yet often more critical, tail classes. This issue is particularly severe at the optimization level. Mainstream optimizers (e.g., SGD [3; 23], AdamW [15]) are fundamentally designed under the assumption of uniform data distributions, treating gradients from all classes indiscriminately. Consequently, the optimization trajectory is overwhelmingly dominated by head classes, while the weak gradient signals from tail classes are marginalized [29].

Driven by the critical need to mitigate the detrimental effects of extreme class imbalance, the community has formulated a broad spectrum of countermeasures. Traditionally, the most prevalent strategies involve manipulating the dataset composition via resampling techniques [5; 38] or calibrating decision boundaries through tailor-made objective functions [4; 17]. Recently, there has been a growing trend to tackle this issue directly at the optimization level. For instance, some studies incorporate meta-learning-based gradient reweighting (e.g., Meta-Weight-Net [27]) to dynamically adjust update steps for minority classes. Other advanced works focus on smoothing the sharp loss landscape characteristic of tail classes. A foundational approach is Sharpness-Aware Minimization (SAM) [8], which explicitly penalizes sharp minima by optimizing for uniformly low loss neighborhoods. Notable extensions tailored for long-tailed learning include Focal-SAM [11], which introduces manually crafted focal-style sharpness penalties, and ProMO [1], which integrates the Muon orthogonal optimizer [2; 16] to help tail classes escape sharp minima.

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.

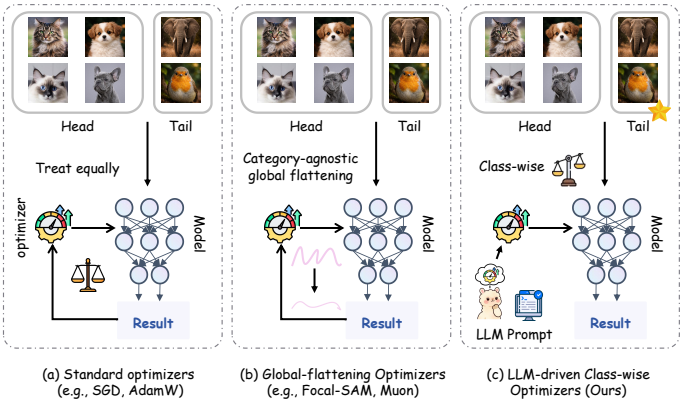


Figure 1: Comparison of optimizer updating strategies.

Despite their empirical success, these tailored optimization interventions are fundamentally constrained by three primary limitations. ❶ **Reliance on Hand-Crafted Rules:** A prominent bottleneck is their heavy reliance on hard-coded, rule-based configurations. For instance, Focal-SAM requires experts to manually formulate focal-style category penalty equations, whereas ProMO depends on an intricate, heuristically designed sinusoidal schedule to alternate between different optimizers. Such rigid mechanisms necessitate exhaustive trial-and-error for hyperparameter tuning and inherently limit their generalizability across varied dataset distributions. ❷ **Confined Actionable Dimensions:** Furthermore, the actionable dimension of these existing methods is strictly confined. Even for adaptive approaches like Meta-Weight-Net, the intervention is exclusively restricted to allocating scalar weights for gradient steps. This merely scales update magnitudes, ignoring fundamental structural modifications to the optimizer such as layer-specific manipulations, stability protections, or moving average mechanisms. Consequently, a vast landscape of effective optimization architectures remains undiscovered. ❸ **Category-Agnostic Treatment:** As illustrated in Figure 1, standard optimizers treat all classes equally. While global-flattening optimizers (e.g., Focal-SAM, Muon) apply global constraints to flatten update directions, they fail to actively amplify tail-class features. In contrast, our optimizer employs class-wise adaptive enhancement, automatically identifying tail classes and applying dynamic weighting to achieve an optimal head-tail balance.

► **Can large language models (LLMs) provide optimization strategies for long-tailed learning?** To address the aforementioned limitations, we explore the potential of large language models (LLMs) in designing optimization algorithms. Specifically, as illustrated in Figure 2, we tested a standard Web LLM alongside a currently popular "vibe coding" tool, the Code Agent. Our analysis reveals that both possess significant design potential. The Code Agent exhibits slightly better performance, primarily because it can directly write, run, and debug code while perceiving the specific project environment. However, although these generated optimizers achieve acceptable performance comparable to several manually designed baselines, there remains a discernible gap compared to the optimal performance. *This preliminary exploration proves that LLMs are highly capable of serving as optimization algorithm generators, while also highlighting that there is still substantial room for further methodological refinement*.

Furthermore, inspired by AutoML, we propose **Auto-LTOpt**, a novel LLM-based automated optimizer design framework for long-tailed learning. Instead of relying on human heuristics, we define a broad search space that includes gradient manipulation operations, momentum mechanisms, orthogonalization, and their parameters. There is no need to manually prescribe how to weight tail gradients or balance head and tail classes. Instead, the evolutionary process autonomously explores effective gradient adjustment rules. The evolution acts as a form of "natural selection": we take performance improvement on the validation set as the survival criterion. In each evolutionary round, diverse optimizer variants are generated, with poorly performing ones eliminated and superior candidates preserved. Through crossover and mutation operations, the algorithm continuously searches for better adaptation logic for long-tailed distributions. Throughout evolution, zero human intervention is introduced. There are no predefined rules that tail gradients must be amplified, nor specified thresholds for gradient clipping. The algorithm adjusts its gradient processing strategies entirely based on training feedback. This constitutes the fundamental difference between our work and traditional manually designed optimizers: we do not "teach the optimizer what to do", but enable the optimizer to "learn how to adapt on its own".

Contributions. ❶ ***New Optimization Paradigm.*** We propose a novel automated framework called AutoLTOpt, which combines large language models with optimizer design **for the first time**. This approach automates the search for effective optimizers and significantly reduces the cost of manually

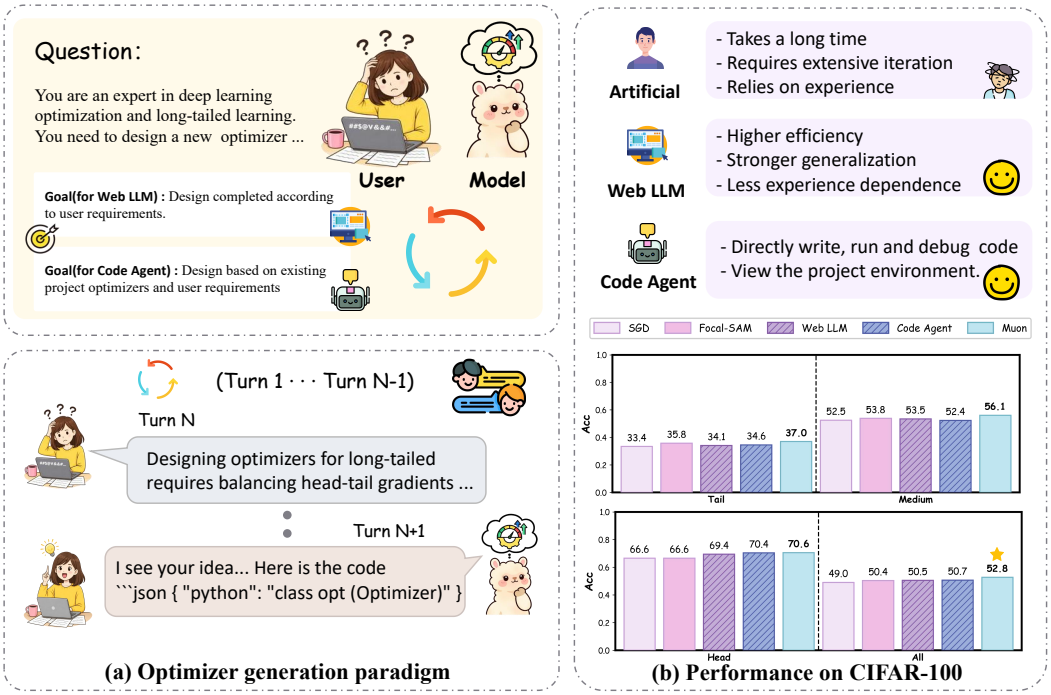


Figure 2: The optimizer generation paradigm and preliminary performance. (a) Comparison among artificial design, Web LLMs, and Code Agents. (b) Performance comparison on CIFAR-100.

designing optimization strategies, providing a novel perspective for efficient long-tailed learning. ❷ ***Autonomous Program Evolution.*** For the field of optimizer design, we prove that program evolution can autonomously discover effective optimization strategies without human intervention. This achieves precise matching between tasks and optimizers, providing a reusable automated framework for future complex scenarios such as imbalanced classification, federated learning, and multimodal learning. ❸ ***Novel Theoretical Insights.*** We provide detailed convergence analysis and theoretical explanations regarding the optimization strategies discovered by our framework, offering profound new insights to inspire future research in extreme class imbalance optimization. ❹ ***Empirical Study.*** Thorough evaluations on standard imbalanced datasets confirm that AutoLTOpt consistently achieves superior performance compared to current state of the art general and specialized optimizers.

2 Related Work

2.1 Long-Tailed Learning

Long-tailed learning is a core research direction for addressing class imbalance. Existing methods mainly explore from data, model, and optimization perspectives. On the data level, re-sampling [12; 26; 33], and data augmentation [31; 32] are adopted to adjust the class distribution and alleviate bias caused by imbalance, but they often lead to overfitting or information loss. On the model level, strategies such as class reweighting [6], and metric learning [9] are used to strengthen the focus on tail classes. For optimizer adaptation to long-tailed distributions, SAM-based methods including ImbSAM [39], CC-SAM [40], and Focal-SAM [11]improve tail generalization by flattening the loss landscape, yet suffer from a trade-off between control granularity and computation efficiency. ProMO [1] further optimizes tail-class convergence by switching strategies, but still relies on manually predefined heuristic rules.

2.2 Automated Algorithm Design with Large Language Models

The unprecedented code generation and logical reasoning capabilities of large language models [18; 24; 34] have sparked a new wave of automated algorithm design. By framing the search process

as a natural language prompting task, researchers have successfully utilized these models to discover novel heuristic algorithms and mathematical solutions [19; 36]. Notable frameworks AEL [13], ReEvo [37]leverage iterative feedback loops, where the model acts as an evolutionary generator that refines algorithm pseudocode based on empirical performance metrics. Recent studies have demonstrated the potential of automated generation to invent new gradient descent variants and optimization strategies without human intervention [22; 28]. Our work extends this promising frontier into the highly specialized domain of imbalanced classification, utilizing program evolution to autonomously design optimizers tailored for long-tailed distributions.

3 Methodology: Automated Optimizer Evolution

In this section, we detail the proposed automated framework for optimizer design. As illustrated in Figure 3, inspired by recent advancements in automated algorithm discovery, our framework operates through two tightly coupled interactive modules: a language model-driven generation module and a downstream training and evaluation module. We first introduce our novel multi-granularity search space, followed by the generation operators and the evaluation pipeline.

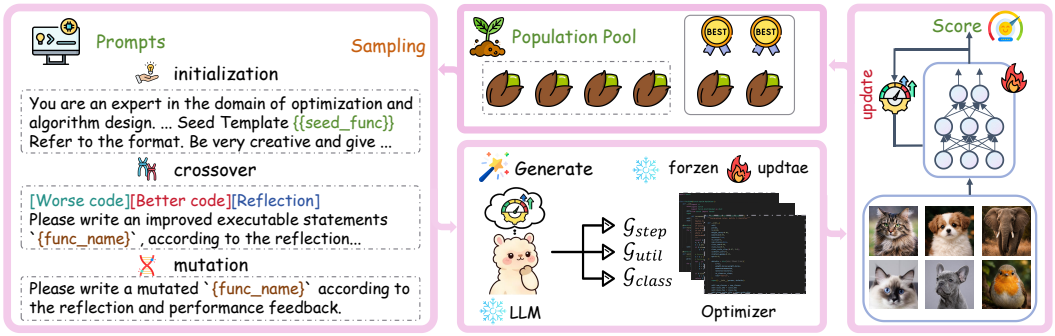


Figure 3: **Overview of the Auto-LTOpt**. The framework consists of a prompt-guided LLM generation module (including initialization, crossover, and mutation operators) and a downstream evaluation module that scores the generated optimizers to update the population pool.

3.1 Tri-level Evolutionary Search Space

Traditional automated search spaces are often rigidly constrained or overly broad, leading to either limited novelty or high syntactic failure rates [20]. To balance exploration freedom with execution stability, we structuralize the optimizer search space $\mathbb{O}$ into three distinct operational granularities. This allows the evolutionary algorithm to target specific algorithmic components safely. An optimizer instance is defined as a hierarchical triplet $\mathcal{O} = \langle \mathcal{G}_{step}, \mathcal{G}_{util}, \mathcal{G}_{class} \rangle$:

► **Core Functional Level ($\mathcal{G}_{step}$).** This granularity isolates the foundational parameter update equation, predominantly located within the step function. The language model modifies the core algebraic interactions between raw gradients $\nabla \mathcal{L}_t$, historical momentum buffers m_t, and the parameter space Θ_t. A generated update rule at step t can be abstracted as:

$$\Theta_{t+1} = \Theta_t - \eta \cdot \mathcal{G}_{step}\left(\nabla \mathcal{L}_t, m_t, \dots\right) \tag{1}$$

► **Utility Module Level ($\mathcal{G}_{util}$).** This level encompasses independent mathematical subroutines invoked during the optimization process. For instance, finding an optimal orthogonalization projection, computing adaptive gradient clipping thresholds, or designing custom weight decay penalties. By isolating these features into modular functions, the framework can evolve complex mathematical transformations without corrupting the primary training loop.

► **Global Class Level ($\mathcal{G}_{class}$).** The macroscopic level manages the overarching architecture of the optimizer. This includes initializing necessary hyperparameter configurations, allocating optimal state tensors in memory, and topologically orchestrating how the core functions $\mathcal{G}_{step}$ and utility modules $\mathcal{G}_{util}$ interact across different neural network layers.

3.2 Language Model Guided Generation Module

To explore the search space $\mathbb{O}$, we employ a pretrained large language model $\mathcal{L}$ as our primary search engine. Let Π denote the prompt space. The generation module constructs specific instruction templates $P \in \Pi$ to guide $\mathcal{L}$ in generating and refining optimizer algorithms. At any generation t, we maintain a dynamic population of optimizer programs $\mathcal{P}_t = \{\mathcal{O}_1^{(t)}, \ldots, \mathcal{O}_N^{(t)}\}$. The evolution from $\mathcal{P}_t$ to $\mathcal{P}_{t+1}$ is mathematically governed by three distinct stochastic operators applied by $\mathcal{L}$:

Initialization Operator $\mathcal{I}$. At generation $t = 0$, the language model is provided with a task description prompt $P_{task} \in \Pi$ and a knowledge base of standard optimizers $\mathcal{K}$. The initialization operator acts as a stochastic sampling function to generate an initial diverse population of N valid optimizer algorithms:

$$\mathcal{P}_0 = \mathcal{I}(\mathcal{L}, P_{task}, \mathcal{K}) = \left\{ \mathcal{O}_i^{(0)} \mid \mathcal{O}_i^{(0)} \sim \mathcal{L}(\cdot \mid P_{task} \cup \mathcal{K}) \right\}_{i=1}^{N} \tag{2}$$

Crossover Operator $\mathcal{X}$. To synthesize superior offspring, we select highly performant parent algorithms, denoted as $\mathcal{O}_a^{(t)}$ and $\mathcal{O}_b^{(t)}$, from the current population $\mathcal{P}_t$. The language model acts as an intelligent crossover operator mapping the parents into a new child algorithm $\mathcal{O}_{child}^{(t+1)}$. Given a crossover prompt $P_{cross} \in \Pi$, $\mathcal{L}$ comprehends the mathematical logic of both parents to produce a child that inherits their combined strengths:

$$\mathcal{O}_{child}^{(t+1)} = \mathcal{X}\left(\mathcal{L}, \mathcal{O}_a^{(t)}, \mathcal{O}_b^{(t)}, P_{cross}\right) \sim \mathcal{L}\left(\cdot \mid P_{cross} \cup \mathcal{O}_a^{(t)} \cup \mathcal{O}_b^{(t)}\right) \tag{3}$$

Mutation Operator $\mathcal{M}$. To escape local optima, we randomly select an elite individual $\mathcal{O}_{elite}^{(t)} \in \mathcal{P}_t$ and prompt the language model to perform local structural mutations. Let $P_{mut} \in \Pi$ be the mutation prompt. The language model is specifically instructed to mutate the algorithm at one target granularity $\gamma \in \{\mathcal{G}_{step}, \mathcal{G}_{util}, \mathcal{G}_{class}\}$ to explore novel mathematical variants. This transformation is formulated as:

$$\mathcal{O}_{mutated}^{(t+1)} = \mathcal{M}\left(\mathcal{L}, \mathcal{O}_{elite}^{(t)}, \gamma, P_{mut}\right) \sim \mathcal{L}\left(\cdot \mid P_{mut} \cup \mathcal{O}_{elite}^{(t)} \cup \gamma\right) \tag{4}$$

3.3 Training and Evaluation Module

The generated optimizer strategies must interact closely with the neural network training process to verify their effectiveness. For each candidate optimizer $\mathcal{O}_i$ generated by the language model, we instantiate it to train a proxy network f_θ parameterized by $\theta \in \mathbb{R}^d$ on the imbalanced training set $\mathcal{D}_{train}$.

We formulate the training procedure as a discrete dynamical system governed by the candidate optimizer $\mathcal{O}_i$. Given an initial parameter state θ_0, the parameter trajectory at optimization step k is rigorously defined by the mapping:

$$\theta_{k+1} = \mathcal{O}_i\left(\theta_k, \nabla_\theta \mathcal{L}(\theta_k, \mathcal{B}_k), \mathcal{S}_k\right) \tag{5}$$

where $\mathcal{B}_k \subset \mathcal{D}_{train}$ represents the sampled mini batch, $\mathcal{L}$ denotes the empirical objective function, and $\mathcal{S}_k$ maintains the internal optimizer states.

The complete training trajectory after K total steps yields the final optimized parameter state θ_K. To accurately measure the capability of the generated algorithm, we define an evaluation functional Φ that maps the trained network to its performance metric on the validation set $\mathcal{D}_{val}$. Furthermore, to ensure robustness against infeasible programs, we introduce a verification predicate $\mathbb{V}(\mathcal{O}_i) \in \{0, 1\}$ that actively monitors syntactic validity and numerical stability during execution.

The ultimate fitness score $\mathcal{F}(\mathcal{O}_i)$ assigned to the candidate optimizer is explicitly formulated as a conditional function:

$$\mathcal{F}(\mathcal{O}_i) = \begin{cases} \Phi(\theta_K, \mathcal{D}_{val}), & \text{if } \mathbb{V}(\mathcal{O}_i) = 1 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

This rigorous formulation guarantees that any optimizer triggering execution errors or numerical divergence is immediately halted and strictly penalized with a zero score.

3.4 AUTONOMOUS EVOLUTIONARY LOOP

The generation and evaluation modules are coupled into a continuous closed loop dynamical system. Unlike standard evolutionary loops that only evaluate the current generation, we maintain a globally persistent elite reservoir $\mathcal{E}_t$ of a strictly fixed capacity K. This reservoir continuously archives the absolute best algorithms discovered across all historical generations.

At the conclusion of generation t, the newly synthesized population $\mathcal{P}_t$ is completely evaluated to yield a discrete fitness set. We then merge these new candidates with the historical elite reservoir $\mathcal{E}_{t-1}$ inherited from the previous step. A truncation selection operator $\mathcal{S}_{elite}$ extracts the top K performing individuals from this union to dynamically update the active elite reservoir:

$$\mathcal{E}_t = \mathcal{S}_{elite}(\mathcal{E}_{t-1} \cup \mathcal{P}_t, \mathcal{F}, K) = \{\mathcal{O} \in \mathcal{E}_{t-1} \cup \mathcal{P}_t \mid \text{rank}_{\mathcal{F}}(\mathcal{O}) \leq K\} \tag{7}$$

where the initial reservoir $\mathcal{E}_0$ is constructed by selecting the top K programs strictly from $\mathcal{P}_0$.

This fixed capacity reservoir survives deterministically to prevent catastrophic forgetting of previously discovered knowledge. For the subsequent generation $t + 1$, the computational evaluation slots are entirely populated by new offspring synthesized exclusively from this globally optimal archive. We formally define the offspring generation dynamic as a union of the crossover offspring and the mutated variants sampled directly from $\mathcal{E}_t$:

$$\mathcal{P}_{t+1} = \mathcal{X}(\mathcal{L}, \mathcal{E}_t, P_{cross}) \cup \mathcal{M}(\mathcal{L}, \mathcal{E}_t, \gamma, P_{mut}) \tag{8}$$

This iterative update operates without any human intervention and continues until a predefined convergence threshold is satisfied or a maximum generation horizon T_{max} is exhausted. Ultimately, the framework autonomously outputs the supreme optimizer architecture $\mathcal{O}^*$ directly from the pinnacle of the final elite reservoir:

$$\mathcal{O}^* = \arg \max_{\mathcal{O} \in \mathcal{E}_{T_{max}}} \mathcal{F}(\mathcal{O}) \tag{9}$$

4 EXPERIMENT

4.1 EXPERIMENTAL SETTINGS

Datasets. We evaluate our method on four standard long-tailed benchmarks: CIFAR-10-LT, CIFAR-100-LT (IR $\in \{100, 50, 10\}$) [4], ImageNet-LT [14], and iNaturalist 2018 [30]. These encompass both synthetic imbalances and large-scale real-world scenarios. Detailed statistics are provided in **Appendix C.1**. For CIFAR-100-LT, evolutionary fitness and optimizer selection use a balanced subset of the official training split; final accuracy is reported on the separate official test split.

Evaluation Metrics. We report the overall top-1 test accuracy. For fine-grained analysis, we also report accuracy on three splits based on training sample sizes: Many (> 100), Medium ($20 \sim 100$), and Few (< 20).

Compared Optimizers. To rigorously evaluate the effectiveness of our automatically discovered optimizer (**Ours**), we compare it against four representative optimization algorithms: (1) **SGD** [23] with momentum, the standard baseline for training deep neural networks; (2) **SAM** [8], a generic state-of-the-art optimizer that improves generalization by penalizing sharp minima; (3) **Focal-SAM** [11], a specialized long-tailed variant of SAM that dynamically reweights sharpness penalties based on class frequencies; and (4) **Muon** [2; 16], a recently proposed orthogonal optimizer demonstrating strong capability in helping tail classes escape sharp minima.

Orthogonal Training Frameworks. Since Auto-LTOpt operates strictly at the optimization level, it is orthogonal to existing data-level and loss-level interventions. To demonstrate its plug-and-play versatility, we integrate and evaluate all optimizers across a broad spectrum of established long-tailed learning paradigms. These include the standard Cross-Entropy (**CE**), cost-sensitive reweighting methods like Class-Balanced Loss (**CB**) [6] and Deferred Reweighting (**CE-DRW**) [4], logit-adjustment techniques such as Balanced Softmax (**BS**) [21] and Logit Adjustment (**LA**) [17], as well as advanced contrastive learning frameworks including Balanced Contrastive Learning (**BCL**) [41] and Prototypical Contrastive Learning (**ProCo**) [7].

Table 1: Top-1 accuracy (%) (↑) results for *Head*, *Medium*, *Tail* and overall classes on CIFAR-100 LT dataset with imbalance ratios $\rho = 0.1$, $\rho = 0.02$, and $\rho = 0.01$. Best results for the *All* and Δ columns in each group are highlighted in red.

Loss	Method	CIFAR-100 LT $\rho = 0.1$					CIFAR-100 LT $\rho = 0.02$					CIFAR-100 LT $\rho = 0.01$				
		Head	Medium	Tail	All	Δ	Head	Medium	Tail	All	Δ	Head	Medium	Tail	All	Δ
BCL	SGD	67.0	58.0	—	63.4	(+0.0)	65.2	48.9	40.4	53.9	(+0.0)	66.6	52.5	33.4	49.0	(+0.0)
	SAM	67.4	58.3	—	63.7	(+0.3)	65.2	48.4	39.8	53.7	(-0.2)	66.0	52.8	34.5	49.4	(+0.4)
	Focal-SAM	67.6	58.5	—	63.6	(+0.2)	65.4	50.3	40.7	54.6	(+0.7)	66.6	53.8	35.7	50.4	(+1.4)
	Muon	69.2	61.2	—	66.0	(+2.6)	68.0	52.3	44.1	57.3	(+3.4)	70.6	56.1	37.0	52.8	(+3.8)
	Ours	69.8	60.9	—	**66.2**	**(+2.8)**	69.8	51.7	44.3	**57.6**	**(+3.7)**	71.6	56.2	37.5	**53.5**	**(+4.5)**
BS	SGD	67.1	58.2	—	63.5	(+0.0)	65.1	47.6	39.3	53.1	(+0.0)	65.7	51.4	33.6	48.5	(+0.0)
	SAM	67.6	58.7	—	63.8	(+0.3)	64.5	49.1	39.0	53.6	(+0.5)	66.5	52.2	34.7	49.5	(+1.0)
	Focal-SAM	67.7	58.5	—	64.1	(+0.6)	64.4	50.2	40.7	54.3	(+1.2)	65.6	53.7	33.9	49.4	(+0.9)
	Muon	68.5	59.3	—	64.8	(+1.3)	66.4	51.6	43.8	**56.0**	**(+2.9)**	67.8	55.0	35.6	51.1	(+2.6)
	Ours	69.0	59.1	—	**65.0**	**(+1.5)**	66.6	50.6	43.3	55.8	(+2.7)	68.8	55.5	35.3	**51.4**	**(+2.9)**
CB	SGD	70.3	48.5	—	61.6	(+0.0)	70.4	40.0	21.6	48.9	(+0.0)	73.9	49.4	18.1	44.2	(+0.0)
	SAM	71.0	48.3	—	61.9	(+0.3)	70.6	40.9	18.1	48.9	(+0.0)	74.0	50.1	18.4	44.4	(+0.2)
	Focal-SAM	71.2	48.2	—	62.0	(+0.4)	71.7	40.4	19.4	49.4	(+0.5)	75.7	49.8	17.5	44.7	(+0.5)
	Muon	71.6	48.8	—	62.3	(+0.7)	73.1	42.0	20.8	50.8	(+1.9)	76.1	51.9	19.2	46.0	(+1.8)
	Ours	71.4	49.7	—	**62.6**	**(+1.0)**	72.7	43.2	20.9	**51.2**	**(+2.3)**	75.9	53.5	19.1	**46.5**	**(+2.3)**
CE	SGD	70.2	47.3	—	61.1	(+0.0)	70.7	39.6	16.6	48.1	(+0.0)	74.5	50.4	16.2	44.0	(+0.0)
	SAM	70.7	47.8	—	61.5	(+0.4)	70.6	41.1	16.0	48.7	(+0.6)	74.7	50.7	15.7	43.9	(-0.1)
	Focal-SAM	71.0	47.7	—	61.7	(+0.6)	72.0	41.4	16.1	49.4	(+1.3)	75.4	50.6	16.1	44.2	(+0.2)
	Muon	71.5	49.3	—	62.6	(+1.5)	73.0	42.5	18.3	50.7	(+2.6)	76.9	52.0	17.6	45.7	(+1.7)
	Ours	71.5	50.6	—	**63.2**	**(+2.1)**	72.5	43.8	19.2	**51.0**	**(+2.9)**	77.4	53.6	17.1	**46.3**	**(+2.3)**
CE-DRW	SGD	67.4	56.9	—	63.2	(+0.0)	64.3	47.3	36.2	52.3	(+0.0)	67.0	54.1	30.8	48.6	(+0.0)
	SAM	67.0	58.7	—	63.7	(+0.5)	65.0	49.7	36.6	53.6	(+1.3)	65.5	53.6	32.2	48.6	(+0.0)
	Focal-SAM	66.4	58.7	—	63.3	(+0.1)	66.1	49.9	42.2	54.2	(+1.9)	65.1	52.7	34.1	49.0	(+0.4)
	Muon	67.5	59.0	—	64.1	(+0.9)	66.4	51.0	40.6	54.6	(+2.3)	68.7	55.8	32.1	50.2	(+1.6)
	Ours	68.1	60.0	—	**64.8**	**(+1.6)**	67.1	51.2	43.5	**55.2**	**(+2.9)**	68.7	56.2	32.4	**50.4**	**(+1.8)**
LA	SGD	67.2	57.3	—	63.2	(+0.0)	64.8	48.6	38.4	53.3	(+0.0)	66.2	52.3	33.0	48.8	(+0.0)
	SAM	66.9	58.2	—	63.3	(+0.1)	65.5	49.5	39.4	54.2	(+0.9)	65.4	51.3	34.8	48.9	(+0.1)
	Focal-SAM	67.6	58.5	—	64.0	(+0.8)	64.2	50.2	41.8	54.4	(+1.1)	67.1	52.7	34.6	49.8	(+1.0)
	Muon	67.8	58.2	—	63.9	(+0.7)	66.1	50.8	43.0	55.7	(+2.4)	67.3	56.0	35.9	51.3	(+2.5)
	Ours	69.0	59.4	—	**65.1**	**(+1.9)**	67.5	51.5	41.4	**56.3**	**(+3.0)**	68.0	55.6	35.8	**51.4**	**(+2.6)**
ProCo	SGD	68.6	59.2	—	64.8	(+0.0)	66.8	49.6	43.4	55.1	(+0.0)	69.6	55.2	34.3	51.1	(+0.0)
	SAM	68.6	59.8	—	65.0	(+0.2)	65.9	50.9	42.8	55.6	(+0.5)	68.6	53.4	36.0	51.0	(-0.1)
	Focal-SAM	67.4	60.9	—	65.2	(+0.4)	65.9	51.6	41.3	55.3	(+0.2)	67.5	53.2	34.7	50.1	(-1.0)
	Muon	69.7	60.8	—	65.9	(+1.1)	68.0	52.3	45.1	57.3	(+2.2)	70.4	55.7	38.1	53.1	(+2.0)
	Ours	70.0	61.3	—	**66.5**	**(+1.7)**	68.5	52.5	46.5	**57.9**	**(+2.8)**	71.4	58.1	36.5	**53.5**	**(+2.4)**

Implementation Details. To autonomously discover optimization strategies, we utilize the ReEvo [37] framework powered by the `gpt-5-nano` language model, conducting the evolutionary search 100 generations with an initial population of 8, a dynamic population size of 10, and a mutation rate of 0.5. For evaluation, we adopt ResNet-32 for CIFAR-10/100-LT, ResNet-50 for ImageNet-LT, and a pre-trained ResNet-50 for iNaturalist 2018. The CIFAR models are trained using a batch size of 256, a Nesterov momentum of 0.9, and a weight decay of 2×10^{-4}. Additional training details (e.g., learning-rate schedules, head-specific settings, and weights), along with long-tailed distribution statistics, are provided in **Appendix C.1**.

4.2 Benchmark Results

CIFAR-100-LT. Table 1 reports the accuracy on CIFAR-100-LT under varying imbalance ratios ($\rho \in \{0.1, 0.02, 0.01\}$). Our method consistently outperforms all competitors across seven diverse baseline frameworks. Beyond overall accuracy, the results reveal three critical properties: ❶ ***Robustness to Extreme Imbalance.*** The relative advantage amplifies as data scarcity worsens, with gains surging up to +4.5% (e.g., with BCL at $\rho = 0.01$). ❷ ***Mitigating the Head-Tail Trade-off.*** Unlike typical strategies that sacrifice majority classes, our method achieves Pareto improvements by simultaneously boosting both Head and Tail accuracies. ❸ ***Plug-and-play Orthogonality.*** While traditional flat-minima optimizers (e.g., SAM) occasionally degrade under severe imbalance (e.g.,

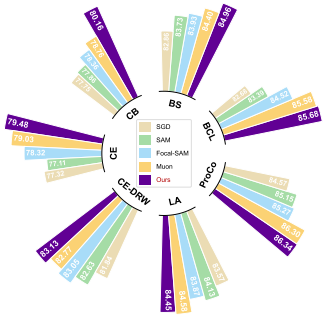
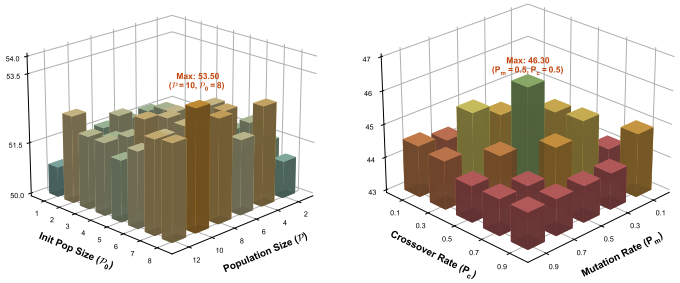


Figure 4: Performance comparison of all classes on the CIFAR10-LT dataset with an Imbalance Ratio (IR) of 100.



(a) Pop sizes & Init pop rates

(b) Crossover & Mutation rates

Figure 5: Hyperparameter analysis regarding (a) the population size $\mathcal{P}$ and initial population size $\mathcal{P}_0$, and (b) crossover rate P_c and mutation rate P_m.

-0.1% in CE at $\rho = 0.01$), our approach maintains stable, state-of-the-art gains across all evaluated loss functions.

CIFAR-100-LT. We evaluate the proposed method on the long-tailed image classification benchmark, CIFAR10-LT, with a severe Imbalance Ratio (IR) of 100. As shown in Figure 4, our method outperforms existing baselines in overall accuracy, demonstrating its robustness to severe data imbalance. Due to space constraints, additional experimental results are deferred to Appendix D.1.

ImageNet-LT and iNaturalist 2018. Table 2 confirms our method's scalability on large-scale datasets. Across diverse baselines (e.g., CE, BS, BCL), it consistently achieves state-of-the-art overall accuracy. Crucially, it delivers balanced gains in Head and Medium classes without sacrificing Tail performance, effectively overcoming the head-tail trade-off in complex, real-world distributions.

Table 2: Top-1 accuracy (%) (↑) results on ImageNet-LT and iNaturalist 2018 datasets. Best results for the *All* columns in each group are highlighted in red.

Loss	Method	ImageNet-LT				iNaturalist 2018			
		Head	Medium	Tail	All	Head	Medium	Tail	All
CE	SGD	64.0	33.8	5.8	41.6 (+0.0)	73.9	63.5	55.5	61.0 (+0.0)
	SAM	64.5	34.5	6.2	42.2 (+0.6)	74.2	64.1	56.0	61.5 (+0.5)
	Focal-SAM	64.9	34.9	6.8	42.7 (+1.1)	74.5	64.6	56.4	62.1 (+1.1)
	Muon	65.8	36.1	9.4	43.9 (+2.3)	74.8	65.8	57.9	62.7 (+1.7)
	Ours	66.5	37.2	10.1	**44.6 (+3.0)**	75.5	66.0	58.5	**63.0 (+2.0)**
BS	SGD	60.9	48.8	32.1	51.0 (+0.0)	65.7	67.4	67.5	67.3 (+0.0)
	SAM	61.5	49.3	32.5	51.3 (+0.3)	66.2	67.9	68.0	67.8 (+0.5)
	Focal-SAM	61.8	49.7	32.8	51.7 (+0.7)	66.5	67.3	68.4	67.8 (+0.5)
	Muon	62.5	51.2	33.6	52.8 (+1.8)	67.4	69.1	70.5	68.8 (+1.5)
	Ours	63.2	51.9	33.5	**53.1 (+2.1)**	67.4	68.8	71.0	**69.2 (+1.9)**
CE-DRW	SGD	61.7	47.3	28.8	50.1 (+0.0)	68.2	67.3	66.4	67.0 (+0.0)
	SAM	62.1	48.0	29.4	50.8 (+0.7)	68.6	67.8	67.0	67.6 (+0.6)
	Focal-SAM	62.4	48.5	29.8	51.3 (+1.2)	68.9	68.4	66.5	67.7 (+0.7)
	Muon	63.0	49.3	30.5	52.2 (+2.1)	69.3	68.8	68.2	68.7 (+1.7)
	Ours	62.8	50.1	30.4	**52.7 (+2.6)**	70.1	69.4	68.4	**69.3 (+2.3)**
BCL	SGD	65.3	53.5	36.3	55.6 (+0.0)	69.4	72.4	71.8	71.8 (+0.0)
	SAM	65.8	54.1	36.9	56.3 (+0.7)	70.5	72.8	72.3	72.2 (+0.4)
	Focal-SAM	66.2	54.5	37.4	56.8 (+1.2)	69.3	73.4	71.8	72.7 (0.9)
	Muon	66.9	55.3	38.5	57.9 (+2.3)	71.3	74.0	72.4	73.4 (+1.6)
	Ours	67.5	57.1	39.6	**58.3 (+2.7)**	71.6	73.9	74.4	**74.0 (+2.2)**

5 Further Analysis

5.1 Hyperparameter Analysis

We evaluate the sensitivity of key evolutionary hyperparameters in Figure 5. Figure 5(a) demonstrates that an appropriate balance between the population size $\mathcal{P}$ and initial size $\mathcal{P}_0$ (e.g., $\mathcal{P} = 10, \mathcal{P}_0 = 8$) maximizes accuracy by ensuring sufficient search diversity. Similarly, Figure 5(b) indicates that a balanced combination of crossover (P_c) and mutation (P_m) rates yields the best performance. Furthermore, Figure 6 illustrates the convergence over iterations. Notably, $\mathcal{G}_{class}$ achieves the highest accuracy compared to $\mathcal{G}_{step}$ and $\mathcal{G}_{util}$. This is because $\mathcal{G}_{class}$ possesses a larger and more fine-grained search space, providing greater expressive capacity to capture complex patterns, which ultimately leads to superior performance upon convergence.

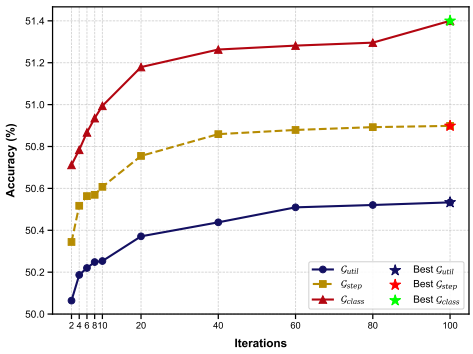
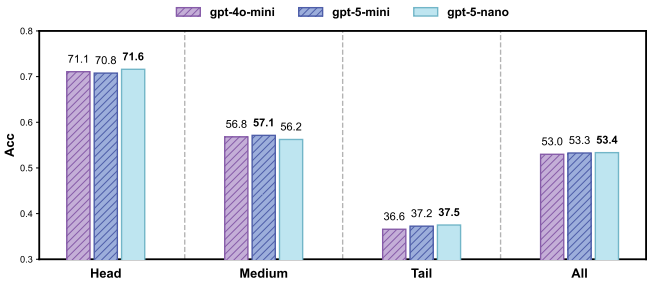


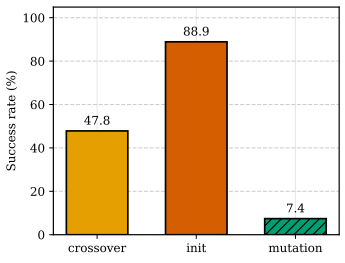
Figure 6: Convergence analysis of different search spaces ($\mathcal{G}_{util}$, $\mathcal{G}_{step}$, $\mathcal{G}_{class}$) over iterations.

5.2 LLM Evolvers and Operator Yield Rates

We evaluate the impact of employing different LLMs (GPT-4o-mini, GPT-5-mini, and GPT-5-nano) as the evolver. As shown in Figure 7a, the choice of LLM exhibits minimal impact on the final convergence and subset accuracy. Our method remains effective with low-cost evolvers such as GPT-5-nano, reducing expenses without sacrificing performance. Figure 7b details the yield rate (i.e., the proportion of executable and verifiable programs). The Initialization operator exhibits the highest success rate (88.9%), as it builds upon safe, established mathematical templates. The Crossover operator maintains a solid yield (47.8%) by conservatively recombining proven code blocks. Conversely, the Mutation operator presents the lowest yield (7.4%). This reflects its aggressive, exploratory nature, which frequently introduces syntax or numerical errors, yet remains mathematically indispensable for helping the algorithm break out of stagnation.



(a) Performance comparison of different LLM evolvers across Head, Medium, and Tail subsets.



(b) Success Rate of Evolutionary Operators

Figure 7: Empirical analysis and performance comparison. **(a)** Performance comparison of different LLM evolvers. **(b)** The yield rate (proportion of executable and verifiable programs) categorized by the three core LLM generation operators: Initialization, Crossover, and Mutation.

6 Conclusion

We proposed Auto-LTOpt, a novel framework that leverages large language models and program evolution to autonomously design optimizers for long-tailed learning without human intervention. Both theoretical analysis and extensive experiments confirm that Auto-LTOpt consistently outperforms state-of-the-art handcrafted baselines, establishing a robust, efficient, and human-free paradigm for imbalanced classification.

AI USE STATEMENT

Large language models are integral to the proposed framework, where they generate and refine candidate optimizers. Their use in preparing this manuscript was limited to improving grammar, clarity, and readability. The authors checked the technical claims and reported results, approved the final text.

REFERENCES

[1] Anonymous. Long-tailed learning with muon optimizer. *OpenReview*, 2025. URL `https://openreview.net/forum?id=go388T3QjQ`.

[2] Jeremy Bernstein and Laker Newhouse. Modular duality in deep learning. In *International Conference on Machine Learning*, pages 3920–3930. PMLR, 2025.

[3] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer, 2010.

[4] Kaidi Cao, Colin Wei, Adrien Gaidon, Nikos Arechiga, and Tengyu Ma. Learning imbalanced datasets with label-distribution-aware margin loss. *Advances in neural information processing systems*, 32, 2019.

[5] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16: 321–357, 2002.

[6] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9268–9277, 2019.

[7] Chaoqun Du, Yulin Wang, Shiji Song, and Gao Huang. Probabilistic contrastive learning for long-tailed visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(9):5890–5904, 2024.

[8] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2020.

[9] Muhammad Abdullah Jamal, Matthew Brown, Ming-Hsuan Yang, Liqiang Wang, and Boqing Gong. Rethinking class-balanced methods for long-tailed visual recognition from a domain adaptation perspective. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7610–7619, 2020.

[10] Yin Lei, Chen Shan, and Wancheng Ge. Edgecasednet: An enhanced detection architecture for edge case perception in autonomous driving. *Plos one*, 20(12):e0338638, 2025.

[11] Sicong Li, Qianqian Xu, Zhiyong Yang, Zitai Wang, Linchao Zhang, Xiaochun Cao, and Qingming Huang. Focal-sam: Focal sharpness-aware minimization for long-tailed classification. *arXiv preprint arXiv:2505.01660*, 2025.

[12] Bo Liu, Haoxiang Li, Hao Kang, Gang Hua, and Nuno Vasconcelos. Breadcrumbs: Adversarial class-balanced sampling for long-tailed recognition. In *European conference on computer vision*, pages 637–653. Springer, 2022.

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

[14] Ziwei Liu, Zhongqi Miao, Xiaohang Zhan, Jiayun Wang, Boqing Gong, and Stella X Yu. Large-scale long-tailed recognition in an open world. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2537–2546, 2019.

[15] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[16] Jiaan Luo, Ziyi Tang, Feng Hong, Qiang Hu, Shuai Xiao, and Jiangchao Yao. Long-tailed learning with muon optimizer, 2026. URL https://openreview.net/forum?id=go388T3QjQ.

[17] Aditya Krishna Menon, Sadeep Jayasumana, Ankit Singh Rawat, Himanshu Jain, Andreas Veit, and Sanjiv Kumar. Long-tail learning via logit adjustment. *arXiv preprint arXiv:2007.07314*, 2020.

[18] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*, 2022.

[19] Alexander Novikov, Ngân Vu, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A gemini-powered coding agent for designing advanced algorithms, 2025.

[20] Esteban Real, Chen Liang, David So, and Quoc Le. Automl-zero: Evolving machine learning algorithms from scratch. In *International conference on machine learning*, pages 8007–8019. Pmlr, 2020.

[21] Jiawei Ren, Cunjun Yu, Xiao Ma, Haiyu Zhao, Shuai Yi, et al. Balanced meta-softmax for long-tailed visual recognition. *Advances in neural information processing systems*, 33:4175–4186, 2020.

[22] Xuanfei Ren, Allen Nie, Tengyang Xie, and Ching-An Cheng. Polca: Stochastic generative optimization with llm. *arXiv preprint arXiv:2603.14769*, 2026.

[23] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[24] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[26] Jiang-Xin Shi, Tong Wei, Yuke Xiang, and Yu-Feng Li. How re-sampling helps for long-tail learning? *Advances in Neural Information Processing Systems*, 36:75669–75687, 2023.

[27] Jun Shu, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, and Zongben Xu. Meta-weight-net: Learning an explicit mapping for sample weighting. In *Advances in neural information processing systems*, volume 32, 2019.

[28] Nikola Surjanovic, Alexandre Bouchard-Côté, and Trevor Campbell. Autogd: Automatic learning rate selection for gradient descent. *arXiv preprint arXiv:2510.09923*, 2025.

[29] Jingru Tan, Changbao Wang, Buyu Li, Quanquan Li, Wanli Ouyang, Changqing Yin, and Junjie Yan. Equalization loss for long-tailed object recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11662–11671, 2020.

[30] Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie. The inaturalist species classification and detection dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8769–8778, 2018.

[31] Binwu Wang, Pengkun Wang, Wei Xu, Xu Wang, Yudong Zhang, Kun Wang, and Yang Wang. Kill two birds with one stone: Rethinking data augmentation for deep long-tailed learning. In *the twelfth international conference on learning representations*, 2024.

[32] Pengkun Wang, Zhe Zhao, HaiBin Wen, Fanfu Wang, Binwu Wang, Qingfu Zhang, and Yang Wang. Llm-autoda: Large language model-driven automatic data augmentation for long-tailed problems. *Advances in Neural Information Processing Systems*, 37:64915–64941, 2024.

[33] Yiru Wang, Weihao Gan, Jie Yang, Wei Wu, and Junjie Yan. Dynamic curriculum learning for imbalanced data classification. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5017–5026, 2019.

[34] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[35] Zhichuan Xu, Jie Song, Cheng Bi, Yuxin Zhang, Xin Zheng, Meng Xiao, Jiale Du, Rongrong Wu, and Bairong Shen. Computer vision for rare diseases: A scoping review. 2025.

[36] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2023.

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

[38] Boyan Zhou, Quan Cui, Xiu-Shen Wei, and Zhao-Min Chen. Bbn: Bilateral-branch network with cumulative learning for long-tailed visual recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9719–9728, 2020.

[39] Yixuan Zhou, Yi Qu, Xing Xu, and Hengtao Shen. Imbsam: A closer look at sharpness-aware minimization in class-imbalanced recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11345–11355, 2023.

[40] Zhipeng Zhou, Lanqing Li, Peilin Zhao, Pheng-Ann Heng, and Wei Gong. Class-conditional sharpness-aware minimization for deep long-tailed recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3499–3509, 2023.

[41] Jianggang Zhu, Zheng Wang, Jingjing Chen, Yi-Ping Phoebe Chen, and Yu-Gang Jiang. Balanced contrastive learning for long-tailed visual recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6908–6917, 2022.

Appendix

Table of Contents

A Preliminaries **14**

B Theoretical Deduction **15**

B.1 Convergence of Optimizer Evolution 15

B.2 Expressive Capacity and Search Space Granularity 16

C Experimental Setup Details **17**

C.1 Detailed Implementation and Training Setup 17

C.2 Comparison Baselines 19

D Additional Experimental Results **21**

D.1 Performance on CIFAR-10-LT 21

D.2 Implementation of the Discovered Optimizer 21

D.3 Ablation Study of the Evolved Components 23

F Limitations and Future Work **28**

A PRELIMINARIES

Consider a training dataset $\mathcal{D}_{train} = \{(x_i, y_i)\}_{i=1}^N$ over label space $\mathcal{Y} = \{1, \dots, C\}$. Let $N_c = \sum_{i=1}^N \mathbb{I}(y_i = c)$ denote the sample size for class c, and $\mathcal{L}_c(\theta) = \mathbb{E}_{x|y=c}[\ell(f_\theta(x), c)]$ denote the expected loss for class c. Assuming monotonically decreasing frequencies $N_1 \geq N_2 \geq \cdots \geq N_C$, a long-tailed distribution exhibits an extreme imbalance ratio:

$$\rho = \frac{N_1}{N_C} \gg 1. \tag{10}$$

Under Empirical Risk Minimization, the objective for a model f_θ parameterized by θ is inherently weighted by class frequencies:

$$\min_\theta \mathcal{L}(\theta) = \sum_{c=1}^C \frac{N_c}{N} \mathcal{L}_c(\theta). \tag{11}$$

For any partition of the label space into disjoint sets $\mathcal{C}_{head}$ and $\mathcal{C}_{tail}$, class frequencies weight their contributions to the empirical gradient differently:

$$\nabla\mathcal{L}(\theta) = \sum_{c\in\mathcal{C}_{head}} \frac{N_c}{N} \nabla\mathcal{L}_c(\theta) + \underbrace{\sum_{c\in\mathcal{C}_{tail}} \frac{N_c}{N} \nabla\mathcal{L}_c(\theta)}_{\text{tail contribution}}. \tag{12}$$

Standard gradient descent applies the update $\theta_{t+1} = \theta_t - \eta\nabla\mathcal{L}(\theta)$. The frequency weighting can reduce the contribution of sufficiently rare classes when their gradient norms are controlled, motivating adaptive optimization strategies.

To quantify the effect of class frequencies on the empirical gradient, we introduce the following lemmas.

Assumption A.1 (Smoothness and Lower Boundedness)**.** *We assume that the expected loss function for each class c, denoted as $\mathcal{L}_c(\theta)$, is L-smooth and lower bounded. Specifically, there exists a constant $L > 0$ such that for any $\theta, \theta' \in \mathbb{R}^d$:*

$$\|\nabla\mathcal{L}_c(\theta) - \nabla\mathcal{L}_c(\theta')\| \leq L\|\theta - \theta'\|. \tag{13}$$

Furthermore, there exists a global minimum $\mathcal{L}^* > -\infty$ such that $\mathcal{L}(\theta) \geq \mathcal{L}^*$ for all θ.

Lemma A.2 (Gradient Skewness Bound)**.** *Let $\nabla\mathcal{L}_{bal}(\theta) = \frac{1}{C}\sum_{c=1}^C \nabla\mathcal{L}_c(\theta)$ represent the ideal, class-balanced expected gradient. Under the empirical risk minimization with an imbalance ratio $\rho = N_1/N_C \gg 1$, the deviation of the standard empirical gradient $\nabla\mathcal{L}(\theta)$ from the balanced gradient is strictly bounded by the distribution skewness:*

$$\|\nabla\mathcal{L}(\theta) - \nabla\mathcal{L}_{bal}(\theta)\| \leq \sum_{c=1}^C \left|\frac{N_c}{N} - \frac{1}{C}\right| \cdot \|\nabla\mathcal{L}_c(\theta)\|. \tag{14}$$

Proof Outline. This follows directly from the triangle inequality and the definition of the expected empirical gradient. Since head classes have $N_c/N \gg 1/C$, their gradient norms disproportionately dominate the optimization trajectory, pulling the parameter updates away from the ideal balanced manifold. □

Lemma A.3 (Tail Gradient Suppression)**.** *Let $N_1 = N_{\max}$, $N_C = N_{\min}$, and $\rho = N_1/N_C$. For a constant $\kappa \geq 1$, define $\mathcal{C}_{tail} = \{c : N_c \leq \kappa N_C\}$. With $p_c = N_c/N$ and learning rate $\eta > 0$, the class-frequency-weighted tail contribution satisfies*

$$\mathbb{E}\left[\left\|\eta \sum_{c\in\mathcal{C}_{tail}} p_c \nabla\mathcal{L}_c(\theta_t)\right\|\right] \leq \frac{\eta\kappa|\mathcal{C}_{tail}|}{\rho}\, \mathbb{E}\left[\sup_{c\in\mathcal{C}_{tail}} \|\nabla\mathcal{L}_c(\theta_t)\|\right]. \tag{15}$$

Proof. Since $N \geq N_1 = \rho N_C$, every $c \in \mathcal{C}_{tail}$ satisfies $p_c = N_c/N \leq \kappa N_C/N \leq \kappa/\rho$. The bound follows from the triangle inequality and $\sum_{c\in\mathcal{C}_{tail}} \|\nabla\mathcal{L}_c(\theta_t)\| \leq |\mathcal{C}_{tail}| \sup_{c\in\mathcal{C}_{tail}} \|\nabla\mathcal{L}_c(\theta_t)\|$. □

Remark. For the least frequent class, $1/(\rho C) \leq p_C \leq 1/\rho$; the lower inequality is strict for a nonuniform distribution. The bound in Lemma A.3 gives an $\mathcal{O}(1/\rho)$ rate only when κ, $|\mathcal{C}_{tail}|$, η, and the expected supremum of the tail-class gradient norms remain bounded as ρ varies. It does not by itself establish a limitation of momentum or the necessity of a particular optimizer modification.

Lemma A.4 (Imbalanced Descent Lemma)**.** *Define the head-class aggregated gradient* $\nabla\mathcal{L}_{head}(\theta_t) = \sum_{c\in\mathcal{C}_{head}} p_c \nabla\mathcal{L}_c(\theta_t)$. *Under Assumption A.1, for a uniform optimization step* $\theta_{t+1} = \theta_t - \eta\nabla\mathcal{L}(\theta_t)$*, the classical descent lemma yields the following inequality separated into head and tail components:*

$$\begin{aligned}\mathcal{L}(\theta_{t+1}) \leq \mathcal{L}(\theta_t) &- \eta\|\nabla\mathcal{L}_{head}(\theta_t)\|^2 \\ &- 2\eta\left(\sum_{c\in\mathcal{C}_{tail}} p_c\langle\nabla\mathcal{L}_c(\theta_t), \nabla\mathcal{L}_{head}(\theta_t)\rangle\right) \\ &- \eta\|\nabla\mathcal{L}_{tail}(\theta_t)\|^2 + \frac{L\eta^2}{2}\|\nabla\mathcal{L}(\theta_t)\|^2.\end{aligned} \tag{16}$$

Remark. The cross-inner-product term $\langle\nabla\mathcal{L}_c(\theta_t), \nabla\mathcal{L}_{head}(\theta_t)\rangle$ demonstrates that if tail gradients conflict with the dominant head gradients (i.e., the inner product is negative), the loss for tail classes can strictly increase during training. This mathematically justifies why exploring operations like gradient orthogonalization in our search space $\mathcal{G}_{util}$ is highly critical.

B Theoretical Deduction

B.1 Convergence of Optimizer Evolution

In practice, commonly used performance metrics for evaluation are typically bounded, such as classification accuracy and the F_1-score. Under this observation, we can conclude that the best fitness sequence, defined as below, converges along the evolutionary process.

Proposition B.1. *Assume that the evaluation function* $\Phi : \mathbb{O} \to \mathbb{R}$ *is bounded on the validation set* $\mathcal{D}_{val}$*. We define the best fitness at generation* t *as*

$$B_t \coloneqq \max_{\mathcal{O}\in\mathcal{E}_t} \mathcal{F}(\mathcal{O}). \tag{17}$$

Then the best fitness sequence $\{B_t\}_{t\geq 1}$ *converges.*

Proof. By the elitist update rule (7), the elite reservoir $\mathcal{E}_t$ consists of the K highest-fitness individuals in $\mathcal{E}_{t-1} \cup \mathcal{P}_t$. Since $\mathcal{E}_{t-1} \subset \mathcal{E}_{t-1} \cup \mathcal{P}_t$, we have

$$\max_{\mathcal{O}\in\mathcal{E}_{t-1}\cup\mathcal{P}_t} \mathcal{F}(\mathcal{O}) \geq \max_{\mathcal{O}\in\mathcal{E}_{t-1}} \mathcal{F}(\mathcal{O}) = B_{t-1}. \tag{18}$$

In particular, $\mathcal{E}_t$ contains an optimizer attaining $\max_{\mathcal{O}\in\mathcal{E}_{t-1}\cup\mathcal{P}_t} \mathcal{F}(\mathcal{O})$, therefore

$$B_t = \max_{\mathcal{O}\in\mathcal{E}_t} \mathcal{F}(\mathcal{O}) = \max_{\mathcal{O}\in\mathcal{E}_{t-1}\cup\mathcal{P}_t} \mathcal{F}(\mathcal{O}) \geq B_{t-1}. \tag{19}$$

Hence, the best fitness sequence $\{B_t\}_{t\geq 1}$ is monotonically non-decreasing.

Obviously, the sequence $\{B_t\}_{t\geq 1}$ is bounded if Φ is bounded. Therefore, the desired conclusion follows immediately from the monotone convergence theorem. □

Proposition B.1 establishes the convergence of $\{B_t\}_{t\geq 1}$, and we now investigate its limit and the corresponding optimal optimizer. We first define $\mathcal{O}_{val} = \{\mathcal{O} \in \mathbb{O} \mid \mathbb{V}(\mathcal{O}) = 1\}$ as the admissible set of optimizers. Suppose that the set of globally optimal optimizers $\mathcal{G} = \{\mathcal{O} \in \mathcal{O}_{val} \mid \mathcal{F}(\mathcal{O}) = \mathcal{F}^*\}$ is non-empty, where $\mathcal{F}^* \coloneqq \sup_{\mathcal{O}\in\mathcal{O}_{val}} \mathcal{F}(\mathcal{O})$ is the optimum (best performance metric).

Assumption B.2. *(Uniform positive probability of hitting the optimal set) There exists a constant* $p > 0$*, such that for any generation* t*, the following probability inequality holds:*

$$\mathbb{P}(\mathcal{P}_{t+1} \cap \mathcal{G} \neq \varnothing \mid \mathcal{E}_t \cap \mathcal{G} = \varnothing) \geq p. \tag{20}$$

Assumption B.2 states that, as long as the elite reservoir $\mathcal{E}_t$ has not yet discovered any globally optimal optimizer, the generation module retains a uniformly positive probability of producing at least one element of $\mathcal{G}$ at each generation.

The following theorem characterizes that, as the evolution proceeds, our designed AutoLTOpt converges to the optimal performance and is able to identify an optimal optimizer.

Theorem B.3. *Under the Assumption B.2, it holds that*

$$B_t \xrightarrow{a.s.} \mathcal{F}^*. \tag{21}$$

Moreover, let $\widehat{\mathcal{O}}_t \in \arg\max_{\mathcal{O}\in\mathcal{E}_t} \mathcal{F}(\mathcal{O})$. *Then*

$$\exists\, T > 0,\ \forall\, t \geq T,\ \widehat{\mathcal{O}}_t \in \mathcal{G}, a.s. \tag{22}$$

which also implies that $\lim_{t\to\infty} \mathbb{P}(\widehat{\mathcal{O}}_t \in \mathcal{G}) = 1$.

Proof. We first define the failure event $A_t \coloneqq \{\mathcal{E}_t \cap \mathcal{G} = \varnothing\}$, i.e., the event that the elite reservoir $\mathcal{E}_t$ does not contain any globally optimal optimizer at generation t. On the event A_t, if the next population $\mathcal{P}_{t+1}$ contains an element of $\mathcal{G}$, then by the elitist truncation selection rule, such an optimal optimizer must be included in $\mathcal{E}_{t+1}$. Hence,

$$\mathbb{P}(A_{t+1} \mid A_t) \leq 1 - \mathbb{P}(\mathcal{P}_{t+1} \cap \mathcal{G} \neq \varnothing \mid A_t) \leq 1 - p. \tag{23}$$

By elitist preservation, once a globally optimal optimizer enters the elite reservoir, at least one such optimizer remains in all subsequent generations. We therefore have $A_{t+1} \subset A_t$. Then it follows that

$$\mathbb{P}(A_{t+1}) = \mathbb{P}(A_{t+1} \cap A_t) = \mathbb{P}(A_t)\mathbb{P}(A_{t+1} \mid A_t) \leq (1-p)\mathbb{P}(A_t). \tag{24}$$

By induction,

$$\mathbb{P}(A_t) \leq (1-p)^t \mathbb{P}(A_0) \leq (1-p)^t, \tag{25}$$

which implies $\mathbb{P}(A_t) \to 0$ as $t \to \infty$. By continuity of probability measures for decreasing events, we have

$$\mathbb{P}\Big(\bigcap_{t=0}^{\infty} A_t\Big) = \lim_{t\to\infty} \mathbb{P}(A_t) = 0. \tag{26}$$

The event $\bigcap_{t=0}^{\infty} A_t$ corresponds to the event that no globally optimal optimizer is ever discovered. Note that its complement is

$$\Big(\bigcap_{t=0}^{\infty} A_t\Big)^c = \bigcup_{t=0}^{\infty} A_t^c \iff \exists\, T > 0,\ \forall\, t \geq T,\ \widehat{\mathcal{O}}_t \in \mathcal{G}. \tag{27}$$

Hence, Eq. (22) holds. This also indicates that $\mathbb{I}_{A_t^c} \xrightarrow{\text{a.s.}} 1$. Since almost sure convergence implies convergence in probability, we also have $\mathbb{I}_{A_t^c} \xrightarrow{\mathbb{P}} 1$, and hence $\mathbb{P}(A_t^c) \to 1$. Furthermore, $A_t^c = \{\mathcal{E}_t \cap \mathcal{G} \neq \varnothing\} \iff \Big\{\widehat{\mathcal{O}}_t \in \mathcal{G}\Big\}$, we can obtain $\lim_{t\to\infty} \mathbb{P}(\widehat{\mathcal{O}}_t \in \mathcal{G}) = 1$.

Finally, for all $t > T$, events A^c imply that there exists $\mathcal{O}_t^* \in \mathcal{E}_t \cap \mathcal{G}$ such that $\mathcal{F}(\mathcal{O}_t^*) = \mathcal{F}^*$. Therefore, $B_t \geq \mathcal{F}^*$. On the other hand, by definition of $\mathcal{F}^*$, we always have $B_t \leq \mathcal{F}^*$. Thus,

$$\exists\, T > 0,\ \forall\, t \geq T,\ B_t = \mathcal{F}^*, a.s. \tag{28}$$

This gives the convergence (21). □

B.2 Expressive Capacity and Search Space Granularity

In our framework, the search space $\mathbb{O}$ is structurally partitioned into three distinct granularities: the utility module level $\mathcal{G}_{util}$, the core functional level $\mathcal{G}_{step}$, and the global class level $\mathcal{G}_{class}$. Empirically, as shown in Section 5.1, searching within $\mathcal{G}_{class}$ yields the highest terminal performance. We now provide a theoretical justification for this observation by analyzing the expressive capacity of these hierarchical search spaces.

Let $\mathbb{O}_{util}$, $\mathbb{O}_{step}$, and $\mathbb{O}_{class}$ denote the set of all valid optimizer programs that can be generated by mutating the utility functions, the core step functions, and the entire optimizer class, respectively. By definition, a mutation at the global class level can encompass any local modification at the step or utility levels, such as redefining internal state tensors alongside custom utility mathematical projections. This establishes a strict hierarchical nesting of the search spaces.

Lemma B.4 (Hierarchical Inclusion of Search Spaces)**.** *The multi-granularity search spaces form a nested topological structure, meaning the representational capacity of lower-level spaces is strictly subsumed by higher-level spaces:*

$$\mathbb{O}_{util} \subset \mathbb{O}_{step} \subset \mathbb{O}_{class}. \tag{29}$$

Building upon this inclusion property, we demonstrate that expanding the actionable dimension of the search space monotonically increases the theoretical upper bound of the optimizer's performance.

Proposition B.5 (Monotonicity of Optimal Fitness)**.** *Let $\mathcal{F}^*_\gamma \coloneqq \sup_{\mathcal{O}\in\mathbb{O}_\gamma} \mathcal{F}(\mathcal{O})$ denote the globally optimal fitness achievable within a specific search space granularity $\gamma \in \{util, step, class\}$. It rigorously holds that:*

$$\mathcal{F}^*_{util} \leq \mathcal{F}^*_{step} \leq \mathcal{F}^*_{class}. \tag{30}$$

Proof. For any valid optimizer configuration $\mathcal{O} \in \mathbb{O}_{util}$, Lemma B.4 dictates that $\mathcal{O}$ is also a valid element of $\mathbb{O}_{step}$. Consequently, the supremum of the evaluation functional $\mathcal{F}$ over the broader set $\mathbb{O}_{step}$ must be at least as large as its supremum over the restricted subset $\mathbb{O}_{util}$. Thus:

$$\sup_{\mathcal{O}\in\mathbb{O}_{util}} \mathcal{F}(\mathcal{O}) \leq \sup_{\mathcal{O}\in\mathbb{O}_{step}} \mathcal{F}(\mathcal{O}) \implies \mathcal{F}^*_{util} \leq \mathcal{F}^*_{step}. \tag{31}$$

By applying the identical algebraic logic to the superset relation between $\mathbb{O}_{step}$ and $\mathbb{O}_{class}$, we trivially obtain $\mathcal{F}^*_{step} \leq \mathcal{F}^*_{class}$. Combining these inequalities completes the proof. □

Remark. Proposition B.5 mathematically confirms that a higher-level structural search space ($\mathcal{G}_{class}$) possesses strictly greater representational power to model complex, long-tailed gradient dynamics than localized mutations ($\mathcal{G}_{util}$ or $\mathcal{G}_{step}$). While a larger search space inherently introduces a more complex optimization landscape, the elite-preserving evolutionary loop formulated in Theorem B.3 guarantees that the generation module can avoid performance degradation and progressively approximate this higher theoretical upper bound $\mathcal{F}^*_{class}$. This provides a solid mathematical foundation for the superior convergence accuracy observed in our empirical evaluations.

C Experimental Setup Details

C.1 Detailed Implementation and Training Setup

C.1.1 Hardware and Evolutionary Search

All experiments, including the LLM-driven optimizer evolution and downstream training evaluations, are implemented in PyTorch and executed on a single NVIDIA A800-SXM4-80GB GPU. For the automated discovery process, we utilize the ReEvo [37] framework powered by the `gpt-5-nano` language model. The evolutionary search is conducted over 100 generations. To balance exploration and exploitation, we configure an initial population size of 8, maintain a dynamic population size of 10 throughout the evolutionary process, and apply a mutation rate of 0.5.

C.1.2 Datasets and Preprocessing

Table 3: Detailed statistics of the datasets used in our experiments. For CIFAR-100-LT, # Train denotes the original training split before validation holdout and long-tail subsampling. The imbalance ratio ρ denotes N_{max}/N_{min}.

Dataset	# Classes	# Train	# Test	Imbalance Ratio (ρ)
CIFAR-10-LT	10	50K	10K	**100, 50, 10**
CIFAR-100-LT	100	50K	10K	**100, 50, 10**
ImageNet-LT	1,000	115.8K	50K	**256**
iNaturalist 2018	8,142	437.5K	24.4K	**500**

As detailed in table 3, we extensively evaluate our method on four benchmark datasets representing both synthetic and real-world long-tailed distributions. The statistics of these datasets are summarized in the standard protocols below.

CIFAR-10-LT and CIFAR-100-LT. These datasets are synthetically imbalanced versions derived from the original CIFAR-10 and CIFAR-100 datasets. Following standard construction protocols [4], the training set is created by reducing the number of samples per class according to an exponential decay function $n_k = n_{head} \cdot \mu^k$, where k denotes the class index. We conduct experiments under three specific Imbalance Factors (IF), defined as the ratio between the most frequent and least frequent classes ($N_{\max}/N_{\min}$), set to 100, 50, and 10, respectively. The image resolution is 32×32 pixels. Note that the validation set remains balanced to ensure fair evaluation across all classes.

CIFAR-100-LT data usage. The long-tailed training data are used to train each candidate optimizer's proxy network. Evolutionary fitness is computed on a class-balanced validation set sampled from the original CIFAR-100 training split, using `balanced_val=True` and `val_samples_per_class=100`. This yields 100 validation images per class (10,000 in total). The sampled validation images are excluded from every long-tailed training subset, so training and validation have no images in common. The same validation scores determine which candidates enter the elite reservoir and which optimizer is selected at the end of evolution. The official CIFAR-100 test split is separate from both training and validation; it is used for final top-1 accuracy reporting and is not used for evolutionary fitness or optimizer selection.

ImageNet-LT. ImageNet-LT is a large-scale long-tailed subset sampled from the ImageNet-2012 classification dataset [14]. It simulates a Pareto distribution with a power value $\alpha = 6$, containing 115,846 training images spanning 1,000 categories. The dataset exhibits severe class imbalance with an imbalance ratio of 256, where the class frequency ranges drastically from a maximum of 1,280 images to a minimum of 5 images. The validation set consists of 50,000 balanced images.

iNaturalist 2018. To verify the effectiveness of our method on real-world imbalance, we utilize the iNaturalist 2018 dataset [30]. Unlike the synthetic benchmarks above, iNaturalist 2018 presents a naturally occurring extremely long-tailed distribution based on real-world species frequency. It is a fine-grained classification dataset containing 437,513 training images from 8,142 species, with a massive imbalance ratio of 500. We utilize the official validation set comprising 24,426 images (3 per class) for evaluation.

Preprocessing and Augmentation. For **CIFAR-10/100-LT**, we apply standard data augmentation including random cropping with a padding of 4 pixels and random horizontal flipping. For the large-scale datasets (**ImageNet-LT** and **iNaturalist 2018**), we employ standard data augmentation including random resized cropping to 224×224 and horizontal flipping during the training stage. During inference, images are resized to 256×256 followed by a center crop of 224×224. All input images are normalized using the dataset-specific mean and standard deviation.

C.1.3 Network Architectures and Optimization Protocol

For a fair comparison with existing state-of-the-art baselines, we strictly adhere to standard network selections and optimization hyperparameters.

Backbone Networks: We adopt ResNet-32 for CIFAR-10-LT and CIFAR-100-LT, ResNet-50 for ImageNet-LT, and a pre-trained ResNet-50 for iNaturalist 2018.

Training Hyperparameters: All models are trained using the generated **Auto-LTOpt** as well as the baseline optimizers equipped with Nesterov momentum of 0.9 and a weight decay of 2×10^{-4}. We set the mini-batch sizes to 256 for the CIFAR datasets, 128 for ImageNet-LT, and 512 for iNaturalist 2018.

Learning Rate Schedules: For the CIFAR datasets, models are trained for 200 epochs. The initial learning rate is set to 0.1 and follows a multi-step decay schedule, decaying by a factor of 0.1 at epochs 160 and 180 (i.e., decayed to 0.01 and 0.001, respectively). For ImageNet-LT and iNaturalist 2018, we employ standard cosine annealing schedules.

C.1.4 REPRODUCIBILITY

We report top-1 accuracy on the official test splits. The full source code, including the LLM generation prompts, evolutionary framework, and the exact discovered PyTorch optimizer implementations, will be made publicly available upon publication.

C.2 COMPARISON BASELINES

To comprehensively evaluate the effectiveness of our proposed Auto-LTOpt, we establish a rigorous evaluation protocol encompassing two distinct dimensions: orthogonal training frameworks (loss functions) and state-of-the-art optimizers.

Orthogonal Training Frameworks. Since Auto-LTOpt operates exclusively at the optimization level, it can be seamlessly integrated with existing data-level and loss-level interventions. We evaluate all optimizers across the following representative long-tailed learning paradigms:

- ❶ ***Cross-Entropy Loss (CE).*** The standard baseline for classification tasks. It treats all classes and samples equally without accounting for the imbalanced distribution, often resulting in significant overfitting on frequent head classes and poor generalization on tail classes.
- ❷ ***Deferred Reweighting (CE-DRW) [4].*** Improves upon standard CE by introducing a two-stage training schedule. It trains normally in the initial stage to learn robust representations, and applies class-balanced reweighting in the second stage to shift the model's focus towards minority classes.
- ❸ ***Class-Balanced Loss (CB) [6].*** A cost-sensitive reweighting method that adjusts the loss based on the effective number of samples for each class, smoothly bridging the gap between uniform weighting and inverse class frequency weighting.
- ❹ ***Balanced Softmax (BS) [21].*** Addresses the label bias problem from a theoretical perspective. It derives an unbiased loss function by adjusting the logits based on the prior class frequencies during training, explicitly accommodating the long-tailed distribution.
- ❺ ***Logit Adjustment (LA) [17].*** A margin-based approach that modifies the network's logits by subtracting a class-frequency-dependent offset, enforcing a larger decision margin for tail classes to compensate for their lack of training samples.
- ❻ ***Balanced Contrastive Learning (BCL) [41].*** Leverages supervised contrastive learning to learn balanced feature representations. It ensures that all classes, regardless of their frequency, have well-separated feature clusters by optimizing sample-to-class margins in the embedding space.
- ❼ ***Prototypical Contrastive Learning (ProCo) [7].*** An advanced contrastive framework that estimates probabilistic prototypes for each class to augment the contrastive loss, effectively mitigating the representation bias caused by extreme class imbalance.

Compared Optimizers. To demonstrate the direct superiority of our autonomously evolved optimizer, we compare **Auto-LTOpt** against the following fundamental and state-of-the-art optimization algorithms:

- ❶ ***SGD [23].*** The standard Stochastic Gradient Descent with Nesterov momentum, serving as the universal baseline for training deep neural networks.
- ❷ ***SAM [8].*** Sharpness-Aware Minimization, a generic state-of-the-art optimizer that improves generalization by explicitly penalizing sharp minima and searching for uniformly low loss neighborhoods.
- ❸ ***Focal-SAM [11].*** A specialized long-tailed variant of SAM that dynamically reweights the sharpness penalties based on class frequencies to prevent the suppression of tail classes.
- ❹ ***Muon [2].*** A recently proposed orthogonal optimizer that leverages Newton-Schulz iterations. It has demonstrated strong empirical capabilities in helping tail classes escape sharp local minima.

Table 4: Top-1 accuracy (%) (↑) results for *Head*, *Medium*, *Tail* and overall classes on CIFAR-10 LT dataset with imbalance factor (IF) of 100. Best results in the last two columns are highlighted in red.

Loss	**Method**	**CIFAR-10 LT IF = 100**				
		Head	Medium	Tail	All	Δ
BCL	SGD	84.19	76.65	—	82.68	(+0.00)
	SAM	84.33	79.65	—	83.39	(+0.71)
	Focal-SAM	85.25	81.60	—	84.52	(+1.84)
	Muon	86.76	80.85	—	85.58	(+2.90)
	Ours	87.04	80.25	—	**85.68**	**(+3.00)**
BS	SGD	84.17	77.60	—	82.86	(+0.00)
	SAM	84.36	81.20	—	83.73	(+0.87)
	Focal-SAM	83.84	84.30	—	83.93	(+1.07)
	Muon	85.12	81.65	—	84.40	(+1.54)
	Ours	85.41	83.15	—	**84.96**	**(+2.10)**
CB	SGD	82.88	57.25	—	77.75	(+0.00)
	SAM	82.54	59.15	—	77.86	(+0.11)
	Focal-SAM	82.50	61.80	—	78.36	(+0.61)
	Muon	83.75	58.80	—	78.76	(+1.01)
	Ours	83.78	65.70	—	**80.16**	**(+2.41)**
CE	SGD	81.75	59.60	—	77.32	(+0.00)
	SAM	82.41	55.90	—	77.11	(-0.21)
	Focal-SAM	82.88	61.20	—	78.32	(+1.00)
	Muon	83.11	62.70	—	79.03	(+1.71)
	Ours	83.28	64.30	—	**79.48**	**(+2.16)**
CE-DRW	SGD	83.31	75.95	—	81.84	(+0.00)
	SAM	83.97	78.05	—	82.63	(+0.79)
	Focal-SAM	84.00	79.30	—	83.05	(+1.21)
	Muon	84.74	74.90	—	82.77	(+0.93)
	Ours	84.23	78.75	—	**83.13**	**(+1.29)**
LA	SGD	84.14	81.35	—	83.57	(+0.00)
	SAM	84.21	83.80	—	84.13	(+0.56)
	Focal-SAM	83.84	84.00	—	83.87	(+0.30)
	Muon	85.60	80.50	—	**84.58**	**(+1.01)**
	Ours	85.35	80.85	—	84.45	(+0.88)
ProCo	SGD	85.95	79.05	—	84.57	(+0.00)
	SAM	86.15	81.15	—	85.15	(+0.58)
	Focal-SAM	85.80	83.15	—	85.27	(+0.70)
	Muon	87.39	81.95	—	86.30	(+1.73)
	Ours	87.35	82.30	—	**86.34**	**(+1.77)**

D ADDITIONAL EXPERIMENTAL RESULTS

In this section, we provide extended empirical evaluations to complement the main findings. We focus specifically on the performance dynamics of our autonomously discovered optimizer when applied to the CIFAR-10-LT dataset under severe class imbalance.

D.1 PERFORMANCE ON CIFAR-10-LT

To rigorously assess the generalizability of Auto-LTOpt on smaller-scale but highly skewed distributions, we conduct experiments on the CIFAR-10-LT benchmark with an Imbalance Factor (IF) of 100. The results, detailing the Top-1 accuracy across various orthogonal training frameworks, are comprehensively summarized in Table 4.

Consistent Superiority Across Frameworks. The empirical results demonstrate that Auto-LTOpt (**Ours**) consistently achieves state-of-the-art performance across the vast majority of evaluated loss functions. When compared to the standard SGD baseline, our optimizer provides substantial and stable performance lifts (Δ). Notably, under the Balanced Contrastive Learning (BCL) framework, Auto-LTOpt achieves a remarkable +3.00% absolute improvement over SGD, reaching an overall accuracy of 85.68%. Similar significant gains are observed with Class-Balanced Loss (CB) and standard Cross-Entropy (CE), yielding improvements of +2.41% and +2.16%, respectively.

Comparison with Advanced Optimizers. Beyond the standard baseline, Auto-LTOpt proves highly competitive against advanced, manually crafted optimizers. While SAM and Focal-SAM provide marginal improvements (and occasionally degrade performance, as seen in CE), Auto-LTOpt consistently outperforms them. Furthermore, compared to the recent orthogonal optimizer, Muon, our method secures the top position in six out of the seven evaluated paradigms. For instance, in the Balanced Softmax (BS) and Prototypical Contrastive Learning (ProCo) settings, Auto-LTOpt surpasses Muon by +0.56% and +0.04%, respectively.

Robustness and Adaptability. It is worth noting that under the Logit Adjustment (LA) loss, Auto-LTOpt achieves a highly competitive overall accuracy of 84.45%, functioning as a strong runner-up to Muon (84.58%) while still outperforming SGD and SAM variants. This slight variance is expected given the automated nature of our search space, yet the overarching trend firmly establishes that the evolutionary strategies discovered by Auto-LTOpt are highly robust, adaptable, and capable of unlocking the potential of various long-tailed objective functions without requiring human-in-the-loop tuning.

D.2 IMPLEMENTATION OF THE DISCOVERED OPTIMIZER

Below is the exact PyTorch implementation of the superior optimizer autonomously discovered by our framework. Note that the evolutionary process synthesized sophisticated logic, including dynamic gradient clipping and adaptive row-wise normalization specifically tailored for tail-class feature enhancement.

PyTorch Implementation of Auto-LTOpt

```
class AutoLTOpt(torch.optim.Optimizer):
    def __init__(self, params, lr=0.02, weight_decay=0.01, momentum=0.95, nesterov=True,
     ns_steps=5, rank=0, world_size=1):
        self.rank = rank
        self.world_size = world_size
        defaults = dict(lr=lr, weight_decay=weight_decay, momentum=momentum, nesterov=
     nesterov, ns_steps=ns_steps)
        params: list[Tensor] = [*params]
        param_groups = []
        for size in {p.numel() for p in params}:
            b = torch.empty(world_size, size, dtype=torch.bfloat16, device="cuda")
            group = dict(params=[p for p in params if p.numel() == size],
                         update_buffer=b, update_buffer_views=[b[i] for i in range(
     world_size)])
            param_groups.append(group)
        super().__init__(param_groups, defaults)

    @torch.no_grad()
```

```
    def step(self):
        for group in self.param_groups:
            update_buffer = group['update_buffer']
            update_buffer_views = group['update_buffer_views']
            params = group['params']
            handle = None
            params_world = None

            def update_prev():
                if handle is not None:
                    handle.wait()
                for p_world, g_world in zip(params_world, update_buffer_views):
                    p_world.mul_(1 - group['lr'] * group['weight_decay'])
                    p_world.add_(g_world.view_as(p_world), alpha=-group['lr'] * max(1,
 p_world.size(-2) / p_world.size(-1))**0.5)

            for base_i in range(len(params))[::self.world_size]:
                if base_i + self.rank < len(params):
                    p = params[base_i + self.rank]
                    g = p.grad
                    if g is None:
                        g = update_buffer_views[self.rank]
                    state = self.state[p]
                    if 'momentum_buffer' not in state:
                        state['momentum_buffer'] = torch.zeros_like(g)
                    buf = state['momentum_buffer']
                    m = group.get('momentum', 0.9)
                    buf.lerp_(g, 1 - m)
                    nesterov = group.get('nesterov', False)
                    g = g.lerp_(buf, m) if nesterov else buf
                    if g is not None and g.dim() == 4:
                        g = g.view(len(g), -1)
                    a, b, c = (3.4445, -4.7750, 2.0315)
                    X = g.bfloat16()
                    transposed = False
                    if g.size(-2) > g.size(-1):
                        X = X.mT
                        transposed = True
                    X = X / (X.norm(dim=(-2, -1), keepdim=True) + 1e-7)
                    for _ in range(group['ns_steps']):
                        A = X @ X.mT
                        B = b * A + c * A @ A
                        X = a * X + B @ X
                    if transposed:
                        X = X.mT
                    if g.dim() == 2:
                        if g.size(0) > 0:
                            row_norm = g.norm(dim=1)
                            mean = row_norm.mean()
                            std = row_norm.std(unbiased=False) + 1e-7
                            z = (row_norm - mean) / (std + 1e-7)
                            tail_mask = z < -0.5
                            if tail_mask.any():
                                w = 1.0 + (-z[tail_mask].clamp(min=-5.0)) * 0.6
                                w = w.clamp(min=0.0, max=2.0)
                                g_tail = g[tail_mask] * w.reshape(-1, 1)
                                g = g.clone()
                                g[tail_mask] = g_tail
                            if 'row_balance' not in state:
                                state['row_balance'] = 1.0
                            else:
                                density = tail_mask.float().mean() if tail_mask.numel() >
 0 else 0.0
                                state['row_balance'] = 0.9 * state['row_balance'] + 0.1 *
 (1.0 + float(density))
                    elif g.dim() == 1:
                        g = g * (1.0 + 0.05 * (g.abs().mean() < g.abs().max() * 0.5).
float())
                    g = X.flatten()
                    g_norm = g.norm()
                    if g_norm > 5.0:
                        g = g / (g_norm + 1e-6) * 5.0
                    bal_factor = state.get('row_balance', 1.0) if 'row_balance' in state
 else 1.0
                    g = g * bal_factor
                    if base_i > 0:
                        update_prev()
                    handle = dist.all_gather_into_tensor(update_buffer, g, async_op=True)
                    params_world = params[base_i: base_i + self.world_size]
```

```
                else:
                    g = update_buffer_views[self.rank]
            update_prev()
```

D.3 Ablation Study of the Evolved Components

To demystify the performance gains of Auto-LTOpt and understand the contribution of the operations autonomously synthesized by the large language model, we conduct an ablation study. We systematically dismantle the full Auto-LTOpt algorithm (`full`) by selectively disabling its components. The experiments are conducted under the ProCo framework on CIFAR-100-LT ($\rho = 0.01$). The detailed performance degradation is recorded in Table 5.

We categorize the reported ablated components into two fundamental mathematical mechanisms:

1. Dimensionality-Aware Sparsity and Stability Protections. Unlike human-designed uniform gradient rules, Auto-LTOpt applies dimension-specific heuristics.

- `no_dim1_boost`: Disables the magnitude-based boosting heuristic specifically applied to 1D tensors (e.g., biases or normalization parameters).
- `no_norm_clip`: Disables the adaptive L_2-norm clipping (thresholded at 5.0) for 1D gradients.
- `no_none_grad` / `strict_grad`: Disables the placeholder logic for missing gradients, forcing strict gradient evaluation.

The ablation reveals that removing the 1D sparsity boost (`no_dim1_only`) causes a $\downarrow 0.38\%$ drop. More interestingly, while clipping is a standard stability technique, disabling it alone (`no_norm_clip`) only causes a minor $\downarrow 0.11\%$ drop. However, when combined with the removal of 1D boosting (`no_dim1_no_clip`), the degradation deepens to $\downarrow 0.17\%$, indicating a synergistic relationship between dimensionality-aware scaling and clipping protections.

2. Newton-Schulz Optimization and Algorithmic Synergies. To isolate the contribution of our LLM-generated post-processing from the base orthogonalization engine, we introduce extreme ablations:

- `ns_helper`: Replaces the explicit Newton-Schulz iterations with an alternative helper function (`zeropower_via_newtonschulz5`).
- `only_ns_momentum`: The most aggressive ablation, stripping away the post-processing operations while retaining the base Newton-Schulz orthogonalization, standard momentum, and distributed gathering.

Substituting the Newton-Schulz logic (`ns_helper`) introduces a moderate degradation ($\downarrow 0.44\%$). Reverting the optimizer to its barebones state (`only_ns_momentum`) causes the largest performance drop in the study, reducing overall accuracy by $\downarrow 0.98\%$.

D.4 Evolutionary Dynamics and Computational Cost

To gain deeper insights into the automated search process, we comprehensively analyze the evolutionary dynamics, operator behaviors, and the practical computational overhead incurred during the LLM-driven search phase.

Performance Trade-offs and Operator Yields. We first examine the optimization behavior of the successfully generated candidates. As shown in Figure 8, plotting the Head (Many-shot) against the Tail (Few-shot) accuracy reveals a distinct cluster of high-performing optimizers in the top-right region. This demonstrates that Auto-LTOpt successfully mitigates the notorious head-tail trade-off, significantly boosting tail performance without strictly sacrificing head accuracy.

Search Stability and Robust Early Pruning. We further monitor the execution outcomes across all 100 generations in Figure 9a. The search process maintains a steady stream of successful evaluations alongside various programmatic failures (e.g., syntax invalidity, training crashes). This diverse outcome distribution confirms that the LLM actively explores a vast, unconstrained search space

Table 5: Ablation study of Auto-LTOpt (ProCo framework) on CIFAR-100-LT ($\rho = 0.01$). We report the Top-1 accuracy (%) across subsets. The best and second-best results in each column are highlighted in **red** and blue respectively. The variants are sorted by the overall performance drop (Δ).

Ablation Variant	Subset Accuracy (%)			Overall	
	Head	Medium	Tail	All	Δ
Full (Auto-LTOpt)	**71.40**	**58.10**	36.50	**53.45**	—
no_norm_clip	70.83	57.60	37.02	53.34	↓ 0.11
only_ns_momentum_ns_helper	70.27	57.07	37.77	53.31	↓ 0.14
no_dim1_no_clip	70.07	56.57	**38.23**	53.28	↓ 0.17
only_ns_momentum_none_strict	70.50	56.80	37.55	53.21	↓ 0.24
strict_grad_ns_helper	70.13	56.90	37.58	53.14	↓ 0.31
no_postprocess_keep_balance	70.23	56.07	38.10	53.13	↓ 0.32
helper_strict_grad	70.30	57.07	37.27	53.12	↓ 0.33
minimal_plus_balance	71.23	56.50	36.95	53.10	↓ 0.35
no_clip_only	69.53	56.47	**38.23**	53.09	↓ 0.36
no_dim1_only	71.23	56.10	37.17	53.07	↓ 0.38
ns_helper	71.00	56.40	36.98	53.01	↓ 0.44
no_bal_global_scale	70.50	56.53	37.12	52.96	↓ 0.49
no_tail_no_dim1	69.43	56.83	37.55	52.90	↓ 0.55
helper_no_clip	70.83	55.60	37.25	52.83	↓ 0.62
no_dim1_boost	71.00	55.37	37.30	52.83	↓ 0.62
no_tail_no_balance	70.27	56.27	36.88	52.71	↓ 0.74
only_ns_momentum	70.67	55.93	36.23	52.47	↓ 0.98

Table 6: Detailed computational cost analysis of the evolutionary search phase across generations. Time metrics are accumulated strictly from successfully completed training runs (approx. 18–22 mins per valid candidate). Invalid syntax or unstable programs that fail prematurely are excluded from the time calculation. In total, 109 candidates were evaluated, with 34 successfully producing valid training time logs.

Generations	Candidates		Training Cost		
	Evaluated	Valid	Total (h)	Total (min)	Avg./Run (s)
0 (Init.)	9	8	2.65	159.1	1193.4
1 – 10	10	3	1.04	62.3	1246.9
11 – 20	10	1	0.31	18.5	1110.4
21 – 30	10	4	1.33	79.9	1199.0
31 – 40	10	3	0.91	54.7	1094.1
41 – 50	10	4	1.37	82.5	1236.9
51 – 60	10	2	0.61	36.7	1101.0
61 – 70	10	1	0.33	19.8	1187.4
71 – 80	10	0	0.00	0.0	—
81 – 90	10	2	0.64	38.6	1158.0
91 – 100	10	6	1.82	109.1	1090.6
Total Summary	**109**	**34**	**11.01**	**661.2**	**1166.8**

rather than conservatively generating trivial, safe variants. A critical driver of our search efficiency under this unconstrained exploration is the strict downstream evaluation module (Algorithm **??**). Among the 109 evaluated candidates, only 34 successfully completed the training pipeline (averaging 1166.8 seconds per valid run). The remaining 75 candidates—often containing syntax errors or triggering numerical instability (NaN/Inf) early in the training dynamics—were swiftly halted and penalized with a zero score.

Overall Computational Efficiency. As depicted in Figure 9b, the accumulated GPU training time remains remarkably low across most generation buckets. Because syntactically invalid and numerically unstable programs are aggressively caught and pruned *before* executing full training epochs, the overall computational cost is strictly bounded. Remarkably, the total accumulated training time for all valid programs was merely 11.01 hours on a single NVIDIA A800 GPU. Given that the discovered optimizer yields substantial, reusable performance improvements (e.g., up to +3.00% absolute gain on CIFAR-100-LT), this one-time upfront search cost is highly economical, effectively eliminating the exhaustive human trial-and-error typically required for manual optimizer tuning.

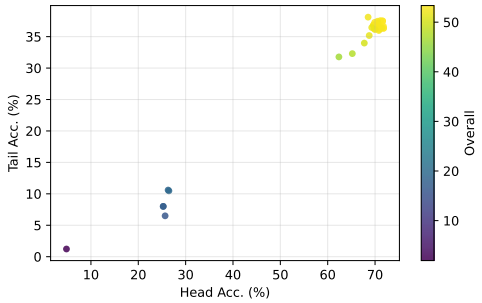
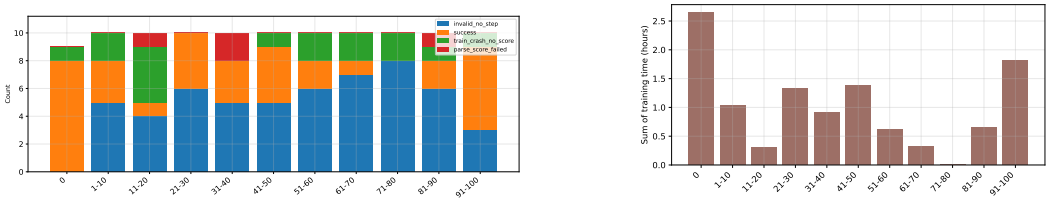


Figure 8: Empirical analysis of the generated optimizers, illustrating the Head vs. Tail Accuracy Trade-off. This scatter plot illustrates the performance distribution of successfully evolved optimizers across Head (Many-shot) and Tail (Few-shot) classes. The color map denotes the overall balanced accuracy. It demonstrates that our framework discovers optimizers capable of boosting tail performance without strictly sacrificing head accuracy.



(a) Outcome Distribution across Generations

(b) Training Time Cost across Generations

Figure 9: Evolutionary stability and computational cost analysis. **(a)** The stacked bar chart details the absolute counts of various execution outcomes (e.g., success, syntax invalidity, numerical crashes) within each generation bucket, providing insights into the stability of the search process over time. **(b)** The accumulated GPU training hours for successfully evaluated candidates across different generation intervals. This confirms that our rigorous early-rejection mechanism keeps the overall computational overhead highly economical.

E LLM Prompt Templates for Optimizer Evolution

In this section, we detail the exact prompt templates used in our evolutionary framework. The language model acts as both an *algorithm generator* (synthesizing and mutating code) and a *reflector* (providing constructive hints based on performance feedback). Variables enclosed in angle brackets, such as `<func_name>` or `<better_code>`, are dynamically populated during the search process.

E.1 Problem Description and Optimization Goal

All generation prompts share the same core problem description and strict constraints to ensure the LLM understands the long-tailed learning context and PyTorch optimizer API requirements:

Problem Description:
The problem focuses on Long-tail Learning, where the dataset exhibits a highly skewed class distribution characterized by a few "head" classes with abundant samples and many "tail" classes with very few samples. This imbalance typically causes models to bias towards head classes, leading to poor generalization on the tail.
The primary objective is to enhance the overall performance of the model while ensuring a delicate balance across all categories. Specifically, the goal is to significantly boost the accuracy of "tail" and "medium" classes while simultaneously ensuring that the performance of "head" classes is not compromised. A successful solution must achieve equitable performance across the entire distribution without causing performance degradation in any specific group.
Component to evolve: The optimizer's `step(self)` method that applies updates to matrix-like backbone parameters in a long-tailed CIFAR training pipeline.
Goal (plain language):

- Improve the `step(self)` to mitigate **class imbalance** in long-tailed recognition.
- Improve **balanced accuracy** across head, medium, and tail classes.
- Keep training stable, no NaN/Inf, no performance drop on head classes.

Strict constraints (MUST FOLLOW 100%):

1. **ONLY USE THESE EXISTING KEYS IN group:** `"update_buffer"`, `"update_buffer_views"`, `"weight_decay"`, `"lr"`, `"momentum"`, `"nesterov"`, `"ns_steps"`.
2. If any additional variables are needed, **YOU MUST DEFINE THEM LOCALLY INSIDE the `step()` function**. DO NOT rely on external data or external functions.

E.2 Evolutionary Operator Prompts

Prompt 1: Initialization Operator ($\mathcal{I}$)

System Prompt:
You are an expert in the domain of optimization and algorithm design. Your task is to design high-quality algorithms that can effectively solve specific problems. Your response outputs Python code and nothing else. Format your code as a Python code string: ```` ```python ... ``` ````.
User Prompt:
Write function or module for the following problem: *[Insert Problem Description and Strict Constraints Here]*
Seed Template:
`<seed_func>`
Refer to the format of a trivial design above. Be very creative and give `<func_name>`. The output MUST be a JSON string with the following rules: 1. The keys are the function/class names from the input list. 2. The value of each key is the improved code wrapped in a Python code block. 3. Ensure the JSON is valid, with proper escaping of quotes and newlines. 4. Do not include any extra text outside the JSON string. 5. Any package that is used must be imported in advance. 6. If you need helper routines, define them inside the same definition (e.g. nested `def`); do not add new instance methods that take `self`.
[Mandatory] Preserve the signatures `step` as required by the existing optimizer API and training loop.
[Mandatory] Update parameters in-place from `param.grad` (skip or guard when `grad` is None); keep `self.state` tensors aligned with their parameters; `step` return value must follow `Optimizer.step` conventions.

Prompt 2: Reflection Operator (Short-term & Long-term)

System Prompt:
You are an expert in the domain of optimization and algorithm design. Your task is to give hints to design better solutions.
User Prompt (Short-term Reflection):
Below are two `<func_name>` executable statements for the following problem: *[Insert Problem Description]*. You are provided with two code versions below, where the second version performs better than the first one.
[Worse code] `<worse_code>`
[Better code] `<better_code>`
[Worse code performance] `<worse_performance>`
[Better code performance] `<better_performance>`
You respond with some hints for designing better solutions, based on the two code versions and their performance feedback. Use less than 50 words.

User Prompt (Long-term Reflection):
Below is your prior long-term reflection on designing solutions for the following problem: *[Insert Problem Description]*.
`<prior_reflection>`
Below are some newly gained insights:
`<new_reflection>`
Write constructive hints for designing better solutions, based on prior reflections and new insights and using less than 50 words.

Prompt 3: Crossover Operator ($\mathcal{X}$)

System Prompt: *[Same as Generator System Prompt]*
User Prompt:
[Insert Problem Description and Strict Constraints Here]
[Worse code]
`<func_signature0>`
`<worse_code>`
[Better code]
`<func_signature1>`
`<better_code>`
[Reflection]
`<reflection>`
[Improved code]
Please write an improved executable statements `<func_name>`, according to the reflection. The output MUST be a JSON string with the following rules: *[Same JSON formatting rules as Initialization]*.

Prompt 4: Mutation Operator ($\mathcal{M}$)

System Prompt: *[Same as Generator System Prompt]*
User Prompt:
[Insert Problem Description and Strict Constraints Here]
[Prior reflection]
`<reflection>`
[Code]
`<func_signature1>`
`<elitist_code>`
[Performance Feedback]
`<elitist_performance>`

[Improved code]
Please write a mutated executable statements `<func_name>`, according to the reflection and performance feedback. The output MUST be a JSON string with the following rules: *[Same JSON formatting rules as Initialization]*.

F LIMITATIONS AND FUTURE WORK

While our proposed Auto-LTOpt framework demonstrates superior performance and establishes a novel paradigm for automated optimizer design in long-tailed recognition, we acknowledge several limitations that warrant future investigation:

- **Computational Overhead of Evolutionary Search:** Although the final discovered optimizer (Auto-LTOpt) is highly efficient during standard training, the upfront automated search phase is computationally intensive. The evolutionary loop requires instantiating and training proxy neural networks for hundreds of generations to evaluate the fitness of each LLM-generated candidate. While we mitigate this by using smaller proxy datasets (e.g., CIFAR) and fewer epochs during the search, scaling the evolutionary process directly to massive datasets (e.g., full ImageNet) or billion-parameter models remains prohibitively expensive.
- **Cross-Modality Transferability and Hardcoded Heuristics:** As shown in the code implementation (Appendix D.2), the executed optimizer uses numerical constants for gradient scaling and clipping. It remains an open question whether these values transfer to different modalities (such as Natural Language Processing) or architectures (such as Transformers) without requiring a re-evolution run.
- **Interpretability of Synthesized Logic:** Although we provide convergence analysis for the evolutionary framework (Theorem B.3), the microscopic mathematical rationale behind the executed LLM-generated operations remains partially opaque. Developing controlled evaluations and analytical tools to interpret these algorithms is a critical next step.