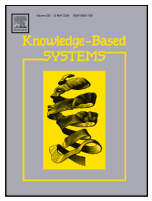
Knowledge-Based Systems 328 (2025) 114210



Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys





Searching for the best student architecture in a knowledge distillation framework

Steve Bakos *, Heidar Davoudi

Department of Computer Science, Ontario Tech University, Oshawa, Canada

ARTICLE INFO

Keywords:
Reinforcement learning
Student models
Neural architecture search
Automl
Deep learning
Knowledge distillation
Hyperparameter optimization

ABSTRACT

Knowledge distillation aims to find a smaller model (i.e., student) that can perform at the level of a larger model (i.e., teacher). While the student model is highly beneficial in resource-constrained environments, finding the optimal student remains challenging due to the extensive search required through potential architectures and hyperparameters. To address this, we introduce a novel framework that integrates a *caching mechanism* and *proximity analysis* into Reinforcement Learning (RL) for Neural Architecture Search (NAS) and Hyperparameter Optimization (HPO) in knowledge distillation. This approach improves computational efficiency by avoiding redundant evaluations and estimating the performance of similar configurations. Our results, benchmarked against foundational and modern evolutionary search methods, demonstrate that the proposed framework can reduce full training evaluations by over 75 % relative to a standard RL search, offering a robust advantage in computationally or operationally constrained environments.

1. Introduction

Knowledge distillation (KD) [1] offers a promising solution for training smaller *student* models to mimic larger *teacher* models, effectively reducing model complexity while maintaining performance. This approach is particularly useful in resource-constrained environments, where deploying large, complex models may not be feasible. However, the manual design of an optimal student model, encompassing both its architecture and training hyperparameters, remains a significant bottleneck due to the vast and complex search space.

The core challenge lies in efficiently navigating this combined search space. Traditional approaches often decouple Neural Architecture Search (NAS) [2,3] from Hyperparameter Optimization (HPO) [4,5], leading to a sequential optimization process. While NAS aims to automate the design of network topologies and HPO seeks optimal training settings (e.g., learning rate, batch size), this sequential treatment can be suboptimal. The ideal hyperparameters are often architecture-dependent, and conversely, the best architecture might only emerge under specific hyperparameter configurations. This interdependence, as highlighted by benchmarks such as NAS-HPO-Bench-II [6] and JAHS-Bench-201 [7], alongside foundational joint search methodologies [8,9], underscores the necessity of a **joint optimization** approach. Indeed, NAS-HPO-Bench-II provides empirical evidence that joint optimization methods can significantly outperform sequential strategies [6], and it has been argued that optimal hyperparameter settings can be even more critical than the architectural choices themselves when considered in isolation [7]. Such findings strongly advocate for considering architectural choices and hyperparameters simultaneously.

Furthermore, the landscape of state-of-the-art (SOTA) NAS includes a diverse array of highly efficient techniques, ranging from sophisticated evolutionary algorithms and reinforcement learning to one-shot approaches and gradient-based methods. Prominent gradient-based methods, such as Differentiable Architecture Search (DARTS) [10] and its successors (e.g., Xu et al. [11], Chen et al. [12], Hao and Pei [13]), have significantly advanced search speed by optimizing a continuous relaxation of the architectural space, often within a super-network framework. While these and many other cutting-edge NAS techniques achieve remarkable efficiency (often measured in GPU hours), they frequently exhibit characteristics that differentiate them from the problem addressed in our work:

1. **Sequential vs. Joint Optimization:** Many advanced NAS methodologies, including numerous differentiable and one-shot approaches, primarily focus on identifying the optimal architecture first. HPO is then typically performed as a subsequent, separate step, rather than being integrated into a joint search.
2. **Fixed Evaluation Budget:** Our work operates under a fixed budget of discrete model evaluations (e.g., a limited number of student models fully trained and assessed). This paradigm directly reflects scenarios where the total number of full, independent model

* Corresponding author.
E-mail addresses: steven.bakos@ontariotechu.net (S. Bakos), heidar.davoudi@ontariotechu.ca (H. Davoudi).

https://doi.org/10.1016/j.knosys.2025.114210
Received 7 May 2025; Received in revised form 19 July 2025; Accepted 29 July 2025
Available online 5 August 2025
0950-7051/© 2025 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

training instances-each incurring a substantial, indivisible computational cost-is the primary resource constraint, a common situation in academic research or during iterative development with fixed experimental budgets.
3. **Clear Sample Efficiency Metric:** Such an evaluation budget provides a clear metric for sample efficiency, quantifying how effectively a search strategy utilizes each costly opportunity to train and evaluate a complete model. This is particularly pertinent in KD where the student training cycle is the dominant expense [2,3], and contrasts with many SOTA NAS methods optimized for cumulative GPU hours or super-network training rather than the count of full, individual model runs.

These distinctions highlight a specific niche for methods, like ours, that are designed for the joint optimization of architectures and hyperparameters under a strict evaluation budget, particularly relevant in contexts like KD where individual student evaluations are a key cost factor.

RL has shown promise in navigating complex search spaces for NAS and HPO. However, RL agents face the exploration-exploitation trade-off, which can lead to computational inefficiency. Agents might expend significant resources evaluating unpromising configurations or re-evaluating similar ones, a challenge amplified by the need to train each candidate configuration.

To address these specific challenges-the need for joint optimization, the constraint of a fixed evaluation budget, and the inherent inefficiencies of RL-based search-we propose a novel RL-based framework. Our approach integrates a *caching mechanism* and *proximity analysis* to enhance the efficiency of joint NAS and HPO, specifically tailored for KD. The cache prevents redundant evaluations of identical configurations, while proximity analysis estimates the performance of similar, unevaluated configurations, thereby reducing unnecessary training. This allows the RL agent to focus resources on the most promising and diverse areas of the joint search space.

Our framework operates in two phases: a *planning phase*, where the RL agent designs a student model (architecture and initial hyperparameters) and decides on evaluation, and an *execution phase*, where caching and proximity mechanisms are employed if the model is trained.

The key contributions of our work are:

1. We design a novel end-to-end RL-based framework that performs **joint NAS and HPO** for KD, specifically addressing the challenge of a fixed evaluation budget.
2. We introduce an integrated **caching mechanism and proximity analysis** strategy within the RL loop to significantly improve search efficiency by avoiding redundant evaluations and estimating performance for similar configurations.
3. Our approach is tailored for **discrete search spaces** and scenarios where full model evaluations are costly, contrasting with continuous relaxation methods like DARTS and many other SOTA NAS paradigms.
4. Comprehensive empirical evaluation demonstrates significant efficiency improvements (e.g., reduced number of student model trainings) while maintaining high performance in comparison to standard search techniques.
5. We conduct a comprehensive **training stability analysis** to provide a deeper evaluation of the search algorithms, assessing not only their final performance but also their consistency and reliability throughout the search process.

Our approach alleviates the computational burden of the exploration-exploitation trade-off in RL for joint NAS-HPO, ensuring efficient use of a limited evaluation budget.

2. Related work

KD [1] is a pivotal technique for compressing large *teacher* models into smaller *student* models while preserving performance. This is crucial for deployment in resource-constrained environments. Enhancements to KD include FitNets [14] using intermediate supervision and attention transfer [15] matching feature maps. A comprehensive survey by Gou et al. [16] details various KD strategies, emphasizing that significantly compressed students can still achieve high accuracy [17,18].

The automation of designing these student models involves NAS and HPO. We review these areas, their intersection with KD, and the role of RL, particularly focusing on the challenges our work addresses.

2.1. Neural architecture search (NAS)

NAS aims to automate neural network design. Early methods, often RL-based [2] or evolutionary [19], achieved impressive results but at substantial computational cost. To improve efficiency, techniques like parameter sharing (Efficient Neural Architecture Search (ENAS) [20]), proxy tasks, and direct search on target hardware (ProxylessNAS [21]) were developed.

Gradient-based NAS methods, such as DARTS [10], significantly reduced search costs by relaxing the discrete architectural search space into a continuous one and using gradient descent. However, DARTS and its variants (e.g., Xu et al. [11], Chen et al. [12], Hao and Pei [13]) primarily focus on discovering the architecture, with HPO typically following sequentially. Moreover, their operational paradigm based on super-network training and optimization over GPU hours differs from scenarios constrained by a fixed number of full model evaluations. Recent SOTA NAS methods continue to evolve, with some exploring evolutionary strategies guided by gradients [22] or focusing on specific applications like Generative Adversarial Networks (GANs) [23]. While these methods advance NAS efficiency and performance, many retain a sequential HPO step or are not directly tailored for joint optimization under strict evaluation budgets.

2.2. Hyperparameter optimization (HPO)

HPO seeks optimal training configurations. Random search [4] and Bayesian optimization (BO) [24] are foundational. For efficiency in deep learning, multi-fidelity methods like Hyperband [25] and BOHB [5] (which combines BO and Hyperband) dynamically allocate resources, training promising configurations for more epochs. These methods are powerful for HPO alone but do not inherently search for architectures.

2.3. Joint NAS and HPO (NAS-HPO)

The understanding that architectures and hyperparameters are profoundly interdependent [8,26] has spurred significant research into **joint NAS-HPO**. This approach acknowledges that the optimal HPs can vary significantly across different architectures, and vice-versa [9]. Treating NAS as an HPO problem where architectural choices are categorical hyperparameters allows methods like BOHB to search the joint space effectively [5]. The development of specialized benchmarks has been crucial in validating the benefits of joint optimization. For instance, NAS-HPO-Bench-II [6] was created to study these joint spaces and provides empirical evidence demonstrating that joint optimization methods can substantially outperform sequential optimization strategies, confirming the strong dependency between architecture and training HPs (e.g., learning rate, batch size). Similarly, JAHS-Bench-201 [7] was introduced as the first collection of surrogate benchmarks for joint architecture and hyperparameter search, further enabling research into this combined problem and highlighting that well-chosen hyperparameters can be more critical than the architecture itself if optimized in isolation. Further research continues to explore diverse strategies, including evolutionary and BO-based methods, for navigating this complex joint space, often while considering multiple objectives such as accuracy and model size [9]. Wu et al. [27] also proposed a joint search

for correlated time series forecasting, underscoring the limitations of sequential approaches. These collective works strongly advocate for the benefits of joint optimization. However, many existing joint methods rely on methodologies (like extensive surrogate modeling or multi-fidelity evaluations that assume flexible budget allocation) that differ from our RL-based approach focused on a fixed number of explicit model evaluations.

2.4. NAS and HPO in knowledge distillation

Combining NAS and/or HPO with KD to find optimal student models is a more specialized area. Some works incorporate KD into the NAS loop [28] or design student architectures specifically for KD. For instance, Distillation without Training (DisWOT) [29] offers a training-free method to find student architectures by measuring similarity to the teacher at initialization. Automatic Multi-step Distillation (AMD) [30] focuses on automatically determining intermediate teacher-assistant sizes. While these methods address KD-specific NAS, they may not perform joint HPO or use the RL paradigm with explicit evaluation budgets and efficiency mechanisms like caching and proximity analysis in the same manner as our framework. Our approach is complementary, as its efficiency mechanisms can potentially be combined with various NAS or training-free techniques.

2.5. RL for joint NAS/HPO under evaluation constraints

RL has been a popular choice for NAS [2,20], framing architecture generation as a sequential decision problem. However, standard RL-NAS can be sample-inefficient, re-evaluating similar configurations. Our work specifically targets this inefficiency within a **joint NAS-HPO context for KD, constrained by a fixed number of student model evaluations**. While some RL approaches for HPO exist [31], they often do not integrate NAS. The novelty of our framework lies in the combination of RL for joint search with tailored efficiency mechanisms (caching and proximity analysis) designed for this specific evaluation paradigm, contrasting with SOTA NAS methods like DARTS that optimize for different constraints (e.g., GPU hours) and typically separate NAS from HPO.

3. Proposed method

3.1. Framework overview

Given the *teacher* model in the KD framework, our primary objective is to identify the optimal *student* model-one that performs at the level of the teacher while being significantly smaller. Fig. 1 illustrates the proposed Knowledge Discovery Policy Learning (KD Policy Learning) framework for finding the best student model in KD. This framework combines NAS, HPO, and KD into a unified RL framework, designed to efficiently explore the extensive search space of student architectures and key hyperparameters in the training stage.

Definition 3.1. (**Configuration**): A configuration c comprises (i) an *architecture* A (Definition 3.2), (ii) two scalar hyper-parameters-learning rate (LR) and number of epochs (EP), and (iii) the numeric feature vector $\mathbf{f}(c) = [\mathrm{LR}, \mathrm{EP}, \mathbf{a}]^\top$ used in Section 3.5. Here $\mathbf{a} \in \mathbb{Z}^{L_{\max}}$ is a fixed-length encoding of the architecture (see below).

Definition 3.2. (**Architecture**): An architecture A is an ordered sequence of up to $L_{\max}$ layer specifications

$$A = \langle (\ell_1, w_1), (\ell_2, w_2), \ldots, (\ell_L, w_L) \rangle, \quad L \le L_{\max},$$

where each layer is defined by $\ell_i \in \{\text{Dense}, \text{Conv}, \text{Pooling}, \text{Dropout}\}$ and a width or filter count $w_i \in \mathbb{N}$. Here $L_{\max}$ is a user-specified constant that caps the depth of any student network; it determines the length of the $\mathbf{a}$ vector and therefore the maximum number of (layer type, width) pairs the search can propose.

We note that our search space consists of foundational layer types. This was chosen to create a clear and interpretable environment to validate our efficiency mechanisms. Future work should expand this space to include more modern components like residual connections or attention mechanisms to enhance the representational capacity of the searched models.

The proposed methodology makes a clear distinction between the *planning phase* and the *execution phase*, allowing for effective exploration while minimizing computational overhead. In the planning phase, the *agent* decides on the student architecture and then transitions to the training phase to evaluate the model. During the execution phase, the

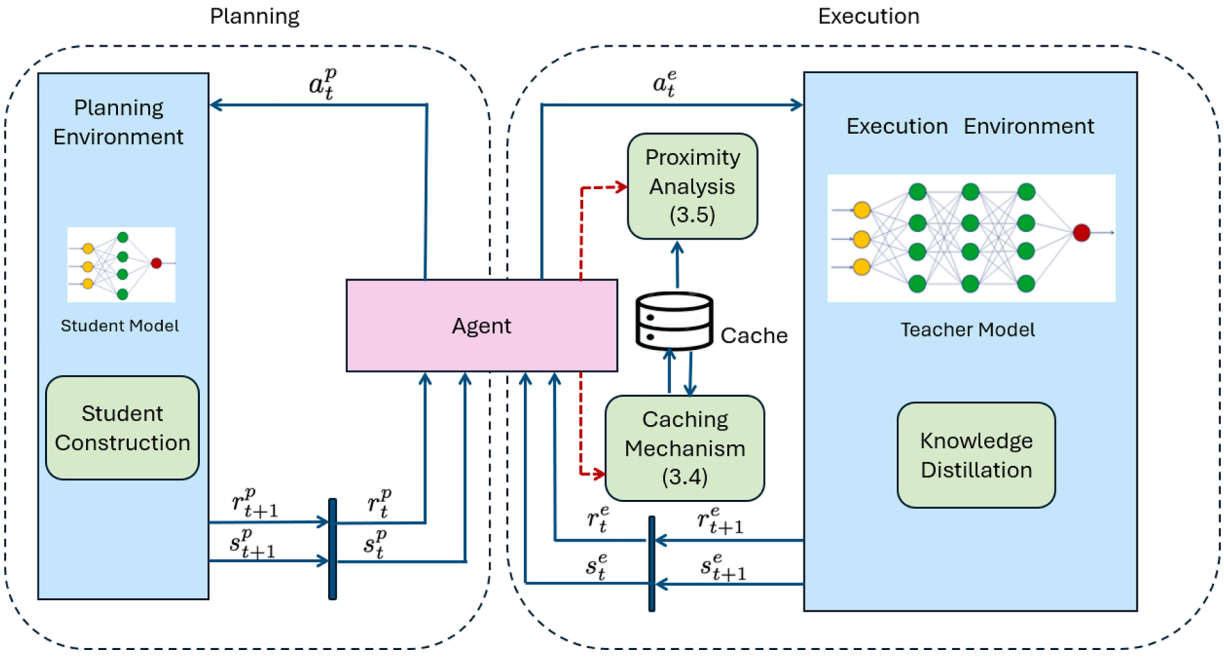


Fig. 1. KD policy learning framework.

agent employs proximity analysis and a caching mechanism to determine whether the proposed model should be evaluated using actual training data or if its performance can be estimated effectively. If the architecture requires evaluation in the actual training step, the agent searches for the optimal hyperparameters (i.e., the number of epochs and the learning rate). In particular, the environments in the planning and execution stages encapsulate the student *construction* and *KD* respectively.

3.2. Planning phase

The planning phase focuses on the construction of potential student neural network architectures by the agent. During this phase, the agent sequentially takes actions that define the architecture's layers or to proceed to the execution phase.

Actions in planning (a_t^p): The action a_t^p at time step t in the planning phase specifies the operation, which is *Add Layer* or *Train*. In case the operation is *Add Layer*, a_t^p is also augmented with layer types (e.g., dense, convolutional, dropout, pooling), and layer parameters (e.g., number of nodes/filters). If the operation is *Train*, we proceed to the execution phase.

States in planning (s_t^p): The state s_t^p at time step t in the planning phase includes the student architecture built so far.

In particular, at each step t of the episode, the agent observes the current state s_t^p. Based on this state, the agent takes an action a_t^p to modify the architecture (through *Add Layer* operation). The environment updates the configuration according to a_t^p, resulting in a new state s_{t+1}^p.

During the planning phase, the agent receives no immediate reward for its actions ($r_t^p = 0$). This design encourages the agent to focus on constructing effective architectures without being influenced by short-term incentives. To prevent the agent from acting greedily by immediately proceeding to the execution phase to obtain a reward, we employ two key strategies: By using an *exponential reward* function in the execution phase (see Section 3.3), even small improvements in performance result in significantly higher rewards. This incentivizes the agent to invest effort in constructing better architectures during the planning phase, as deeper or more complex architectures may lead to greater rewards. Moreover, the agent's loss function incorporates an entropy term, with the entropy coefficient tuned to encourage exploration. This fosters diversity in action selection, reducing the chances of repeatedly choosing the same actions. Consequently, the agent is less likely to consistently pick actions leading directly to execution and is more inclined to explore additional layers or architectural choices before training.

The planning phase ends when the agent decides to proceed to the execution phase by selecting the appropriate action. If the agent reaches the maximum allowed number of layers ($L_{\max}$) without deciding to proceed, the environment forces the transition to the execution phase.

3.3. Execution phase

The execution phase begins once a complete configuration is determined. In this phase, the agent's proposed configuration is evaluated by assessing its performance and providing a corresponding reward.

The evaluation process involves several steps:

1. **Memory Check**: The configuration is checked against the stored configurations in the cache memory (Section 3.4). If the configuration exists in cache and has been sufficiently evaluated, its stored performance metrics are retrieved.
2. **Similarity Assessment**: If the configuration is not found in the cache or has not been evaluated enough times, it undergoes a proximity analysis (Section 3.5). An estimated validation accuracy and reward may be obtained based on similarity to previously evaluated configurations.
3. **Full Evaluation**: If the configuration is deemed novel or potentially high-performing, it undergoes a full evaluation (Section 3.6). This involves training the model and obtaining its actual performance metrics.

Actions in execution (a_t^e): The action a_t^e at time step t in the execution phase specifies the hyperparameters (i.e., learning rate, number of epochs).

States in execution (s_t^e): The observation space in the execution phase includes the complete architecture representation, actual or estimated performance metrics, and the number of parameters.

Rewards in execution (r_t^e): We define two reward functions-*partial* and *complete*-whose value depends on the origin of the performance signal. If a configuration is already in the cache we return its average reward $\bar{r}_m$; if it is not cached but has close neighbours we return the proximity-estimated reward $\bar{r}_s$; otherwise we train the model, obtain its validation accuracy $A \in [0, 1]$, and compute the reward directly. In the complete variant we also incorporate the student's parameter count N relative to the bounds $N_{\min}$ and $N_{\max}$ defined by the action space.

For the *partial* reward function, the reward is calculated as:

$$r_t^e = \begin{cases} \bar{r}_m & \text{if retrieved from cache (Section3.4),} \\ \bar{r}_s & \text{if estimated by proximity analysis (Section3.5),} \\ 10^A & \text{if fully evaluated (Section3.6).} \end{cases} \tag{1}$$

When a configuration is fully trained, we use an exponential reward function (10^A). This non-linear transformation creates a steeper reward gradient to more strongly incentivize improvements at higher accuracy levels. While a direct linear mapping of accuracy to reward is a foundational approach [2], non-linear transformations that amplify reward differences in high-performance regions are an established practice for pushing an agent beyond local optima of "good enough" solutions [32]. As we demonstrate in our ablation study (M), this exponential formulation empirically outperforms both linear and quadratic rewards in our framework.

For the *complete* reward function, we incorporate a size penalty to encourage the agent to find models that are both accurate and compact. The size penalty P is defined as:

$$P = \frac{N - N_{\min}}{N_{\max} - N_{\min}} \tag{2}$$

where N is the number of parameters of the model, $N_{\min}$ is the minimum possible number of parameters a student model can have, and $N_{\max}$ is the maximum possible number of parameters a student model can have, given the layers and configurations available through the action space. These values are determined prior to training by analyzing the action space. By constructing models using the combinations of layers and parameters allowed in the action space-without actually training them-we can determine the minimum and maximum possible sizes (in terms of the number of parameters) that a student model can have. This parameter-based penalty provides a simple, hardware-agnostic proxy for model complexity. We acknowledge that for specific hardware deployments, more direct metrics like inference latency or FLOPs would be more representative and are a key direction for future research.

The complete reward function is then defined as:

$$r_t^e = \begin{cases} \bar{r}_m & \text{if retrieved from cache (Section 3.4)} \\ \bar{r}_s & \text{if estimated by proximity analysis (Section 3.5)} \\ 10^A - 10^P & \text{if fully evaluated (Section 3.6)} \end{cases} \tag{3}$$

In this case, if the model is evaluated in full, the agent gains a high reward for accurate predictions (through 10^A) but also suffers a penalty based on P, which grows with model size. By structuring the execution phase in this way, we balance the need for accurate performance assessment with computational efficiency, allowing the agent to explore the search space effectively while favoring both high accuracy and compactness.

3.4. Caching mechanism

The Caching Mechanism operates during the execution phase to prevent redundant evaluations of identical configurations.

Definition 3.4.1. (Cache/Memory): Memory $\mathcal{M}$ contains a set of triples $(c, \bar{r}_m, \text{count})$, where c is the configuration, $\bar{r}_m$ is the average reward, and count is the number of times the configuration has been evaluated.

The Memory/Cache $\mathcal{M}$ has two procedures: Cache Check and Cache Update. In Cache Check, a complete configuration c is checked against the cache/memory $\mathcal{M}$. If the configuration has been evaluated before and has sufficient evaluation counts n (Algorithm 1, line 3), we retrieve the stored average reward $\bar{r}_m$ for configuration c. Otherwise, we return $Null$. In this case, we need a full evaluation to gather more data.

Algorithm 1 Cache Check(c, $\mathcal{M}$, n).

1: **if** $c \in \mathcal{M}$ **then**
2: $(\text{count}, \bar{r}_m) \leftarrow \mathcal{M}[c]$
3: **if** count $\geq n$ **then**
4: **return** $\bar{r}_m$
5: **end if**
6: **end if**
7: **return** Null

Algorithm 2 Cache Update(c, r, $\mathcal{M}$, n).

1: **if** $c \in \mathcal{M}$ **then**
2: $(\text{count}, \bar{r}_m) \leftarrow \mathcal{M}[c]$
3: **else**
4: Initialize $\bar{r}_m \leftarrow 0$, count $\leftarrow 0$
5: **end if**
6: **if** count $< n$ **then**
7: $\bar{r}_m \leftarrow \dfrac{\bar{r}_m \times \text{count} + r}{\text{count} + 1}$ ▷ Update Avg. Reward
8: count $\leftarrow$ count $+ 1$
9: Update $\mathcal{M}[c] \leftarrow (\bar{r}_m, \text{count})$
10: **end if**

In Cache Update (Algorithm 2), if count $< n$, we update the memory with the new performance metrics and increment the evaluation count for the configuration c. Also, the average reward $\bar{r}_m$ is updated incrementally.

3.5. Proximity analysis

Let $\mathbf{w} = [w_1, \ldots, w_n]^\top (\geq 0)$ be the feature-weight vector applied to the n Normalized features (architecture encoding, learning-rate, epochs). In the *baseline* variant we set $\mathbf{w} = \mathbf{1}$ (i.e. $w_j = 1\ \forall j$); the dynamic variant in Section 3.5.1 learns $\mathbf{w}$ online.

The Proximity Analysis estimates the performance of configurations that are similar to those previously assessed. This process occurs during the execution phase, following the caching mechanism (If a configuration isn't found in memory or hasn't been adequately evaluated, we determine its similarity to existing configurations).

Algorithm 3 outlines the procedure for proximity analysis. First, we represent configurations as feature vectors that include architectural features, learning rate, and number of epochs. All features are normalized using standard scaling (lines 4 and 6). Next, we compute the Euclidean distances between the current configuration and all configurations in memory (line 7). We then identify the k nearest neighbors (with $k = 3$) based on these distances, using their rewards to estimate the performance of the current configuration (lines 14 and 15).

We calculate the average distance $\bar{d}$ to the nearest neighbors and the standard deviation σ_d of all distances. A configuration c is considered not *novel* if $\bar{d} < 0.5\sigma_d$, and it is deemed not *promising* if the average reward

Algorithm 3 Proximity Analysis($c, \mathcal{M}, k$).

1: **if** $\mathcal{M}$ is empty **then**
2: **return** Null
3: **end if**
4: $\mathbf{f}_c \leftarrow \text{Normalize}(\text{Vector}(c))$
5: **for** each configuration c_i in $\mathcal{M}$ **do**
6: $\mathbf{f}_{c_i} \leftarrow \text{Normalize}(\text{Vector}(c_i))$
7: $d_i = \sqrt{\sum_{j=1}^{n} w_j (f_{c,j} - f_{c_i,j})^2}$ ▷ distance from c to all $c_i \in \mathcal{M}$
8: $\bar{r}_{c_i} \leftarrow \mathcal{M}[c_i]$
9: **end for**
10: **if** $|\mathcal{M}| < k$ **then**
11: $k \leftarrow |\mathcal{M}|$
12: **end if**
13: **if** $k > 0$ **then**
14: $I \leftarrow$ Indices of k nearest neighbors based on d_i
15: $\bar{r}_s = \dfrac{1}{k} \sum_{i \in I} \bar{r}_{c_i}$
16: $\sigma_d = \text{STD}(\{d_i : i = 1 \ldots |\mathcal{M}|\})$
17: $p_{90\%} = 90\text{th Percentile}(\{\bar{r}_{c_i} : i = 1 \ldots |\mathcal{M}|\})$
18: $\bar{d} = \text{Average}(\{d_i : i \in I\})$
19: **if** $\bar{d} < 0.5\sigma_d$ **and** $\bar{r}_s \leq p_{90\%}$ **then**
20: **return** $\bar{r}_s$
21: **end if**
22: **return** Null
23: **end if**
24: **return** Null

$\bar{r}_s$ does not exceed a predefined high-performance threshold (set at the 90th percentile of rewards). If neither condition is met, we return the estimated performance $\bar{r}_s$; otherwise, we return $Null$.

Under the uniform-weight baseline we simply have $\mathbf{w} = \mathbf{1}$, so the distance metric reduces to the ordinary Euclidean norm.

3.5.1. Proximity analysis with dynamic feature weights

Unlike the baseline variant, here we *learn* the weight vector $\mathbf{w}$ instead of fixing $\mathbf{w} = \mathbf{1}$. The underlying feature space is the **numeric feature vector** $\mathbf{f}(c) = [\texttt{LR}, \texttt{EP}, \mathbf{a}]^\top$ defined in Definition 3.1. Thus the n dimensions consist of (i) learning-rate LR, (ii) epoch count EP, and (iii) the $L_{\max}$-length architecture encoding $\mathbf{a}$ from Definition 3.2.

To determine the importance of each feature, we train a Random Forest regressor [33] on historical (config, reward) pairs stored in the cache. The regressor provides raw feature-importance scores, which we normalise to obtain the weight vector $\mathbf{w} = [w_1, \ldots, w_n]^\top$, where each w_j represents the relative influence of feature j.

Weight-update schedule. We update $\mathbf{w}$ every $\eta = 5$ full (non-cached) evaluations, provided that at least $m = 10$ configurations have been evaluated so far. At each update the Random Forest is retrained from scratch using all cached data, and its hyper-parameters are tuned via a small randomised search with cross-validation to preserve predictive accuracy.

With these learned weights we substitute the weighted Euclidean distance of Algorithm 3; all remaining steps (neighbour search, novelty check, reward estimate) are identical to the baseline procedure.

3.6. Full evaluation

When a configuration undergoes full evaluation, the environment constructs the neural network model based on the specified architecture. It then trains the model using the selected hyperparameters and performs KD from the teacher model. We employ a KD loss function that combines the standard Cross-Entropy (CE) loss with the Kullback-Leibler (KL) divergence between the teacher's and student's output distributions [1]. After training, the model's performance is evaluated on a validation set to obtain validation accuracy A. Depending on the chosen

reward type (*partial* or *complete*), the reward r_t^e is calculated as described in Section 3.3.

3.7. KD policy learning

Algorithm 4 outlines the learning procedure for the proposed framework, where the main loop (line 2) integrates the planning and execution phases to efficiently construct and evaluate student models. Each *episode* represents the process of building and assessing a single student model. In the planning phase, the agent incrementally constructs neural network architectures by adding layers (line 9). Once the agent either decides to proceed to execution or reaches the maximum number of layers ($L_{\max}$), it initializes the hyperparameters (line 12), appends them to the architecture to form the configuration c (line 13), and transitions to the execution phase with c (line 14). In the execution phase, the agent checks the cache (line 20) to determine if the configuration has already been sufficiently evaluated (detailed in Algorithm 1). If not, the agent attempts to estimate the reward using proximity analysis (detailed in Algorithm 3). If proximity analysis cannot provide an estimate (returns $Null$), full training with KD is performed (line 28). Depending on whether the partial or complete reward function is being used, the agent calculates r_t^e accordingly (lines 29–33).

Algorithm 4 KD policy learning.

1: **Initialize** agent, environment, memory $\mathcal{M}$, evaluation threshold n.
2: **for** each episode **do**
3: **Planning Phase**:
4: architecture $\leftarrow \emptyset$
5: $l \leftarrow 0$
6: **while** True **do**
7: Agent observes s_t^p and takes action a_t
8: **if** a_t is **Add Layer** and $l + 1 \leq L_{\max}$ **then**
9: Add layer to architecture
10: $l \leftarrow l + 1$
11: **else**
12: Agent initializes hyperparameters
13: $c \leftarrow$ {architecture, hyperparameters}
14: Proceed to execution phase with c
15: **break** loop
16: **end if**
17: Provide zero reward $r_t^p = 0$ to agent
18: **end while**
19: **Execution Phase**:
20: $\bar{r}_m \leftarrow \text{CacheCheck}(c, \mathcal{M}, n)$
21: **if** $\bar{r}_m \neq \text{Null}$ **then**
22: $r_t^e \leftarrow \bar{r}_m$
23: **else**
24: $\bar{r}_s \leftarrow \text{ProximityAnalysis}(c, \mathcal{M})$
25: **if** $\bar{r}_s \neq \text{Null}$ **then**
26: $r_t^e \leftarrow \bar{r}_s$
27: **else**
28: $A \leftarrow \text{Knowledge Distillation}(c)$
29: **if** Using *partial* reward **then**
30: $r_t^e \leftarrow 10^A$
31: **else if** Using *complete* reward **then**
32: Compute size penalty P
33: $r_t^e = 10^A - 10^P$
34: **end if**
35: **end if**
36: CacheUpdate(c, r_t^e, $\mathcal{M}$, n)
37: **end if**
38: Agent updates policy using reward r_t^e
39: **end for**

4. Empirical evaluation

In this section, we present a detailed empirical analysis of our proposed methods. We first describe the experimental setup (Section 4.1), then provide results for CIFAR-10 (Section 4.2.1) and IMDb (Section 4.2.2), covering accuracy/time trade-offs, KL divergence, and our `PPO + Dynamic Prox` approach's feature-weight evolution. We then analyze the time/accuracy trade-offs using z-scores (Section 4.4) and discuss how caching and proximity reduce overhead (Section 4.3), followed by an expanded discussion (Section 5) and conclusion (Section 6). All appendices are included thereafter.

4.1. Experiment setup

We conducted experiments on two tasks: **CIFAR-10** [34] for image classification (using MobileNetV2 [35] as the teacher) and **IMDb** [36] for sentiment analysis (using DistilBERT [18] as the teacher). For full details on dataset splits and teacher model configurations, please see E and F. Across both tasks, we evaluated **4,000** student models under two reward variants: a *partial* reward (accuracy-only) and a *complete* reward (accuracy minus size penalty; see Section 3.3). The methods compared are: **Grid**, **Random Search**, **Random Agent**, **Regularized Evolution** [19], **Bayesian Single** (single-environment GP), **Bayesian Parallel** (multi-environment: 8 for CIFAR-10, 4 for IMDb), **Proximal Policy Optimization (PPO)** [37], plus our **PPO + Cache**, **PPO + Proximity**, and **PPO + Dynamic Prox** variants. We measured total wall-clock time (in hours) and the highest validation accuracy achieved. Further implementation details, such as hardware configuration, action space, and loss function settings, are provided in the Appendices.

Importantly, the *student models are not seeded*, ensuring each architecture is trained from random initial weights. This choice allows us to observe a distribution of possible performance values for identical architectures, which benefits both our caching mechanism (storing multiple performance outcomes) and proximity estimates. Consequently, some methods (e.g., `Random Agent`) may occasionally appear to yield high accuracy simply due to favorable random initializations, but overall performance trends remain consistent across repeated evaluations.

4.2. Efficiency and accuracy

We first examine CIFAR-10 results (Section 4.2.1), where larger architectures typically yield higher accuracy, and then IMDb results (Section 4.2.2), where small networks usually suffice. We show each method's accuracy/time trade-offs, discuss KL divergences (comparing exploration distributions to PPO's), and analyze `PPO + Dynamic Prox`'s feature-weight shifts.

4.2.1. CIFAR-10 results

Validation Accuracy Over Time: Fig. 2 illustrates each method's *best* validation accuracy over time. Under the **partial** reward, `PPO + Proximity` and `PPO + Dynamic Prox` rapidly approach `PPO`'s accuracy, saving around 75 % of total evaluation time. `PPO + Cache` also achieves near-PPO accuracy with ~70 % less time. The `Regularized Evolution` baseline proves to be a very strong performer, achieving the highest accuracy overall while being significantly faster than the standard PPO agent. Bayesian methods can discover reasonable architectures but remain relatively slow or inconsistent in discrete, higher-dimensional spaces. Under the **complete** reward (right plot), which penalizes large models, `PPO + Proximity` and `Regularized Evolution` track PPO's accuracy closely while cutting time by over 75 %.

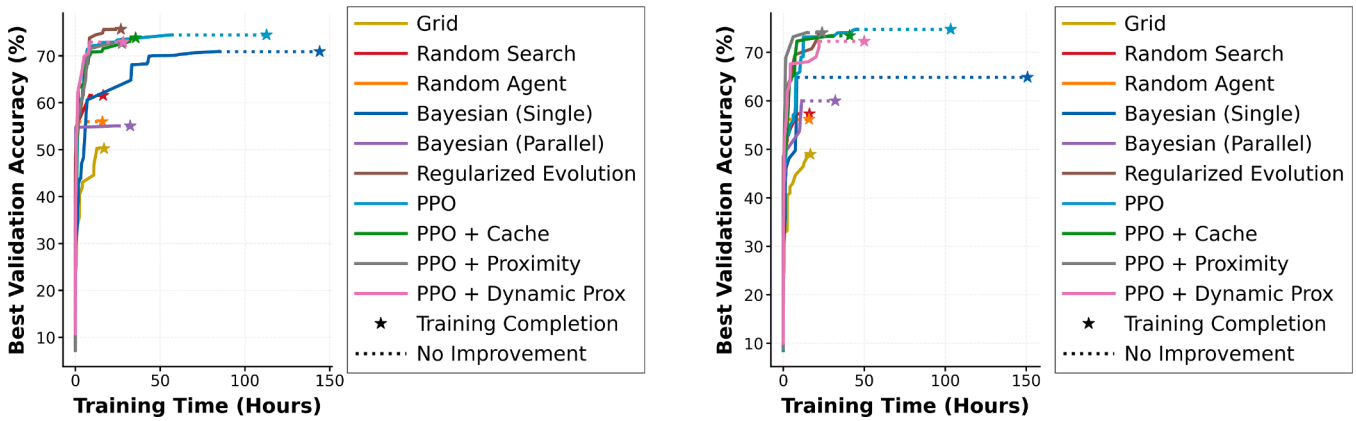


Fig. 2. Best Validation Accuracy Over Time (CIFAR-10). **Left:** Partial reward (accuracy-only). **Right:** Complete reward (with size penalty).

Table 1
CIFAR-10 Results (Partial vs. Complete Reward) for 4000 Models. **Bold** indicates the highest accuracy. Percentages in parentheses with arrows (↑↓) show the relative change compared to the PPO baseline.

	Partial Reward		Complete Reward	
Algorithm	Time (hrs)	Acc (%)	Time (hrs)	Acc (%)
Grid	16.9 (↓ 85.0 %)	50.2 (↓ 24.2 %)	16.7 (↓ 83.8 %)	49.0 (↓ 25.8 %)
Random Search	16.4 (↓ 85.4 %)	61.6 (↓ 12.9 %)	16.2 (↓ 84.4 %)	57.4 (↓ 17.4 %)
Random Agent	15.9 (↓ 85.9 %)	56.0 (↓ 18.5 %)	15.9 (↓ 84.6 %)	56.2 (↓ 18.6 %)
Regularized Evolution	26.9 (↓ 76.2 %)	**75.7** (↑ 1.2 %)	24.1 (↓ 76.7 %)	74.0 (↓ 0.8 %)
Bayesian Single	144.0 (↑ 27.8 %)	70.9 (↓ 3.5 %)	150.8 (↑ 46.0 %)	64.9 (↓ 9.9 %)
Bayesian Parallel	32.3 (↓ 71.3 %)	55.1 (↓ 19.4 %)	32.1 (↓ 68.9 %)	60.0 (↓ 14.8 %)
PPO	112.7 (-)	74.5 (-)	103.3 (-)	**74.8** (-)
PPO + Cache	35.5 (↓ 68.5 %)	73.8 (↓ 0.6 %)	41.0 (↓ 60.3 %)	73.4 (↓ 1.3 %)
PPO + Proximity	27.5 (↓ 75.6 %)	72.7 (↓ 1.8 %)	23.9 (↓ 76.8 %)	74.1 (↓ 0.7 %)
PPO + Dynamic Prox	28.3 (↓ 74.9 %)	73.0 (↓ 1.5 %)	50.0 (↓ 51.6 %)	72.3 (↓ 2.5 %)

Time/Accuracy Trade-Offs: Table 1 quantifies these outcomes. Under the partial reward, `Regularized Evolution` is the top performer, achieving the highest accuracy (75.7 %) with a 76.2 % reduction in time compared to PPO. Our PPO variants also yield significant time savings (∼ 75 %) with only a minor drop in accuracy. Once the size penalty is introduced (complete reward), `PPO + Proximity` and `Regularized Evolution` both remain highly competitive, delivering accuracy near the PPO baseline while drastically cutting time. `Bayesian Single` achieves decent accuracy but at a higher cost, while `Bayesian Parallel` is faster but less accurate on average.

KL Divergence: B shows how each method's parameter/accuracy distributions compare to `PPO`'s via KL divergence. Under partial reward, `PPO + Dynamic Prox` is usually closest (lowest KL), while under complete reward `PPO + Cache` more closely mirrors PPO's distribution. Random or BO approaches often diverge more from PPO.

Feature Weights (Dynamic Prox, CIFAR-10): Using `PPO + Dynamic Prox`, we track how a Random Forest regressor assigns importance to three key features: *architecture*, *learning rate*, and *number of epochs*. Across both partial and complete reward conditions (see L), *architecture* consistently receives the highest weight-rising above 50 % on average-while learning rate gradually increases in importance as the search progresses. This outcome indicates that, on CIFAR-10, changes to a model's architecture (e.g., depth, width, or layer types) correlate most strongly with final performance outcomes, making it the main driver of accuracy variations. Hence, the agent invests considerable effort in exploring different structural configurations early on, leading to the largest gains. By the late phase of training, architecture still dominates but learning rate becomes more influential (up to 37.4 % under the partial reward), suggesting that once promising architectures are found, hyperparameter tuning can further boost accuracy. In short, for image classification on CIFAR-10, structural choices matter most initially, whereas refinements to learning rate become increasingly impactful after sufficiently capable models are identified.

4.2.2. IMDb results

Validation Accuracy Over Time: As shown in Fig. 3, on IMDb the agent quickly finds that relatively small student models can achieve high accuracy. Under the **partial** reward, `Random Agent` happens to find a high accuracy (85.4 %) early on, likely aided by unseeded weight initialization, though it does so inconsistently. `PPO + Cache` and `PPO + Proximity` converge to near-equal accuracy as PPO, but finish over 92 % faster. In contrast to its performance on the image task, `Regularized Evolution` is significantly slower than the PPO baseline here, a point we analyze further in the discussion. Under the **complete** reward, `Bayesian Parallel` leads slightly (85.3 %) but is slower than our caching/proximity approaches.

Time/Accuracy Trade-Offs: Table 2 confirms the complex trade-offs on this task. Under the partial reward, `Regularized Evolution` achieves the highest accuracy (85.5 %) but takes 19.0 % *more* time than the PPO baseline, illustrating a clear inversion of its efficiency on CIFAR-10. `Random Agent` also finds a high-accuracy model, likely by chance. In contrast, our `PPO + Cache` and `PPO + Proximity` methods remain over 92 % faster than PPO while maintaining near-equal accuracy. Under the complete reward, `Bayesian Parallel` attains the top accuracy (85.3 %) but at a high time cost, while our caching and proximity approaches again provide significant speed-ups with only minor accuracy gaps.

KL Divergence & Dynamic Proximity: On the IMDb sentiment analysis task, `PPO + Dynamic Prox` reveals a different emphasis. Under the partial reward, learning rate quickly becomes the primary driver (roughly 77.8 % mean importance), overshadowing architecture, which contributes only 16.6 % in the early phase and 19.5 % in mid phase. When the complete reward introduces a size penalty, learning rate importance grows even further, exceeding 84.8 % in the early hours and surpassing 92.6 % by mid phase; architecture thus drops below 10 % in influence. Because IMDb's accuracy does not substantially improve with larger models, the agent converges rapidly on compact architectures and devotes

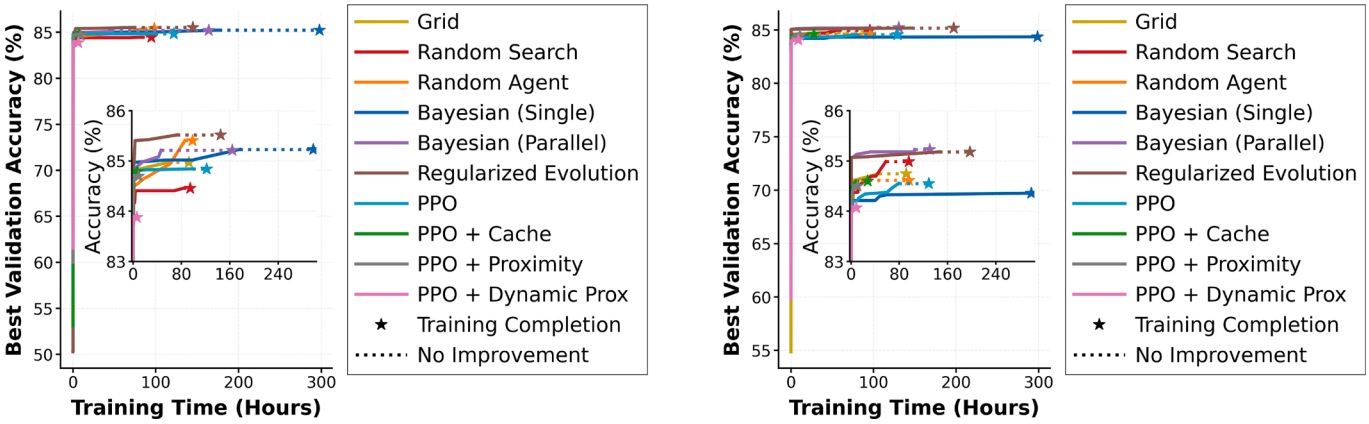


Fig. 3. Best Validation Accuracy Over Time (IMDb). **Left:** Partial reward; **Right**: Complete reward.

Table 2
IMDb Results (Partial vs. Complete Reward) for 4000 Models. **Bold** indicates the highest accuracy. Percentages in parentheses with arrows (↑↓) show the relative change compared to the PPO baseline.

	Partial Reward		Complete Reward	
Algorithm	Time (hrs)	Acc (%)	Time (hrs)	Acc (%)
Grid	92.5 (↓ 24.0 %)	85.0 (↑ 0.2 %)	91.5 (↓ 29.1 %)	84.8 (↑ 0.2 %)
Random Search	94.6 (↓ 22.3 %)	84.5 (↓ 0.4 %)	95.6 (↓ 25.9 %)	85.0 (↑ 0.4 %)
Random Agent	98.3 (↓ 19.3 %)	85.4 (↑ 0.7 %)	96.0 (↓ 25.6 %)	84.6 (↑ 0.0 %)
Regularized Evolution	144.8 (↑ 19.0 %)	**85.5** (↑ 0.8 %)	197.1 (↑ 52.8 %)	85.2 (↑ 0.7 %)
Bayesian Single	297.8 (↑ 144.7 %)	85.2 (↑ 0.5 %)	298.7 (↑ 131.6 %)	84.4 (↓ 0.2 %)
Bayesian Parallel	164.1 (↑ 34.8 %)	85.2 (↑ 0.5 %)	130.6 (↑ 1.2 %)	**85.3** (↑ 0.8 %)
PPO	121.7 (-)	84.8 (-)	129.0 (-)	84.6 (-)
PPO + Cache	7.0 (↓ 94.3 %)	84.8 (↓ 0.0 %)	27.6 (↓ 78.6 %)	84.6 (↓ 0.0 %)
PPO + Proximity	9.2 (↓ 92.4 %)	84.7 (↓ 0.1 %)	10.8 (↓ 91.6 %)	84.5 (↓ 0.1 %)
PPO + Dynamic Prox	6.2 (↓ 95.0 %)	84.3 (↓ 0.6 %)	8.7 (↓ 93.2 %)	84.1 (↓ 0.6 %)

most attention to hyperparameter tuning. Parameter-count trends in C (Fig. C.10) confirm that the best solutions remain small, often below 250K parameters; this aligns with the time-versus-accuracy results in Fig. 3 and supports our conclusion that large student networks yield minimal gains in IMDb.

4.3. Evaluation method usage over time

Each PPO-based method (plain PPO, `PPO + Cache`, `PPO + Proximity`, `PPO + Dynamic Prox`) gradually replaces *full* training runs with *cached* or *similarity*-based estimates. Fig. 4 shows this trend for the **partial reward** function. Across the 4,000-model budget on **CIFAR-10**, cached or estimated evaluations account for ≈ 83 % of all decisions in `PPO + Proximity` and `PPO + Dynamic Prox` by the halfway mark, cutting wall-clock time by over 75 % with a mean accuracy loss below 1.5 %. On **IMDb** the shift happens even faster: more than 90 % of evaluations are non-full after only 600 models, which explains the > 90 % runtime reduction reported in Table 2. A similar pattern is observed under the **complete reward** function, as shown in Fig. 5. While the size penalty can alter the agent's exploration strategy, the fundamental efficiency benefit remains. The system consistently learns to rely on caching and proximity analysis after an initial exploration phase, regardless of the reward structure.

These numbers confirm that once the agent has collected a modest cache of examples, the combination of caching and proximity estimation dominates the work-load, leaving full training runs for genuinely novel or potentially high-payoff configurations. Overall, the curves show a consistent two-phase pattern: an *exploration phase* dominated by full evaluations while the cache is sparse, followed by an *exploitation phase* where cached or similarity-based shortcuts dominate. The transition occurs sooner on IMDb because the task saturates with much smaller models, allowing quick reuse of prior knowledge; on CIFAR-10 the agent needs a longer full-evaluation period to populate the cache with diverse architectures.

4.4. Time/accuracy trade-offs (Z-score analysis)

The horizontal axis in Figs. 6 and 7 reports how many standard deviations (z-score) a method's validation accuracy deviates from PPO's mean, while the vertical axis shows the analogous z-score for wall-clock time reduction. Points in the green quadrant dominate PPO on *both* metrics; yellow quadrants indicate a trade-off; pink marks underperformance on both.

4.4.0.1. CIFAR-10 (image). In Fig. 6 (left)-the *partial* reward-our caching & proximity variants and `Regularized Evolution` all land in the green Better Region, achieving positive z-scores on both axes. `Regularized Evolution`, `PPO + Cache`, `PPO + Proximity`, and `PPO + Dynamic Prox` all demonstrate superior performance over the PPO baseline. All other fast methods lie in the upper-yellow band (speed gains, accuracy losses), whereas `Bayesian (Single)` appears in the lower-right yellow quadrant-accurate but slow. Moving to Fig. 6 (right)-the *complete* reward-`Regularized Evolution`, `PPO + Cache`, and `PPO + Proximity` all occupy the green quadrant, while `PPO + Dynamic Prox` moves to the lower-right yellow quadrant, as it is slightly slower (Time Z = -0.06) but maintains a strong accuracy score (Accuracy Z = +0.71).

4.4.0.2. IMDb (NLP). Fig. 7 (left) shows that under the *partial* reward, `PPO + Cache`, `PPO + Proximity`, and `PPO + Dynamic Prox` yield the largest time gains but with lower accuracy, placing them in the upper-left yellow quadrant. Notably, `Random Agent` lands in the green "Better Region," being slightly faster (Time Z = +0.06) with high accuracy (Accuracy Z = +1.04). Regularized Evolution offers an accuracy-time trade-off, appearing in the lower-right yellow quadrant as it is slower (Time Z = -0.46) but achieves the highest accuracy score of all methods

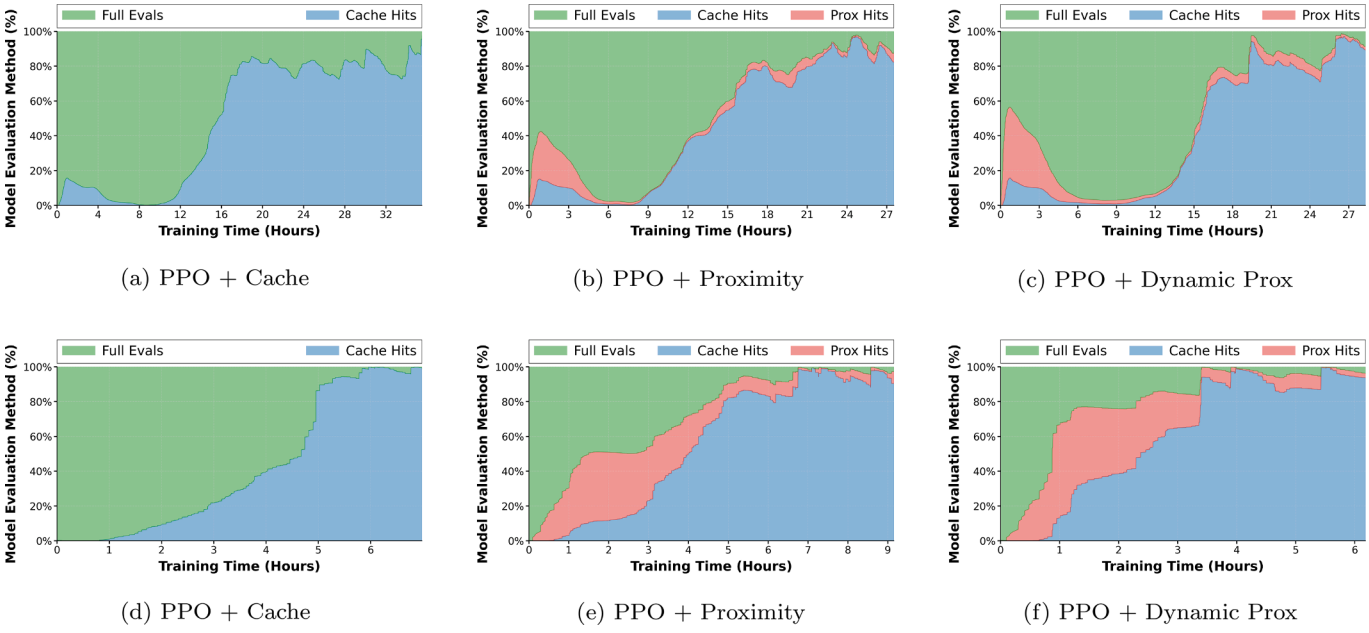


Fig. 4. Evaluation-method usage over time for **CIFAR-10** (top row) and **IMDb** (bottom row) using the **Partial Reward** function.

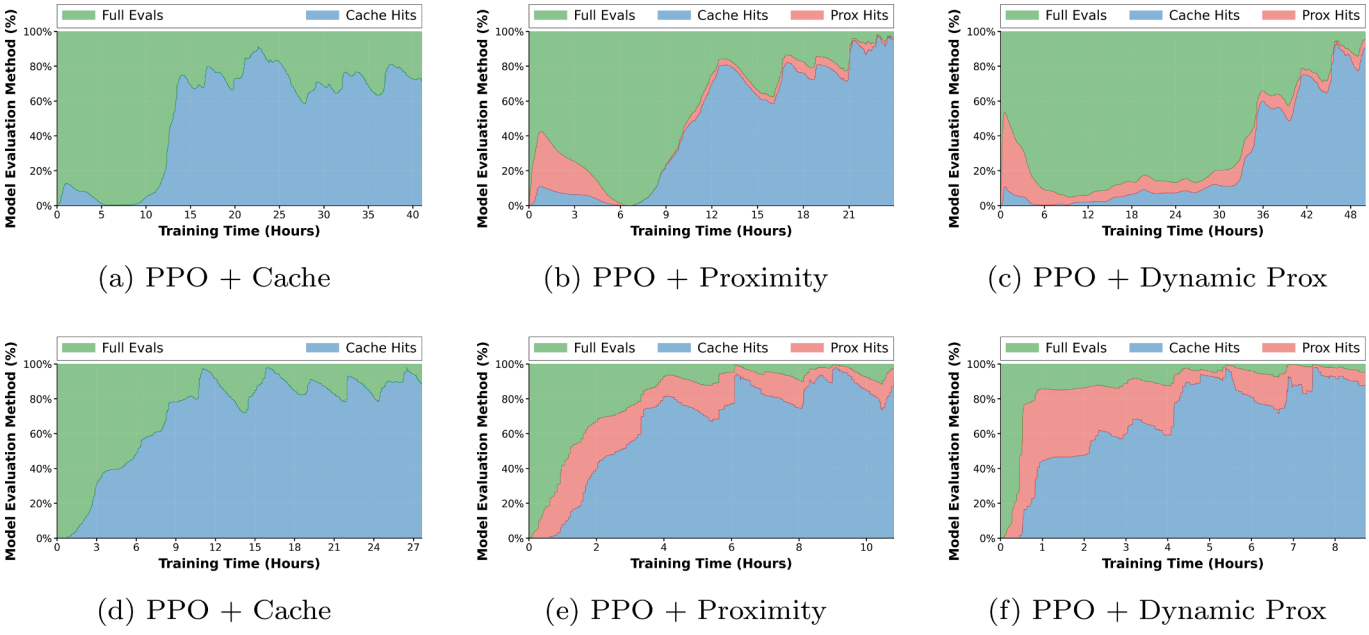


Fig. 5. Evaluation-method usage over time for **CIFAR-10** (top row) and **IMDb** (bottom row) using the **Complete Reward** function.

(Accuracy Z = +1.27). In Fig. 7 (right)-the *complete* reward-`Bayesian Parallel` attains the top accuracy ($+1.50\,\sigma$) yet incurs a time deficit, while caching/proximity variants keep their large time savings with only minor accuracy drops. Regularized Evolution again offers a trade-off between high accuracy (Accuracy Z = +1.36) and slower performance (Time Z = -0.99). No method enters the green quadrant, underscoring IMDb's tight accuracy ceiling once compact students are found.

Across all four panels, the caching and proximity techniques consistently deliver the largest positive time-reduction y-scores. On CIFAR-10, they-along with `Regularized Evolution`-match or surpass PPO on both axes. On IMDb, they intentionally trade a fractional accuracy loss for runtime reductions of up to 95 %. These results substantiate the core claim that caching and similarity estimates dramatically accelerate RL-driven distillation without materially harming model quality.

4.5. Training stability analysis

Beyond final performance, it is crucial to understand the reliability of the search process. To provide deeper insight into the consistency of each algorithm, particularly given the single-trial nature of the experiments, we conducted a detailed training stability analysis. We define a composite **Stability Score** as $1/(1 + \mathrm{CV})$, where CV is the Coefficient of Variation of the validation accuracy over a rolling window. A higher score (closer to 1) indicates greater stability.

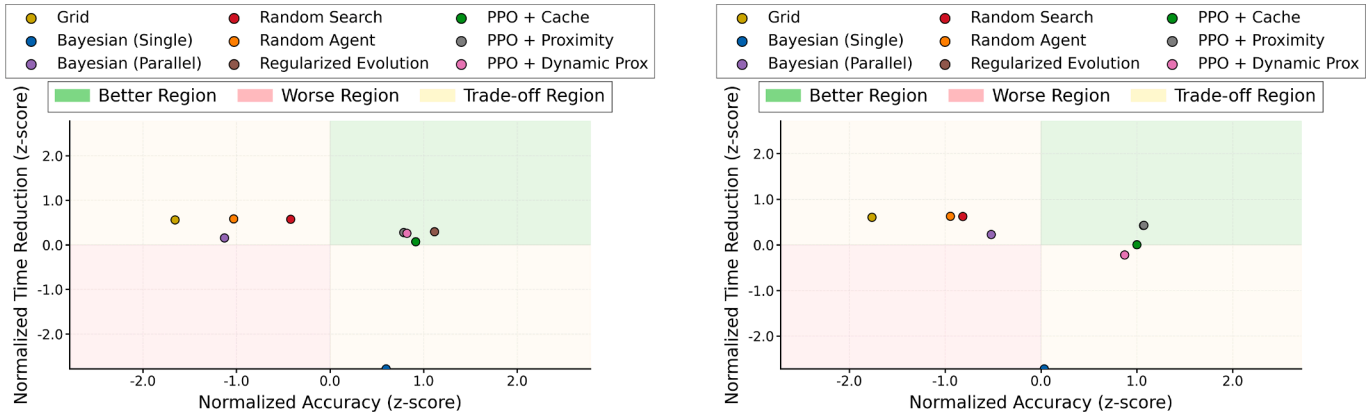


Fig. 6. CIFAR-10: Normalized Accuracy (z-score) vs. Normalized Time-Reduction (z-score). **Left**: partial reward; **Right**: complete reward. Both axes are centred on PPO at $(0, 0)$. Green upper-right $= \geq 1\sigma$ faster *and* more accurate than PPO; pink lower-left $= \geq 1\sigma$ slower *and* less accurate; yellow regions denote speed-accuracy trade-offs.

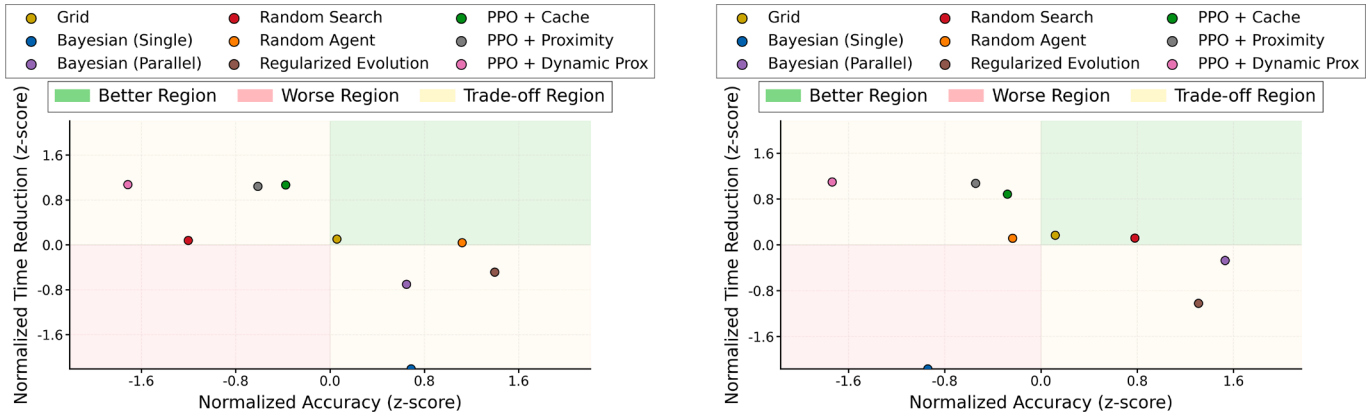


Fig. 7. IMDb: Normalized Accuracy (z-score) vs. Normalized Time-Reduction (z-score). **Left**: partial reward; **Right**: complete reward. Quadrant colours correspond to Fig. 6.

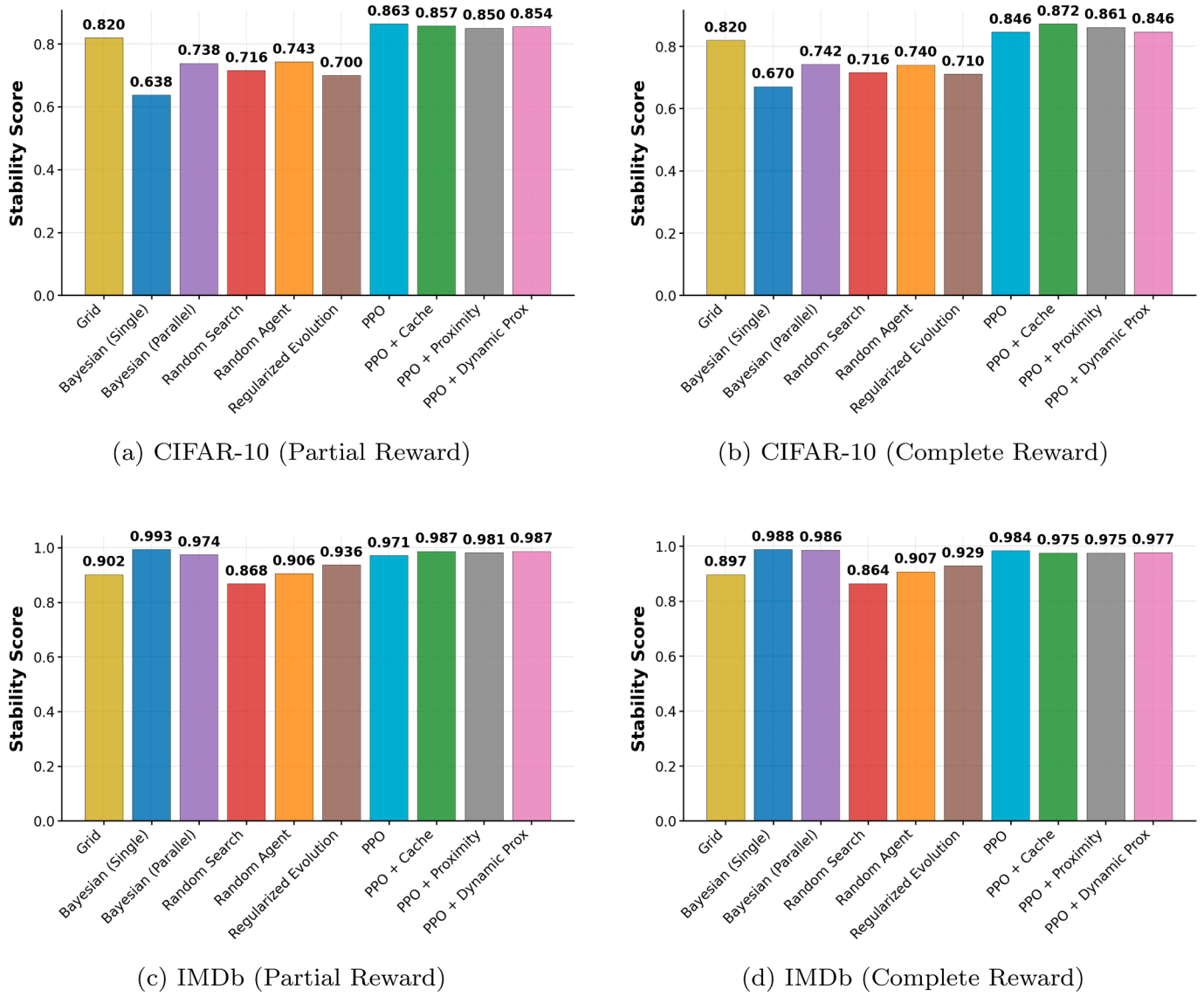
Fig. 8 shows that our PPO-based methods exhibit significantly higher training stability, especially on the more complex CIFAR-10 task. For instance, under partial rewards, the baseline PPO achieved a high Stability Score of 0.86, a benchmark that was closely matched by its variants. In stark contrast, Bayesian (Single) registered a score of only 0.64, indicating a far more erratic search process. This suggests that the KD-PL framework not only accelerates the search but also makes it more reliable and predictable. A full breakdown of the stability methodology and detailed temporal plots are available in A.

5. Discussion

5.1. Limitations and future work

While our study demonstrates a promising framework, we acknowledge several limitations and have identified clear directions for future work based on them.

- **Single-Trial Experiments:** A primary limitation of our main empirical evaluation is that the results for most baselines are based on single runs. Due to the significant computational cost (e.g., a single run of the baseline PPO agent on CIFAR-10 required up to 100 hours), a comprehensive multi-run analysis across all methods was not feasible. To address this and validate the statistical robustness of our findings, we conducted a dedicated multi-seed analysis on our most efficient method (`PPO + Dynamic Prox`), the results of which are detailed in N. While the broader results should be interpreted as an exploratory but promising demonstration, this multi-seed study confirms the reproducibility of our framework's performance.
- **Comparison with Foundational Baselines:** Our experiments compare the framework against foundational search paradigms (Grid, Random, BO, PPO) and a modern evolutionary algorithm (Regularized Evolution). This was a deliberate methodological choice driven by the specific **fixed-evaluation-budget paradigm** under investigation. Many prominent SOTA techniques, particularly gradient-based methods like DARTS and its successors, are designed for a different paradigm involving continuous architectural relaxation and are benchmarked by cumulative GPU-hours. As such, they are not directly comparable under a strict budget of discrete, full model evaluations, which is the core metric for sample efficiency in this work. Therefore, our focus was on first validating the framework's efficiency mechanisms against established paradigms that operate naturally within this discrete evaluation context.
- **Absence of Ablation Studies:** While we have included ablation studies on key hyperparameters in the knowledge distillation loss function (J) and the RL agent's reward function (M), a more comprehensive analysis covering all design choices was not feasible. The computational demands of our experiments were prohibitive; as shown in our results, baseline PPO experiments alone required over 100 hours of GPU time per task, making an exhaustive ablation study requiring multiple re-runs for each variant a task well beyond our available computational resources.
- **Simplified Search Space and Cost Proxy:** The architectural search space, while sufficient for our benchmark tasks, excluded modern components like residual connections. Similarly, the model cost



(a) CIFAR-10 (Partial Reward)

(b) CIFAR-10 (Complete Reward)

(c) IMDb (Partial Reward)

(d) IMDb (Complete Reward)

Fig. 8. Final Stability Score Comparison. Higher scores indicate greater training stability. The PPO-based methods consistently demonstrate higher stability than volatile baselines on the more complex CIFAR-10 task (top row), while most methods prove stable on the simpler IMDb task (bottom row).

proxy was based on parameter count, which does not directly capture hardware-specific metrics like latency or energy consumption.

Addressing these limitations constitutes the direct path for future work. The next phase of this research will focus on (1) conducting large-scale, multi-run benchmarks to establish statistical significance, (2) benchmarking our framework against SOTA methods under a carefully controlled evaluation-budget paradigm where applicable, and (3) extending the framework to include more sophisticated, latency-aware reward functions for hardware-specific deployments.

5.2. Analysis of framework performance

Our empirical results demonstrate that *caching* and *proximity* substantially reduce training overhead without sacrificing much accuracy. In practice, these two mechanisms allow the RL agent to reuse or estimate the performance of already-seen configurations, thus avoiding fully training identical or near-duplicate student models. This efficiency advantage is particularly useful in large search spaces, as highlighted by our evaluations on image classification (CIFAR-10) and sentiment analysis (IMDb).

To further contextualize our framework's performance, we introduced Regularized Evolution (RE) [19] as a strong, methodologically-appropriate NAS baseline. On the CIFAR-10 task, RE was highly efficient, discovering a top-performing model in only 26.9 hours, significantly faster than the standard PPO agent. This performance highlights an operational bottleneck in our PPO implementation: because the PPO agent takes actions sequentially, it can underutilize parallel workers. In contrast, batch-oriented algorithms like RE ensure maximum hardware utilization. Interestingly, this dynamic was inverted on the IMDb task, where RE was one of the slowest methods. This is attributable to the bottleneck created by our centralized teacher model for NLP. In this scenario, RE must make a costly network request for every single evaluation. Our PPO variants, however, learn to circumvent this bottleneck by leveraging their cache and proximity mechanisms, ultimately performing far more efficiently. This provides a compelling demonstration of our framework's ability to adapt to and overcome real-world operational constraints.

Within this broader context, our `Dynamic Prox` variant offers deeper insight into which design elements drive performance most strongly. By tracking *feature-weight evolution* over the course of training, we observe different trends in CIFAR-10 (image classification) versus IMDb (sen-

timent analysis), as well as clear distinctions between partial rewards (accuracy-only) and complete rewards (accuracy minus size penalty). For CIFAR-10 under the partial reward, early exploration (0–6 hours) is dominated by *architecture* (around 75.5 %), while learning rate has a more modest impact (20.5 %) and epoch count is minimal (4.0 %). In the mid phase (6–12 hours), architecture remains important (66.5 %), although learning rate grows to 27.6 %. By the late phase, the agent begins fine-tuning its training hyperparameters more aggressively, with architecture settling to 55.6 % while learning rate rises to 37.4 %. When the complete reward is used, a similar pattern emerges but with somewhat less architectural dominance: architecture averages 64.9 % in the early phase, dips to 51.8 % in mid, then rises to 58.3 % late. Learning rate correspondingly increases its relative importance once the agent has identified architectures that balance accuracy and size constraints.

By contrast, on IMDb, the agent more quickly finds that relatively small networks can already achieve high accuracy, thereby reducing the necessity of searching for more complex architectures. Under a partial reward, learning rate rapidly claims the greatest share of importance (77.8 % on average), surpassing architecture (16.6 % in the early phase and $\sim 19.5\,\%$ in the mid phase). This trend intensifies once we introduce a size penalty, with learning rate eventually comprising around 84.8 % of feature importance in the early hours and rising to 92.6 % in the mid phase. As training proceeds, the system devotes fewer resources to exploring new architectures and focuses primarily on hyperparameter refinements that can further boost accuracy for the already-compact models. Parameter-count analyses (C, Fig. C.10) confirm that the top-performing IMDb solutions generally remain below 250K parameters, reinforcing our observation that searching for bigger student models yields limited benefit in this domain.

5.3. Robustness of efficiency gains and the impact of seeding

A key consideration in this work is the use of single experimental runs for most baselines. While multiple seeds are the gold standard for verifying the robustness of the final accuracy metric, we argue that the primary conclusion of this paper-the order-of-magnitude improvement in time efficiency-is fundamentally robust to the choice of seed. This robustness stems from our framework's success at addressing inherent algorithmic and operational bottlenecks that are independent of stochastic initialization.

The non-learning baselines (`Grid`, `Random Search`, `Random Agent`) serve as a crucial experimental control. As they have no learning overhead and can evaluate candidates in a maximally parallel, non-blocking fashion, their runtimes (e.g., ~16-17 hours on CIFAR-10, Table 1) establish a practical benchmark for maximum evaluation throughput under ideal conditions.

In contrast, the learning-based baselines introduce deterministic bottlenecks that are independent of any random seed:

- **Algorithmic Complexity:** The `Bayesian (Single)` baseline is bound by the computational complexity of its Gaussian Process surrogate, which scales cubically ($O(n^3)$) with the number of evaluated points (as detailed in H.3). This ensures a slow runtime as the search progresses, a factor evidenced by its 144-hour and 297-hour execution times on CIFAR-10 and IMDb, respectively.
- **Operational Constraints:** The standard `PPO` agent must perform 4000 full, costly student evaluations. As detailed in our experimental setup (Section 4.1), each full evaluation is a non-trivial time sink. Furthermore, the synchronous nature of the PPO implementation (D) and the network latency from the centralized teacher in the IMDb setup (H) create unavoidable delays for every full evaluation.

Different random seeds would cause the agent to explore a different path, leading to minor fluctuations in the final accuracy. However, a new seed would not change these fundamental realities. The baseline methods will still be required to perform thousands of time-consuming evaluations.

The structural nature of our framework's efficiency gains is visually demonstrated in Figs. 4 and 5. These plots show a consistent and predictable behavioral shift where the framework learns to forego costly full evaluations in favor of near-instantaneous cache hits and proximity estimates. For the IMDb task, the time savings are particularly stark; the `PPO + Dynamic Prox` variant reduces search time from PPO's 121.7 hours to just 6.2 hours (a 95 % reduction) under the partial reward (Table 2). This is a direct consequence of the framework's ability to circumvent the teacher-inference queue for over 90 % of the evaluations-a structural advantage, not a stochastic one.

We empirically validate this claim with a five-seed run of our most efficient method, presented in N. The results confirm that the final validation accuracy is highly stable across runs, with a 95 % confidence interval of only $\pm 0.12\,\%$. While runtime naturally varies, the core finding of high accuracy at a vastly reduced time cost is shown to be reproducible. Therefore, while multiple runs for all baselines would provide tighter confidence intervals, the evidence confirms that the KD-PL framework's mechanisms provide a structural, and therefore reproducible, reduction in search time on the order of 75–95 %. Our Training Stability Analysis (Section 4.5) further supports the reliability of this search process.

6. Conclusion

We introduced a framework for KD, based on RL, that uses caching and proximity analysis to boost search efficiency under a fixed evaluation budget. Our experiments on CIFAR-10 and IMDb show these methods can reduce full training evaluations by 75–90 % while maintaining high accuracy. Using a `Dynamic Prox` variant, we also demonstrated how the framework can dynamically determine which search components-architecture or hyperparameters-are most critical to a given task.

Our approach is well-suited for the discrete and budget-constrained challenge of finding optimal student models, offering a clear advantage over standard methods that fail to leverage knowledge from prior trials. Our comparison with a modern evolutionary algorithm further reveals that while our RL-based efficiency mechanisms provide a reliable and substantial performance boost, the choice of the underlying search paradigm remains a critical, task-dependent consideration. Although our empirical validation is based on single experimental runs, our analysis in Section 5.3 shows the results are underpinned by structural efficiency gains that address fundamental algorithmic and operational bottlenecks, suggesting the findings are robust. The work, therefore, confirms the framework's significant potential. As outlined in our discussion, future work will focus on scaling these results via large-scale benchmarks and developing more sophisticated, performance-aware reward functions.

CRediT authorship contribution statement

Steve Bakos: Writing â original draft, Visualization, Validation, Software, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Heidar Davoudi:** Writing â review & editing, Supervision, Resources, Methodology, Funding acquisition, Conceptualization.

Data availability

Data will be made available on request.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Training stability and performance analysis

This appendix provides a comprehensive analysis of the training stability for each algorithm across all experimental conditions. This analysis provides deeper insight into the reliability and consistency of the search process, moving beyond the final performance metrics presented in the main text.

A.1. Methodology

Our analysis examines the validation accuracy trajectories of each algorithm across its entire run. Using rolling statistics computed over windows of 200 consecutive evaluations, we track the **Stability Score**, a composite metric defined as $1/(1 + \text{CV})$, where CV is the Coefficient of Variation. This provides an intuitive scale from 0 to 1, where higher scores indicate greater consistency and reliability. Other reported metrics include the average and maximum standard deviation of accuracy within the rolling window, and the total count of significant accuracy fluctuations.

A.2. Summary tables

The following tables summarize the key stability metrics for each algorithm across the four experimental conditions. Best performance in each category is highlighted in bold.

A.3. Stability and performance plots

A.3.1. CIFAR-10 – partial reward

The results for the CIFAR-10 task under the partial reward highlight significant differences in stability among the search algorithms. As summarized in Table A.1, the PPO-based methods demonstrate superior stability, with the baseline PPO achieving the highest Stability Score of 0.863.

Table A.1
Stability Summary for all algorithms on CIFAR-10 under the partial reward.

Algorithm	Stability Score	Avg Std Dev (%)	Max Std Dev (%)	Total Fluctuations
Grid	0.820	**5.250**	**10.130**	3256
Random Search	0.716	8.860	11.430	3695
Random Agent	0.743	6.250	10.060	3472
Regularized Evolution	0.700	19.640	27.420	3700
Bayesian (Single)	0.638	17.830	23.850	3728
Bayesian (Parallel)	0.738	6.820	9.820	3436
PPO	**0.863**	9.190	22.630	2981
PPO + Cache	0.857	9.530	19.470	2420
PPO + Proximity	0.850	9.990	22.120	2371
PPO + Dynamic Prox	0.854	9.840	21.500	**2250**

Table A.2
Stability Summary for all algorithms on CIFAR-10 under the complete reward.

Algorithm	Stability Score	Avg Std Dev (%)	Max Std Dev (%)	Total Fluctuations
Grid	0.820	**5.250**	**9.060**	3236
Random Search	0.716	8.980	11.690	3689
Random Agent	0.740	6.380	9.060	3483
Regularized Evolution	0.710	18.010	24.440	3718
Bayesian (Single)	0.670	15.210	18.960	3761
Bayesian (Parallel)	0.742	6.670	9.970	3363
PPO	0.846	10.340	23.470	3211
PPO + Cache	**0.872**	8.380	15.800	2635
PPO + Proximity	0.861	9.450	18.350	**2134**
PPO + Dynamic Prox	0.846	9.420	18.350	2856

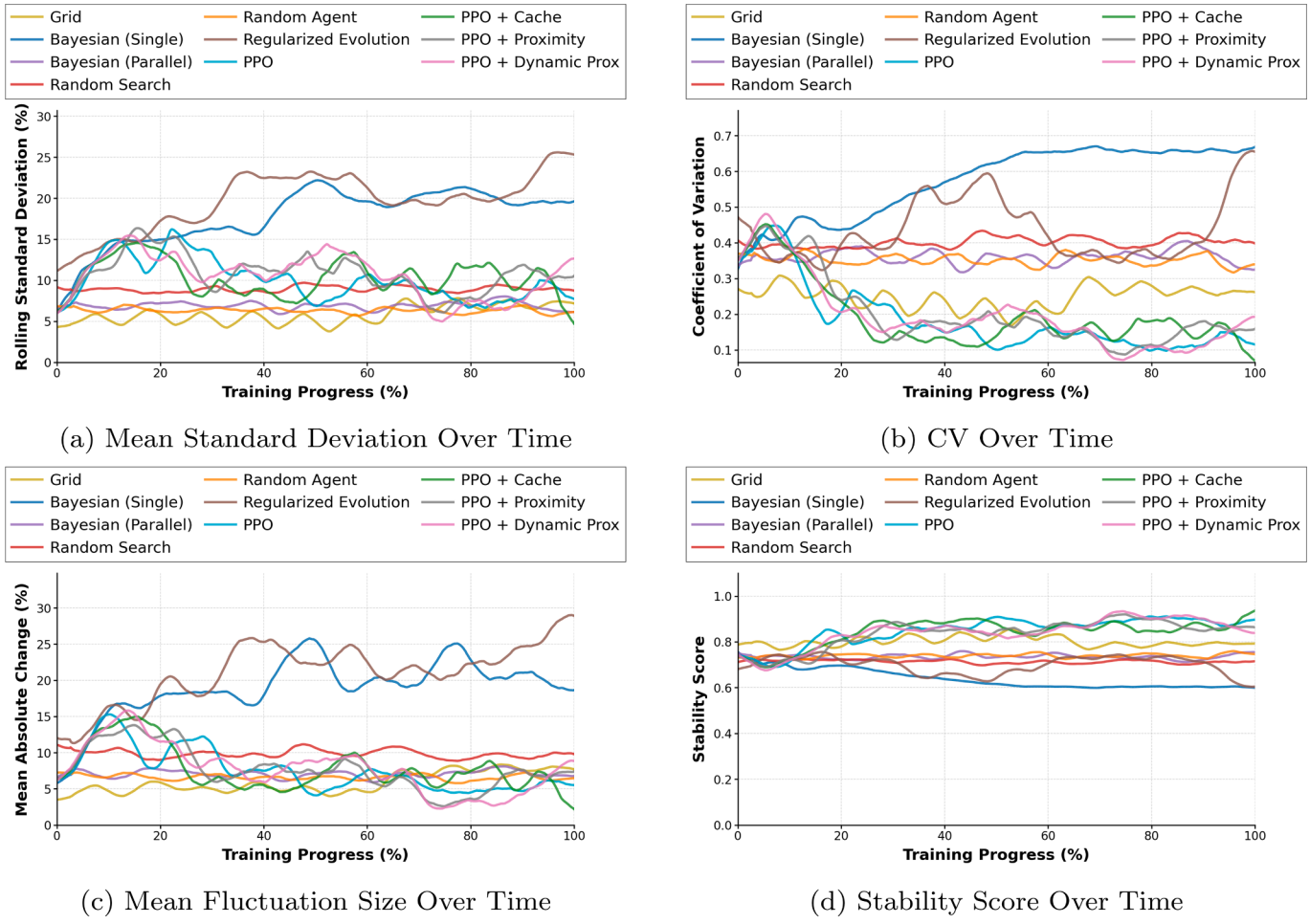
Table A.3
Stability Summary for all algorithms on IMDb under the partial reward.

Algorithm	Stability Score	Avg Std Dev (%)	Max Std Dev (%)	Total Fluctuations
Grid	0.902	8.520	12.460	2855
Random Search	0.868	11.450	14.410	3152
Random Agent	0.906	8.100	**10.790**	3149
Regularized Evolution	0.936	5.650	19.460	1556
Bayesian (Single)	**0.993**	**0.590**	15.650	**222**
Bayesian (Parallel)	0.974	2.220	13.360	1176
PPO	0.971	2.490	15.450	2151
PPO + Cache	0.987	1.140	21.230	354
PPO + Proximity	0.981	1.620	15.920	578
PPO + Dynamic Prox	0.987	1.140	14.640	283

In stark contrast, Bayesian (Single) is the most unstable with a score of 0.638. Regularized Evolution's high average standard deviation (19.640 %) indicates a highly explorative but volatile search process. The temporal plots in Fig. A.1 visualize this disparity, showing the PPO family's stability scores remaining consistently high, while the scores for Bayesian (Single) and Regularized Evolution are lower and more erratic. The rising curve of mean accuracy for the PPO variants (Fig. A.2) indicates genuine policy improvement, whereas the large gap between the mean and best performance for the more unstable methods suggests their successes were rare outliers in an otherwise inefficient search.

A.3.2. CIFAR-10 – complete reward

When the size penalty is introduced on CIFAR-10, the stability dynamics remain broadly similar. The PPO-based methods continue to show superior stability compared to the baselines (Fig. A.3). Notably, as seen in Table A.2, PPO + Cache emerges with the highest Stability Score (0.872). Regularized Evolution remains one of the more volatile methods, confirming that its search is highly explorative. The performance context in Fig. A.4 again shows the PPO agent effectively learning to propose better models over time, demonstrating its ability to navigate the more complex trade-off between accuracy and model size without losing stability.



(a) Mean Standard Deviation Over Time

(b) CV Over Time

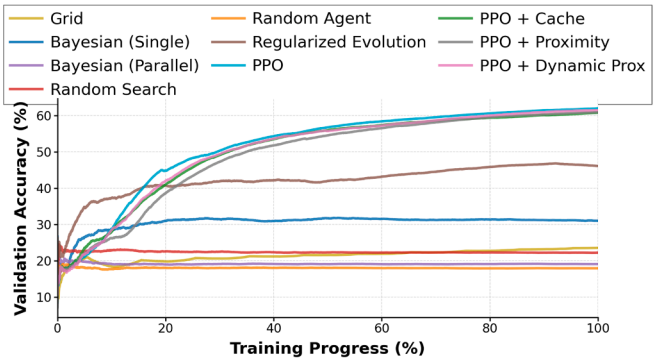
(c) Mean Fluctuation Size Over Time

(d) Stability Score Over Time

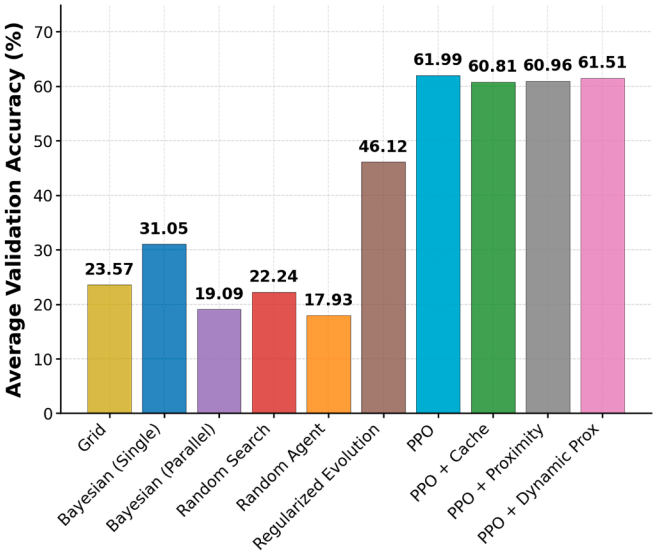
Fig. A.1. Temporal evolution of stability metrics for all algorithms on CIFAR-10 with Partial Rewards.

Table A.4
Stability Summary for all algorithms on IMDb under the complete reward.

Algorithm	Stability Score	Avg Std Dev (%)	Max Std Dev (%)	Total Fluctuations
Grid	0.897	8.950	13.780	2623
Random Search	0.864	11.820	14.970	3227
Random Agent	0.907	7.940	**10.690**	3002
Regularized Evolution	0.929	6.330	17.970	1690
Bayesian (Single)	**0.988**	**1.010**	11.340	815
Bayesian (Parallel)	0.986	1.190	16.720	1645
PPO	0.984	1.330	15.570	686
PPO + Cache	0.975	2.100	13.450	665
PPO + Proximity	0.975	2.100	14.910	543
PPO + Dynamic Prox	0.977	1.930	16.330	**461**

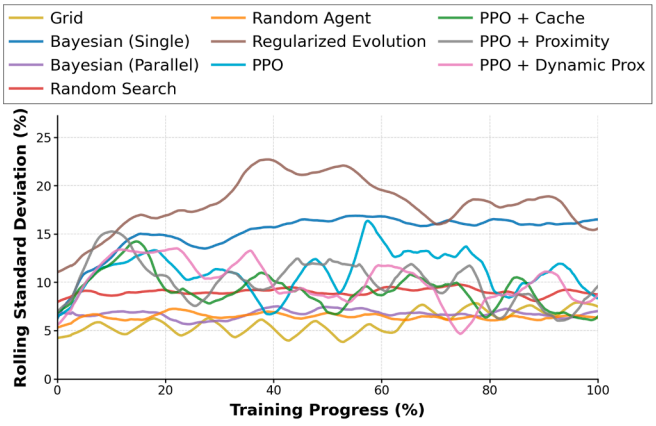


(a) Mean Accuracy Over Time

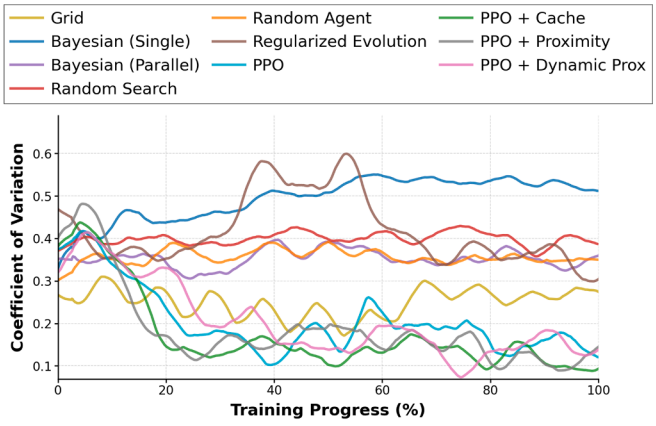


(b) Overall Mean Accuracy Comparison

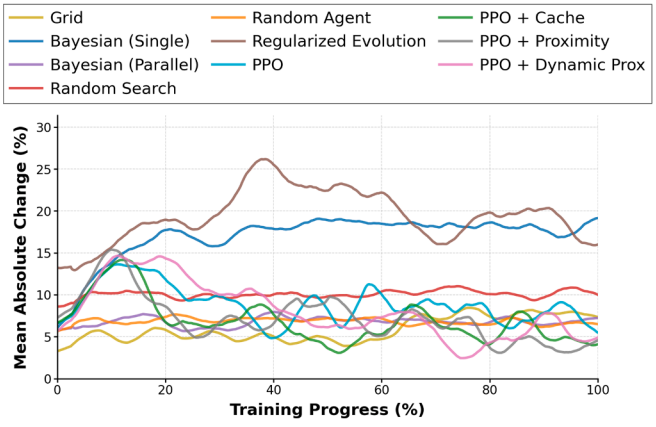
Fig. A.2. Performance Context Analysis for all algorithms on CIFAR-10 with Partial Rewards.



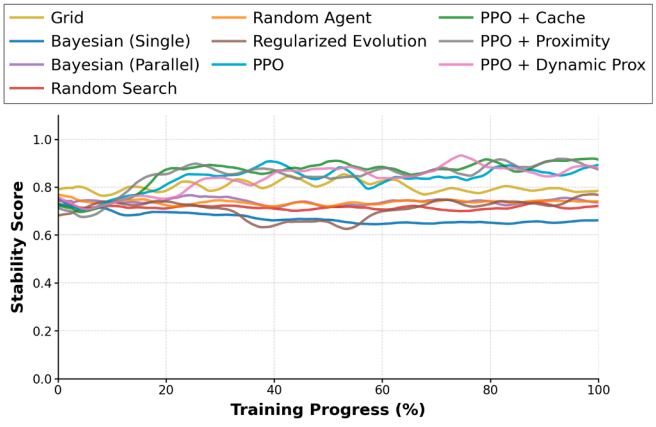
(a) Mean Standard Deviation Over Time



(b) CV Over Time

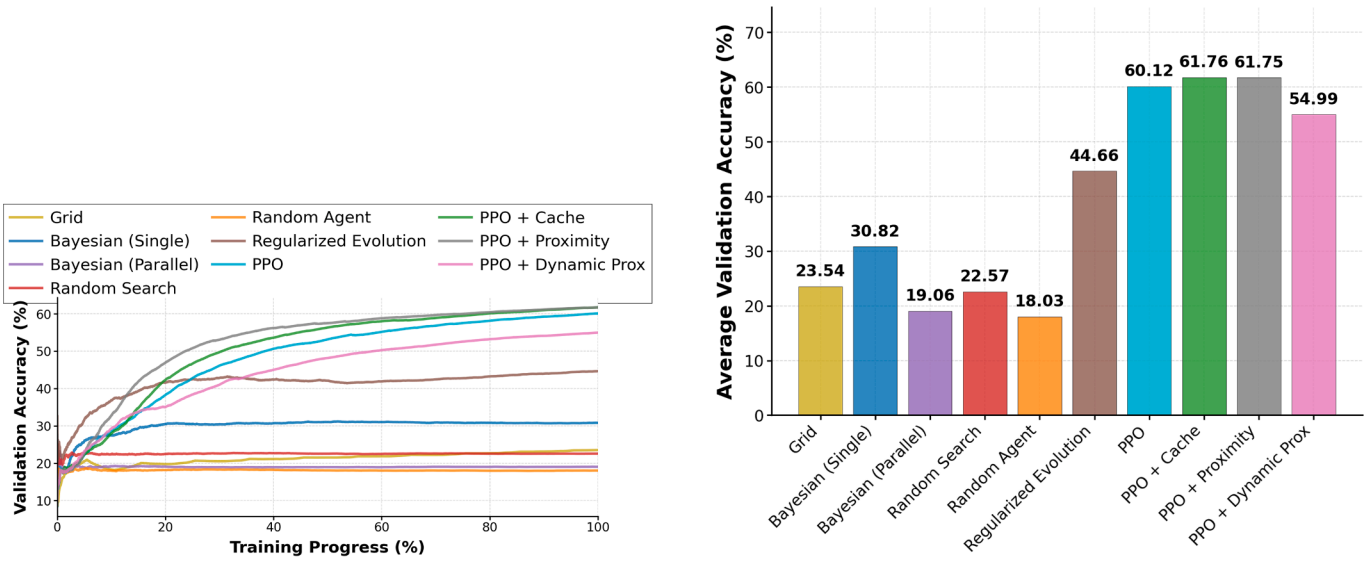


(c) Mean Fluctuation Size Over Time



(d) Stability Score Over Time

Fig. A.3. Temporal evolution of stability metrics for all algorithms on CIFAR-10 with Complete Rewards.



(a) Mean Accuracy Over Time (b) Overall Mean Accuracy Comparison

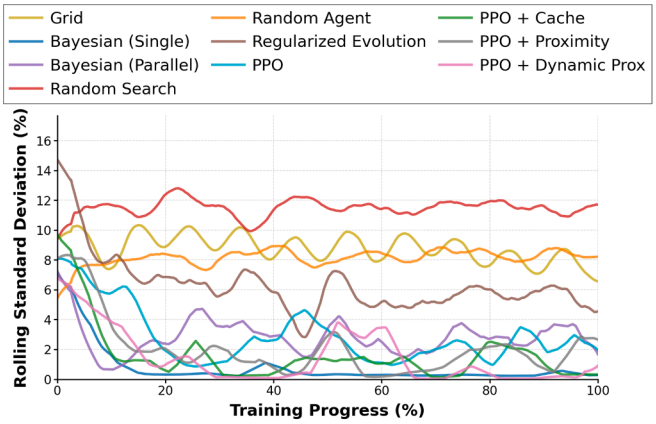
Fig. A.4. Performance Context Analysis for all algorithms on CIFAR-10 with Complete Rewards.

A.3.3. IMDb – partial reward

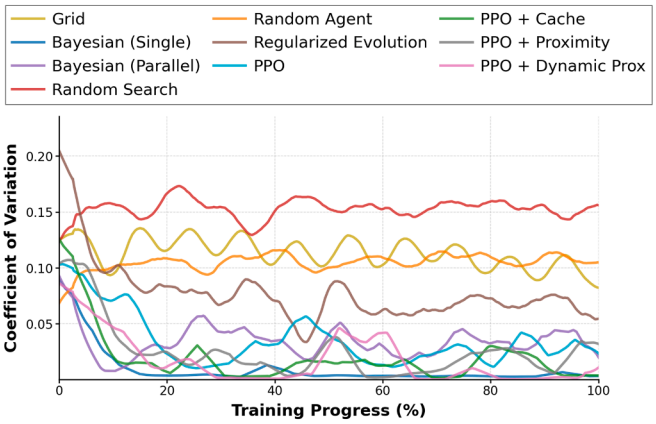
The stability analysis for the IMDb task reveals a markedly different landscape. As shown in Fig. A.5, most algorithms achieve very high stability scores quickly, indicating the search space is likely simpler. Intriguingly, Bayesian (Single) achieves the highest stability score (0.993), with its sequential nature proving advantageous here. Regularized Evolution also demonstrates strong stability with a score of 0.936, outperforming the simpler baselines like Random Search. The performance context plots in Fig. A.6 support this; the mean accuracy for most methods plateaus relatively early, suggesting good solutions are found quickly and there is less to be gained from prolonged exploration.

A.3.4. IMDb – complete reward

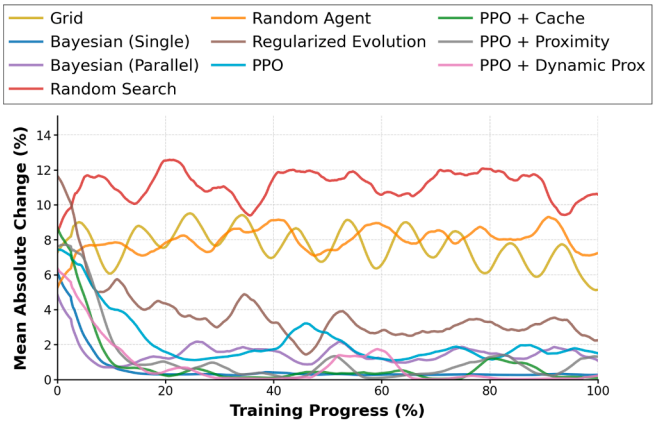
The trends observed under the partial reward for IMDb continue when the size penalty is introduced (Fig. A.7). The stability scores remain high across the board, with Bayesian (Single) again achieving the highest score (0.988), as detailed in Table A.4. The size penalty does not appear to destabilize the search processes, likely because the agents naturally gravitate towards smaller models on this task. Regularized Evolution (0.929) remains more stable than the simple baselines, and our PPO variants with caching and proximity demonstrate excellent and reliable performance, confirming their robustness across different tasks and reward structures, as seen in Fig. A.8.



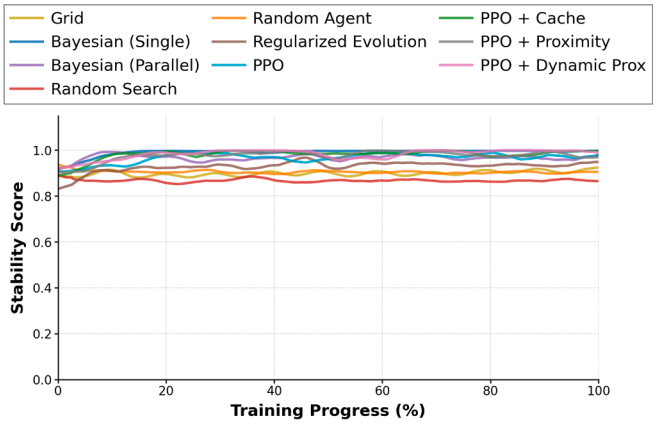
(a) Mean Standard Deviation Over Time



(b) CV Over Time

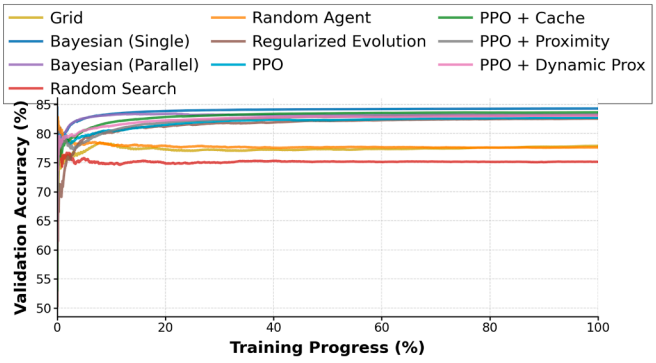


(c) Mean Fluctuation Size Over Time

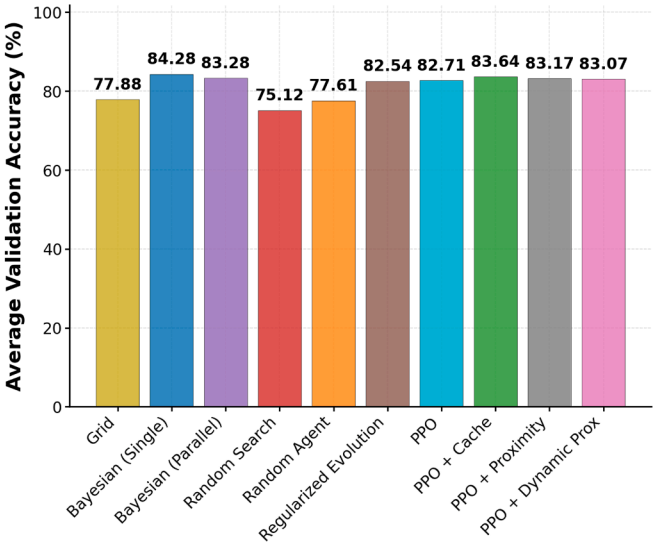


(d) Stability Score Over Time

Fig. A.5. Temporal evolution of stability metrics for all algorithms on IMDb with Partial Rewards.

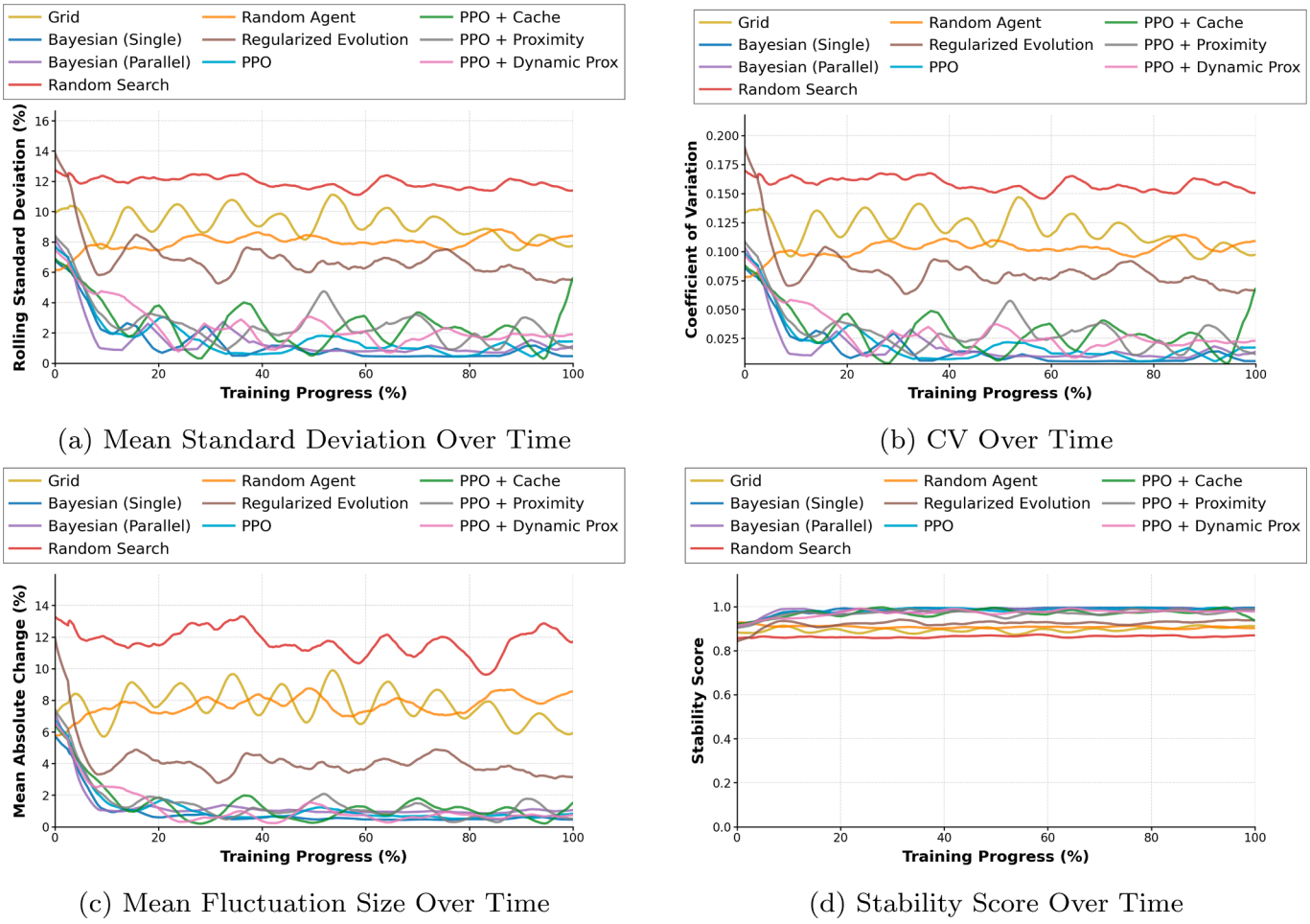


(a) Mean Accuracy Over Time



(b) Overall Mean Accuracy Comparison

Fig. A.6. Performance Context Analysis for all algorithms on IMDb with Partial Rewards.



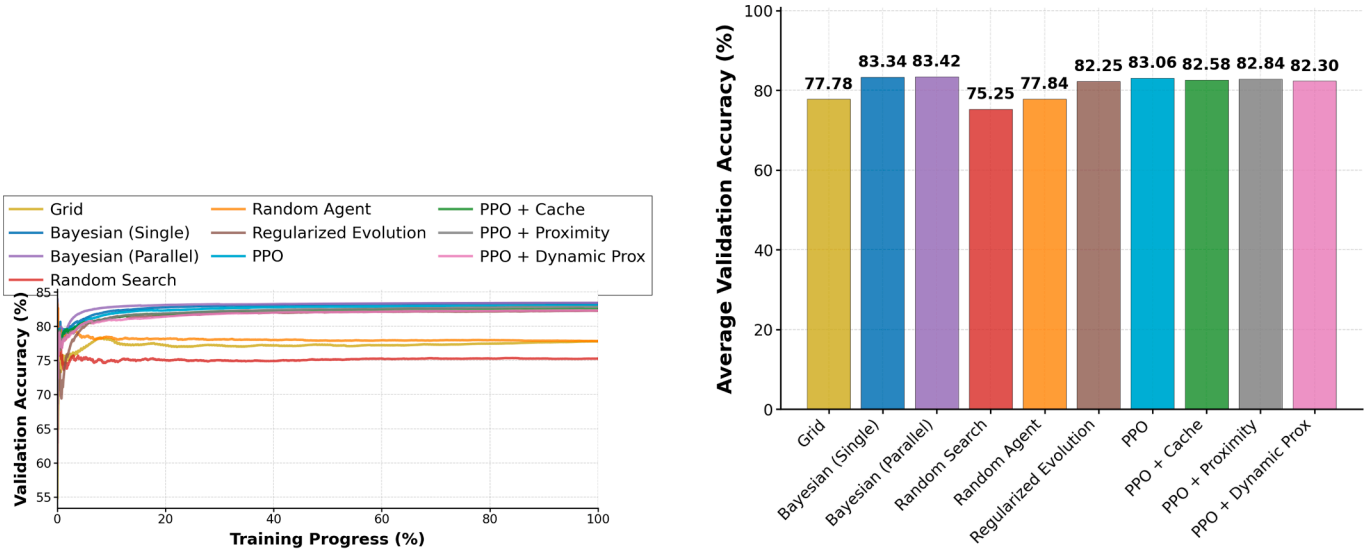
(a) Mean Standard Deviation Over Time

(b) CV Over Time

(c) Mean Fluctuation Size Over Time

(d) Stability Score Over Time

Fig. A.7. Temporal evolution of stability metrics for all algorithms on IMDb with Complete Rewards.



(a) Mean Accuracy Over Time

(b) Overall Mean Accuracy Comparison

Fig. A.8. Performance Context Analysis for all algorithms on IMDb with Complete Rewards.

Appendix B. KL divergence metrics

Table B.5
CIFAR-10 KL Divergence (Partial vs. Complete). Lower = more similar to PPO.

	Partial			Complete		
Algorithm	Params	Acc	Avg	Params	Acc	Avg
Bayesian (Parallel)	4.4746	2.5030	3.4888	4.3058	2.3869	3.3464
Random Agent	4.4156	2.5506	3.4831	4.1710	2.3979	3.2845
Grid	3.8750	2.5354	3.2052	3.5795	2.2603	2.9199
Random Search	3.2599	2.2567	2.7583	2.8867	1.9966	2.4416
Bayesian Single	1.6881	1.3541	1.5211	2.5640	1.3580	1.9610
Regularized Evolution	1.1196	0.6768	0.8982	1.4844	0.5642	1.0243
PPO + Proximity	0.0119	0.0171	0.0145	0.5868	0.0884	0.3376
PPO + Cache	0.0112	0.0130	0.0121	0.1376	0.0322	**0.0849**
PPO + Dynamic Prox	0.0065	0.0103	**0.0084**	1.6242	0.4273	1.0257

Table B.6
IMDb KL Divergence (Partial vs. Complete). Lower = more similar to PPO.

	Partial			Complete		
Algorithm	Params	Acc	Avg	Params	Acc	Avg
Random Search	1.4261	1.2757	1.3509	2.6326	2.1070	2.3698
Bayesian (Single)	1.1085	1.0950	1.1018	0.1979	0.1164	0.1571
Grid	0.5921	0.8492	0.7206	1.6330	1.6514	1.6422
Regularized Evolution	0.4419	0.2365	0.3392	0.4955	0.9026	0.6991
PPO + Cache	0.3763	0.5271	0.4517	0.5102	0.0428	0.2765
PPO + Dynamic Prox	0.2173	0.3689	0.2931	0.5740	0.3551	0.4646
Random Agent	0.1972	0.9336	0.5654	1.1145	1.6272	1.3708
Bayesian (Parallel)	0.2523	0.0723	0.1623	0.2326	0.1852	0.2089
PPO + Proximity	0.0827	0.1934	**0.1380**	0.1996	0.0396	**0.1196**

Appendix C. Parameter count analysis

Figs. C.9 (CIFAR-10) and C.10 (IMDb) illustrate parameter-count trends over time.

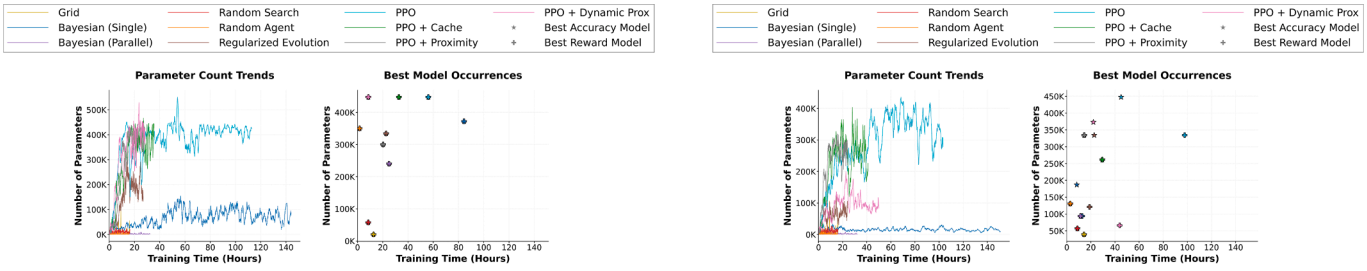


Fig. C.9. Parameter Count Trends & Best-Model Occurrences (CIFAR-10). **Left:** Partial reward; **Right:** Complete reward. Larger models help accuracy, but the size penalty can shift the optimum.

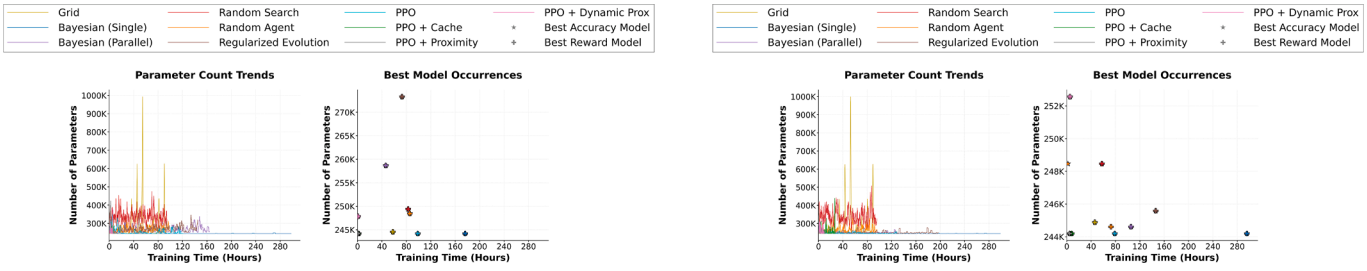


Fig. C.10. Parameter Count Trends & Best-Model Occurrences (IMDb). **Left:** Partial reward; **Right:** Complete reward. Smaller architectures dominate, so the agent seldom explores large architectures.

Appendix D. Reinforcement learning agent details

In our experiments, we used the PPO algorithm [37] from Stable Baselines3 [38]. We selected PPO due to its particular suitability for our joint NAS-HPO problem. Its on-policy, actor-critic nature is adept at handling the sparse and delayed rewards inherent to our two-phase design, where actions in the planning phase are only rewarded after the execution phase. Furthermore, PPO's clipped surrogate objective provides crucial training stability, mitigating policy divergence in the high-variance reward landscape of architecture search. This stability, combined with an entropy bonus to encourage exploration, allows the agent to effectively leverage the accelerated learning cycle enabled by our caching and proximity mechanisms to thoroughly explore the search space.

PPO is an actor-critic method that optimizes a clipped surrogate objective. Both the actor (policy) and critic (value function) networks have three hidden layers with 128 units each (ReLU activations), which suffices for our NAS/HPO tasks.

Appendix E. Dataset descriptions

CIFAR-10: 60,000 images, 10 classes, 50k train (80–20 split) + 10k test.
IMDb: 50,000 reviews, 25k train (75–25 split) + 25k test, balanced positive/negative.

Appendix F. Teacher models

MobileNetV2 [35] ($\sim$ 2.2M params) for CIFAR-10; DistilBERT [18] ($\sim$ 66M params) for IMDb.

Appendix G. Hardware configuration

Experiments ran on two identical servers:

- Intel Xeon W-2223 CPU @ 3.6 GHz (8 cores)
- 32 GB RAM
- NVIDIA RTX A4000 GPU

Appendix H. Implementation details

We used a fixed seed (42) for PPO itself, but student models remain unseeded, allowing random initial weights to capture performance variability. CIFAR-10 had up to 8 parallel student environments referencing a local MobileNetV2; IMDb had 4 parallel students referencing a centralized DistilBERT teacher across the network. Bayesian Single re-fits a GP after each iteration, whereas Bayesian Parallel requests multiple points per batch.

H.1. Image classification task

Eight parallel instances referencing MobileNetV2 locally. This eliminates teacher-network bottlenecks and scales well.

H.2. Sentiment analysis task

Four parallel student environments referencing a single, centralized DistilBERT teacher. The teacher runs on a separate system, incurring potential network overhead.

H.3. Bayesian optimization

BO can be parallelized only partially. Bayesian Single re-fits a GP sequentially each iteration; Bayesian Parallel does batched suggestions but can be less accurate or more costly in discrete/high-dimensional spaces.

Appendix I. Action space and hyperparameters

- **Layer Types**: Dense, Conv, Dropout, Pooling (up to 6 layers).
- **Units**: {8, 16, 32, 64, 128}.
- **Learning Rates**: {1e-4, 1e-3, 5e-3, 1e-2}.
- **Epochs**: {1, 2, 3, 4, 5}.

Dropout rate = (units / 256).

Appendix J. Loss function and knowledge distillation

We employed a knowledge distillation strategy that combines Cross-Entropy (CE) loss with Kullback-Leibler (KL) divergence, a standard technique proposed by Hinton et al. [1]. The core of this method involves a temperature parameter, T, which softens the probability distributions from both the teacher and student models. A higher temperature reveals the nuanced inter-class relationships learned by the teacher, creating a richer training signal. The final loss function is a weighted combination of the standard CE loss, calculated against the hard ground-truth labels, and the KL divergence loss, which aligns the student's softened predictions with the teacher's:

$$L_{\text{total}} = (1-\alpha)\,\text{CE}(y_{\text{true}}, \sigma(z_s)) + \alpha\,\text{KL}(\sigma(z_t/T)||\sigma(z_s/T)).$$

Here, z_s and z_t represent the logits from the student and teacher models, respectively, and σ is the softmax function. We set the temperature $T = 3.0$ across all experiments. This choice is guided by foundational work in knowledge distillation, which shows that intermediate temperatures are most effective when the student model is significantly smaller than the teacher. For a student with a radically smaller number of parameters, temperatures in the range of 2.5 to 4 have been demonstrated to work significantly better than higher or lower values [1]. As our experiments involve a significant capacity gap between the teacher and student models, our selection of $T = 3.0$ is consistent with this recommended range.

The weighting parameter α is crucial as it balances the student's learning between the factual ground-truth labels and the distilled knowledge from the teacher. The choice of α was tailored to the specific characteristics of our tasks. For the CIFAR-10 task, a balanced $\alpha = 0.5$ was used, reflecting a scenario where the teacher and student architectures were more comparable. In contrast, for the IMDb sentiment analysis task, we set a high $\alpha = 0.9$. This value was deliberately chosen to force the smaller student models to rely heavily on the rich, supervisory signal from the powerful DistilBERT teacher, using its "dark knowledge" as a strong regularizer [1]. To validate these choices and analyze the sensitivity of performance to these hyperparameters, we conducted a post-hoc ablation study, the results of which are detailed below.

J.1. Ablation study on KD hyperparameters

To empirically validate our chosen hyperparameters, we performed a post-hoc ablation study by conducting a grid search over a range of values for α and T. The experiment was run for both the CIFAR-10 and IMDb tasks using a representative, fixed student for each. The results of this grid search are presented in Tables J.7 and J.8.

Table J.7
Ablation results for KD hyperparameters on CIFAR-10. Values represent the best validation accuracy. The underlined value indicates the setting used in our main experiments, while the **bolded** value represents the best result in the grid.

$\alpha \setminus$ T	1	3	5	10	15
0.1	0.528	0.481	0.512	0.482	0.540
0.3	0.549	0.563	0.562	0.553	0.510
0.5	0.552	0.548	0.462	0.508	0.517
0.7	0.559	0.544	0.486	0.550	0.412
0.9	0.513	**0.565**	0.423	0.564	0.475

Table J.8
Ablation results for KD hyperparameters on IMDb. Values represent the best validation accuracy. The underlined value indicates the setting used in our main experiments, while the **bolded** value represents the best result in the grid.

$\alpha \setminus$ T	1	3	5	10	15
0.1	0.812	0.823	0.817	0.809	0.815
0.3	0.827	0.819	0.819	0.810	0.812
0.5	0.823	0.817	0.815	0.816	0.808
0.7	0.824	0.822	0.818	0.815	0.816
0.9	0.821	**0.828**	0.823	0.817	0.819

J.1.1. Sensitivity Analysis and Interpretation

The ablation study provides insight into the sensitivity of our models to the knowledge distillation hyperparameters. For the CIFAR-10 task, the main experiments used $\alpha = 0.5$ and $T = 3$, resulting in an accuracy of 0.548. The ablation study confirmed this was a strong choice, even though the optimal result in the grid was 0.565 (at $\alpha = 0.9, T = 3$). With a standard deviation of 0.043 across all configurations, the results indicate that the distillation process is not overly sensitive to the exact choice of α and T. This stability suggests that our selected hyperparameters provided a robust and effective baseline for the student search.

Similarly, for the IMDb task, our main experiments used $\alpha = 0.1$ and $T = 3$, which yielded an accuracy of 0.823. Although a slightly higher accuracy of 0.828 was found at $\alpha = 0.9, T = 3$, the performance across the hyperparameter grid was remarkably stable. The standard deviation across all 25 configurations was only 0.005, confirming the robustness of our chosen parameters and suggesting that the model is not highly sensitive to these specific settings.

Appendix K. Additional results: explored configurations

Figs. K.11–K.14 present the *partial-reward* distributions discussed in Sections 4.2.1–4.2.2, while Figs. K.15–K.18 show the new *complete-reward* counterparts. Across both rewards, CIFAR-10 favours larger architectures (wider parameter spread), whereas IMDb quickly saturates with compact students. Crucially, caching & proximity skip only duplicate or near-duplicate trials-they do *not* impede exploration of the broader search space.

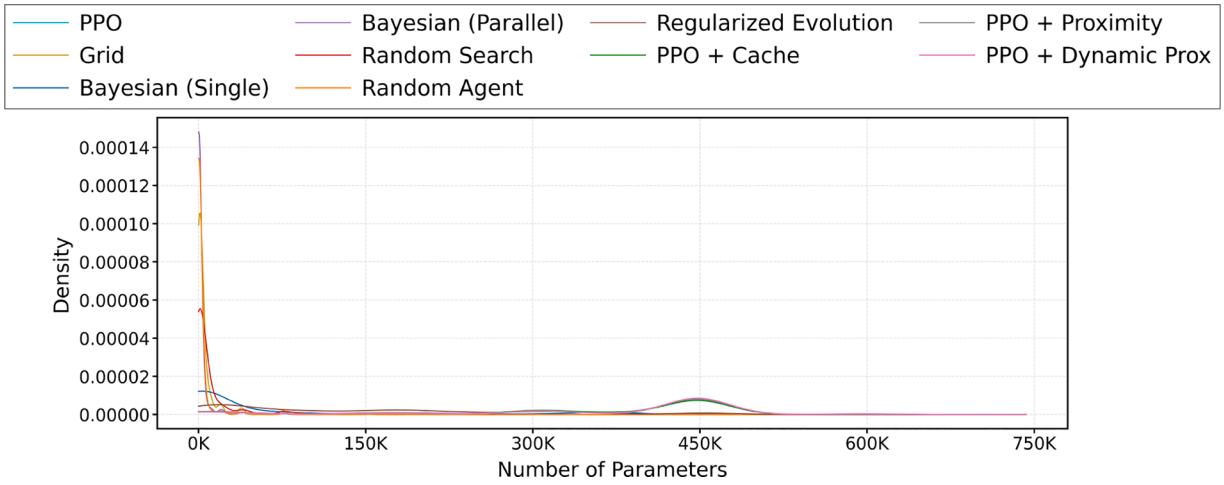


Fig. K.11. CIFAR-10 (Partial): parameter distribution.

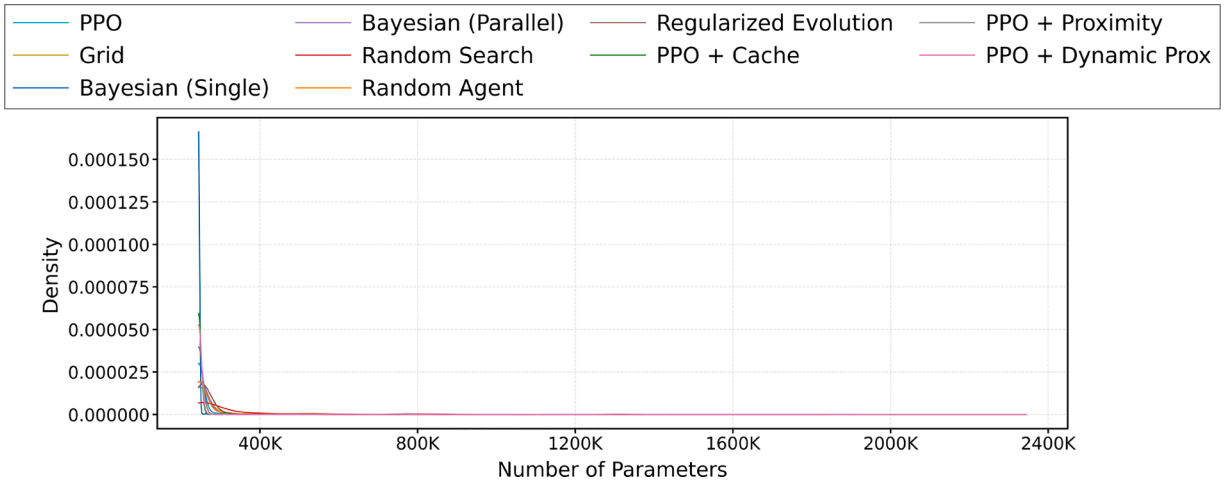


Fig. K.12. IMDb (Partial): parameter distribution.

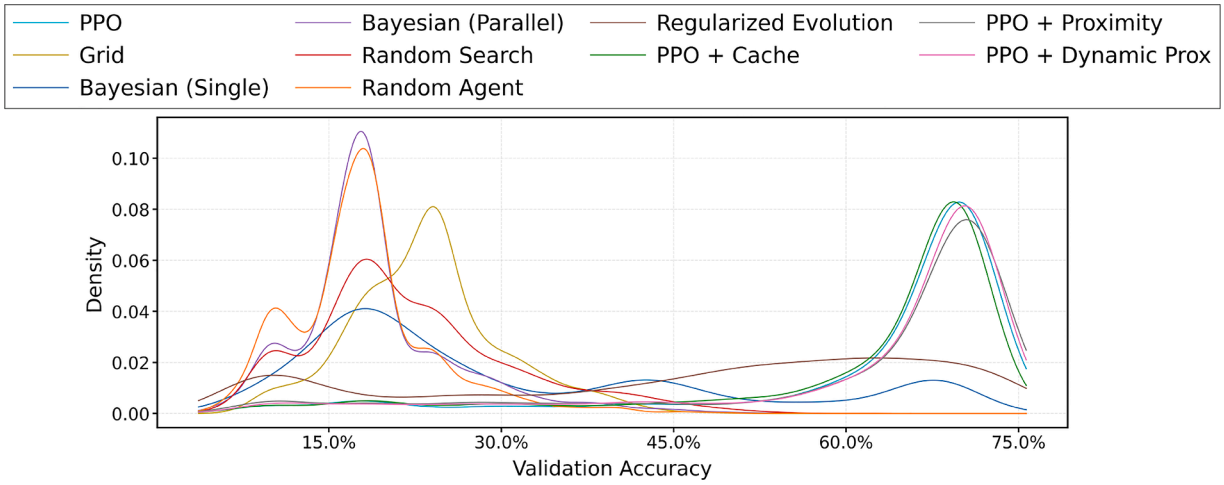


Fig. K.13. CIFAR-10 (partial): accuracy distribution.

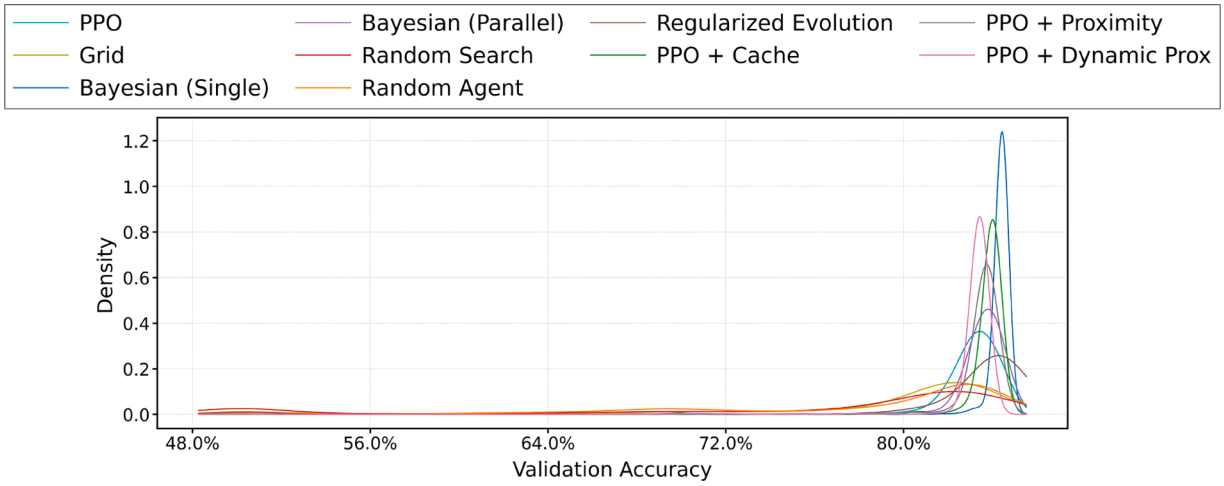


Fig. K.14. IMDb (partial): accuracy distribution.

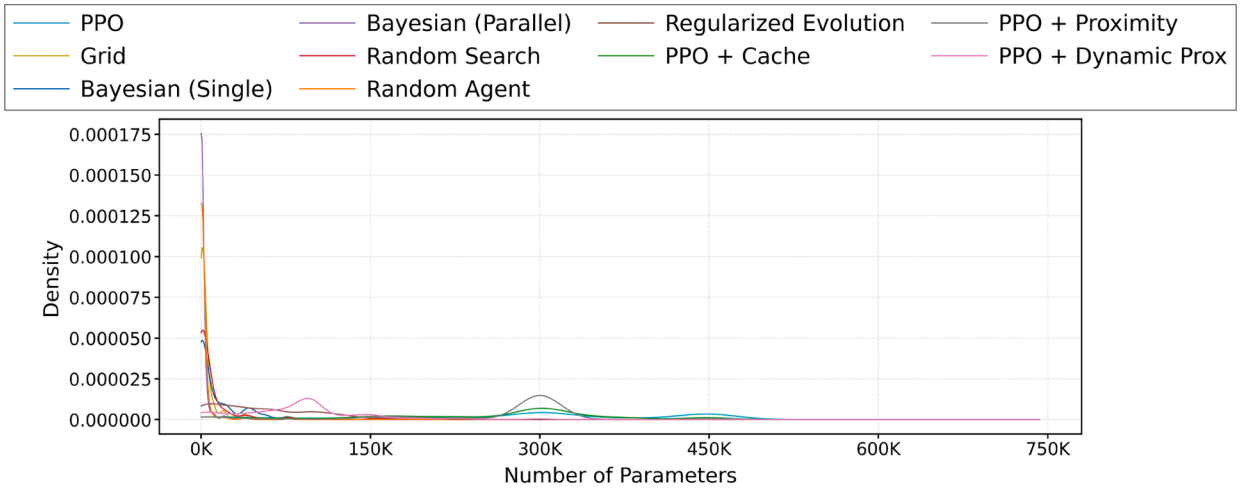


Fig. K.15. CIFAR-10 (complete): parameter distribution.

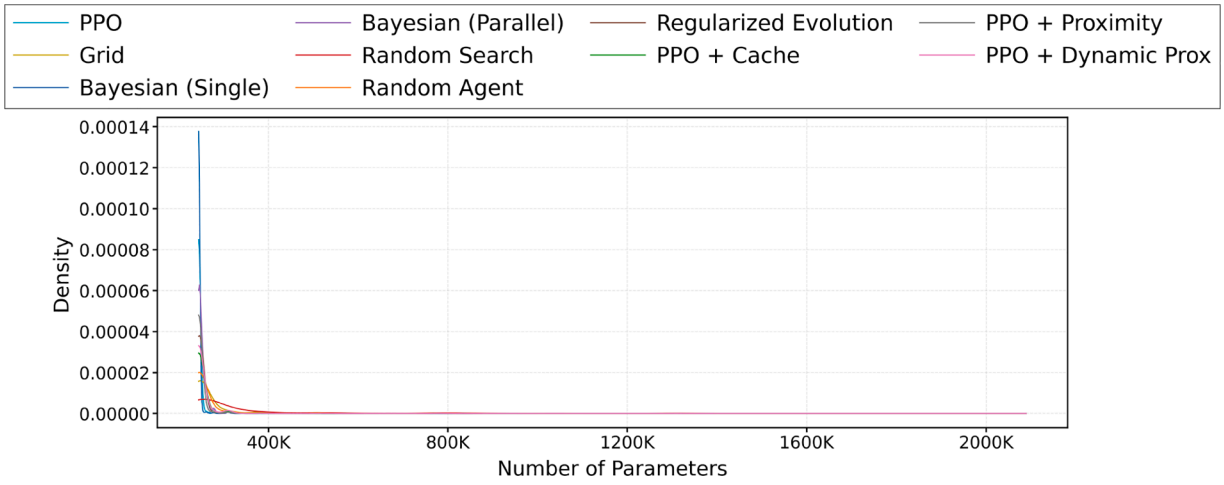


Fig. K.16. IMDb (complete): parameter distribution.

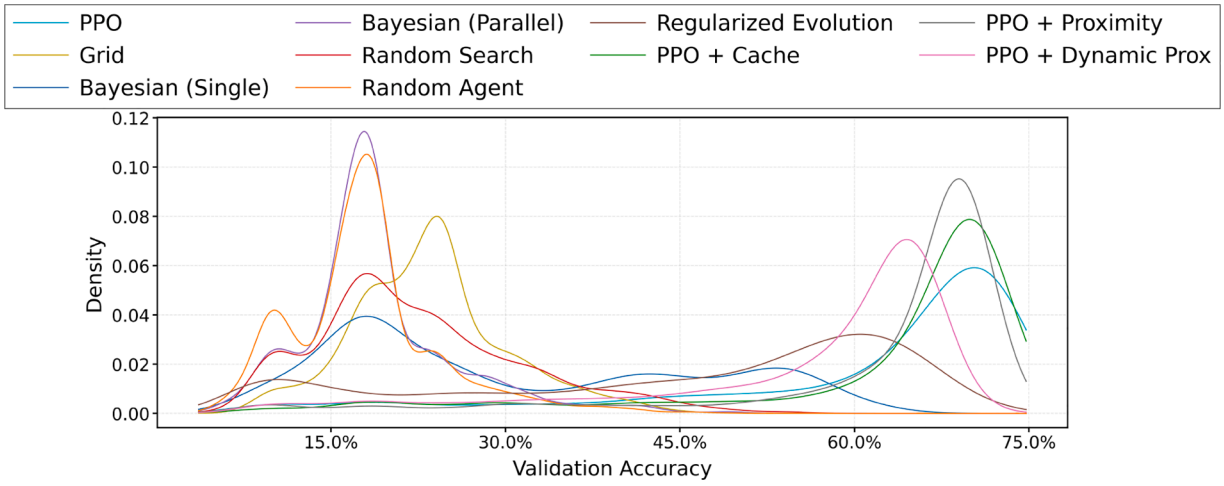


Fig. K.17. CIFAR-10 (complete): accuracy distribution.

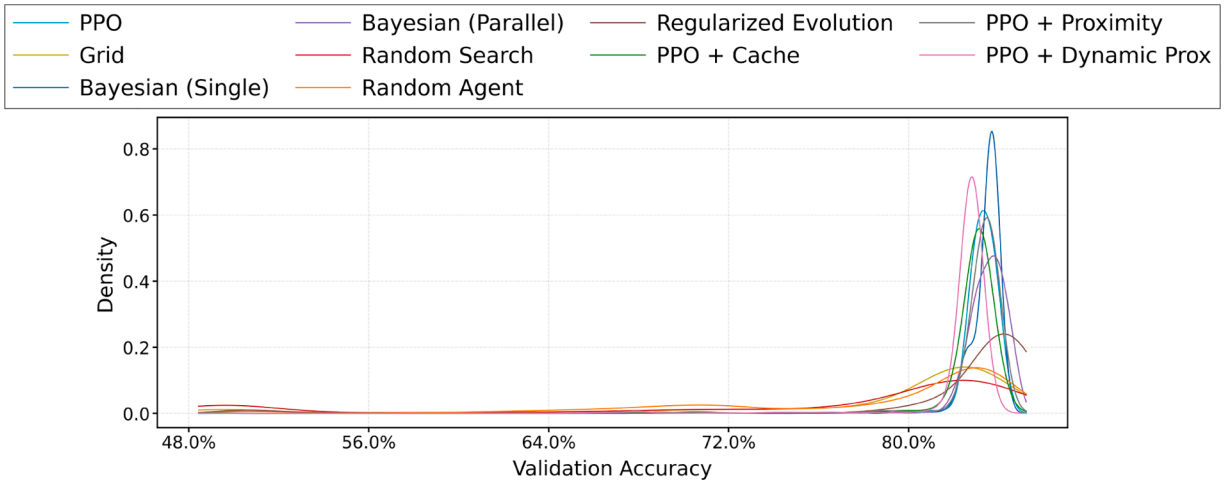


Fig. K.18. IMDb (complete): accuracy distribution.

Appendix L. Dynamic proximity feature weights

Below, we show how `PPO + Dynamic Prox`'s Random Forest regressor adapts feature weights over time for **CIFAR-10** (Fig. L.19) and **IMDb** (Fig. L.20).

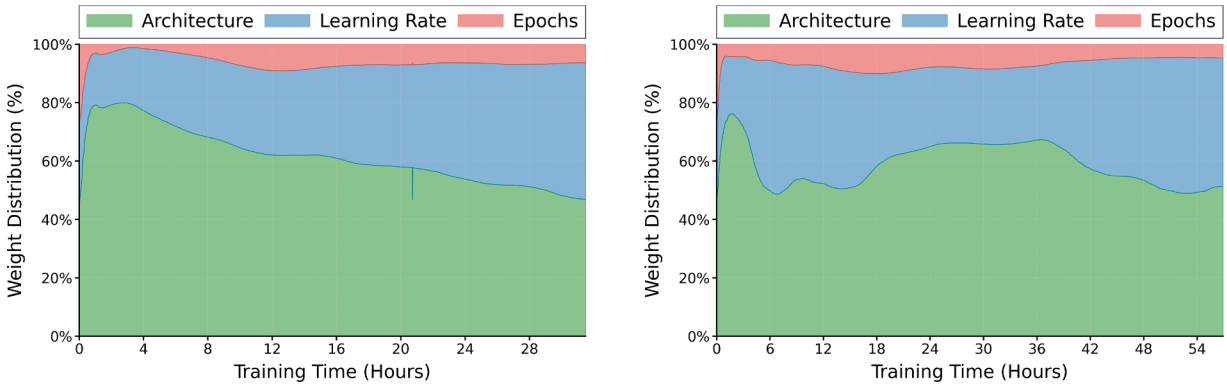


Fig. L.19. Feature Weight Distribution Over Time (%): **CIFAR-10, Dynamic Prox**. **Left**: Partial reward; **Right**: Complete reward.

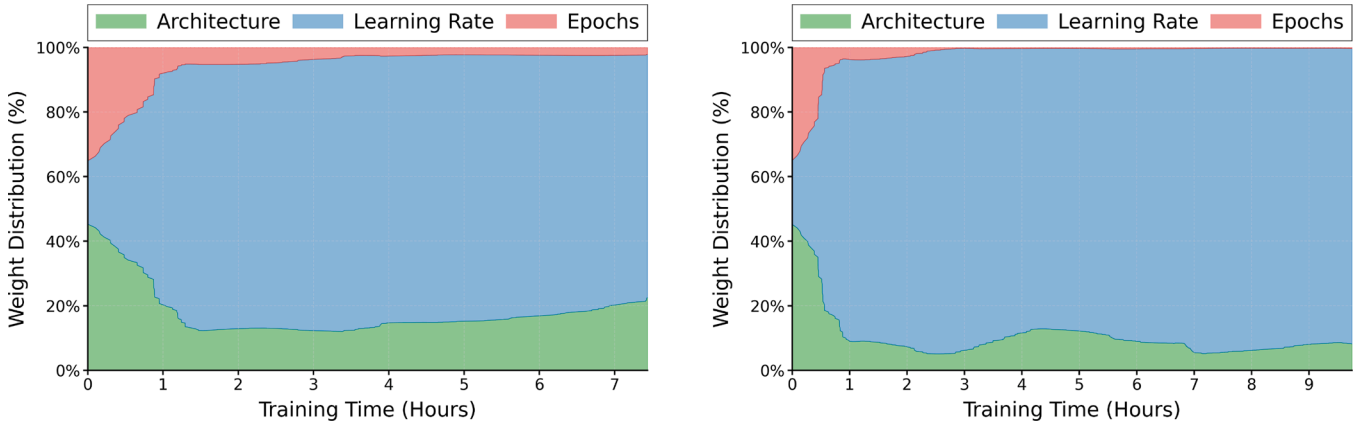


Fig. L.20. Feature Weight Distribution Over Time (%): **IMDb, Dynamic Prox**. **Left**: Partial reward; **Right**: Complete reward.

For CIFAR-10, architecture eventually dominates under both rewards, reflecting the impact of deeper/wider nets. Learning rate remains moderate, while epochs generally fade. For IMDb, small models work well, so learning rate becomes paramount, with architecture having a smaller effect. These observations mirror Sections 4.2.1–4.2.2.

Appendix M. Ablation study on reward function design

To empirically validate our choice of an exponential reward function, we conducted an ablation study. Here, we compare the standard **linear reward function** proposed in the original NAS framework [2] against several **non-linear transformations** (quadratic and our exponential function). Non-linear rewards are an established technique for encouraging an agent to fine-tune high-performing candidates [32]. The study was run using the PPO agent on the CIFAR-10 task for 24 hours for both the partial and complete reward schemes.

M.1. Reward formulations

In this study, we compare three distinct reward functions. For each function, we define both a *partial* variant based only on the student model's validation accuracy ($A \in [0, 1]$) and a *complete* variant that also incorporates the normalized size penalty (P), as defined in Section 3.3. The formulations are as follows:

- **Linear Reward:** This is the baseline function, which scales the reward linearly with accuracy. The complete version subtracts a linear penalty.

$$R = \begin{cases} 10 \cdot A & \text{if partial} \\ 10 \cdot A - 10 \cdot P & \text{if complete} \end{cases}$$

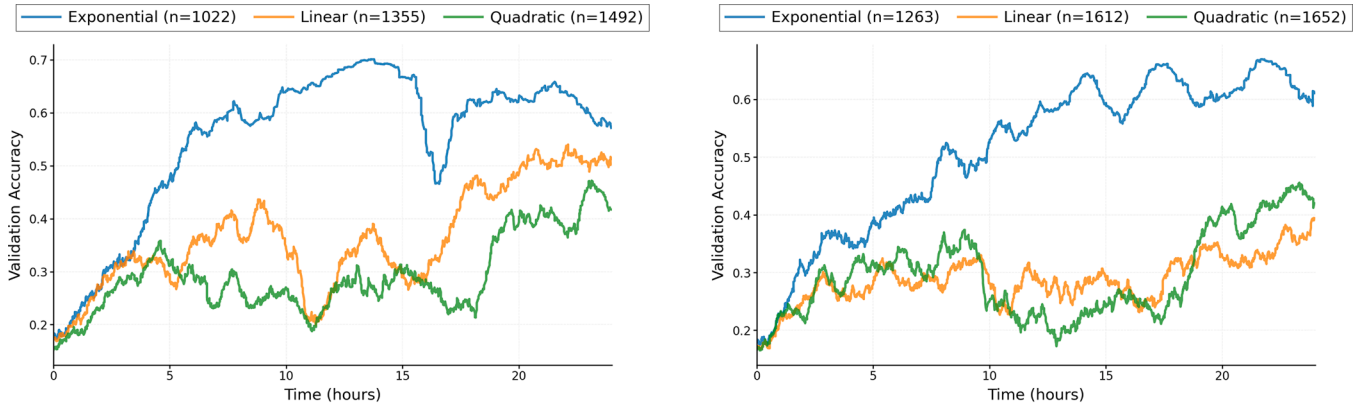
- **Quadratic Reward:** This function squares the accuracy term to more strongly incentivize improvements at higher accuracy levels. For the complete version, the penalty term is also squared.

$$R = \begin{cases} 10 \cdot A^2 & \text{if partial} \\ 10 \cdot A^2 - 10 \cdot P^2 & \text{if complete} \end{cases}$$

- **Exponential Reward:** This is the function used in our main experiments, which applies a steep exponential curve to both the accuracy and penalty terms.

$$R = \begin{cases} 10^A & \text{if partial} \\ 10^A - 10^P & \text{if complete} \end{cases}$$

The results, shown in Fig. M.21, demonstrate the clear superiority of the exponential reward function. In both experimental settings, the agent guided by the exponential reward consistently discovered architectures with significantly higher validation accuracy than the agents guided by linear or quadratic rewards. This suggests that the steeper reward gradient provided by the exponential function is highly effective at guiding the agent's early-stage learning, allowing it to more rapidly escape regions of low performance and exploit promising architectural motifs. These findings provide strong empirical support for the reward function design used in our main experiments.



(a) IMG-Partial Reward Scheme (b) IMG-Complete Reward Scheme

Fig. M.21. Comparison of PPO agent performance on the CIFAR-10 task over 24 hours using different reward function transformations. The validation accuracy is smoothed using a rolling average with a window of 50 steps. The legend indicates the total number of models (n) evaluated by each agent within the 24-hour period.

Appendix N. Multi-seed analysis for statistical robustness

To rigorously evaluate the statistical robustness of our findings, we conducted a multi-seed analysis on our most efficient method, `PPO + Dynamic Prox`, for the NLP partial reward task. We performed five full runs of the experiment, each with a different random seed, over the full 4000 model budget. This analysis examines the consistency of the final performance metrics, the behavior of the efficiency mechanisms over time, and the stability of the learned search strategy.

N.1. Performance and runtime consistency

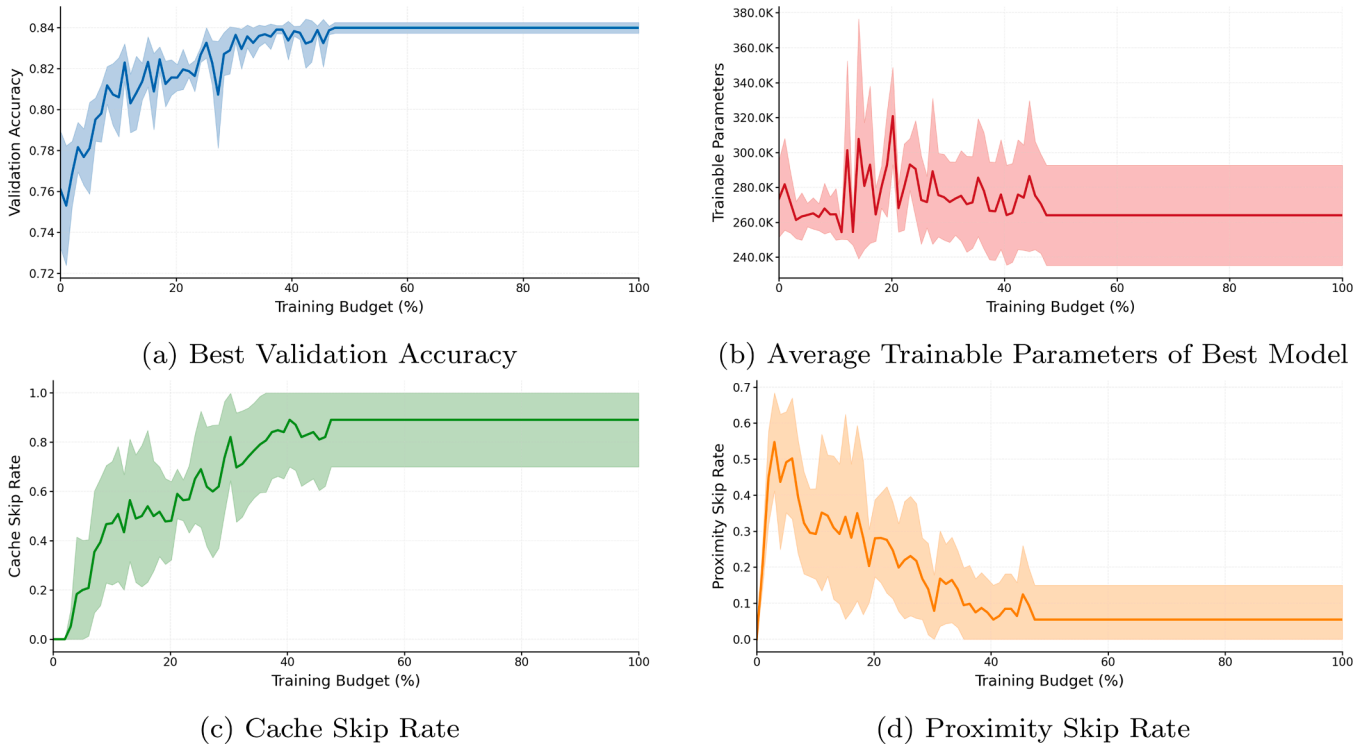
The aggregated performance results, summarized in Table N.9, demonstrate the reliability of our approach. Most critically, the best validation accuracy achieved is highly consistent across all five runs, with a mean of 84.76 % and a tight 95 % confidence interval of ±0.12 %. This confirms that the high performance reported in our main results is a reproducible outcome of our framework's design. As expected, the total runtime showed a higher variance, with a mean of 8.24 hours and a 95 % confidence interval of ±2.40 hours, reflecting the stochastic nature of the search path taken in each run.

N.2. Stability of efficiency mechanisms

The temporal evolution of key metrics, visualized in Fig. N.22, provides insight into the behavior of our efficiency mechanisms. The cache skip rate (Fig. N.22c) climbs steadily as the search progresses, handling over 80 % of evaluations by the 30 % budget mark. Conversely, the proximity skip rate (Fig. N.22d) is most active during the initial exploration phase (before the 30 % budget mark) when the cache is still sparse. Once the cache is well-populated, proximity-based skips become less frequent. Together, these mechanisms are highly effective, with the framework ultimately skipping a full model evaluation over 95 % of the time on average (Table N.9). The consistency of these trends across the five seeds demonstrates that this two-phase exploration-exploitation behavior is a stable and inherent property of the framework.

Table N.9
Aggregated statistics for the 5-seed run of `PPO + Dynamic Prox` on the NLP partial reward task.

Metric	Mean ± 95 % CI
Best Validation Accuracy (%)	84.76 ± 0.12
Time to Complete (hours)	8.24 ± 2.40
Average Cache Skip Rate	0.8397 ± 0.0759
Average Proximity Skip Rate	0.1128 ± 0.0630



(a) Best Validation Accuracy

(b) Average Trainable Parameters of Best Model

(c) Cache Skip Rate

(d) Proximity Skip Rate

Fig. N.22. Temporal evolution of key metrics for the `PPO + Dynamic Prox` method on the NLP partial reward task. The solid line represents the mean across 5 seeds, and the shaded area represents the 95 % confidence interval.

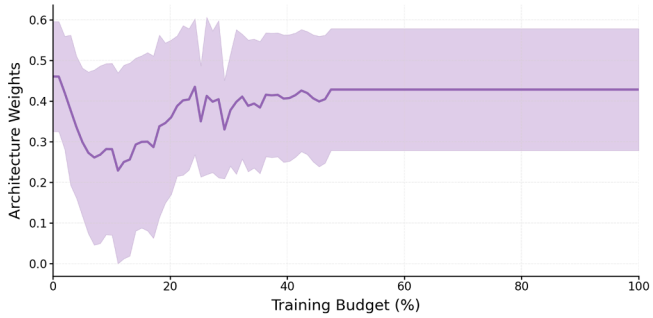
Table N.10
Aggregated feature weight statistics for the 5-seed run of `PPO + Dynamic Prox` on the NLP partial reward task.

Feature	Mean Weight ± 95 % CI
Architecture	0.3359 ± 0.2621
Learning Rate	0.6236 ± 0.2663
Number of Epochs	0.0406 ± 0.0338

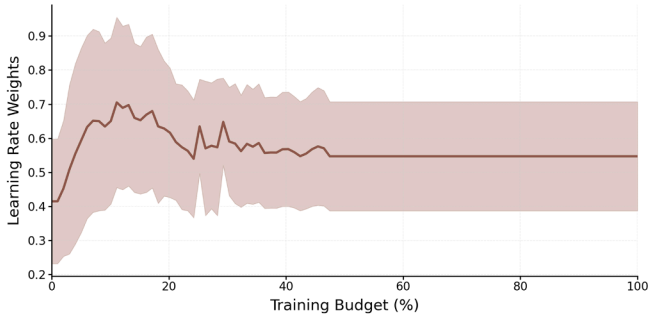
N.3. Consistency of the learned search strategy

We also examined the behavior of the Random Forest regressor across the five seeded runs to analyze the stability of the learned feature importances. As both the PPO agent and the regressor were initialized with the same seed in each run, this analysis provides insight into the consistency of the strategy that guides the proximity analysis.

The aggregated results are summarized in Table N.10 and visualized in Fig. N.23. The findings are consistent with the analysis in our main paper for the IMDb task: the agent consistently learns that **Learning Rate** is the most dominant feature, assigning it an average weight of 62.36 %. **Architecture** retains secondary importance at 33.59 %, while the **Number of Epochs** is correctly identified as the least influential factor. While the confidence intervals for the feature weights are wide during the initial exploration phase, reflecting the high variance of the early search, they narrow as the search progresses. This demonstrates that the learned search strategy becomes stable and reproducible as more data is collected.



(a) Architecture Weights



(b) Learning Rate Weights



(c) Number of Epochs Weights

Fig. N.23. Temporal evolution of feature importance weights for the `PPO + Dynamic Prox` method on the NLP partial reward task. The solid line represents the mean across 5 seeds, and the shaded area represents the 95 % confidence interval.

References

[1] G.E. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, arXiv preprint arXiv: 1503.02531, 2015.

[2] B. Zoph, V.Le Quoc, Neural architecture search with reinforcement learning, in: 5th International Conference on Learning Representations, ICLR, 2017.

[3] T. Elsken, J.H. Metzen, F. Hutter, Neural architecture search: a survey, J. Mach. Learn. Res. 20 (2019) 21.

[4] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 (2012) 281–305.

[5] S. Falkner, A. Klein, F. Hutter, Bohb: robust and efficient hyperparameter optimization at scale, Proceedings of the 35th International Conference on Machine Learning, ICML 2018 80 (2018) 1436–1445.

[6] Y. Hirose, N. Yoshinari, S. Shirakawa, NAS-HPO-Bench-II: a benchmark dataset on joint optimization of convolutional neural network architecture and training hyperparameters, Asian Conference on Machine Learning, ACML 2021 157 (2021) 1349–1364.

[7] A. Bansal, D. Stoll, M. Janowski, A. Zela, F. Hutter, Jahs-bench-201: a foundation for research on joint architecture and hyperparameter search, Adv. Neural Inf. Process. Syst. 35 (2022) 38788–38802.

[8] A. Zela, A. Klein, S. Falkner, F. Hutter, Towards automated deep learning: Efficient joint neural architecture and hyperparameter search, arXiv preprint arXiv: 1807.06906, 2018.

[9] J. Guerrero-Viu, S. Hauns, S. Izquierdo, G. Miotto, S. Schrodi, A. Biedenkapp, T. Elsken, D. Deng, M. Lindauer, F. Hutter, Bag of baselines for multi-objective joint neural architecture search and hyperparameter optimization, arXiv preprint arXiv: 2105.01015, 2021.

[10] H. Liu, K. Simonyan, Y. Yang, Darts: differentiable architecture search, in: 7th International Conference on Learning Representations, ICLR 2019.

[11] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, H. Xiong, Pc-darts: partial channel connections for memory-efficient architecture search, in: 8th International Conference on Learning Representations, ICLR 2020, 2020.

[12] X. Chen, L. Xie, J. Wu, Q. Tian, Progressive differentiable architecture search: Bridging the depth gap between search and evaluation, in: International Conference on Computer Vision, ICCV 2019, 2019, pp. 1294–1303.

[13] D. Hao, S. Pei, Mig-darts: towards effective differentiable architecture search by gradually mitigating the initial-channel gap between search and evaluation, Neural Comput. Appl 37(8) (2025) 6085–6096.

[14] A. Romero, N. Ballas, S.E. Kahou, A. Chassang, C. Gatta, Y. Bengio, Fitnets: hints for thin deep nets, in: 3rd International Conference on Learning Representations, ICLR 2015.

[15] S. Zagoruyko, N. Komodakis, Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer, in: 5th International Conference on Learning Representations, ICLR 2017.

[16] J. Gou, B. Yu, S.J. Maybank, D. Tao, Knowledge distillation: a survey, Int. J. Comput. Vis. 129(6) (2021) 1789–1819.

[17] R. Tang, Y. Lu, L. Liu, L. Mou, O. Vechtomova, J. Lin, Distilling task-specific knowledge from BERT into simple neural networks, arXiv preprint arXiv: 1903.12136, 2019.

[18] V. Sanh, L. Debut, J. Chaumond, T. Wolf, Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter, arXiv preprint arXiv: 1910.01108, 2019.

[19] E. Real, A. Aggarwal, Y. Huang, V. Quoc, Le, Regularized evolution for image classifier architecture search, in: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, 2019, pp. 4780–4789.

[20] H. Pham, M.Y. Guan, B. Zoph, Q.V. Le, J. Dean, Efficient neural architecture search via parameter sharing, Proceedings of the 35th International Conference on Machine Learning, ICML 2018 80 (2018) 4092–4101.

[21] H. Cai, L. Zhu, S. Han, Proxylessnas: direct neural architecture search on target task and hardware, in: 7th International Conference on Learning Representations, ICLR 2019.

[22] Y. Xue, X. Han, F. Neri, J. Qin, D. Pelusi, A gradient-guided evolutionary neural architecture search, IEEE Trans. Neural Netw. Learn. Syst. 36(3) (2025) 4345–4357.

[23] Y. Xue, W. Tong, F. Neri, P. Chen, T. Luo, L. Zhen, X. Wang, Evolutionary architecture search for generative adversarial networks based on weight sharing, IEEE Trans. Evol. Comput. 28(3) (2024) 653–667.

[24] J. Snoek, H. Larochelle, R.P. Adams, Practical bayesian optimization of machine learning algorithms, Adv. Neural Inf. Process. Syst. 25 (2012) 2960–2968.

[25] L. Li, K.G. Jamieson, G. Desalvo, A. Rostamizadeh, A. Talwalkar, Hyperband: a novel bandit-based approach to hyperparameter optimization, J. Mach. Learn. Res. 18 (2017) 52.

[26] A. Klein, F. Hutter, Tabular benchmarks for joint architecture and hyperparameter optimization, arXiv preprint arXiv: 1905.04970, 2019.

[27] X. Wu, D. Zhang, M. Zhang, C. Guo, B. Yang, C.S. Jensen, Joint neural architecture and hyperparameter search for correlated time series forecasting, CoRR, abs/2211.16126, 2022.

[28] G. Chen, W. Choi, X. Yu, T.X. Han, M. Chandraker, Learning efficient object detection models with knowledge distillation, In Advances in Neural Information Processing Systems 30, 2017, pp. 742–751.

[29] P. Dong, L. Li, Z. Wei, Diswot: student architecture search for distillation without training, in: Conference on Computer Vision and Pattern Recognition, CVPR 2023, 2023, pp. 11898–11908.

[30] C. Han, Q. Wang, S.A. Dianat, M. Rabbani, M. Raghuveer, Y. Rao, Q. Fang, L. Guan, D. Huang, Liu, Amd: automatic multi-step distillation of large-scale vision models, Computer Vision - ECCV 2024 15123 (2024) 431–450.

[31] M. Jaderberg, V. Dalibard, S. Osindero, W.M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, K. Kavukcuoglu, Population based training of neural networks, arXiv preprint arXiv: 1711.09846, 2017.

[32] B. Zoph, V. Vasudevan, J. Shlens, V. Quoc, Le, Learning transferable architectures for scalable image recognition, in: Conference on Computer Vision and Pattern Recognition, 2018, pp. 8697–8710. CVPR 2018.

[33] K. Tin, Ho, Random decision forests, in: Third International Conference on Document Analysis and Recognition, ICDAR 1995, 1995, pp. 278–282.

[34] A. Krizhevsky, Learning multiple layers of features from tiny images, Technical Report, University of Toronto, 2012.

[35] M. Sandler, A.G. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: inverted residuals and linear bottlenecks, in: Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520. CVPR 2018.

[36] A.L. Maas, R.E. Daly, P.T. Pham, D. Huang, A.Y. Ng, C. Potts, Learning word vectors for sentiment analysis, in: The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, 2011, pp. 142–150.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv: 1707.06347, 2017.

[38] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, N. Dormann, Stable-baselines3: reliable reinforcement learning implementations, J. Mach. Learn. Res. 22(8) (2021) 1–268.