LANGUAGE-GUIDED TUNING: CONFIGURATION OPTIMIZATION FOR AUTOMATED ML RESEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Automated machine learning research depends on repeatedly proposing, executing, and evaluating changes to an experiment. Configuration optimization is a central part of this process, requiring decisions across model architecture, training strategy, feature engineering, and hyperparameters. We introduce Language-Guided Tuning (LGT), a framework of large language model (LLM) agents that organizes these decisions as a language-guided configuration experiment loop. An Advisor proposes edits from training traces, an Evaluator compares candidate outcomes, and an Optimizer converts feedback into guidance for subsequent proposals. Textual feedback complements numerical optimization by recording the rationale for a proposed intervention, while a validation-based retention rule separates proposing an edit from accepting it. LGT has the best reported mean performance in the result tables across seven datasets and eleven baselines covering untuned training and classical tuning; the reported evaluation also includes a MiniViT feasibility study. The framework makes proposed edits and their feedback inspectable, providing a configuration-level component for automated research workflows. The current evidence concerns task performance and recorded trajectories; matched LLM-agent comparisons, controlled agent-role ablations, and independent verification of the evaluation protocol remain necessary.

1 INTRODUCTION

Automated research systems increasingly select interventions, produce executable candidates, measure outcomes, and decide what to investigate next. Even with a fixed research question, an experiment requires choices of architecture, data transformation, training strategy, and hyperparameters. We study this configuration experimentation layer of automated ML research, connecting explicit edits to training feedback and using the resulting record to inform subsequent proposals.

These choices interact: augmentation can change suitable learning rates or architectures, while loss functions affect optimizer behavior (Shorten & Khoshgoftaar, 2019; Wang et al., 2022). Established methods model dependencies and adapt to observations. Bayesian optimization guides trials using past evaluations; Population Based Training, Self-Tuning Networks, and population-based bandits adapt configurations during training (Frazier, 2018; Jaderberg et al., 2017; MacKay et al., 2019; Parker-Holder et al., 2020). Neither online adaptation, LLM-based proposals, nor multiple agents alone distinguishes our approach.

We propose **Language-Guided Tuning (LGT)**, which organizes configuration experiments around three roles: an **Advisor** proposes edits, an **Evaluator** assesses candidate outcomes, and an **Optimizer** distills feedback into guidance. Their shared history links interventions across these four domains to measurements and retention decisions. Retained and rejected candidates provide context for subsequent proposals. These records expose stated rationales without establishing explanation fidelity or the causes of performance gains.

LGT combines architecture-level iterations following completed training cycles with epoch-boundary changes to compatible training and feature configurations. This schedule distinguishes architecture changes from edits that can reuse the current training state. Architecture changes can affect reusable optimizer state, while augmentation can alter appropriate training strategies, motivating a shared record of coupled interventions.

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.

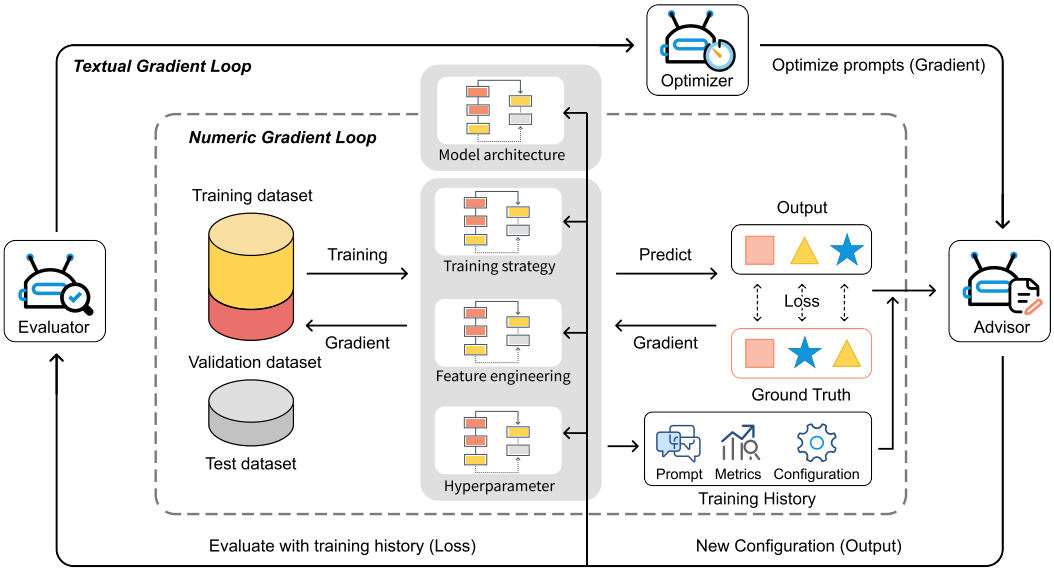


Figure 1: Overall framework of Language-Guided Tuning (LGT). The system operates through dual loops: an inner numeric gradient loop for traditional training, and an outer textual gradient loop where three LLM agents (Advisor, Evaluator, Optimizer) coordinate to optimize configurations across four dimensions (model architecture, training strategy, feature engineering, hyperparameters).

A compatible candidate is retained only when its validation score improves by a margin η_t and the Evaluator approves it; otherwise the incumbent is restored (Algorithm 1). A fixed scoring protocol and full state restoration protect the recorded incumbent validation score. Population-risk convergence and stationarity require additional assumptions (Appendix E). The margin is a retention criterion, not a calibrated significance test. Single-domain ablations characterize reported domain contributions; isolating the value of three-role decomposition and cross-domain coordination still requires controlled comparisons.

Across seven datasets, LGT is at or above eleven comparison baselines in reported mean performance, with larger margins on some tabular and generation metrics than on easier classification tasks. These experiments evaluate configuration-level outcomes. An outer research agent could supply a hypothesis, task, and budget and receive a configuration with its experiment record; end-to-end research integration remains untested.

(1) Two training timescales: LGT combines architecture-level iterations after completed training cycles with compatible configuration edits at epoch boundaries.

(2) Conditional validation-gated retention: LGT combines a numerical margin with Evaluator approval under explicit scoring and restoration conditions.

(3) A shared four-domain interface: LGT records proposed changes and outcomes across four domains in a common experiment history.

2 Related Works

2.1 Configuration Tuning Methods

Grid and random search explore specified spaces without adapting to previous scores (Liashchynskyi & Liashchynskyi, 2019; Bergstra & Bengio, 2012). Bayesian optimization, Optuna, and SMAC3 guide search using past evaluations (Frazier, 2018; Akiba et al., 2019; Lindauer et al., 2022); Hyperband and BOHB allocate evaluation effort across candidates (Li et al., 2018; Falkner et al., 2018). DARTS targets architectures, while AutoML combines pipeline choices (Liu et al., 2019; Karmaker et al., 2021; Zöller & Huber, 2021). LGT adds a shared language-conditioned interface and textual records across four domains, without assuming that other optimizers cannot represent coupled variables.

2.2 Automated Research Agents

The AI Scientist-v2 uses template-free experimentation and agentic tree search (Yamada et al., 2025); the subsequent journal study examines template-based and template-free workflows from idea generation through manuscript preparation (Lu et al., 2026). AI-Researcher coordinates literature analysis, hypothesis development, implementation, and documentation, introducing Scientist-Bench for guided and open-ended research tasks (Tang et al., 2025).

AIDE uses tree search in code space for machine learning engineering (Jiang et al., 2025). AlphaEvolve combines LLM-generated code, evaluator feedback, and evolutionary search (Novikov et al., 2025). Empirical Research Assistance (ERA) combines LLMs and tree search to develop scientific software for tasks with a specified quality metric (Aygün et al., 2026). Karpathy's *autoresearch* repository provides a software example of a fixed-budget edit–run–retain loop (Karpathy, 2026); MLE-bench evaluates agents on Kaggle-style engineering tasks (Chan et al., 2025).

These systems share a proposal–execution–evaluation cycle but modify different objects and pursue different research scopes. LGT focuses on configuration interventions within a given task, including architecture proposals containing model code. Its experiment record offers a potential interface to broader agents choosing research hypotheses; it does not assume that every candidate in those systems requires complete retraining.

2.3 LLM-Based Optimization

OPRO generates candidates from prior solutions and scores; AgentHPO and OptiMindTune use LLMs for hyperparameter tuning (Yang et al., 2024; Liu et al., 2025a; Madiraju & Madiraju, 2025). OptFormer learns optimization behavior with a transformer (Chen et al., 2022). TextGrad and GEPA use language-mediated feedback (Yuksekgonul et al., 2025; Agrawal et al., 2026), while AutoDAN-Turbo separates generation, scoring, and strategy summarization (Liu et al., 2025b). CALM co-evolves candidate heuristics and the language model (Huang et al., 2026); MAT-Agent uses bandit mechanisms for training control (Zhang et al., 2025). Directional feedback and LLaMoCo study complementary feedback and language-conditioned optimization mechanisms (Nie et al., 2024; Ma et al., 2024). Building on these precedents, LGT organizes numerical training outcomes and configuration edits at two timescales through a shared history and retention rule.

3 Methods

3.1 Configuration Experiments at Two Timescales

Figure 1 combines numerical training with language-guided configuration experiments. Let $\mathcal{C} = \mathcal{C}_A \times \mathcal{C}_F \times \mathcal{C}_T \times \mathcal{C}_H$ comprise model architecture, feature engineering, training strategy, and hyperparameters. Each configuration $c \in \mathcal{C}$ parametrizes a model f_c that maps inputs $x \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$. Here f_c denotes the predictor obtained under a specified training protocol. The objective is

$$c^* = \arg\min_{c \in \mathcal{C}} \mathcal{L}(c) = \arg\min_{c \in \mathcal{C}} \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(f_c(x), y)] \tag{1}$$

where ℓ is a fixed task-level evaluation loss and $\mathcal{D}$ is the data distribution. Losses obtained under different training objectives are not directly interchangeable: configuration interventions must be compared using the same evaluation criterion.

The four domains expose distinct, interacting choices. Architecture includes layer types, connectivity, and structural parameters subject to computational and structural constraints. Feature engineering selects available transformations, including geometric changes, noise injection, and domain-specific operations. Training strategy comprises loss functions, optimizers, and learning-rate schedulers; hyperparameters include bounded learning rates, weight decay, and related settings. Appendix K.1 preserves their formal spaces, component objectives, and projected hyperparameter update.

LGT uses a hierarchical schedule. Architecture proposals occur between training cycles, informed by the completed cycle; compatible feature, strategy, and hyperparameter proposals occur at epoch boundaries within a cycle. Each proposal conditions on the previously retained configuration. This distinction agrees with the iteration and epoch axes in Figure 3 and does not assume that arbitrary

architecture changes preserve the current checkpoint. Numerical training updates weights, while the outer language-mediated process proposes configuration interventions at both timescales.

3.2 Proposal, Assessment, and Guidance

LGT separates three roles with distinct inputs and outputs. The **Advisor** receives the current configuration, training metrics, optimization history, and prompt context. It returns a natural-language configuration recommendation Δc_t^{NL}, which a dedicated configuration parser turns into a candidate. Only this parsed Advisor output modifies the configuration; the other roles produce assessments and guidance rather than configurations.

The **Evaluator** compares the candidate and incumbent using their configurations, measured outcomes, and available history. Its textual response $\mathcal{I}_{\mathrm{eval}}^{\mathrm{NL}}$ is parsed into a binary decision $\mathrm{success}_t$ and a suggestion s_t^{NL}. The decision contributes to the retention gate below, while the suggestion can address configuration fields, constraints, or future proposals. Text alone cannot trigger retention.

The **Prompt Optimizer** uses measured performance, Advisor context, and the Evaluator's suggestion to synthesize role-specific guidance. The effective Advisor input combines a fixed instruction template with variable guidance and experiment context. Consistent with the supplied Optimizer template, the update concerns this context and does not rewrite the instruction template. The prompt-update notation describes guidance synthesis; further implementation-level rewriting needs verification.

Appendix K.2 gives the original agent codomains, parsers, and per-role prompt mappings. Complete templates and recorded outputs appear in Appendices F and G. The roles make proposal, assessment, and guidance explicit in the experiment record. They provide an architectural rationale; matched single-agent and role-removal controls are needed to establish whether three roles are necessary.

3.3 Experiment History and Feedback

The shared history links configuration choices c_t, measured training outcomes $\mathcal{M}_t$, proposed edits, assessments, and prompt context:

$$\mathcal{H}_t = \mathcal{H}_{t-1} \cup \{(c_t, \mathcal{M}_t, \Delta c_t^{\mathrm{NL}},\ \mathcal{I}_{\mathrm{eval}}^{\mathrm{NL}}, \mathrm{prompt}_t)\} \tag{2}$$

This record supplies context for dependencies between configuration components and for subsequent proposals. In a research workflow, an incumbent configuration and training record define the starting point, the parsed edit defines the intervention, and the measured candidate outcome and retention decision record its result. Retained and rejected attempts can inform the next proposed intervention.

The feedback loop turns observed outcomes into guidance, but the presence of feedback alone does not establish successful self-improvement. Its value requires comparison with alternative proposal mechanisms under matched budgets. Improved performance on a known task is a configuration-level result, not itself evidence of scientific novelty or successful autonomous research.

3.4 Candidate Execution and Retention

Let $z_t^\star = (c_t^\star, w_t^\star, o_t^\star, q_t^\star)$ denote the incumbent configuration, weights, optimizer state, and scheduler state. For a compatible edit, $\widetilde{z}_{t+1}$ is produced by applying the edit and training for one epoch from this state. Let S_{val} be a higher-is-better score under a fixed validation protocol and $\eta_t \geq 0$ the improvement margin. The formal retention rule is

$$\begin{aligned} a_t &= \mathbf{1}[\mathrm{success}_t = 1 \ \wedge\ S_{\mathrm{val}}(\widetilde{z}_{t+1}) > S_{\mathrm{val}}(z_t^\star) + \eta_t]\,, \\ z_{t+1}^\star &= \begin{cases} \widetilde{z}_{t+1}, & a_t = 1, \\ z_t^\star, & a_t = 0. \end{cases} \end{aligned} \tag{3}$$

The formal rule requires the numerical comparison while allowing the LLM to veto a candidate. Rejection requires restoration of all mutable state relevant to continuation, including randomness and preprocessing state when applicable. Changes to layer widths, parameter shapes, or optimizer families require a specified transfer or restart rule; copying old weights alone does not cover these transitions. Algorithm 1 describes the incumbent loop. Implementation-specific transfer and scoring details remain to be verified against execution artifacts.

Under an unchanged scoring protocol and complete state restoration, the rule protects the recorded incumbent validation score. Population-risk convergence and stationarity require the additional assumptions in Appendix E; the margin is not a calibrated significance test. Epoch-level decisions impose a horizon: slower early training may improve a longer-run objective. This unresolved tradeoff motivates comparisons with multi-epoch or full-cycle decisions.

4 EXPERIMENTS

We evaluate configuration-level outcomes on seven datasets and a supplementary MiniViT study (Appendix H). The evaluation concerns task performance and recorded behavior, rather than end-to-end autonomous research.

4.1 DATASETS

We evaluate LGT on seven datasets spanning three fundamental machine learning tasks (Table 1, Appendix A). For image classification, we use MNIST (Deng, 2012) and CIFAR-10 (Krizhevsky, 2009) to assess performance on visual recognition tasks. For tabular data, we include classification tasks (Iris (Fisher, 1936), Water Potability (Tharmalingam, 2023)) and regression tasks (House Price (Imran, 2023), Wine Quality (Cortez et al., 2009)) to demonstrate effectiveness on structured data. For text generation, we employ Tiny Shakespeare (Karpathy, 2015) to evaluate sequence modeling capabilities. This evaluation spans different data modalities and task types, providing a set of configuration experiments on compact, familiar benchmarks. The dataset identities, splits, and metric definitions constrain how broadly these outcomes can be generalized.

Table 1: Dataset characteristics.

Dataset	Samples	Features	Task Type	Classes
MNIST	70,000	784	Classification	10
CIFAR-10	60,000	3,072	Classification	10
Iris	150	4	Classification	3
Water Potability	3,276	9	Classification	2
House Price	50,000	5	Regression	–
Wine Quality	1,599	11	Regression	–
Tiny Shakespeare	40,000	–	Generation	–

4.2 EXPERIMENTAL SETUP

Default LLM: All agents use DeepSeek-V3.2 (DeepSeek-AI, 2025) at temperature 0.2.

The reported inference setup serves Llama-3-8B-Instruct and Qwen3-8B locally through vLLM, using temperature 0.2 and top-p 0.95 to match the DeepSeek-V3.2 generation settings. GPT-oss-120B is accessed through its public endpoint at the same temperature. These details specify the model-serving arrangements behind the backbone comparison (Appendix H.2).

Data access: The intended train/validation/test split is 72/8/20; agents and retention should use only training and validation records. Appendix A instead describes 80/20 without the validation subdivision, and Figure 2 includes test-loss trajectories. Test independence from adaptive selection therefore remains unestablished. Split indices and generation-window construction need reconciliation (Appendices J and L).

Search spaces: LGT explores 2–5 layers with 32–512 neurons, geometric and noise-based transformations, loss/optimizer/scheduler choices, class weights from 0.1 to 10.0, and learning rates from 10^{-4} to 10^{-1}. Full space descriptions are retained in Appendix L.

The training-strategy search considers Adam, SGD, RMSprop, AdamW, Adagrad, and AdaFactor. Geometric feature interventions include rotation and scaling, alongside noise injection and task-specific preprocessing. The intended validation set is formed by holding out 10% of the 80% training allocation. For Tiny Shakespeare, reproducing the split additionally requires the order of raw-text partitioning and construction of overlapping windows to assess shared context across partitions.

On MiniViT, the architecture search spans patch sizes 2, 4, or 8 and embedding dimensions 32, 64, 128, or 192, together with choices of depth, attention heads, MLP ratio, and dropout. The remaining three domains reuse the main experiment's search spaces, without prompt-level modification to the Advisor, Evaluator, or Optimizer.

Evaluation: The record specifies up to 50 configurations, a 10-epoch horizon, and ten runs with seeds 42–51. Metrics cover accuracy/F1/AUC, MAE/MSE/R^2, and perplexity/BLEU/token accuracy.

Reported variation does not establish significance or equal budgets: candidate training, rejected attempts, restarts, validation, and language calls all contribute to cost. Appendix J lists unresolved aggregation and protocol checks.

Table 2: Classification performance comparison.

	MNIST			CIFAR-10		
Method	**Acc.**	**F1**	**AUC**	**Acc.**	**F1**	**AUC**
No Tuning	$96.77_{0.2}$	$96.78_{0.2}$	$98.37_{0.08}$	$69.01_{2.4}$	$69.01_{2.5}$	$93.60_{1.2}$
Random Search	$97.87_{0.1}$	$97.87_{0.1}$	$99.96_{0.04}$	$74.57_{4.6}$	$74.48_{4.5}$	$96.95_{0.8}$
Grid Search	$97.75_{0.1}$	$97.75_{0.1}$	$99.96_{0.05}$	$76.82_{1.4}$	$76.70_{1.4}$	$97.19_{0.2}$
Bayesian Opt.	$97.67_{0.3}$	$97.66_{0.3}$	$98.45_{0.20}$	$72.45_{2.1}$	$72.23_{2.2}$	$94.95_{0.9}$
Optuna	$97.79_{0.1}$	$97.79_{0.1}$	$99.96_{0.04}$	$76.83_{4.9}$	$76.80_{0.8}$	$97.24_{0.1}$
SMAC3	$97.61_{0.2}$	$97.61_{0.2}$	$99.95_{0.05}$	$74.97_{2.3}$	$75.02_{2.3}$	$96.81_{0.5}$
KerasTuner	$97.78_{0.1}$	$97.78_{0.1}$	$99.96_{0.04}$	$77.69_{1.3}$	$77.59_{1.4}$	$97.34_{0.2}$
Hyperband	$97.32_{0.2}$	$97.31_{0.2}$	$99.93_{0.06}$	$75.14_{3.4}$	$74.97_{3.3}$	$97.04_{0.6}$
BOHB	$97.47_{0.3}$	$97.46_{0.3}$	$99.94_{0.05}$	$77.42_{0.9}$	$77.34_{1.0}$	$97.43_{0.1}$
PBT	$97.05_{0.2}$	$97.05_{0.2}$	$99.92_{0.07}$	$77.11_{2.3}$	$76.95_{2.3}$	$97.40_{0.3}$
DARTS	$97.34_{0.5}$	$97.34_{0.5}$	$98.67_{0.21}$	$75.89_{3.4}$	$75.67_{3.5}$	$96.45_{1.3}$
LGT	$\mathbf{98.99}_{0.8}$	$\mathbf{98.99}_{0.8}$	$\mathbf{99.99}_{0.10}$	$\mathbf{79.64}_{2.0}$	$\mathbf{79.61}_{2.1}$	$\mathbf{98.42}_{0.7}$

	Water Potability			Iris		
Method	**Acc.**	**F1**	**AUC**	**Acc.**	**F1**	**AUC**
No Tuning	$65.17_{2.8}$	$50.94_{3.1}$	$65.33_{1.9}$	$93.33_{4.2}$	$92.95_{3.8}$	$96.60_{2.1}$
Random Search	$65.15_{2.0}$	$52.11_{3.6}$	$66.05_{2.5}$	$94.33_{3.7}$	$94.45_{3.2}$	$96.78_{1.8}$
Grid Search	$65.89_{2.9}$	$61.67_{3.2}$	$66.78_{2.0}$	$95.67_{2.1}$	$94.23_{2.9}$	$97.45_{2.0}$
Bayesian Opt.	$66.67_{2.7}$	$62.45_{2.9}$	$67.23_{1.8}$	$95.67_{1.2}$	$94.45_{1.7}$	$97.78_{1.5}$
Optuna	$66.50_{1.4}$	$45.42_{3.0}$	$68.39_{2.2}$	$90.33_{4.6}$	$90.00_{5.1}$	$98.40_{1.6}$
SMAC3	$66.16_{1.1}$	$47.96_{3.2}$	$68.92_{2.1}$	$94.67_{3.7}$	$94.62_{3.8}$	$99.53_{0.5}$
KerasTuner	$63.05_{1.4}$	$50.77_{3.8}$	$66.06_{2.1}$	$95.33_{4.0}$	$95.34_{4.0}$	$98.67_{1.7}$
Hyperband	$63.89_{1.9}$	$51.03_{3.2}$	$65.48_{1.7}$	$95.00_{3.1}$	$94.99_{3.1}$	$99.33_{0.9}$
BOHB	$66.13_{1.9}$	$50.98_{2.6}$	$68.52_{2.0}$	$95.00_{4.0}$	$94.96_{4.1}$	$99.30_{0.7}$
PBT	$66.78_{1.3}$	$46.72_{2.9}$	$69.05_{2.2}$	$95.67_{3.3}$	$95.65_{3.4}$	$99.27_{0.7}$
DARTS	$66.12_{3.1}$	$61.89_{3.3}$	$66.89_{2.2}$	$94.33_{2.8}$	$94.67_{2.1}$	$98.12_{1.2}$
LGT	$\mathbf{66.92}_{2.5}$	$\mathbf{63.52}_{2.7}$	$\mathbf{69.60}_{1.6}$	$\mathbf{96.67}_{1.8}$	$\mathbf{96.67}_{1.7}$	$\mathbf{99.54}_{0.8}$

Table 3: Regression and text generation performance comparison.

	House Price			Wine Quality		
Method	**MAE**	**MSE**	**R^2**	**MAE**	**MSE**	**R^2**
No Tuning	$0.458_{0.026}$	$0.329_{0.030}$	$0.425_{0.03}$	$0.706_{0.032}$	$0.828_{0.060}$	$-0.267_{0.05}$
Random Search	$0.441_{0.029}$	$0.313_{0.031}$	$0.451_{0.03}$	$0.680_{0.030}$	$0.781_{0.055}$	$-0.210_{0.04}$
Grid Search	$0.435_{0.027}$	$0.305_{0.029}$	$0.463_{0.03}$	$0.665_{0.030}$	$0.746_{0.052}$	$-0.160_{0.05}$
Bayesian Opt.	$0.420_{0.026}$	$0.289_{0.027}$	$0.505_{0.02}$	$0.632_{0.028}$	$0.691_{0.048}$	$-0.090_{0.04}$
Optuna	$0.418_{0.025}$	$0.286_{0.026}$	$0.511_{0.02}$	$0.629_{0.027}$	$0.686_{0.047}$	$-0.080_{0.04}$
SMAC3	$0.419_{0.026}$	$0.287_{0.026}$	$0.508_{0.02}$	$0.630_{0.028}$	$0.689_{0.047}$	$-0.075_{0.01}$
KerasTuner	$0.430_{0.028}$	$0.298_{0.028}$	$0.489_{0.02}$	$0.642_{0.029}$	$0.711_{0.050}$	$-0.120_{0.04}$
Hyperband	$0.432_{0.028}$	$0.300_{0.029}$	$0.485_{0.03}$	$0.653_{0.030}$	$0.731_{0.052}$	$-0.150_{0.07}$
BOHB	$0.417_{0.024}$	$0.284_{0.025}$	$0.515_{0.02}$	$0.617_{0.026}$	$0.668_{0.045}$	$0.010_{0.03}$
PBT	$0.414_{0.023}$	$0.279_{0.024}$	$0.525_{0.02}$	$0.609_{0.025}$	$0.655_{0.042}$	$0.035_{0.06}$
DART/DARTS	$0.408_{0.022}$	$0.262_{0.025}$	$0.539_{0.02}$	$0.588_{0.024}$	$0.613_{0.040}$	$0.081_{0.03}$
LGT	$\mathbf{0.402}_{0.021}$	$\mathbf{0.254}_{0.024}$	$\mathbf{0.557}_{0.02}$	$\mathbf{0.547}_{0.027}$	$\mathbf{0.515}_{0.045}$	$\mathbf{0.212}_{0.03}$

	Tiny Shakespeare		
Method	**PPL**	**BLEU-4**	**Acc.**
No Tuning	$2.96_{0.12}$	$0.681_{0.023}$	$82.45_{1.8}$
Random Search	$2.43_{0.09}$	$0.724_{0.018}$	$84.12_{1.5}$
Grid Search	$2.38_{0.08}$	$0.731_{0.016}$	$84.56_{1.3}$
Bayesian Opt.	$2.51_{0.11}$	$0.712_{0.020}$	$83.89_{1.6}$
Optuna	$2.35_{0.07}$	$0.738_{0.015}$	$84.78_{1.2}$
SMAC3	$2.47_{0.10}$	$0.719_{0.019}$	$84.23_{1.4}$
KerasTuner	$2.33_{0.06}$	$0.742_{0.014}$	$84.91_{1.1}$
Hyperband	$2.54_{0.13}$	$0.708_{0.021}$	$83.67_{1.7}$
BOHB	$2.41_{0.08}$	$0.728_{0.017}$	$84.34_{1.3}$
PBT	$2.29_{0.05}$	$0.749_{0.013}$	$85.12_{1.0}$
DART/DARTS	$2.62_{0.14}$	$0.695_{0.022}$	$83.21_{1.9}$
LGT	$\mathbf{1.72}_{0.04}$	$\mathbf{0.907}_{0.008}$	$\mathbf{92.50}_{0.7}$

5 Results

5.1 Overall Performance Comparison

Tables 2 and 3 report means and standard deviations over ten runs against untuned training and classical tuning methods. LGT has the best displayed mean in every metric column. On CIFAR-10, accuracy rises from 69.01 to 79.64, a 10.63-point gain over No Tuning and a 1.95-point gain over KerasTuner, the strongest displayed accuracy baseline. These are task-level comparisons, without statistical-significance or equal-budget guarantees. The MNIST AUC aggregation requires reconciliation (Appendix J).

The regression and generation results show the task-dependent margins behind this overall pattern. On House Price, LGT reports lower MAE (0.402 versus 0.408) and higher R^2 (0.557 versus 0.539) than DARTS, the strongest displayed baseline on these metrics. On Wine Quality, the corresponding values are MAE 0.547 and R^2 0.212 for LGT, versus 0.588 and 0.081 for DARTS. For Tiny Shakespeare, LGT reports perplexity 1.72, BLEU-4 0.907, and token accuracy 92.50, compared with 2.29, 0.749, and 85.12 for PBT. Thus the comparison covers continuous prediction errors and sequence-modeling metrics as well as classification. The reported advantages vary across tasks and metrics, rather than constituting one uniform performance margin.

As a configuration component for automated research, LGT records what was proposed, why, and which outcome followed. The tables do not isolate language guidance from action-space, proposal-budget, or pretrained-prior differences, and omit matched LLM-driven tuning/coding-agent baselines. Common-budget, common-space controls are needed to attribute the observed gains to the proposal mechanism. Extended interpretation is in Appendix L.

5.2 Dynamic Convergence Analysis

Figure 2 illustrates recorded training trajectories for LGT and No Tuning across six datasets. The plotted examples reach lower final losses for LGT, including 0.0469 versus 1.5827 on MNIST and 0.0337 versus 0.7041 on Iris. These trajectories illustrate how performance can evolve during a configuration experiment. They provide neither a convergence proof nor a matched wall-clock comparison. Test-loss curves require reconciliation with the held-out reporting protocol before serving as independent generalization evidence.

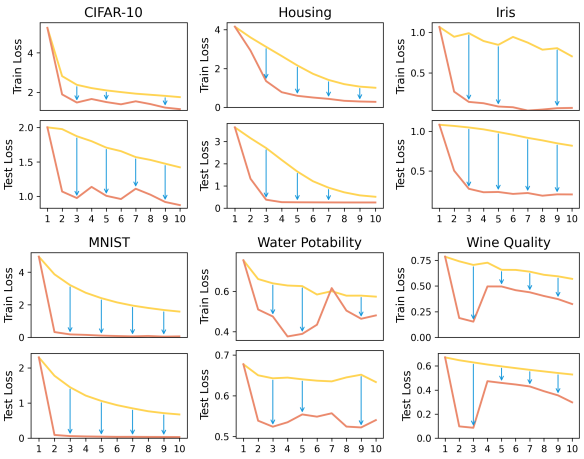


Figure 2: Recorded loss trajectories across six datasets for LGT and the untuned baseline. The plotted training and test losses describe these runs; their relation to candidate selection is discussed in the text.

The trajectory analysis makes the timing of configuration interventions relevant to automated research: an agent must choose both what to change and when to evaluate it. LGT conditions subsequent proposals on observed training behavior, rather than keeping a fixed configuration throughout the plotted run. The descending curves relative to No Tuning are consistent with useful adaptive interventions in these examples. Figure 2, however, does not show grid-search or random-search trajectories, so it cannot establish greater stability than those methods. LLM inference, candidate training, and validation also incur costs. The figures support inspection of the observed sequences, while controlled proposal mechanisms and matched time budgets are necessary to determine why their performance differs.

Figure 3 records the evolution of configuration domains across architectural iterations and epoch-level adjustments. The two timelines distinguish model-structure proposals from compatible training updates. Iteration 1 shows filter1 changing from 8 to 16, filter2 from 16 to 32, addition of kernel3, and dropout changing from 0.2 to 0.4; Iteration 3 scales the architecture further and changes dropout from 0.4 to 0.3. In the epoch timeline, the record begins with cross-entropy and Adam at $t = 1$, changes to focal loss and AdamW at $t = 3$, and returns to Adam at $t = 7$. Feature choices move from flipping to rotation, while the displayed learning rates change from 0.01 to 0.02 to 0.02. For an automated investigator, this is a trace of proposed interventions and their ordering. It does not verify that class imbalance motivated or benefited from the loss change, that the updates avoided conflicts, or that tensor-shape changes reused state correctly; those claims require the corresponding measurements and implementation records.

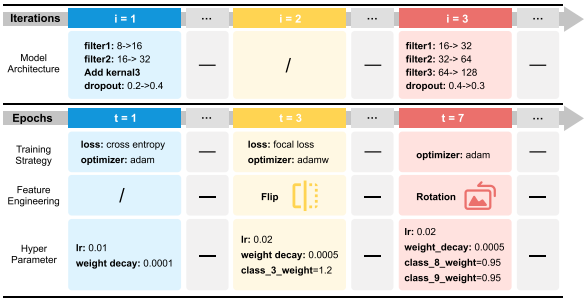


Figure 3: Recorded configuration changes across four dimensions, shown at architecture-iteration and training-epoch timescales.

5.3 Textual Evolution and Interpretability

Figure 4 shows the textual record produced by the configuration experiment loop. Natural-language proposals and feedback make the sequence of decisions available for inspection, so a researcher can compare an agent's stated reason with the change actually made and the reported result. This inspectability is illustrated by the figure; explanation fidelity requires an additional evaluation.

The textual feedback in Figure 4 exposes how the three roles describe an experiment. The Advisor states a rationale for a proposed configuration change, the Evaluator comments on the observed metrics, and the Optimizer turns that assessment into subsequent guidance. Such records can support error analysis as well as successful iteration: the figure describes CIFAR-10 as balanced in one place but invokes class imbalance later, and Appendix G includes an incorrect description of the model's dropout. These inconsistencies are visible precisely because the rationale is recorded. The practical research value is the ability to examine and contest an agent's explanation, rather than an assumption that a readable explanation is correct or that feedback necessarily improves future proposals.

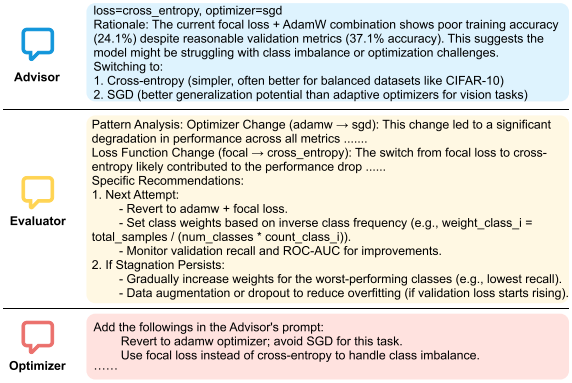


Figure 4: Recorded agent interaction: the Advisor proposes configuration changes, the Evaluator assesses outcomes, and the Optimizer synthesizes guidance for subsequent proposals. The displayed rationales are agent outputs.

Appendix G records both retained candidates and failed sequences. In the successful example, the record pairs `feature_dropout` with Smooth L1 loss and Adam, learning-rate/dropout proposals, and an Evaluator decision to retain the candidate following reported validation improvement. In the failed sequence, degraded validation outcomes prompt guidance to return to AdamW with cross-entropy and use smaller learning-rate increments. The record keeps these recommendations alongside attempted changes and outcomes, including termination without an improved candidate. The configuration loop can return candidate settings, measurements, retention decisions, and guidance for a subsequent experiment, as these examples illustrate.

Table 4: Architecture extension, backbone comparison, and timing. (a) MiniViT on CIFAR-100. (b) LGT accuracy on CIFAR-10 across LLMs. (c) Reported per-epoch training and LLM time on MiniViT/CIFAR-100. Component rows report shares of total epoch time; the total row gives its ratio to training alone.

(a) MiniViT on CIFAR-100

Variant	Acc.	Prec.	AUC	F1
Baseline (No Tuning)	27.22	26.53	91.76	21.32
Architecture only	38.07	39.89	97.09	37.17
Features only	31.27	30.29	95.23	29.72
Strategy only	32.19	33.14	95.69	32.15
Hyperparams only	35.14	34.25	96.92	33.26
Full LGT	**41.03**	**40.93**	**98.23**	**39.06**

(b) LLM backbones (CIFAR-10)

Backbone	Size	Acc.	Δ
No LLM (baseline)	–	69.01	–
Llama-3-8B-Inst.	8B	76.52	+7.51
Qwen3-8B	8B	76.84	+7.83
GPT-oss-120B	120B	78.89	+9.88
DeepSeek-V3.2	MoE	**79.64**	**+10.63**

(c) Per-epoch wall-clock

Component	Time (s)	Share / ratio
Training	85.0	94.4%
LLM inference	4.5	5.0%
Parse + update	0.5	0.6%
Total	**90.0**	**1.06×**

5.4 Generalization, Robustness, and Efficiency

Table 4 examines a compact transformer, alternative proposal LLMs, and local runtime overhead; Appendix H provides the supporting setup.

Architectural generalization to ViTs. To examine configurations beyond the CNN/MLP/LSTM backbones, we apply LGT to a Mini Vision Transformer on CIFAR-100, initialized with patch size 4, embedding dimension 64, 2 attention heads, 2 layers, MLP ratio 4.0, and dropout 0.1. Table 4(a) reports accuracy increasing from 27.22% to 41.03% (+13.81 points) and F1 from 21.32 to 39.06 (+17.74 points). Architecture-only tuning reaches 38.07% accuracy, accounting for 10.85 points of the reported gain; full LGT is 2.96 points higher. This is a feasibility result for configuration experimentation on the specified compact transformer. Without uncertainty estimates or matched role controls, the remaining margin cannot be attributed to cross-domain coordination or extrapolated to large-scale transformer research. Appendix L retains the extended interpretation.

The reported final accepted configuration keeps patch size 4 and uses embedding dimension 128, 4 heads, 4 layers, MLP ratio 3.0, and dropout 0.2. It combines AdamW and a cosine schedule with learning rate 3×10^{-4}, weight decay 0.05, and RandAugment as the dominant feature transformation. This recorded candidate specifies the architecture, training, hyperparameter, and augmentation choices returned together by the configuration experiment (Appendix H.1).

LLM backbones. We evaluate Llama-3-8B-Instruct, Qwen3-8B, GPT-oss-120B, and the default DeepSeek-V3.2 on CIFAR-10. All four improve over No Tuning by at least 7.51 accuracy points, with a 3.12-point range from Llama-3-8B-Instruct to DeepSeek-V3.2. The direction of the reported improvement therefore persists across the tested backbones, an encouraging observation for using the configuration loop with different proposal models. The comparison concerns the loop under these backbone choices; it does not isolate agent-role or gate effects from model quality, sampling, and other shared protocol choices, or order the tested architectures by a single notion of model size.

Computational overhead. The textual loop adds LLM inference and parsing to numerical training. Table 4(c) reports approximately 85.0 s training, 4.5 s LLM inference, and 0.5 s parsing/updates per epoch on MiniViT/CIFAR-100: 90.0 s total, or $1.06\times$ the training component. This provides a local description of the additional language-mediated work. Connecting selected trajectories to total research cost requires counting all candidate epochs, rejected attempts, architecture restarts, validation, and language calls, even when a retained trajectory reaches a target in fewer epochs. Appendix H.3 retains the detailed accounting discussion and identifies missing per-agent latency and token counts. The measured component ratio therefore does not establish overall cost parity or savings against another research agent.

Table 5: Single-domain ablations on MNIST and Housing, alongside untuned and full-system results.

Metric	Baseline (No LLM)	Model Arch. Only	Feature Eng. Only	Training Strat. Only	Hyperparams Only	Full LGT System
MNIST AUC	$98.37_{0.08}$	$99.51_{0.07}$	$98.18_{0.25}$	$99.96_{0.12}$	$99.89_{0.13}$	$\mathbf{99.99}_{0.10}$
Housing MAE	$0.458_{0.026}$	$0.437_{0.029}$	$0.455_{0.034}$	$0.424_{0.027}$	$0.416_{0.025}$	$\mathbf{0.402}_{0.021}$

5.5 Ablation Study and Component Analysis

Table 5 evaluates Model Architecture, Feature Engineering, Training Strategy, and Hyperparameters individually on classification (MNIST) and regression (Housing). These variants remove editable domains, rather than Advisor, Evaluator, or Optimizer roles. The strongest single-domain setting depends on the task: Training Strategy reaches MNIST AUC 99.96, while Hyperparameter tuning gives Housing MAE 0.416. Feature Engineering alone reduces MNIST AUC from the untuned 98.37 to 98.18, an adverse outcome whose cause is not identifiable from the aggregate table.

This task dependence matters for configuration experimentation. The single-domain comparisons show where particular interventions were useful under the reported settings; they motivate treating training choices as hypotheses whose consequences must be measured. Adding a domain need not improve an experiment; the results do not demonstrate optimal allocation of search effort.

The full system has the best displayed mean on both tasks. Its MNIST AUC is 99.99 versus 99.96 for Training Strategy, with reported standard deviations of 0.10 and 0.12; Housing MAE is 0.402 versus 0.416 for Hyperparameter-only tuning, with standard deviations of 0.021 and 0.025. The margins do not identify coordination as their cause, and MNIST AUC aggregation requires verification (Appendix J). Testing role decomposition requires controls that fix the action space, backbone, retention rule, and budget.

6 Conclusion

LGT organizes configuration experiments for automated ML research through language-generated proposals, numerical assessment, and feedback across architecture, feature engineering, training strategy, and hyperparameters. Across seven datasets, it reports the best displayed means against eleven baselines spanning untuned training and classical tuning. Its trajectories record proposed changes and expose limits of their explanations. These task-level findings motivate the loop as a component of broader research workflows, without establishing autonomous scientific discovery. Matched LLM baselines, role controls, full resource accounting, and resolved evaluation checks remain necessary to establish comparative research efficiency and mechanism-level benefits.

AI USE STATEMENT

For this manuscript revision, an LLM assistant supported conceptual framing of the existing method, interpretation of reported results, literature lookup, and manuscript editing. Appendix **??** describes its role and the checks performed for this revision.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0e9e708b6f48e14fd0ac29e167413f76-Abstract-Conference.html.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A Next-Generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2623–2631, 2019. URL https://arxiv.org/abs/1907.10902.

Eser Aygün, Anastasiya Belyaeva, Gheorghe Comanici, Marc Coram, Hao Cui, Jake Garrison, Renee Johnston, Anton Kast, Cory Y. McLean, Peter Norgaard, Zahra Shamsi, David Smalling, James Thompson, Subhashini Venugopalan, Brian P. Williams, Chujun He, Sarah Martinson, Martyna Plomecka, Lai Wei, Yuchen Zhou, Qian-Ze Zhu, Matthew Abraham, Erica Brand, Anna Bulanova, Jeffrey A. Cardille, Chris Co, Scott Ellsworth, Grace Joseph, Malcolm Kane, Ryan Krueger, Johan Kartiwa, Dan Liebling, Jan-Matthis Lueckmann, Paul Raccuglia, Xuefei Julie Wang, Katherine Chou, James Manyika, Yossi Matias, John C. Platt, Lizzie Dorfman, Shibl Mourad, and Michael P. Brenner. An AI system to help scientists write expert-level empirical software. *Nature*, 654(8120):909–916, 2026. doi: 10.1038/s41586-026-10658-6. URL https://doi.org/10.1038/s41586-026-10658-6.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012. URL https://jmlr.org/papers/v13/bergstra12a.html.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 50466–50494, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/7e3767db483c942b883eb4f8cfb74e31-Abstract-Conference.html.

Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Richard Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, Marc'Aurelio Ranzato, et al. Towards learning universal hyperparameter optimizers with transformers. In *Advances in Neural Information Processing Systems*, volume 35, pp. 32053–32068, 2022. doi: 10.52202/068431-2323. URL https://doi.org/10.52202/068431-2323.

Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision support systems*, 47(4): 547–553, 2009. doi: 10.1016/j.dss.2009.05.016. URL https://doi.org/10.1016/j.dss.2009.05.016.

DeepSeek-AI. DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models. *arXiv preprint arXiv:2512.02556*, 2025. URL https://arxiv.org/abs/2512.02556.

Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012. doi: 10.1109/msp.2012.2211477. URL https://doi.org/10.1109/msp.2012.2211477.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *International conference on machine learning*, pp. 1437–1446. PMLR, 2018. URL https://proceedings.mlr.press/v80/falkner18a.html.

Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7 (2):179–188, 1936. doi: 10.1111/j.1469-1809.1936.tb02137.x. URL https://doi.org/10.1111/j.1469-1809.1936.tb02137.x.

Peter I. Frazier. A Tutorial on Bayesian Optimization. *arXiv preprint arXiv:1807.02811*, 2018. URL https://arxiv.org/abs/1807.02811.

Ziyao Huang, Weiwei Wu, Kui Wu, Wei-Bin Lee, and Jianping Wang. CALM: Co-evolution of Algorithms and Language Model for Automatic Heuristic Design. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/75ed738c083f8941cfc990245e4f6265-Abstract-Conference.html.

Muhammad Bin Imran. Housing Price Prediction Data. Kaggle dataset, 2023. URL https://www.kaggle.com/datasets/muhammadbinimran/housing-price-prediction-data.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population Based Training of Neural Networks. *arXiv preprint arXiv:1711.09846*, 2017. URL https://arxiv.org/abs/1711.09846.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. AIDE: AI-Driven Exploration in the Space of Code. *arXiv preprint arXiv:2502.13138*, 2025. doi: 10.48550/arXiv.2502.13138. URL https://arxiv.org/abs/2502.13138.

Shubhra Kanti Karmaker, Md Mahadi Hassan, Micah J Smith, Lei Xu, Chengxiang Zhai, and Kalyan Veeramachaneni. Automl to date and beyond: Challenges and opportunities. *ACM Computing Surveys*, 54(8):1–36, 2021. doi: 10.1145/3470918. URL https://doi.org/10.1145/3470918.

Andrej Karpathy. char-rnn: Multi-layer Recurrent Neural Networks for Character-Level Language Models. GitHub repository, 2015. URL https://github.com/karpathy/char-rnn.

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

Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791. URL https://doi.org/10.1109/5.726791.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018. URL https://jmlr.org/papers/v18/16-558.html.

Petro Liashchynskyi and Pavlo Liashchynskyi. Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS. *arXiv preprint arXiv:1912.06059*, 2019. URL https://arxiv.org/abs/1912.06059.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *Journal of Machine Learning Research*, 23(54): 1–9, 2022. URL https://jmlr.org/papers/v23/21-0888.html.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1806.09055.

Siyi Liu, Chen Gao, and Yong Li. AgentHPO: Large language model agent for hyper-parameter optimization. In *Conference on Parsimony and Learning*, volume 280 of *Proceedings of Machine Learning Research*, pp. 1146–1169. PMLR, 2025a. URL https://proceedings.mlr.press/v280/liu25c.html.

Xiaogeng Liu, Peiran Li, Edward Suh, Yevgeniy Vorobeychik, Zhuoqing Mao, Somesh Jha, Patrick McDaniel, Huan Sun, Bo Li, and Chaowei Xiao. AutoDAN-Turbo: A Lifelong Agent for Strategy Self-Exploration to Jailbreak LLMs. In *International Conference on Learning Representations*, 2025b. URL https://openreview.net/pdf?id=bhK7U37VW8.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. Towards end-to-end automation of AI research. *Nature*, 651(8107): 914–919, 2026. doi: 10.1038/s41586-026-10265-5. URL https://doi.org/10.1038/s41586-026-10265-5.

Zeyuan Ma, Hongshu Guo, Jiacheng Chen, Guojun Peng, Zhiguang Cao, Yining Ma, and Yue-Jiao Gong. LLaMoCo: Instruction Tuning of Large Language Models for Optimization Code Generation. *arXiv preprint arXiv:2403.01131*, 2024. URL https://arxiv.org/abs/2403.01131.

Matthew MacKay, Paul Vicol, Jon Lorraine, David Duvenaud, and Roger Grosse. Self-Tuning Networks: Bilevel Optimization of Hyperparameters using Structured Best-Response Functions. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=r1eEG20qKQ.

Meher Bhaskar Madiraju and Meher Sai Preetam Madiraju. OptiMindTune: A Multi-Agent Framework for Intelligent Hyperparameter Optimization. *arXiv preprint arXiv:2505.19205*, 2025. URL https://arxiv.org/abs/2505.19205.

Allen Nie, Ching-An Cheng, Andrey Kolobov, and Adith Swaminathan. The Importance of Directional Feedback for LLM-based Optimizers. *arXiv preprint arXiv:2405.16434*, 2024. URL https://arxiv.org/abs/2405.16434.

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

Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. KerasTuner. GitHub repository, 2019. URL https://github.com/keras-team/keras-tuner.

Jack Parker-Holder, Vu Nguyen, and Stephen J. Roberts. Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits. In *Advances in Neural Information Processing Systems*, volume 33, pp. 17200–17211, 2020. URL https://papers.neurips.cc/paper_files/paper/2020/hash/c7af0926b294e47e52e46cfebe173f20-Abstract.html.

Mohamad Zaim Awang Pon and Krishna Prakash K K. Hyperparameter tuning of deep learning models in keras. *Sparklinglight Transactions on Artificial Intelligence and Quantum Computing (STAIQC)*, 1(1):36–40, 2021. doi: 10.55011/STAIQC.2021.1104. URL https://www.sparklinglightpublisher.com/index.php/staiqc/article/view/5.

Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):60, 2019. doi: 10.1186/s40537-019-0197-0. URL https://doi.org/10.1186/s40537-019-0197-0.

Jiabin Tang, Lianghao Xia, Zhonghang Li, and Chao Huang. AI-Researcher: Autonomous scientific innovation. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 9481–9520. Curran Associates, Inc., 2025. doi: 10.52202/085713-0320. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/0d904d300a105809a2114d727851e759-Paper-Conference.pdf.

Laksika Tharmalingam. Water Quality and Potability. Kaggle dataset, 2023. URL https://www.kaggle.com/datasets/uom190346a/water-quality-and-potability.

Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, 9(2):187–212, 2022. doi: 10.1007/s40745-020-00253-5. URL https://doi.org/10.1007/s40745-020-00253-5.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist-v2: Workshop-Level Automated Scientific Discovery via Agentic Tree Search. *arXiv preprint arXiv:2504.08066*, 2025. doi: 10.48550/arXiv.2504.08066. URL https://arxiv.org/abs/2504.08066.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/3339f19c5fcee3ad74502947a32be9e6-Abstract-Conference.html.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nature*, 639(8055):609–616, 2025. doi: 10.1038/s41586-025-08661-4. URL https://doi.org/10.1038/s41586-025-08661-4.

Jusheng Zhang, Kaitong Cai, Yijia Fan, Ningyuan Liu, and Keze Wang. MAT-Agent: Adaptive Multi-Agent Training Optimization. In *The Thirty-Ninth Annual Conference on Neural Information Processing Systems*, 2025. doi: 10.52202/085713-2589. URL https://doi.org/10.52202/085713-2589.

Marc-André Zöller and Marco F Huber. Benchmark and survey of automated machine learning frameworks. *Journal of artificial intelligence research*, 70:409–472, 2021. doi: 10.1613/jair.1.11854. URL https://doi.org/10.1613/jair.1.11854.

A Dataset Details

This section provides comprehensive information about all datasets used in our experiments. Table 6 summarizes the key characteristics of each dataset, including size, task type, number of classes, features, and descriptions.

Table 6: Overview of benchmark datasets used for evaluation. The collection includes four classification tasks (MNIST, CIFAR-10, Iris, Water Quality), two regression tasks (Housing, Wine Quality), and one text generation task (Tiny Shakespeare).

Dataset	Characteristics and description
MNIST (LeCun et al., 1998)	**Task Type:** Classification; **Samples:** 70,000; **Features:** 784; **Classes:** 10. Handwritten digit dataset containing 70,000 grayscale images of size 28×28 pixels, split into 60,000 training and 10,000 test images. The dataset includes digits 0–9 with class frequencies determined by the dataset. Images are simple with low noise, making it ideal for evaluating convolutional networks, data augmentation, and regularization strategies.
CIFAR-10 (Krizhevsky, 2009)	**Task Type:** Classification; **Samples:** 60,000; **Features:** 3,072; **Classes:** 10. Color image dataset containing 60,000 32×32 RGB images across 10 classes (*airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck*), with 50,000 training and 10,000 test images (6,000 per class). Despite small image size, the dataset features rich colors and varied object poses, serving as a classic benchmark for CNN architectures and regularization.
Iris (Fisher, 1936)	**Task Type:** Classification; **Samples:** 150; **Features:** 4; **Classes:** 3. One of the earliest datasets in classification literature, widely used in statistics and machine learning. Contains 150 samples of iris flowers from 3 species (50 instances each): *Setosa*, *Versicolor*, and *Virginica*. Features include sepal length, sepal width, petal length, and petal width. One class is linearly separable from the others, while the remaining two are not separable from each other.
Water Quality (Tharmalingam, 2023)	**Task Type:** Classification; **Samples:** 3,276; **Features:** 9; **Classes:** 2. Binary classification dataset for water potability assessment. Contains water quality measurements including pH, hardness, solids, chloramines, sulfate, conductivity, organic carbon, trihalomethanes, and turbidity. The "Potability" column indicates whether water is suitable for human consumption.
Housing (Imran, 2023)	**Task Type:** Regression; **Samples:** 50,000; **Features:** 5; **Classes:** –. Synthetic housing price regression dataset described as containing 50,000 records. Each record includes features such as building area, number of bedrooms, number of bathrooms, neighborhood type, and year built. The target variable is continuous house price.
Wine Quality (Cortez et al., 2009)	**Task Type:** Regression; **Samples:** 1,599; **Features:** 11; **Classes:** –. The red wine dataset containing 1,599 samples characterized by 11 physicochemical attributes: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free/total sulfur dioxide, density, pH, sulphates, and alcohol. The output is a median quality score (0–10) provided by wine experts. Suitable for both regression and classification tasks.
Tiny Shakespeare (Karpathy, 2015)	**Task Type:** Generation; **Samples:** 40,000; **Features:** –; **Classes:** –. A small, classic text dataset commonly used for character-level language modeling. Consists of concatenated works of William Shakespeare, providing approximately 40,000 character sequences for text generation tasks. Widely used for testing recurrent neural networks and transformer architectures.

A.1 Data Preprocessing

The preprocessing below describes the reported experimental procedure. Its 80/20 notation must be reconciled with the validation subdivision in the main text; fitted preprocessing partitions, raw-text window construction, and regression target scaling require implementation-level verification.

Classification Tasks. For classification tasks (MNIST, CIFAR-10, Iris, Water Quality), we employed an 80%/20% train/test split with stratified sampling to maintain class distribution balance. Feature normalization was performed using StandardScaler to achieve zero mean and unit variance across all features. For image datasets (MNIST / CIFAR-10), pixel values were additionally normalized to the

Algorithm 1 Language-Guided Tuning (validation-gated, incumbent form)

Require: initial configuration c_0, Advisor prompt p_0, margin schedule $\{\eta_t\}$, validation score S_{val}
Ensure: accepted configuration $c^\star$

$c^\star \leftarrow c_0$; $\mathcal{H}_0 \leftarrow \emptyset$; $p_1 \leftarrow p_0$
$(\mathcal{M}^\star, \theta^\star) \leftarrow \text{train_epoch}(f_{c_0}, \mathcal{D}_{\text{train}})$ {incumbent metrics + checkpoint}
for $t = 1, 2, \ldots, T$ **do**
  $\Delta c_t^{\text{NL}} \leftarrow \mathbb{A}_{\text{adv}}(c^\star, \mathcal{M}^\star, \mathcal{H}_{t-1}, \mathcal{C};\, p_t)$ {propose an edit}
  $\hat{c}_t \leftarrow \mathcal{P}^{\text{cfg}}_{\text{parse}}(\Delta c_t^{\text{NL}})$ {candidate configuration}
  $(\hat{\mathcal{M}}_t, \hat{\theta}_t) \leftarrow \text{train_epoch}(f_{\hat{c}_t}, \mathcal{D}_{\text{train}} \mid \theta^\star)$ {one epoch from the incumbent checkpoint}
  $\mathcal{I}^{\text{NL}}_{\text{eval}} \leftarrow \mathbb{A}_{\text{eval}}(\hat{c}_t, \hat{\mathcal{M}}_t,\, c^\star, \mathcal{M}^\star,\, \mathcal{H}_{t-1})$ {candidate *vs.* incumbent}
  $\text{success}_t \leftarrow \mathcal{P}^{\text{dec}}_{\text{parse}}(\mathcal{I}^{\text{NL}}_{\text{eval}})$; $s_t^{\text{NL}} \leftarrow \mathcal{P}^{\text{sgg}}_{\text{parse}}(\mathcal{I}^{\text{NL}}_{\text{eval}})$
  if $\text{success}_t = 1$ **and** $S_{\text{val}}(\hat{\mathcal{M}}_t) > S_{\text{val}}(\mathcal{M}^\star) + \eta_t$ **then**
    $c^\star \leftarrow \hat{c}_t$; $\mathcal{M}^\star \leftarrow \hat{\mathcal{M}}_t$; $\theta^\star \leftarrow \hat{\theta}_t$ {accept: promote candidate}
  else
    discard $\hat{c}_t, \hat{\theta}_t$; restore $(c^\star, \theta^\star)$ {reject: roll back to incumbent}
  end if
  $\Delta p_t^{\text{NL}} \leftarrow \mathbb{A}_{\text{opt}}(\hat{\mathcal{M}}_t, \hat{c}_t, p_t, s_t^{\text{NL}})$
  $p_{t+1} \leftarrow \mathcal{P}^{\text{prm}}_{\text{parse}}(\Delta p_t^{\text{NL}})$ {update Advisor guidance context}
  $\mathcal{H}_t \leftarrow \mathcal{H}_{t-1} \cup \{(\hat{c}_t,\, \hat{\mathcal{M}}_t,\, \Delta c_t^{\text{NL}},\, \mathcal{I}^{\text{NL}}_{\text{eval}},\, p_t,\, \text{success}_t)\}$
end for
return $c^\star$

[0, 1] range to facilitate neural network training. Missing values in numerical features, when present, were handled using mean imputation to preserve data integrity while avoiding information loss.

Regression Tasks. Regression tasks (Housing, Wine Quality) followed a similar preprocessing pipeline with an 80%/20% train/test split. Feature normalization using StandardScaler was applied to ensure all input features contributed equally to model training. The target variables were kept in their original scale without transformation to maintain interpretability of predictions. For outlier handling, we flagged values beyond 3 standard deviations from the mean but retained them in the dataset, as real-world regression problems often contain legitimate extreme values that should not be removed.

Generation Task. The text generation task (Tiny Shakespeare) required different preprocessing tailored to sequential data. We employed character-level tokenization, converting the text into sequences of individual characters. Each training sequence was set to 100 characters in length, providing sufficient context for the model to learn language patterns while maintaining computational efficiency. The vocabulary size was determined dynamically based on the unique characters present in the corpus, ensuring complete coverage of the text domain without introducing complexity.

To ensure fair comparison across all methods and maintain experimental rigor, we applied standardized preprocessing procedures to all datasets. The experimental record specifies seeds 42–51 for 10 runs; fixed seeds support reproducibility but do not by themselves establish independence or statistical significance. All baseline methods and LGT used the same train/test splits for fair comparison.

B Epoch-Level Training Algorithm

The complete training algorithm operates each epoch in incumbent form: the loop always holds one *accepted* configuration $c^\star$ together with the checkpoint $\theta^\star$ of its weights and optimizer state. A proposal is trained for one epoch, compared against the incumbent on the validation split, and either promoted to incumbent or discarded by restoring the checkpoint. The formal rule carries forward only a retained proposal. Complete restoration and an unchanged validation protocol preserve the recorded incumbent score after a rejection; this does not guarantee population performance or remove adaptive validation bias. These distinctions specify the scope of Appendix E.

C Baselines

This section primarily introduces some of the baselines we used in our experiments, including baseline models and baseline methods.

Table 7: Overview of model structure configurations. The collection includes MLP models evaluated on four datasets (Iris, Housing, Wine Quality, Water Quality), CNN models on two image datasets (MNIST, CIFAR-10), and an LSTM model on one text dataset (Tiny Shakespeare).

Model Type	Dataset	Layer Number	Dropout	Others
MLP	Iris	2	0.3	hidden_dims: [16, 16]
	Housing	1	0.5	hidden_dims: [2]
	Wine Quality	1	0.5	hidden_dims: [16]
	Water Quality	2	0.2	hidden_dims: [256, 512]
CNN	MNIST	2	0.25	kernel_size: 3 stride: 1 padding: 1 pool_size: 2 fc_layers: [64]
	CIFAR-10	3	0.4	kernel_size: 3 padding: 1 stride: 1 fc_layers:[128] pool_size: 2
LSTM	Tiny Shakespeare	2	0.5	bidirectional: false embedding_dim: 128 hidden_dim: 256

C.1 Baseline Models

Table 7 details the specific configurations of the models used in our experiments. We set distinct parameters for different tasks and defined three levels of model parameters, simple, adapted, and complex, to validate whether our model structure optimization was effective. The table records these starting configurations; the result comparisons assess subsequent tuning outcomes, without establishing that the final capacity is optimal for the task.

C.2 Baseline Methods

We comprehensively evaluate LGT against eleven baseline methods covering untuned training and classical configuration-search approaches:

No Tuning: Provides an untuned reference using default configurations without any optimization. This baseline employs standard architectures with commonly used hyperparameters to establish the performance achievable without tuning effort.

Random Search (Bergstra & Bengio, 2012): Randomly samples configurations from predefined hyperparameter spaces. Despite its simplicity, random search often outperforms grid search in high-dimensional spaces due to its ability to explore diverse configurations.

Grid Search (Liashchynskyi & Liashchynskyi, 2019): Systematically explores discretized configuration spaces through exhaustive enumeration. Exhaustive evaluation can identify the best observed grid point, but the capped candidate budget used here need not exhaust the grid and does not guarantee a global optimum.

Bayesian Optimization (Frazier, 2018): Uses Gaussian process surrogate models to approximate the objective function, selecting new configurations via expected improvement acquisition. This method efficiently balances exploration and exploitation in continuous spaces.

Optuna (Akiba et al., 2019): Employs tree-structured Parzen estimators (TPE) for efficient hyperparameter optimization. Optuna's define-by-run API enables dynamic search space construction and pruning of unpromising trials.

SMAC3 (Lindauer et al., 2022): Sequential Model-based Algorithm Configuration combines random forest surrogate models with aggressive racing strategies. SMAC3 handles both categorical and continuous hyperparameters while providing robust uncertainty estimates.

KerasTuner (O'Malley et al., 2019; Pon & K K, 2021): Provides a software framework with several search algorithms, including Bayesian optimization; the exact algorithm and version used in a comparison are part of its experimental configuration. It includes built-in support for common neural network hyperparameters and early stopping strategies.

Hyperband (Li et al., 2018): Dynamically allocates computational resources using successive halving, enabling efficient exploration of many configurations by early-stopping poor performers. This bandit-based approach significantly reduces total computational cost.

BOHB (Falkner et al., 2018): Combines the benefits of Bayesian optimization with Hyperband's efficient scheduling. BOHB uses kernel density estimators to guide the search while leveraging Hyperband's resource allocation for computational efficiency.

Population Based Training (PBT) (Jaderberg et al., 2017): Jointly optimizes hyperparameters and model weights through evolutionary strategies. PBT maintains a population of models, periodically replacing underperforming members with mutated versions of better performers.

Neural Architecture Search (NAS) (Liu et al., 2019): Optimizes model architecture using differentiable architecture search (DARTS). While primarily designed for architecture optimization, we include it as an architecture-focused comparator.

D Computational Complexity Analysis

Let N be the number of training examples, B the batch size, and $S = \lceil N/B \rceil$ the number of update steps per epoch. Let T be the number of epochs. Let P denote the number of model parameters. Let $\mathcal{C}_{\text{step}}(P, A)$ be the wall-time for one training step of f_c, where A is the activation footprint per step (proportional to B and sequence/feature sizes). Let $\mathcal{A} = \{\text{adv}, \text{eval}, \text{opt}\}$ denote the three agents. For agent $a \in \mathcal{A}$ at epoch t, let $L_a^{(t)}$ be the number of *output* tokens produced, and let κ be the per-token decoding constant for the LLM backend.

Numerical loop cost.

$$\mathsf{Cost}_{\text{train}} \;=\; T \cdot S \cdot \mathcal{C}_{\text{step}}(P, A) \tag{4}$$

$$\mathcal{C}_{\text{step}}(P, A) = \Theta(P) + \Theta(A) \tag{5}$$

Textual loop cost. For the simplified case of one call per role per epoch and decoding-dominated latency, the language component can be written as:

$$\mathsf{Cost}_{\text{LLM}} \;=\; \sum_{t=1}^{T} \sum_{a \in \mathcal{A}} \kappa \, L_a^{(t)} \;=\; O\big(T \, |\mathcal{A}| \, \kappa \, \overline{L}\big) \tag{6}$$

where $\overline{L}$ bounds the agent outputs. This simplified expression excludes prompt prefill, network latency, retries, and variable call counts; those components need not be lower order in practice.

Training-plus-decoding cost under the simplified model.

$$\mathsf{Cost}_{\text{total}} \;=\; TS\mathcal{C}_{\text{step}}(P, A) \;+\; O\big(T \, |\mathcal{A}| \, \kappa \, \overline{L}\big) \tag{7}$$

With bounded output length and the stated call-count assumptions, the simplified textual component grows as $O(T)$, in addition to $O(TS\mathcal{C}_{\text{step}}(P, A))$ training. The per-step approximation also presumes a fixed computational pattern; parameter and activation counts do not by themselves characterize every architecture's runtime. Detailed convergence analysis appears in Appendix E.

A full experiment-search account must additionally include all attempted candidates, rather than only retained epochs. Writing b_t for candidate t's training horizon and $C_{\text{train},t}$ for its per-epoch training cost, an accounting identity is

$$C_{\text{search}} = C_{\text{initial}} + C_{\text{final}} + \sum_{t=1}^{N_{\text{attempts}}} \left(b_t C_{\text{train},t} + C_{\text{eval},t} + C_{\text{LLM},t} + C_{\text{state},t}\right). \tag{8}$$

Here $C_{\text{eval},t}$ includes candidate validation and any incumbent reevaluation, $C_{\text{LLM},t}$ includes all inference and communication, and $C_{\text{state},t}$ includes transfer, restart, and rollback. Initial and final terms include their corresponding construction, training, and scoring where applicable. This identity makes the required cost components explicit; it is not an additional runtime measurement.

E CONVERGENCE ANALYSIS

We study the outer textual loop that proposes configuration updates and accepts them through validation feedback. The statements below are conditional analyses of an idealized acceptance process, not verified guarantees for the reported implementation. Let $F(c)$ denote a fixed population validation loss associated with an evaluated candidate. A configuration must either index a complete prediction state, including its learned weights, or be evaluated under a fixed initialization and training protocol; the four configuration fields alone do not determine the loss of a stateful training run. We retain c_t for this analytical argument and distinguish a fixed response surface from the live state transitions discussed in the main text. At epoch t, the Advisor proposes $\tilde{c}_{t+1}$. For proposals not vetoed by the Evaluator, the score-based acceptance rule is

$$\widehat{F}_t(\tilde{c}_{t+1}) \le \widehat{F}_t(c_t) - \eta_t \quad \Rightarrow \quad c_{t+1} = \tilde{c}_{t+1}, \text{ else } c_{t+1} = c_t, \tag{9}$$

The weak inequality in this analytical rule allows a tie at the margin; the strict gate in Equation 3 is a more restrictive case and is covered by the same conditional statements. Here $\eta_t \ge 0$, $\widehat{F}_t$ is the validation estimate at epoch t, and F is bounded below by $F_\star$. A vetoed proposal retains the incumbent without applying the promotion branch; a candidate that fails the score condition cannot be promoted. Rejection means exact retention of the incumbent state. A fixed deterministic empirical validation loss obeys a monotonicity property under this rule, but an empirical comparison alone does not establish monotonic population loss.

Validation noise model. For a conditional noise analysis, assume $\widehat{F}_t(c) = F(c) + \epsilon_t(c)$ with $\mathbb{E}[\epsilon_t(c) \mid c] = 0$ and $\epsilon_t(c)$ sub-Gaussian with parameter σ_t^2/n_t for n_t validation samples. These marginal conditions alone do not justify a bound for an adaptively selected pair. For the misacceptance bound below, additionally condition on the history and the chosen incumbent–candidate pair, and assume that $\epsilon_t(\tilde{c}_{t+1}) - \epsilon_t(c_t)$ is centered sub-Gaussian with variance proxy at most σ_t^2/n_t. This is a separate assumption about the difference: two individually sub-Gaussian errors need not have the same variance proxy for their difference. Take the displayed schedules and bounds to be deterministic. Fresh evaluation data independent of candidate selection can support such a conditional model; adaptive reuse of the same finite validation set, particularly with cached incumbent scores, does not establish it. The reported experiments do not verify these noise assumptions.

E.1 MONOTONICITY IN EXPECTATION

Lemma 1 (One step descent in expectation)**.** *For integrable compared losses, the following relation holds as an accounting identity (and therefore also as the displayed weak inequality):*

$$\mathbb{E}[F(c_{t+1}) \,|\, c_t] \le F(c_t) - \mathbb{E}[\Delta_t \,|\, c_t]\,, \tag{10}$$

where $\Delta_t := F(c_t) - F(\tilde{c}_{t+1})$ if the update is accepted and 0 otherwise. It is a descent statement only under the additional condition $\mathbb{E}[\Delta_t \mid c_t] \ge 0$. Neither centered validation noise nor the acceptance threshold by itself implies this condition.

Sketch. On acceptance, the definition gives $F(c_{t+1}) = F(c_t) - \Delta_t$; on rejection the same equality holds because $\Delta_t = 0$ and the incumbent is retained. Taking conditional expectations gives equality in the displayed relation. A falsely accepted worsening proposal has $\Delta_t < 0$. Nonnegative conditional drift must therefore be assumed or proved separately; a bound on the probability of false acceptance alone does not make that drift nonnegative. □

E.2 CONVERGENCE OF VALIDATION LOSSES

Theorem 2 (Almost sure convergence of $F(c_t)$)**.** *Assume $F(c_1)$ is finite, F is bounded below, and, almost surely, every accepted update satisfies $F(\tilde{c}_{t+1}) \le F(c_t)$. Then $F(c_t)$ converges almost surely and $\sum_{t=1}^{\infty} \Delta_t < \infty$. In particular, the number of accepted updates with improvement at least $\varepsilon > 0$ is finite and bounded by $\lceil (F(c_1) - F_\star)/\varepsilon \rceil$. This conclusion holds in particular when $\sum_{t=1}^{\infty} \eta_t < \infty$, but summability of the margins is neither needed for this non-worsening case nor sufficient by itself in the noisy case.*

Sketch. Define $X_t := F(c_t)$. On the probability-one event in the assumptions, every accepted increment has $\Delta_t \ge 0$ and every rejection leaves X_t unchanged. Thus X_t is nonincreasing and

bounded below, and it converges. Telescoping gives $\sum_{t=1}^{T} \Delta_t = F(c_1) - F(c_{T+1}) \le F(c_1) - F_\star$. Taking the limit establishes summability and the stated bound on improvements of size at least ε. The acceptance rule with noisy estimates does not supply the non-worsening premise; in particular, $\sum_t \eta_t < \infty$ is not a bound on cumulative estimation errors. □

Misacceptance control. Under the additional conditional difference-noise assumption above, a false acceptance of a candidate with $F(\tilde{c}_{t+1}) > F(c_t)$ requires its estimation-error difference to be at most $-\eta_t$. A conditional sub-Gaussian tail bound, followed by averaging over the history, gives

$$\Pr(\text{false accept at } t) \le \exp\Big(- \tfrac{n_t \eta_t^2}{2\sigma_t^2} \Big).$$

Choose $\eta_t = \sigma_t \sqrt{2(1+\delta)\log t \,/\, n_t}$ for any $\delta > 0$ and $t \ge 2$. Then $\sum_t \Pr(\text{false accept at } t) < \infty$, so by Borel–Cantelli only finitely many false acceptances occur almost surely; thereafter $F(c_t)$ is nonincreasing. With finite objective values and a lower bound, this also gives eventual convergence under these stronger assumptions. It does not imply monotonicity from the first iteration or the preceding theorem's improvement count bound using only the initial loss gap: any positive loss increases must also be charged to that bound. For constant $n_t = n$ and $\sigma_t = \sigma > 0$, this margin schedule is not summable. It therefore cannot be justified by $\sum_t \eta_t < \infty$, and its assumptions are not established by repeatedly querying the validation split used in the reported experiments.

E.3 Discrete and continuous components

Finite discrete choices. Suppose the admissible sets for architecture, training strategy, and feature transformations are finite, and consider a discrete subproblem with the other components and evaluation protocol fixed. Under error-free comparisons and a fixed $\eta > 0$, only finitely many updates can be accepted. The proposal process need not halt: an LLM may repeatedly offer rejected candidates. An η-local-minimum conclusion additionally requires coverage of an explicit neighborhood and a stopping rule that tests all its members. If all such candidates are evaluated and none decreases F by at least η, the returned configuration is an η-local minimum with respect to that neighborhood. Such coverage is not guaranteed by the supplied LLM proposer. With a schedule $\eta_t \downarrow 0$, error-free acceptance still makes F nonincreasing and bounded below, so $F(c_t)$ converges and the number of strict improvements by at least any fixed $\varepsilon > 0$ is finite.

Projected continuous hyperparameters. Write $c = (d, \lambda)$ with discrete part d and continuous hyperparameters $\lambda \in \mathcal{C}_H$ compact. For this conditional statement, take $F(d, \lambda)$ to be a fixed differentiable response surface, rather than assuming that a changing live checkpoint defines such a surface automatically. Assume accepted hyperparameter moves satisfy the sufficient decrease condition

$$F(d_t, \lambda_{t+1}) \le F(d_t, \lambda_t) - \alpha_t \|\lambda_{t+1} - \lambda_t\|^2 + \xi_t, \tag{11}$$

with $\alpha_t \ge \underline{\alpha} > 0$ and $\sum_t \xi_t < \infty$, where $\xi_t \ge 0$. On rejection, set $\lambda_{t+1} = \lambda_t$, so the inequality also holds for that step. The update is projected:

$$\lambda_{t+1} = \Pi_{\mathcal{C}_H}\big[\lambda_t + u_t\big]. \tag{12}$$

Sufficient decrease and projection alone do not imply stationarity: choosing $u_t = 0$ forever can satisfy both at a nonstationary point. We therefore state the additional premises explicitly. Assume that $d_t = d_\star$ eventually, $\mathcal{C}_H$ is nonempty and convex as well as compact, $F(d_\star, \cdot)$ is continuously differentiable on a neighborhood of $\mathcal{C}_H$, and $\Pi_{\mathcal{C}_H}$ is Euclidean projection. For a fixed $\gamma > 0$, define the projected-gradient residual $G_\gamma(\lambda) = \gamma^{-1}\{\lambda - \Pi_{\mathcal{C}_H}[\lambda - \gamma \nabla_\lambda F(d_\star, \lambda)]\}$. Assume also that, for some finite K and nonnegative $r_t \to 0$,

$$\|G_\gamma(\lambda_t)\| \le K \|\lambda_{t+1} - \lambda_t\| + r_t \qquad \text{for all sufficiently large } t. \tag{13}$$

This direction-consistency condition is an additional analytical requirement; it is not a property established for LGT's language-generated proposals.

Theorem 3 (Stationarity of limit points)**.** *Under the sufficient-decrease and additional premises above, every accumulation point of (λ_t) is first order stationary for minimizing $F(d_\star, \cdot)$ over $\mathcal{C}_H$. Moreover $\sum_t \|\lambda_{t+1} - \lambda_t\|^2 < \infty$.*

Sketch. After the discrete part has stabilized, summing the sufficient-decrease inequality and using the lower bound on F, the positive lower bound on α_t, and summability of ξ_t bounds the sum of squared step lengths. The finite prefix contributes only a finite amount. Thus step lengths tend to zero, and the direction-consistency condition gives $G_\gamma(\lambda_t) \to 0$. Compactness supplies accumulation points. Continuity of the gradient and of Euclidean projection implies that every such point $\bar{\lambda}$ satisfies $G_\gamma(\bar{\lambda}) = 0$. The projection optimality condition then gives $\langle \nabla_\lambda F(d_\star, \bar{\lambda}), \lambda - \bar{\lambda} \rangle \geq 0$ for every $\lambda \in \mathcal{C}_H$, which is constrained first-order stationarity. This argument does not claim global optimality or convergence of the entire parameter sequence. □

E.4 Budgeted restarts and inner loop stochasticity

If a structural change resets the inner loop for a fixed budget, define $\widetilde{F}$ as the validation loss after that budget under a specified initialization and evaluation protocol. The same acceptance-rule analysis may be applied to $\widetilde{F}$ only when the corresponding fixed-objective, state-retention, and noise premises hold. A restart changes the experiment being compared and its training cost; it does not justify arbitrary weight-transfer assumptions or an uncharged reset. Stochastic training adds separate variability if the objective averages over training randomness. It is covered by the concentration argument only if the required conditional error-difference bound is established for that combined estimator; validation-sampling concentration alone does not establish it.

F Prompt Templates

This section provides the complete prompt templates for each specialized LGT agent. These prompts define the reasoning capabilities and decision-making processes of our multi-agent system.

F.1 Advisor Agent Prompts

The Advisor agent is responsible for proposing configuration changes across 4 dimensions: feature engineering, training strategy, hyperparameters, and model architecture. Each sub-agent uses specialized prompts tailored to its optimization domain.

F.1.1 Model Architecture Agent

System Prompt

You are an advanced machine learning architect. Your task is to design an improved model architecture based on the provided training metrics history and current config.

Instructions:
1. Focus ONLY on changing the model structure (e.g., layer types, layer sizes, activation functions).
2. Do NOT touch any YAML sections other than 'model'. The rest of the original config must stay byte-wise identical.
3. Inside the 'model' section, you may only change numeric values (ints/floats). Do NOT rename keys, add/remove fields, or re-indent.
4. The 'model.name' field MUST be exactly MODEL_NAME (must match the python file name without .py).
5. Respond with TWO markdown fenced blocks in exact order: (1) python code block, (2) yaml code block.
6. Ensure YAML is valid and preserves the original indentation; no trailing commas.

Do NOT output anything else outside the two fenced blocks.

User Prompt Template

Current config (model section only): {yaml_formatted_model_config}
Metrics history: {truncated_metrics_history}
Current model code: {model_code_context}

F.1.2 Feature Engineering Agent

System Prompt

You are a deep-learning model-training expert. Your task is to decide which single augmentation from the list will most likely improve performance in the next epoch based on the provided training/validation metrics and the augmentation currently in use.

Instructions:
1. Unless the provided metrics already show strong improvement with the current setup, you SHOULD choose an augmentation method different from the current one.
2. Reply 'none' only when applying another augmentation would clearly degrade performance or is redundant.

User Prompt Template

Data type: {Data type}
Available methods: {methods}
Epoch: {epoch}
Current augmentation in use: {method}
Train dataset size: {size}
Latest metrics (JSON): {metrics}

Choose exactly one augmentation method (or 'none').

F.1.3 Training Strategy Agent

System Prompt

You are a highly skilled ML practitioner specializing in optimizing training strategies across diverse tasks. Your task is to propose a potentially better pair of loss function and optimizer for the next epoch based on the provided: (a) task type, (b) current loss function & optimizer, and (c) the latest training/validation metrics.

Instructions:
1. Respond with exactly 'no change' (case-insensitive) only if you are absolutely certain the current combination is already optimal and cannot be improved.
2. In most cases you should suggest a new combination.
3. Return a single line in the format: "loss=LOSS FUNCTION, optimizer=OPTIMIZER" (comma-separated, no extras).
4. Do NOT suggest learning-rate changes or any other hyper-parameters.

User Prompt Template

Task type: {Task type}
PREVIOUS AGENT CHANGES: {PREVIOUS AGENT CHANGES}
Current loss function: {loss function}
Current optimizer: {Current optimizer}

Recent metrics (JSON): {metrics}
Model config: {model config}

F.1.4 HYPERPARAMETER OPTIMIZATION AGENT

System Prompt

You are an advanced machine learning architect specializing in hyperparameter optimization. Your task is to evaluate training status and suggest precise hyperparameter changes based on provided parameters and metrics.

Instructions:
1. Focus ONLY on suggesting hyperparameter changes to improve training performance.
2. Analyze current parameters and metrics to determine optimal adjustments.
3. Output changes in ONE of these formats:
- 'no change'
- OR use multiple lines with 'param: value'
- OR use multiple lines with 'param=value'
4. DO NOT use JSON format under any circumstances.
5. If no changes are needed, respond strictly with 'no change'.

User Prompt Template

PREVIOUS AGENT CHANGES: {PREVIOUS AGENT CHANGES}
Current parameters: {Current parameters}
Training metrics: {Training metrics}
Validation: {Validation metrics}

F.2 EVALUATOR AGENT PROMPTS

The Evaluator agent assesses whether optimization attempts improve on the baseline by comparing numerical metrics on validation performance.

F.2.1 COMPARISON PROMPT

Prompt Template

Attempt {attempt_number} of {max_attempts}.

Compare ONLY the numeric metrics (e.g., loss, accuracy, mse, mae) between baseline and optimized models. Give PRIMARY importance to validation metrics (e.g., accuracy, precision, recall, mse, mae). Validation loss can be referenced but should NOT dominate the decision because loss may change substantially when switching loss functions. Training metrics are secondary and should be used only when validation metrics are missing or inconclusive.

Respond in the following strict format (uppercase keywords):

BETTER_VERSION: BASELINE or OPTIMIZED
SUGGESTION: Short general suggestion if BASELINE is better, otherwise N/A

BASELINE_METRICS: {formatted_baseline_metrics}
OPTIMIZED_METRICS: {formatted_optimized_metrics}

F.2.2 ADVISOR IMPROVEMENT PROMPT

Prompt Template

Analyze the following optimization attempts compared to a fixed baseline. Focus on understanding what parameter changes were effective or ineffective relative to the same baseline metrics.

Baseline metrics: {formatted_baseline_metrics}
{optimization_history_analysis}

Based on these attempts, provide:
1. Pattern analysis: Identify patterns in parameter changes and their effects
2. Effective changes: List parameter changes that showed promise
3. Ineffective changes: List parameter changes that were counterproductive

Now, provide concise recommendations for each enabled agent in the following format:

ENGINEERING_AGENT_SUGGESTION:
- Optimization direction: One sentence describing optimization direction
- Reasoning: Brief explanation of effectiveness

ADAPTIVE_AGENT_SUGGESTION:
- Optimization direction: One sentence describing optimization direction
- Reasoning: Brief explanation of effectiveness

HPO_AGENT_SUGGESTION:
- Optimization direction: One sentence describing optimization direction
- Reasoning: Brief explanation of effectiveness

F.3 OPTIMIZER AGENT PROMPTS

The Optimizer distills evaluator feedback into agent-specific guidance for subsequent proposals. The supplied prompt implements guidance synthesis; it does not alone establish that the full system prompt is rewritten after each attempt.

Prompt Template

You are a prompt optimization assistant. Your goal is to distill feedback coming from a model performance judger into concise, actionable notes for different agent types. You MUST NOT rewrite the advisor prompt itself; instead, summarize the feedback in agent-specific bullet points so that each agent can understand what to adjust. Return ONLY the agent-specific suggestions in the requested format, without any other commentary.

ADVISOR_PROMPT (context): {advisor_prompt}
JUDGER_FEEDBACK: {judger_suggestion}

Please distill the JUDGER_FEEDBACK into agent-specific suggestions.
{agent_format_section}

Do NOT modify or reformat the ADVISOR_PROMPT.

AGENT_SUGGESTIONS:
ENGINEERING_AGENT: Suggestion for feature engineering strategy
ADAPTIVE_AGENT: Suggestion for training strategy (loss function, optimizer)
HPO_AGENT: Suggestion for hyperparameter tuning

Table 8: Recorded multi-domain proposals and a positive Evaluator retention decision.

Agent	Suggestion	Reasoning / Parameters
FEATURE	feature_dropout	Applied feature dropout to address overfitting patterns observed in training metrics.
STRATEGY	loss=smooth_l1, optimizer=adam	**Parameters:** loss_function: smooth_l1, optimizer: adam
HPO	learning_rate: 0.0001, dropout: 0.3	**Reasoning:** The current learning rate (2.5e-05) appears too low, as evidenced by poor R^2 scores (-3.77 training, 0.21 validation). Increasing the learning rate to 0.0001 should help convergence. Additionally, increasing dropout from 0.1 to 0.3 will provide better regularization.
EVALUATOR	use_optimized: true	**Decision:** OPTIMIZED version adopted. All metrics showed improvement over baseline, particularly validation performance.

G AGENT OUTPUT EXAMPLES

This section presents representative agent outputs from optimization iterations, including successful and unsuccessful attempts. These examples illustrate the textual-feedback mechanism and the sequence of proposed interventions, observed outcomes, and revised guidance.

G.1 SUCCESSFUL OPTIMIZATION EXAMPLE

Table 8 records proposals in three tuning domains and an Evaluator decision to retain the candidate. The example exposes the stated reasoning about training dynamics and dependencies, but does not isolate the contribution of coordination or establish that each explanation is correct.

G.1.1 ACTUAL AGENT OUTPUTS

Below are the actual textual outputs from each agent during this successful optimization:

Feature Engineering Agent Output

feature_dropout

Feature dropout helps reduce overfitting in tabular data by randomly masking input features, encouraging the model to learn more robust representations, especially useful here given the poor training R^2 and signs of instability.

Training Strategy Agent Output

loss=smooth_l1
optimizer=adam

Smooth L1 loss is mathematically equivalent to Huber loss and is widely supported across deep learning frameworks with stable gradients, making it a robust alternative that maintains the same insensitivity to outliers while potentially offering better numerical stability or implementation efficiency. Switching from AdamW to Adam removes weight decay, which may be beneficial here: given the small tabular dataset (WineQuality) and moderate model capacity ([256,128,64] with dropout=0.2), explicit weight decay might be unnecessarily constraining optimization, especially since the validation R^2 is already positive (0.21) while training R^2 is catastrophically negative (-3.77), indicating possible over-regularization or optimization instability. Removing weight decay (Adam vs. AdamW) could allow the model to fit the training data more effectively, helping to close the train-val performance gap.

Table 9: Failed optimization attempts triggering feedback refinement process.

Attempt	Agent	Suggestion/Parameters
0	Aug Strategy	flip
	HPO	loss=cross_entropy, optimizer=adamw
	Evaluator	learning_rate: 0.0003, weight_class: 1.1
		Validation accuracy degraded
1	Aug Strategy	flip
	HPO	loss=focal_loss, optimizer=sgd
	Evaluator	learning_rate: 0.001, weight_class: 1.2–1.5
		SGD+focal loss: 7–9% accuracy drop
2	Aug Strategy	flip
	HPO	loss=cross_entropy, optimizer=adamw
	Evaluator	learning_rate: 0.0003, weight_class: 1.1
		Minimal improvement; below baseline
3	Aug Strategy	flip
	HPO	loss=focal_loss, optimizer=sgd
	Evaluator	learning_rate: 0.0002, weight_class: 1.1
		Consistent degradation; terminated

HPO Agent Output

learning_rate: 0.0001
dropout: 0.3

The current learning rate (2.5e-05) appears too low, as evidenced by the poor R^2 scores (-3.77 training, 0.21 validation) and the significant gap between training and validation metrics. Increasing the learning rate to 0.0001 should help the model converge more effectively. Additionally, increasing dropout from 0.1 to 0.3 will provide better regularization given the validation metrics are outperforming training metrics, suggesting potential overfitting.

Evaluator Decision

BETTER_VERSION: OPTIMIZED

All metrics showed improvement over baseline. The optimized configuration successfully addressed the convergence issues and overfitting patterns identified in the baseline run. Validation R^2 improved from 0.21 to 0.58, indicating better generalization.

G.2 Failed Optimization with Iterative Refinement

This example records multiple failed optimization attempts and the Evaluator and Optimizer feedback supplied to later iterations. It illustrates how an experiment outcome can revise the next suggestion, while the recorded failures do not establish successful self-improvement.

G.2.1 Example Feedback Evolution

Below are actual textual outputs showing feedback evolution after Attempt 1:

Evaluator Analysis (Attempt 1)

Pattern Analysis:
The simultaneous switch from AdamW to SGD optimizer combined with focal loss implementation created significant negative impacts across all metrics. The class weight adjustments (particularly for classes 0, 3, 5, 7 at 1.2× and others at 1.1×) failed to compensate for the fundamental optimization strategy change. The learning rate increase from 0.0005 to 0.001 likely exacerbated the instability.

Effective Changes: None identified in this attempt

Ineffective Changes: AdamW→SGD, cross-entropy→focal loss, LR 0.0005→0.001, class weight adjustments

FEATURE_AGENT_SUGGESTION:
- Optimization direction: Focus on class-balanced feature engineering without relying solely on loss weighting.
- Reasoning: Class weight adjustments failed, suggesting synthetic data generation may be more effective.

ADAPTIVE_AGENT_SUGGESTION:
- Optimization direction: Revert to AdamW optimizer and maintain cross-entropy loss with adaptive LR schedules.
- Reasoning: SGD+focal loss caused significant drops; baseline strategy was more suitable.

HPO_AGENT_SUGGESTION:
- Optimization direction: Systematically tune learning rate with smaller increments.
- Reasoning: LR increase with other changes led to instability; need conservative exploration.

Optimizer Refinement (Attempt 1)

Feature Agent:
Focus on class-balanced feature engineering instead of solely relying on loss weighting. Since adjusting class weights proved ineffective, generating synthetic data is likely a more viable solution.
Adaptive Agent:
Revert to the AdamW optimizer while maintaining cross-entropy loss with adaptive learning rate schedules. The significant performance drop observed with SGD and focal loss indicates that the original strategy is more appropriate.
HPO Agent:
Systematically tune the learning rate using smaller increments. The previous instability caused by increasing the learning rate alongside other changes necessitates a more conservative exploration approach.

This example shows how failed attempts are converted into subsequent textual guidance. The evolving feedback is an inspectable part of the research record, but several interventions change together and the sequence ends without an improved candidate. It therefore does not identify the causes of failure or demonstrate that the revised guidance improved the proposer.

H Supplementary Experimental Results

This appendix provides supporting details for the generalization, robustness, and efficiency results reported in Section 5.4.

H.1 ViT Feasibility Extension: Setup and Trajectory

Search space. For the MiniViT feasibility extension on CIFAR-100, the architecture-axis agent operates over: patch size $\in \{2, 4, 8\}$, embedding dim $\in \{32, 64, 128, 192\}$, attention heads $\in \{2, 4, 8\}$, transformer layers $\in \{2, 4, 6\}$, MLP ratio $\in \{2.0, 3.0, 4.0\}$, and dropout $\in [0.0, 0.5]$. The feature-engineering, training-strategy, and hyperparameter axes use the same search spaces as the main experiments, without any prompt-level modification to the Advisor, Evaluator, or Optimizer.

Final accepted configuration. The reported final accepted configuration is: patch size 4, embedding dim 128, 4 attention heads, 4 layers, MLP ratio 3.0, dropout 0.2, AdamW with cosine schedule, learning rate 3×10^{-4}, weight decay 0.05, with RandAugment as the dominant feature transformation.

Table 10: Evolution of evaluator feedback across failed attempts.

Attempt	Evaluator Feedback & Optimizer Refinement
0	**Evaluator Analysis:** Initial attempt showed performance degradation across validation metrics. **Agent-Specific Suggestions:** • *FEATURE:* Implement targeted augmentation focusing on minority classes • *Strategy:* Consider reducing regularization and revisiting loss function selection • *HPO:* Conduct focused hyperparameter search on learning rate and regularization
1	**Pattern Analysis:** The simultaneous switch from AdamW to SGD with focal loss created significant negative impacts. Class weight adjustments (1.2–1.5×) failed to compensate. Learning rate increase from 0.0005 to 0.001 likely exacerbated instability. **Effective Changes:** None identified (all changes degraded performance) **Ineffective Changes:** AdamW→SGD, cross-entropy→focal loss, LR increase, class weights **Refined Suggestions:** • *FEATURE:* Focus on class-balanced augmentation without relying on loss weighting • *Strategy:* Revert to AdamW with cross-entropy; explore adaptive LR schedules • *HPO:* Tune learning rate with smaller increments; targeted class weight approach
2 & 3	**Cumulative Analysis:** After four attempts, patterns emerged: (1) Class weights (1.1–1.5×) uniformly degraded performance; (2) AdamW+cross-entropy showed better stability than SGD+focal loss; (3) Conservative LR reduction (0.0003) showed least degradation; (4) Validation loss decreased with SGD+focal but metrics worsened. **Final Recommendations:** • *FEATURE:* Address class imbalance through augmentation rather than loss manipulation • *Strategy:* Maintain AdamW+cross-entropy; explore adaptive LR schedules • *HPO:* Use Bayesian optimization for systematic tuning with smaller increments

We note that LGT independently arrived at scale-up choices (deeper, wider, more heads) consistent with prior ViT scaling studies, without an explicitly ViT-specific prompt; pretrained LLM knowledge may nevertheless contain relevant architecture priors.

H.2 LLM Backbone Robustness: Per-Backbone Details

Inference setup. Llama-3-8B-Instruct and Qwen3-8B were served locally via vLLM with temperature 0.2 and top-p 0.95, matching the DeepSeek-V3.2 generation settings used in the main experiments. GPT-oss-120B was accessed through its public endpoint under the same temperature.

Token usage. The record omits mean output tokens per call for the Advisor, Evaluator, and Prompt Optimizer. The stated relative output length for Llama-3-8B is approximately $0.6\times$ that of DeepSeek-V3.2, but request-level counts are unavailable for verification. This unverified description cannot establish causes such as context handling or support total-token cost comparisons.

Failure modes by backbone. The experimental narrative reports malformed YAML for Llama-3-8B in $< 5\%$ of calls and structurally valid outputs for Qwen3-8B and GPT-oss-120B in the described runs. These rates are unverified because the request counts and parser logs are not available. Rejecting malformed edits prevents their direct application, but does not establish that the search is unaffected or that other accepted proposals cannot degrade generalization.

H.3 Computational Cost: Detailed Breakdown

Per-agent latency. Table 4(c) reports aggregate LLM inference time for MiniViT/CIFAR-100, but the separate Advisor, Evaluator, and Prompt-Optimizer latencies were not specified. Per-role measurements should include API round-trip time and retries as well as generation latency; the reported aggregate cannot recover that breakdown.

Scaling. If the number and cost of language calls per epoch remain fixed while training cost grows, their relative contribution decreases. As an illustrative calculation, a hypothetical 200 s training epoch with the same ≈ 5 s LLM component would have an approximately 2.5% training-relative overhead. This is a conditional extrapolation, not a measured total-search runtime; validation, retries, and state handling must also be counted.

I LIMITATIONS

What we have not compared against. The closest systems to LGT are LLM-driven, and we do not yet benchmark them head-to-head. Three comparisons are missing and would each be informative in a different way: a *single-agent* ablation in which one LLM performs proposal, evaluation, and guidance synthesis, which is the cleanest test of whether the three-role decomposition earns its complexity; TextGrad (Yuksekgonul et al., 2025) and GEPA (Agrawal et al., 2026) restricted to the subset of our search space they can express; and autoresearch (Karpathy, 2026) given an equal wall-clock budget, which would measure how much the epoch-level granularity actually buys. We report the multi-agent design as an architectural choice. The per-axis ablation in Table 5 evaluates editable domains and does not isolate the necessity of the agent roles.

Asymmetric search spaces. LGT edits four domains, while the baseline configurations described in this study focus on narrower interventions, including architecture for DARTS and selected hyperparameters for Optuna and BOHB. Some observed differences may reflect the selected action spaces rather than proposal quality, and the current experiments do not separate these effects. The software frameworks themselves can support broader spaces than a particular baseline configuration.

Scale. Our benchmarks are moderate in size and the backbones compact. Whether epoch-level editing remains stable when a single epoch costs hours, and whether the Advisor's priors transfer to modern architectures at scale, is untested; MLE-bench is the natural next target.

Unbounded history. The mathematical history $\mathcal{H}_t$ records the accumulated interaction sequence. The source description states that agent context is truncated to recent entries when it exceeds the available window; the retained length and truncation policy are not specified. A principled memory mechanism is future work.

Constrained pipelines. LGT assumes the configuration is genuinely free to change. In settings built around fixed pretrained families and standardized training stacks, most of our four axes are frozen by convention, and the available intervention space can narrow to hyperparameter tuning. The value of the broader interface must then be measured against an appropriate narrower control.

Fluent rationales are not verified rationales. The Appendix G examples include a case where the Strategy Agent misstated the model's own dropout rate while giving otherwise sensible advice. A numerical gate can reject an unfavorable measured outcome, but it does not validate the rationale or guarantee generalization. Readers should distinguish recorded explanations from verified explanations when using these traces.

J REPORTING AND IMPLEMENTATION CHECKS

The supplied numerical record supports the comparisons as displayed, but several details remain unresolved. These issues limit conclusions from the current configuration experiments.

Metric aggregation. The MNIST AUC entry 99.99 ± 0.10 needs verification against the per-run values and definition of the dispersion. If both values describe a mean and sample standard deviation over ten runs on a $[0, 100]$ scale, they are incompatible even allowing the displayed rounding: a mean of at least 99.985 gives a maximum sample standard deviation of approximately 0.0474. The supplied table is retained pending verification rather than replacing it with an inferred number. Tiny Shakespeare also requires explicit definitions of sequence splitting, target offsets, decoding or teacher forcing, and BLEU/perplexity aggregation.

Evaluation access. The split indices, fitted preprocessing partitions, model-selection record, and generation-window construction are needed to resolve the main-text/appendix split descriptions and the test-loss trajectories in Figure 2. Repeated adaptive reuse of a small validation set can affect selection even if the test partition is inaccessible to the agents.

State and prompts. The formal loop specifies an incumbent and candidate, but the supplied feedback prompt refers to a fixed baseline. Their implemented relationship, the numerical margin, candidate horizon, malformed-output handling, and restoration of weights, optimizer, scheduler, randomness, and preprocessing state require code-level verification. Architecture changes in Figure 3 additionally require a documented transfer or restart rule.

Mechanism and resource comparisons. Equal-budget LLM-HPO/coding-agent baselines, single-agent and role-removal controls, and epoch-versus-longer-horizon decisions are not present in the reported tables. Proposal counts, parse validity, acceptance rates, and all attempted-candidate costs are needed to compare experimental efficiency. Literature comparisons identify relevant systems; they do not substitute for running those controls.

K Detailed Configuration and Agent Mappings

This section retains component definitions and agent mappings underlying the main-text overview. The configuration objective, experiment-history update, and retention rule remain in the main text.

K.1 Configuration Components

K.1.1 Model Architecture Optimization

The architecture space $\mathcal{C}_A = \{\theta_a | \theta_a \in \mathcal{F}_a\}$ encompasses all neural network configurations, including layer types, connectivity patterns, and structural parameters, where $\mathcal{F}_a$ defines the space of architectures satisfying computational and structural constraints. The architecture optimization function : $\mathcal{M}_t \times \mathcal{C}_A \to \mathcal{C}_A$ analyzes current training metrics $\mathcal{M}_t$ to generate improved architectures. The optimal architecture selection follows:

$$\theta_a^* = \arg\min_{\theta_a \in \mathcal{C}_A} \mathbb{E}[\mathcal{L}(f_{\theta_a}(x), y)] \tag{14}$$

where f_{θ_a} denotes the model with architecture θ_a. This optimization occurs independently before training cycles of other configurations (Appendix F.1.1).

K.1.2 Feature Engineering

The feature engineering space $\mathcal{C}_F = \{\tau : \mathcal{X} \to \mathcal{X}' | \tau \in \mathcal{T}\}$ consists of all possible data transformation functions, where $\mathcal{T}$ represents available methods including geometric transformations, noise injection, and domain-specific operations. The augmentation selection function : $\mathcal{M}_t \times \mathcal{C}_F \to \mathcal{C}_F$ adapts strategies based on current training metrics $\mathcal{M}_t$. The augmentation selection follows:

$$\tau^* = \arg\min_{\tau \in \mathcal{C}_F} \mathbb{E}[\mathcal{L}(f(\tau(x)), y)] \tag{15}$$

where f is the current model. This enables dynamic response to fitting patterns through appropriate augmentation method selection (Appendix F.1.2).

K.1.3 Training Strategy Optimization

The training strategy space $\mathcal{C}_T = \mathcal{L}_{\text{loss}} \times \mathcal{O}_{\text{opt}} \times \mathcal{S}_{\text{sched}}$ encompasses loss functions, optimizers, and learning rate schedulers. The strategy selection function : $\mathcal{M}_t \times \mathcal{C}_T \to \mathcal{C}_T$ adapts training configurations based on current training metrics $\mathcal{M}_t$. The strategy selection follows:

$$(l^*, o^*, s^*) = \arg\min_{(l,o,s) \in \mathcal{C}_T} \mathbb{E}[\mathcal{L}(f_{l,o,s}(x), y)] \tag{16}$$

where $f_{l,o,s}$ denotes the predictor trained using loss l, optimizer o, and scheduler s (Appendix F.1.3). We compare these training interventions' outcomes using the fixed task-level evaluation criterion.

K.1.4 Hyperparameter Optimization

The hyperparameter space $\mathcal{C}_H = \{\lambda | \lambda_i \in [\lambda_i^{\min}, \lambda_i^{\max}]\}$ includes learning rates, weight decay, and other parameters. The hyperparameter update function : $\mathcal{M}_t \times \mathcal{C}_H \to \mathcal{C}_H$ modifies parameters based on training metrics $\mathcal{M}_t$. The epoch-level update:

$$\lambda_{t+1} = \Pi_{\mathcal{C}_H}[\lambda_t + \Delta\lambda_t] \tag{17}$$

where $\Delta\lambda_t$ is proposed with the aim of reducing the expected loss $\mathbb{E}[\mathcal{L}(f_\lambda(x), y)]$, and $\Pi_{\mathcal{C}_H}$ ensures constraint satisfaction (Appendix F.1.4).

K.2 Agent Interfaces and Prompt Mappings

Our framework employs a multi-agent system $\mathcal{A} = \{\mathbb{A}_{\text{adv}}, \mathbb{A}_{\text{eval}}, \mathbb{A}_{\text{opt}}\}$ consisting of three agents: an Advisor ($\mathbb{A}_{\text{adv}}$) that generates configuration modifications, an Evaluator ($\mathbb{A}_{\text{eval}}$) that assesses optimization effectiveness, and a Prompt Optimizer ($\mathbb{A}_{\text{opt}}$) that refines agent decision-making capabilities.

The three agents have distinct codomains, and we keep them separate rather than folding them into one transition function. Writing $\mathcal{M}$ for the space of training metrics, $\mathcal{H}$ for optimization histories, $\mathcal{P}$ for prompts and Σ^* for natural-language strings, the agents are $\mathbb{A}_{\text{adv}} : \mathcal{C} \times \mathcal{M} \times \mathcal{H} \times \mathcal{P} \to \Sigma^*$, $\mathbb{A}_{\text{eval}} : (\mathcal{C} \times \mathcal{M})^2 \times \mathcal{H} \to \Sigma^*$ and $\mathbb{A}_{\text{opt}} : \mathcal{M} \times \mathcal{C} \times \mathcal{P} \times \Sigma^* \to \Sigma^*$. Each returns free text; dedicated parsers $\mathcal{P}^{\text{cfg}}_{\text{parse}}, \mathcal{P}^{\text{dec}}_{\text{parse}}, \mathcal{P}^{\text{sgg}}_{\text{parse}}, \mathcal{P}^{\text{prm}}_{\text{parse}}$ map that text into a candidate configuration, a binary accept/reject decision, a suggestion, and a revised prompt respectively. Only $\mathcal{P}^{\text{cfg}}_{\text{parse}} \circ \mathbb{A}_{\text{adv}}$ ever produces an element of $\mathcal{C}$, so the configuration space is modified exclusively along that path. This creates a closed-loop system where agents communicate through structured pipelines, linking proposed edits, measured outcomes, and textual feedback in a common experiment record. We provide the prompt templates for each agent in Appendix F. In Appendix G, we present the outputs from agent interactions, including cases of successes and failures.

K.2.1 Advisor (Output): config generator

The Advisor ($\mathbb{A}_{\text{adv}}$) analyzes the current training state and generates new configuration recommendations. The advisor function $\mathbb{A}_{\text{adv}} : (c_t, \mathcal{M}_t, \mathcal{C}) \to \Delta c_t^{\text{NL}}$ maps the current configuration, training metrics, and configuration space to natural language configuration recommendations Δc_t^{NL}:

$$\Delta c_t^{\text{NL}} = \mathcal{G}_{\text{LLM}}(\text{prompt}_{\text{adv}}(c_t, \mathcal{M}_t, \mathcal{C})) \tag{18}$$

The candidate configuration $\tilde{c}_{t+1}$ is obtained through parsing:

$$\tilde{c}_{t+1} = \mathcal{P}^{\text{cfg}}_{\text{parse}}(\Delta c_t^{\text{NL}}) \tag{19}$$

where $\mathcal{P}^{\text{cfg}}_{\text{parse}}$ extracts and instantiates modified configuration from the natural language outputs.

K.2.2 Evaluator (Loss): performance assessor

The Evaluator ($\mathbb{A}_{\text{eval}}$) judges the proposed change with both a binary acceptance signal and a suggestion for improvement. The mapping $\mathbb{A}_{\text{eval}} : (c_t,\ \mathcal{M}_t,\ c_{t-1},\ \mathcal{M}_{t-1}) \to \mathcal{I}^{\text{NL}}_{\text{eval}}$ is given by

$$\mathcal{I}^{\text{NL}}_{\text{eval}} = \mathcal{G}_{\text{LLM}}(\text{prompt}_{\text{eval}}(c_t, \mathcal{M}_t,\ c_{t-1}, \mathcal{M}_{t-1})) \tag{20}$$

Parsing yields a decision and a suggestion:

$$\text{success}_t = \mathcal{P}^{\text{dec}}_{\text{parse}}\left(\mathcal{I}^{\text{NL}}_{\text{eval}}\right) \in \{0, 1\}, \tag{21}$$

$$s_t^{\text{NL}} = \mathcal{P}^{\text{sgg}}_{\text{parse}}\left(\mathcal{I}^{\text{NL}}_{\text{eval}}\right), \tag{22}$$

where s_t^{NL} is a natural-language suggestion that may reference specific configuration fields, search constraints, or prompt refinements (Appendix F.2).

K.2.3 Optimizer (Gradient): prompt updater

The Prompt Optimizer ($\mathbb{A}_{\text{opt}}$) produces guidance using the current performance, Advisor context, and Evaluator suggestion. Here $\text{prompt}^t_{\text{adv}}$ denotes the effective Advisor input: a fixed instruction template combined with variable guidance and experiment context. The update concerns this context, consistent with the supplied Optimizer template, which explicitly forbids rewriting the instruction template. The mapping $\mathbb{A}_{\text{opt}} : (\mathcal{H}_t,\ \mathcal{M}_t,\ c_t,\ \text{prompt}^t_{\text{adv}},\ s_t^{\text{NL}}) \to \Delta p_t^{\text{NL}}$:

$$\Delta p_t^{\text{NL}} = \mathcal{G}_{\text{LLM}}\Big(\text{prompt}_{\text{opt}}(\mathcal{M}_t, c_t,\ \text{prompt}^t_{\text{adv}},\ s_t^{\text{NL}})\Big) \tag{23}$$

The next effective Advisor input is composed through:

$$\text{prompt}^{t+1}_{\text{adv}} = \mathcal{P}^{\text{prm}}_{\text{parse}}(\Delta p_t^{\text{NL}}) \tag{24}$$

where $\mathcal{P}^{\text{prm}}_{\text{parse}}$ extracts the role-specific guidance and composes it with the fixed template and experiment context (Appendix F.3). This notation describes guidance synthesis; whether an implementation performs any additional prompt rewriting requires separate verification.

L Extended Evaluation Protocol and Analysis

This section retains the detailed protocol descriptions and result interpretations summarized in Sections 4 and 5. No additional experimental runs are introduced.

L.1 Split protocol and evaluation access

Splits and what the agents may see: For each dataset, the intended split is 80% training and 20% testing, with 10% of training held out for validation: 72/8/20. The Advisor, Evaluator, and retention rule should access training and validation records only; test outcomes should be reserved for final reporting. The descriptions need reconciliation: Appendix A describes an 80/20 split without this validation subdivision, and Figure 2 includes test-loss trajectories. Without split indices and execution records, independence of the final test outcomes from adaptive selection is not established. For Tiny Shakespeare, the order of raw-text splitting and overlapping-window construction must be specified to exclude shared context across partitions.

L.2 Search spaces

Search Spaces: Our optimization framework explores multiple dimensions simultaneously. The architecture search examines networks with 2–5 hidden layers with 32–512 neurons. Feature engineering encompasses geometric transformations (rotation, scaling), data augmentation (noise injection), and task-specific preprocessing strategies. We evaluate different optimization algorithms like Adam, SGD, RMSprop, AdamW, Adagrad, and AdaFactor. Additional hyperparameters include class weights ranging from 0.1 to 10.0, and learning rates spanning from 10^{-4} to 10^{-1}.

L.3 Run budgets and metrics

Evaluation: The experimental record describes a cap of up to 50 configurations and a 10-epoch training horizon for each method. It reports ten runs with random seeds 42–51; repeated runs characterize observed variation but do not alone establish statistical significance or equal computational budgets. Performance metrics are task-specific: classification uses accuracy, F1-score, and AUC; regression uses MAE, MSE, and R^2; generation uses perplexity, BLEU, and token-level accuracy. Candidate training, rejected attempts, architecture restarts, validation, and LLM calls must all enter a common cost account before comparing overall research efficiency. The tabulated results retain the supplied values; Appendix J identifies the artifacts needed to verify their interpretation.

L.4 Overall comparison: extended interpretation

Tables 2 and 3 compare LGT with the reported baselines across all seven datasets. The comparison set covers untuned training and classical optimization methods; it does not include a matched LLM-driven tuning or coding-agent baseline. The entries use the source record's displayed precision and are described there as means and standard deviations over **10** runs. LGT has the best displayed mean in each reported metric column, which is evidence about these task outcomes under the reported settings. It is not a statistical significance result or a demonstration that its search space, language mechanism, and resource budget are equivalent to those of the comparators.

Across classification, regression, and generation, the tables report favorable mean outcomes for LGT, with task-dependent margins. On CIFAR-10, accuracy increases from 69.01 for No Tuning to 79.64 for LGT, a gain of 10.63 percentage points. Against KerasTuner, the strongest displayed accuracy baseline on that dataset, the increase is from 77.69 to 79.64, or 1.95 percentage points. The regression and generation tables likewise report better means for LGT on their metrics, subject to the evaluation-protocol and aggregation checks noted above. In addition to final scores, LGT exposes textual records of *why* an edit was proposed, *what* was changed, and *which* outcome was observed. These fields make the experiment trajectory inspectable (Appendix G.1); an agent's stated rationale remains a hypothesis about the outcome rather than a verified causal explanation.

Viewed as automated research, these results examine whether a language-guided configuration loop can produce useful candidates within a fixed task. The shared record connects architectural, training, feature, and hyperparameter decisions to numerical outcomes and to later proposals. This

complements numerical search with an inspectable account of the experimental process. However, the results do not isolate semantic reasoning from differences in action space, proposal budget, or pretrained model knowledge. A common-budget comparison in the same edit space is needed to attribute an advantage specifically to language guidance.

L.5 Compact-transformer feasibility

To examine configurations beyond the CNN/MLP/LSTM backbones used in Section 5.1, we apply LGT to a Mini Vision Transformer (MiniViT; patch size 4, embedding dimension 64, 2 heads, 2 layers, MLP ratio 4.0, dropout 0.1) on CIFAR-100. Table 4(a) reports accuracy increasing from 27.22% to 41.03% (+13.81 points) and F1 from 21.32 to 39.06 (+17.74 points). Architecture-only tuning reaches 38.07% accuracy, accounting for 10.85 points of the reported gain; Full LGT is 2.96 points higher. This is a feasibility result for configuration experimentation on the specified compact transformer. Without uncertainty estimates for this comparison or matched role controls, it does not establish that cross-domain coordination caused the remaining margin or that the result generalizes to large-scale transformer research.

L.6 Backbone comparison

Robustness across LLM backbones. We evaluate Llama-3-8B-Instruct, Qwen3-8B, GPT-oss-120B, and the default DeepSeek-V3.2 on CIFAR-10. Table 4(b) reports gains over the No Tuning baseline of at least 7.51 percentage points, and a 3.12-point range from Llama-3-8B-Instruct to DeepSeek-V3.2. The direction of the reported improvement therefore persists across the tested backbones, an encouraging observation for using the configuration loop with different proposal models. This comparison does not isolate the effect of the three-agent design or gate from model quality, sampling, and other shared protocol choices; it does not rank these architectures by one notion of model size.

L.7 Runtime accounting

Computational overhead. The textual loop adds LLM inference and parsing costs to training. On MiniViT/CIFAR-100, Table 4(c) reports approximately 85.0 s of training, 4.5 s of LLM inference, and 0.5 s of parsing and updates, totaling 90.0 s and a ratio of approximately $1.06\times$ to the reported training component. These are per-epoch component timings, not an account of the full configuration search. Fewer epochs to a target in selected trajectories would imply lower total cost only after accounting for all candidate epochs, rejected attempts, restarts, validation, and language calls. Appendix H retains the cost discussion and identifies the missing per-agent latency and token counts. Consequently, the present evidence supports a local overhead description but does not establish overall cost parity or savings against another research agent.

L.8 Single-domain ablations: extended analysis

Table 5 presents an ablation study evaluating each tuning dimension in the LGT framework. We evaluate each of the four core optimization components, Model Architecture, Feature Engineering, Training Strategy, and Hyperparameters, in isolation to understand their relative contributions on classification (MNIST) and regression (Housing) tasks.

The strongest single-domain setting varies across the two tasks. For MNIST, Training Strategy reaches AUC 99.96, whereas Feature Engineering alone gives 98.18 against a 98.37 untuned baseline; the latter is an adverse result, whose cause cannot be determined from the aggregate table. For Housing, Hyperparameter tuning reaches MAE 0.416 and is the strongest reported single-domain variant. These observations identify where individual edit domains were useful under the reported settings, without showing that the framework learned how to allocate effort optimally. In a research workflow, the result motivates treating configuration interventions as task-dependent hypotheses whose consequences must be measured, rather than assuming that adding another editable domain always improves an experiment.

The full system has the best displayed mean on both tasks, but its margin over the strongest single-domain variant is small on MNIST: AUC 99.99 versus 99.96, with reported standard deviations of 0.10 and 0.12. On Housing, the full system reports MAE 0.402 versus 0.416 for Hyperparameter-only

tuning, with standard deviations of 0.021 and 0.025. These comparisons do not identify coordination as the cause of the difference, and the MNIST AUC aggregation itself requires verification (Appendix J). The experiment ablates editable domains, not Advisor, Evaluator, or Optimizer roles. A direct test of the latter would keep the action space, backbone, retention rule, and budget fixed while varying the role decomposition; the present tables should not be interpreted as that test.

AI Assistance Disclosure

Generative AI was used in revising this manuscript for literature identification and reference checking, organization and wording of the automated-research framing, analysis of reviewer feedback, mathematical clarification, and LaTeX preparation. No new experimental runs were performed as part of this revision. The paper distinguishes supplied measurements from unresolved implementation and reporting details; AI-assisted revision does not independently validate the experiments. The authors remain responsible for verifying and approving the final manuscript and for completing any additional disclosure required by the submission process.