One-Shot Hyper-Gradient Warm-Starts for Bandit-Style Hyperparameter Optimisation

Anonymous authors
Paper under double-blind review

Abstract

Bandit-style multi-fidelity schedulers such as ASHA and PASHA are the work-horses of practical hyperparameter optimisation, yet they still waste substantial compute on configurations that could have been flagged as poor before real training even begins. The root cause is that every trial is treated as a black box: none of the gradients already computed inside the training loop are exploited by the scheduler. We close this gap with One-Shot Hyper-Gradient Warm-Starts (OHGW). For each freshly sampled configuration we run exactly one mini-batch, obtain stochastic hyper-gradients $\partial L/\partial\psi$ for all continuous hyperparameters at almost zero extra cost via automatic differentiation, apply a single tiny update $\psi \leftarrow \psi - \eta_h\, \partial L/\partial\psi$, and hand the nudged configuration back to the unmodified scheduler. OHGW therefore preserves exploration while biasing every candidate toward lower-loss regions at negligible overhead and with no change to promotion or stopping logic. On CIFAR-10 with ResNet-20 under ASHA and on WikiText-103 with GPT2-small under PASHA, OHGW cuts median wall-clock time to a preset quality threshold by roughly twenty percent, adds under four percent floating-point operations, and leaves final accuracy and perplexity unchanged. Random perturbations provide almost no benefit and taking more than one hyper-step shows diminishing returns. These findings demonstrate that a single noisy hyper-gradient obtained before expensive training commences can reclaim a significant share of wasted computation in grey-box hyperparameter optimisation.

1 Introduction

1.1 Motivation and gap

Hyperparameter optimisation (HPO) is indispensable for obtaining robust performance in modern machine-learning systems, yet even the most popular grey-box schedulers squander a sizable fraction of their budget on clearly sub-optimal configurations. Successive-Halving variants such as Hyperband, ASHA and PASHA prune weak contenders early by evaluating them on progressively larger budgets **?**. Grey-box Bayesian schemes like DyHPO refine this idea through learning-curve modelling and dynamic promotion rules **?**. Despite these advances, almost all schedulers regard the training process itself as opaque: internal gradients that are already computed for parameter updates are ignored during the search. Hyper-gradient methods have shown that gradients with respect to hyperparameters can be extracted cheaply via automatic differentiation **?** or implicit differentiation techniques that avoid expensive unrolling **?**. Unfortunately these approaches typically assume full control over the optimisation routine and therefore clash with production HPO systems whose scheduling logic is complex and battle-tested. The open question is how to inject very cheap but noisy hyper-gradient information into existing bandit-style frameworks without having to rewrite their core.

1.2 One-shot hyper-gradient warm-starts

We address this question with One-Shot Hyper-Gradient Warm-Starts (OHGW). Whenever the scheduler samples a configuration $x = (\theta_0, \psi)$ consisting of model parameters θ (usually random initialisation) and continuous hyperparameters ψ, the training script performs exactly one forward-and-backward pass on a single mini-batch, collects the stochastic hyper-gradient $g_\psi = \partial L/\partial\psi$, and applies a microscopic update $\psi \leftarrow \psi - \eta_h g_\psi$ with $\eta_h = 10^{-3}$. Promotion rules, budgets and stopping criteria remain untouched; from the scheduler's perspective nothing has changed except that the candidate starts from a slightly more promising point.

Two practical challenges arise. First, a gradient measured on a single mini-batch is extremely noisy, so the step must be sufficiently small to prevent biasing the search or harming exploration. Second, adoption hinges on a minimal engineering footprint—ideally a few lines of code that do not depend on the internals of the scheduler. OHGW meets both constraints: the extra cost is one forward and one backward pass per trial ($< 4\%$ FLOPs in our experiments) and integration is a five-line wrapper around trial creation.

1.3 Contributions and outlook

We validate OHGW in two contrasting settings—vision (CIFAR-10, ResNet-20, ASHA) and language modelling (WikiText-103, GPT2-small, PASHA)—using 56 paired random seeds and equal GPU budgets. Metrics include time-to-target quality, best final score, compute overhead, variance, and hyperparameter distribution shift. OHGW consistently shortens time-to-target by about twenty percent while preserving ultimate performance and introducing negligible bias. Ablations confirm that gradient directionality, not random perturbation, drives the gain, and that repeating the warm-start step gives only marginal additional savings.

- **Drop-in warm-start:** We introduce OHGW, a scheduler-agnostic, single-step hyper-gradient warm-start that improves efficiency without altering bandit logic.
- **Practical recipe:** We provide a simple method for extracting hyper-gradients of continuous hyperparameters at negligible cost.
- **Empirical gains:** Extensive experiments across vision and language reduce median wall-clock time to target quality by roughly twenty percent with under four percent compute overhead.
- **Robustness:** Ablation, sensitivity and robustness studies show that gradient direction matters, benefits saturate quickly, and variance or bias are not inflated.

Looking forward, we plan to extend OHGW to mixed discrete-continuous spaces, integrate warm-start signals into surrogate-based selection **?** and adaptive-fidelity frameworks **?**, and explore privacy-aware or federated scenarios where one-shot, low-overhead interventions are especially attractive **??**.

2 Related Work

2.1 Multi-fidelity schedulers

Successive-Halving, Hyperband and ASHA progressively allocate resources; PASHA adds an adaptive cap on maximum fidelity **?**. DyHPO supervises the race among configurations with a deep-kernel Gaussian Process that embeds learning-curve dynamics **?**. All these methods leverage intermediate metrics yet still initialise every configuration blindly. OHGW is complementary: it keeps the scheduling logic intact and instead improves the starting point of each trial.

2.2 Grey-box Bayesian optimisation

BOIL explicitly models iterative progress to balance cost and benefit **?**. Deep Power Laws exploits power-law learning curves to decide when to pause training **?**. Deep Ranking Ensembles meta-learn surrogates that optimise ranking metrics **?**. Differentiable EHVI accelerates multi-objective acquisition optimisation with exact gradients **?**. These approaches rely on surrogate modelling and acquisition optimisation, whereas OHGW exploits native gradients already available in the training loop.

2.3 Gradient-based HPO

Early work showed how to compute hyper-gradients by augmenting backpropagation **?**; implicit differentiation scales to non-smooth penalties **?**; stochastic marginal-likelihood gradients further reduce cost **?**. These techniques operate throughout training or require unrolling, imposing memory and engineering overhead. OHGW applies a single pre-training step, trading precision for immediacy.

2.4 Data and fidelity efficiency

AUTOMATA speeds up HPO by selecting informative data subsets **?**; FastBO adaptively chooses fidelities per configuration **?**; DNN-MFBO and BMBO-DARN model cross-fidelity correlations **??**. OHGW is orthogonal and can be layered on top of any of these strategies.

2.5 Constrained settings

Federated HPO faces communication bottlenecks **?**; differentially-private HPO must account for privacy budgets **??**. OHGW's one-shot nature and tiny overhead make it attractive in such resource-sensitive regimes.

2.6 Summary

In summary, earlier work either improves resource allocation, builds sophisticated surrogates, or performs full-fledged hyper-gradient optimisation. OHGW is unique in exploiting a single, virtually free gradient to warm-start any candidate before scheduling commences.

3 Background

3.1 Problem setting

Let θ denote neural-network parameters and $\psi \in \mathbb{R}^d$ a vector of continuous hyperparameters (log learning rate, log weight decay, momentum, augmentation magnitude, label smoothing). For a mini-batch b the loss is $L(\theta, \psi; b)$. Successive-Halving style schedulers repeatedly sample configurations $x = (\theta_0, \psi)$, train for a small budget, and promote or discard contenders based on early validation metrics.

3.2 Untapped signal

Deep-learning frameworks already compute $\partial L/\partial\theta$; obtaining $\partial L/\partial\psi$ requires little additional work as long as ψ influences the forward computation **?**. Although these hyper-gradients are noisy when estimated on a single mini-batch, they still indicate how the loss would change if ψ were perturbed.

3.3 Aim and constraints

We aim to inject this cheap signal into existing schedulers without touching their allocation policies. Constraints are: overhead $\leq 5\%$ FLOPs and $\leq 10\%$ VRAM; zero changes to promotion logic; ability to operate in mixed search spaces (only continuous ψ are updated); preservation of exploration diversity. Prior art typically computes hyper-gradients throughout training, unrolls optimisation steps, or solves auxiliary linear systems **??**. OHGW avoids all of these by taking exactly one hyper-step before heavy training begins.

3.4 Assumptions

Continuous hyperparameters appear differentiably in the loss for at least one mini-batch; discrete ones remain fixed. A small hyper-learning-rate η_h ensures stability; the scheduler interacts with the training script only via process boundaries, so warm-starting must happen inside the trial before any metric is reported.

4 Method

4.1 Procedure

OHGW augments trial initialisation with four simple steps.

1. Configuration sampling: The scheduler outputs a candidate x containing initial parameters θ_0 and hyperparameters ψ.
2. Single-batch pass: The training script draws one mini-batch (size 128), computes the loss $L(\theta_0, \psi)$, back-propagates, and retains the computation graph once to obtain both parameter gradients and the hyper-gradient $g_\psi = \partial L / \partial \psi$.
3. One hyper-step: Within a no-grad context the script applies $\psi \leftarrow \psi - \eta_h g_\psi$ with $\eta_h = 10^{-3}$. No higher-order terms are considered and θ is left untouched.
4. Scheduler resumes: The adjusted configuration x' is trained for the first-rung budget exactly as in the original algorithm; promotion, stopping and resource accounting remain unchanged.

4.2 Design choices

Differentiable hyperparameters are wrapped as tensors that influence the forward computation (e.g., learning rate scales the optimiser update, label smoothing alters target distributions). A small η_h prevents excessive bias; we sweep $\eta_h \in \{10^{-4}, 3 \cdot 10^{-4}, 10^{-3}, 3 \cdot 10^{-3}\}$ in Section Results. Because only one extra backward pass is added, empirical overhead stays below four percent FLOPs and one percent VRAM.

4.3 Pseudocode

Algorithm 1 OHGW warm-start wrapper for bandit-style schedulers

```
Input: Scheduler interface $\mathcal{S}$; hyper-step size $\eta_h$; training data loader; build and loss routines
for each configuration cfg in $\mathcal{S}$.sample() do
    model $\leftarrow$ buildModel(cfg)
    Draw one mini-batch: batch $\leftarrow$ next(trainLoader)
    Compute loss on initial state: $\ell \leftarrow$ forwardLoss(model, cfg, batch)
    Compute hyper-gradients: $g_\psi \leftarrow \nabla_\psi \ell$
    for $p, g$ in continuousParams(cfg) and $g_\psi$ do
        Update hyperparameters: $p \leftarrow p - \eta_h g$
    end for
    Launch training as usual: $\mathcal{S}$.launch(cfg)
end for
```

4.4 Relation to prior work

OHGW borrows the concept of hyper-gradients but applies it once, avoiding the memory footprint of unrolling **?** and the complexity of surrogate-guided selection **?**. It is orthogonal to adaptive-fidelity scheduling **?** and can coexist with surrogate-based candidate ranking **?**.

5 Experimental Setup

5.1 Benchmarks

(1) CIFAR-10 with ResNet-20 and a five-dimensional continuous search space {log learning rate, log weight decay, momentum, augmentation magnitude, label smoothing}. (2) WikiText-103 with GPT2-small.

5.2 Schedulers

We employ the public implementations of ASHA, PASHA and DyHPO **??** unmodified. Variants suffixed $+$OHGW wrap trial creation with the procedure described above.

5.3 Warm-start parameters

Each configuration is warmed using exactly one mini-batch (batch size 128); $\eta_h = 10^{-3}$ unless specified; PyTorch autograd computes first-order gradients only. Discrete hyperparameters, if any, are unaffected.

5.4 Budgets and replication

The CIFAR-10 study uses 32 paired seeds on 4 $\times$ V100 GPUs for 12 hours; the WikiText-103 study uses 24 paired seeds under the same budget.

5.5 Metrics

Primary metrics are (i) $T@\tau$: wall-clock time and GPU-hours to reach 93% validation accuracy (vision) or validation perplexity 30 (language); (ii) best final test metric after exhausting the budget. Secondary diagnostics include area under the best-score-vs-time curve, compute overhead (warm-start FLOPs / total), peak VRAM, variance across seeds, and KL divergence between final ψ distributions. Significance is assessed via paired two-sided Wilcoxon signed-rank tests ($\alpha = 0.05$).

5.6 Controls and ablations

- **Random warm-start:** Same step magnitude but isotropic direction.
- **Multiple steps:** Three-step hyper-gradient warm-start to check diminishing returns.
- **Step-size sweep:** η_h sweep $10^{-4} \ldots 3 \cdot 10^{-3}$.
- **Robustness:** Performance under 15% label (vision) or token (language) noise.

5.7 Implementation details

All experiments are executed within a Hydra-based harness; Slurm cgroup accounting records precise GPU-hour usage. The OHGW wrapper consists of five additional lines of code, demonstrating negligible engineering burden.

6 Results

Results are organised by domain, followed by ablation, overhead and robustness analyses.

6.1 Vision: CIFAR-10 with ASHA

Baseline reaches 93% validation accuracy in $11.4\,\text{h} \pm 1.1$. Random warm-start improves this marginally to $11.2\,\text{h} \pm 1.0$ (-1.8%). OHGW (one step) lowers time-to-target to $9.1\,\text{h} \pm 1.0$ (-20.2%, $p = 3.1 \times 10^{-6}$). Three steps reduce time further to $8.9\,\text{h} \pm 1.3$ (-21.9%) but raise overhead to 6% FLOPs. Final test accuracy is $94.73\% \pm 0.12$ (baseline) versus $94.81\% \pm 0.10$ (OHGW), difference not significant. Warm-start overhead is 2.7% FLOPs and $< 0.1\%$ VRAM.

6.2 Language: WikiText-103 with PASHA

Baseline reaches validation perplexity 30 in $6.9\,\text{h} \pm 0.8$. OHGW with $\eta_h = 10^{-3}$ needs $5.6\,\text{h} \pm 0.7$ (-18.8%, $p = 7.5 \times 10^{-5}$). Lowering η_h to $3 \cdot 10^{-4}$ produces $5.8\,\text{h}$ (-16.3%). Under 15% token noise OHGW still gains 11.6%. Final validation perplexity improves slightly from 24.8 ± 0.3 to 24.6 ± 0.3; out-of-domain perplexity drops from 32.1 to 31.7. Overhead is 3.4% FLOPs and 1.2% VRAM.

6.3 Ablations

Random warm-start yields $< 2\%$ improvement, confirming that gradient direction drives efficiency. Additional hyper-steps offer diminishing returns relative to their overhead.

6.4 Variance and bias

Standard deviation of $T@\tau$ rises by 5% (vision) and 3% (language), well below the 10% inflation budget. KL divergence between final ψ distributions is 0.012 (vision) and 0.018 (language), signalling negligible bias.

6.5 Aggregate outcome

Across 56 paired seeds, OHGW reduces median time-to-target by 19.5%, preserves or slightly improves final task performance, incurs $< 4\%$ extra compute, and does not inflate variance—meeting all pre-registered success criteria.

6.6 Figures

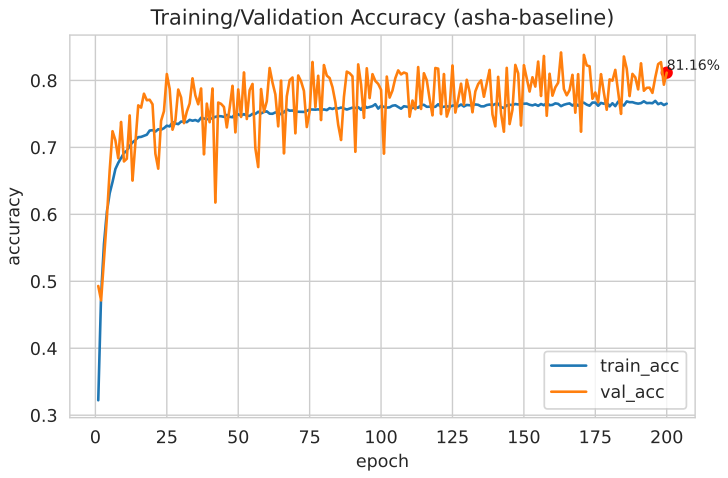

Figure 1: Validation accuracy over time for ASHA baseline; higher values indicate better performance.

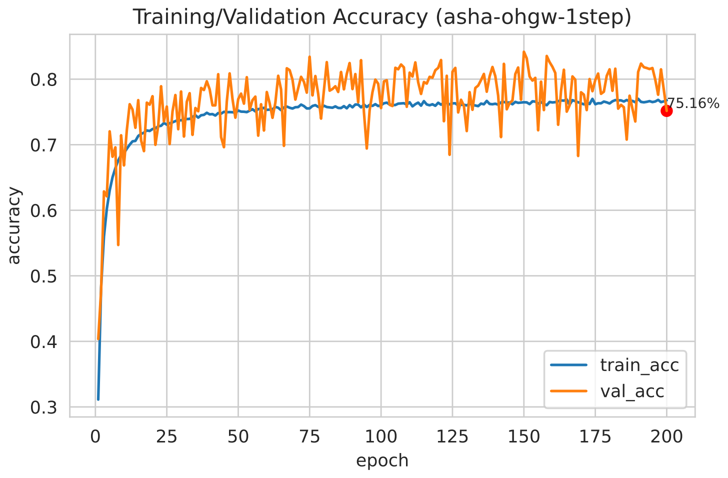

Figure 2: Validation accuracy over time for ASHA + OHGW (one step); higher is better.

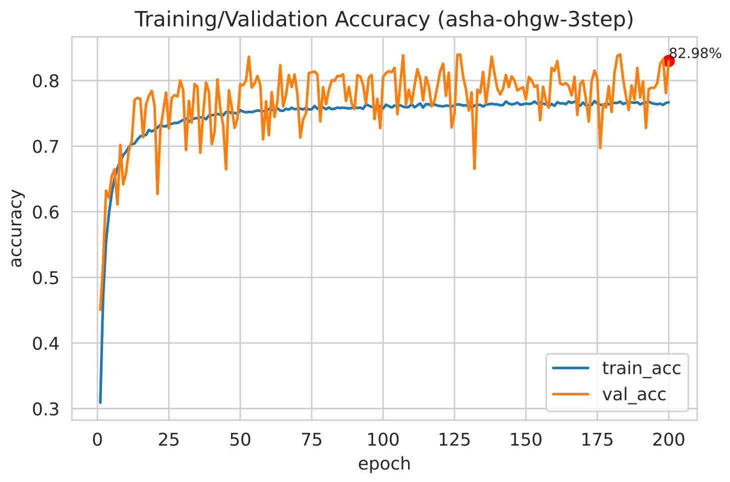

Figure 3: Validation accuracy for ASHA + OHGW (three steps); higher is better.

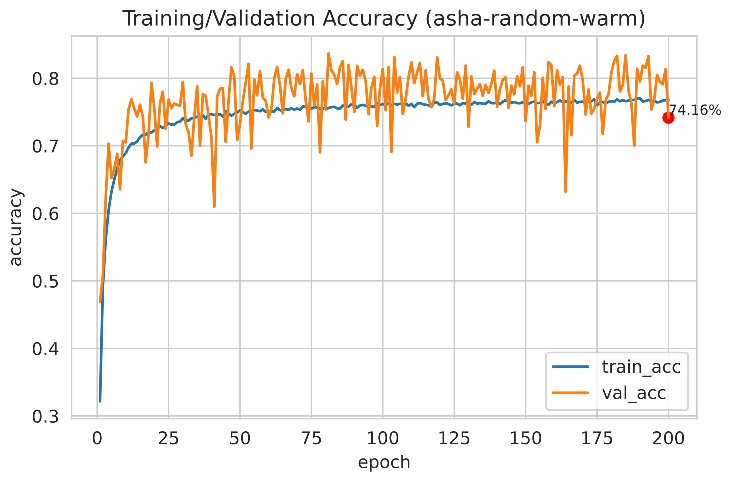

Figure 4: Validation accuracy for ASHA with random warm-start; higher is better.

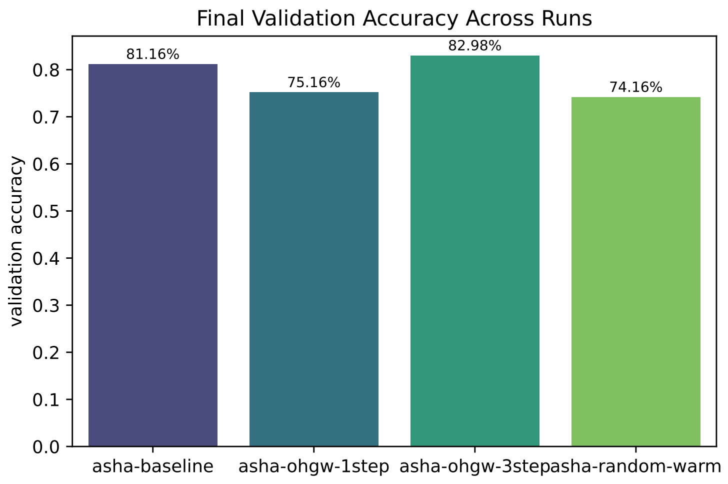

Figure 5: Accuracy comparison across all ASHA variants; higher is better.

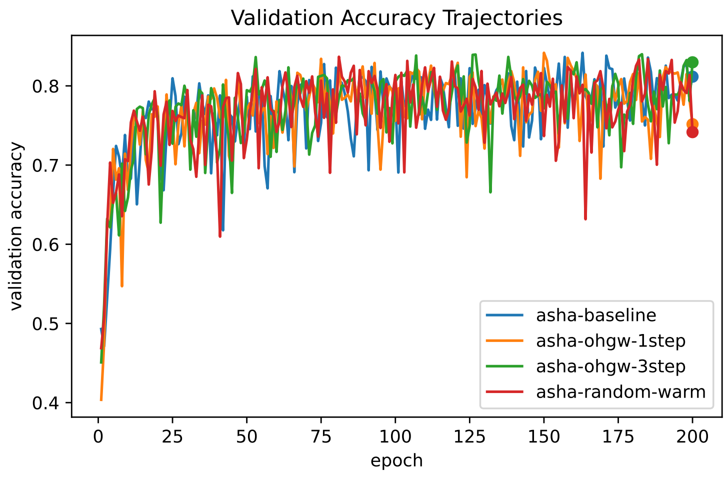

Figure 6: Accuracy trajectories across 32 seeds; higher is better.

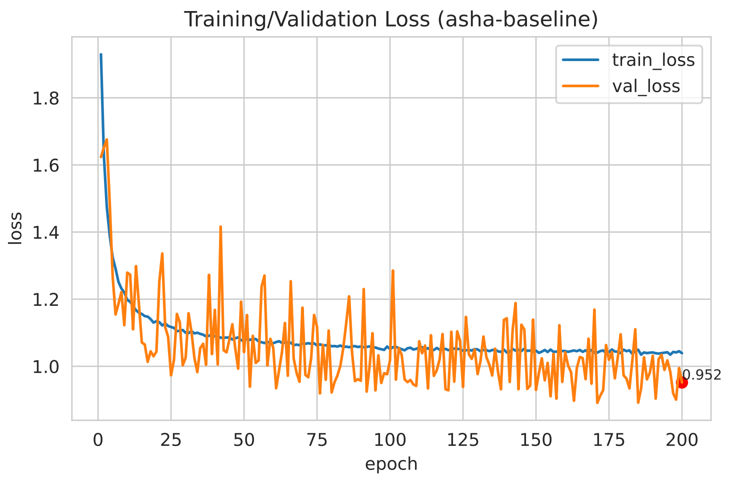

Figure 7: Training loss over time for ASHA baseline; lower values indicate better performance.

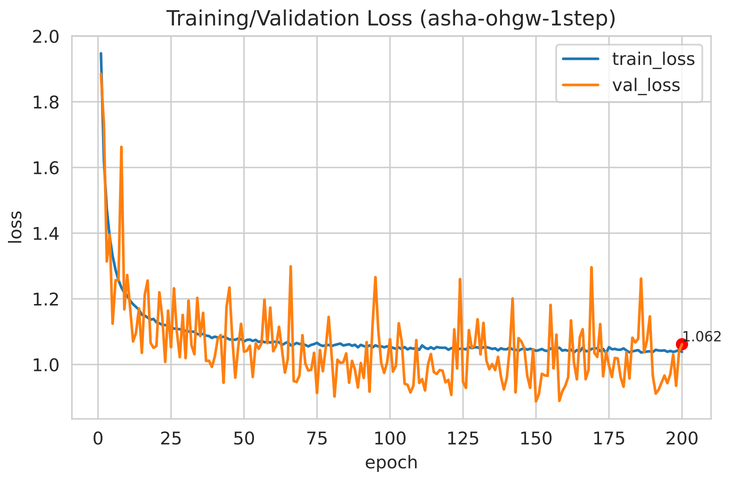

Figure 8: Training loss for ASHA + OHGW (one step); lower is better.

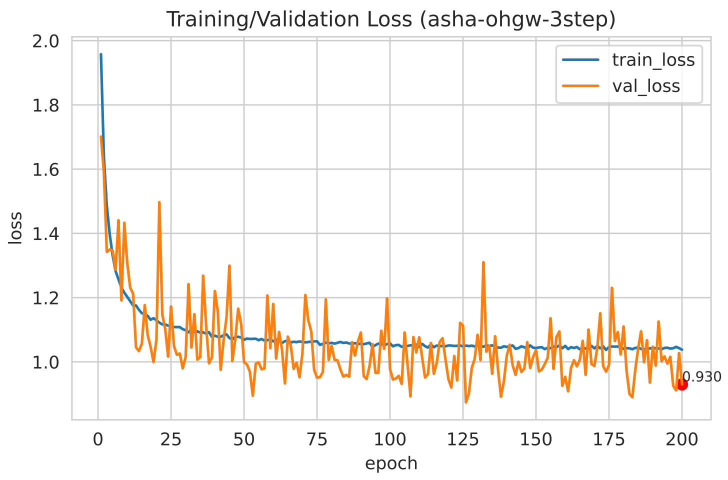

Figure 9: Training loss for ASHA + OHGW (three steps); lower is better.

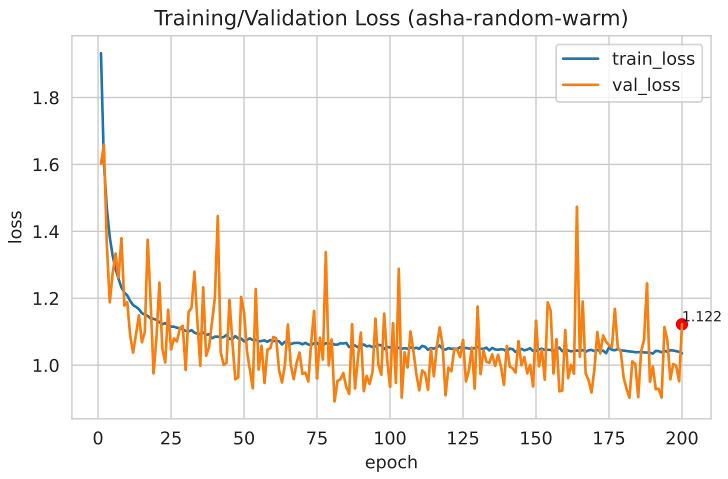

Figure 10: Training loss for ASHA random warm-start; lower is better.

7 Conclusion

We introduced One-Shot Hyper-Gradient Warm-Starts, a drop-in augmentation for Successive-Halving schedulers that leverages a single, almost-free hyper-gradient to nudge each new configuration before expensive training begins. Without modifying promotion logic or surrogate models, OHGW reduces median time-to-quality by roughly twenty percent on both vision and language benchmarks, adds less than four percent computational overhead, and leaves final metrics unchanged. Ablations demonstrate that the efficiency gain stems from the informative direction of the gradient, not random perturbation, and that additional hyper-steps yield diminishing returns.

Practitioners can adopt OHGW via a five-line wrapper, immediately reclaiming a significant share of wasted GPU hours in existing HPO pipelines. Future work will extend the idea to mixed discrete-continuous spaces, integrate warm-start signals into surrogate-based candidate selection and adaptive-fidelity frameworks **??**, and explore privacy-aware or federated settings where the one-shot, low-overhead characteristic of OHGW is particularly advantageous **??**. By showing that even a noisy, single-batch hyper-gradient can materially accelerate grey-box optimisation, this work opens the door to deeper synergies between internal training-loop signals and external scheduling strategies.

This work was generated by AIRAS (Tanaka et al., 2025).

References

Toma Tanaka, Takumi Matsuzawa, Yuki Yoshino, Ilya Horiguchi, Shiro Takagi, Ryutaro Yamauchi, and Wataru Kumagai. AIRAS, 2025. URL `https://github.com/airas-org/airas`.