Reference-Guided Training: Adaptive Gradient Scaling via Per-Sample Loss Comparisons

Israel Rodrigues Soares[*]
Kumamoto University
Kumamoto, Japan

Pedro Benedetti[†]
Kumamoto University
Kumamoto, Japan

Thanda Shwe[‡]
Kumamoto University
Kumamoto, Japan

Israel Mendonça[§]
Kumamoto University
Kumamoto, Japan

Masayoshi Aritsugi[¶]
Kumamoto University
Kumamoto, Japan

Abstract

Empirical Risk Minimization treats training samples uniformly, which can degrade performance when the training data are noisy or heterogeneous. We introduce Reference-Guided Training, a new training objective that uses per-sample loss comparisons with a fixed reference model to reweight gradient contributions without relying on prediction imitation, while preserving per-sample gradient directions and ensuring bounded gradient amplification. We characterize its gradient and curvature properties, and evaluate classification with synthetic label noise on Fashion-MNIST, human annotation noise on CIFAR-10N, and time-series forecasting on M4. The proposed method improves accuracy in many noisy classification settings and reduces forecasting error in most reported regression conditions, with architecture- and reference-dependent exceptions. The analysis suggests that the proposed method benefits from differences in error rankings between the learner and the reference when these differences provide reliable, complementary information about individual samples.

1 Introduction

Empirical Risk Minimization (ERM) is a fundamental principle in supervised learning. It estimates model parameters by minimizing the average loss over the training data, which serves as an empirical estimate of the expected risk, with uniform weighting of samples. This uniform weighting implies that all samples contribute equally to the empirical risk [1, 2].

In practice, real-world datasets often exhibit noise, variability, and imprecision, commonly modeled as aleatoric uncertainty, i.e., noise inherent to the data-generating process [3]. Under such conditions, uniformly weighting all samples can degrade model performance, particularly in the presence of noisy or corrupted data [4–6].

Prior work has addressed the limitations of uniform sample weighting through three main categories. First, loss-based methods modify the loss function to reduce the influence of large residuals, limiting the impact of outliers while preserving the ERM formulation. Representative examples include robust loss functions such as the Huber loss [7] and its generalizations [8]. Second, sample reweighting

[*]israelrodsoares@gmail.com
[†]bndttipedro@gmail.com
[‡]thandashwe@cs.kumamoto-u.ac.jp
[§]israel@cs.kumamoto-u.ac.jp
[¶]aritsugi@cs.kumamoto-u.ac.jp

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

methods adjust the contribution of individual samples based on signals derived from the learner or the training data itself, such as estimated difficulty or observed error patterns. Examples include curriculum learning [9] and self-paced learning (SPL) [10]. Third, external-model-based methods incorporate signals from auxiliary models to guide training, typically transferring knowledge by encouraging the learner to imitate predictions from a reference model, as in Knowledge Distillation (KD) [11, 12], or by teacher–student curriculum learning, where an auxiliary model learns a curriculum or sample weighting policy to guide the student training process [13, 14].

Within reference-based methods, two main patterns emerge. Some approaches directly exploit signals from a reference model, such as predictions or internal representations, while others rely on auxiliary models that must be trained to generate guidance signals, incurring additional computational cost [13, 15]. When predictions are used directly, they are typically treated as targets for imitation rather than as signals for sample reweighting [11, 12]. This raises a natural question: can a reference model guide learning by directly leveraging its predictions to reweight samples within the ERM objective without introducing auxiliary prediction targets? If a reference model performs better on certain samples, the learner's relative underperformance can be used to prioritize samples where it remains inaccurate. Unlike previous approaches, our method uses the reference model's per-sample losses not as targets for imitation, but as signals to modulate sample priority through reweighting.

Reference-based comparisons are common in model evaluation. In time-series forecasting, metrics such as Mean Absolute Scaled Error (MASE) [16] and Overall Weighted Average (OWA) [17] assess predictive performance relative to baseline forecasts. Conventional training objectives, such as cross-entropy and Mean Squared Error (MSE), do not explicitly incorporate reference-model information. This motivates investigating whether reference-relative information can also guide training by adjusting the contributions of individual samples.

We propose a Reference-Guided (RG) training objective that extends standard ERM using information derived from per-sample comparisons with a reference model. The proposed objective depends only on per-sample loss comparisons, making it compatible with any pre-trained reference model — including statistical methods and deep learning models — without requiring architectural modifications to the learner or additional auxiliary model training. Moreover, it is compatible with different supervised loss functions, allowing its application across classification and regression tasks. Code and experimental artifacts are available at `https://github.com/Raelis/reference-guided-training`.

The main contributions of this work are:

- We introduce the RG training objective that extends standard ERM by incorporating per-sample loss comparisons with a fixed reference model, without relying on prediction imitation or auxiliary model training.
- We show that the proposed objective induces adaptive gradient scaling with bounded magnitude and preserved direction, and characterize three distinct amplification regimes governed by the guide-relative per-sample error.
- We evaluate RG on image classification under synthetic label noise in Fashion-MNIST, human annotation noise in CIFAR-10N, and on large-scale time-series forecasting in the M4 forecasting benchmark, demonstrating improvements across the majority of the experiments.
- We investigate how the guide model and its error distribution affect RG, providing a practical criterion for guide selection.

2 Related work

Limitations of uniform sample weighting have been widely studied, particularly in settings affected by noise, outliers, or heterogeneous sample difficulty. Existing approaches can be grouped into three main categories based on how they determine sample influence.

Loss-based methods (robust losses). These methods modify the loss function to reduce sensitivity to outliers or noise, typically by limiting gradient contributions from large errors. Examples include Huber loss [7], Maximum Correntropy Criterion (MCC) [18], ramp loss [19], pinball loss [20], and quantile loss [21]. While effective at reducing the influence of extreme residuals, these approaches rely just on the learner's prediction errors to determine sample influence, not exploiting potentially complementary information from external reference models.

Sample reweighting methods. These methods adjust the contribution of individual samples during training based on signals derived from the learner's state or the data distribution. Curriculum Learning [9] organizes training data from easier to more complex examples to guide optimization, while SPL [10] iteratively selects easier samples based on the model's current state and gradually introduces harder ones. Meta-learning approaches that learn observation weights using validation signals [22, 23] also fall into this category, as they infer weighting schemes through bi-level optimization but still rely only on signals derived from the learner or the data itself.

External-model-based methods. These methods incorporate signals from a separate reference model to guide learning. Knowledge Distillation [11] uses a pre-trained reference model to encourage the learner to imitate its output distribution. MentorNet [15] instead trains an auxiliary network to estimate sample difficulty and assign weights to guide the learner. Teacher–student curriculum learning similarly relies on an auxiliary teacher model trained to learn a curriculum or sample weighting policy [13, 14], incurring additional training cost beyond the reference model itself. Other approaches include RHO-LOSS [24], which selects informative examples using reducible holdout loss, and Rho-1 [25], which applies reference-guided token selection in language model pretraining. SRatio [26] combines information from a complex model with estimates of learner difficulty to reweight training samples.

3 Reference-guided training objective

We introduce a training objective that incorporates external reference information into ERM [2] through per-sample loss comparisons between a learner and a fixed reference model.

3.1 Formal definition

We denote $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ as the training set, where n is the number of samples. We consider a trainable model, referred to as the learner, parameterized by θ, and a fixed reference model, referred to as the guide model, which is independent of θ. Let $\ell(\hat{y}, y)$ be a differentiable loss function. The per-sample losses of the learner and the guide are respectively defined as

$$L_i(\theta) = \ell\big(\hat{y}_i^{\text{learner}}(\theta), y_i\big) \tag{1}$$

$$L_i^{\text{guide}} = \ell\big(\hat{y}_i^{\text{guide}}, y_i\big) \tag{2}$$

where $L_i(\theta)$ depends on θ, while L_i^{guide} is constant with respect to θ.

The difference between learner and guide losses determines whether a sample should receive additional gradient weighting. This comparison is incorporated into the proposed RG per-sample loss, defined as

$$\mathcal{L}_i(\theta) = L_i(\theta) + \lambda\, \phi_\delta\big(L_i(\theta) - \gamma L_i^{\text{guide}}\big) \tag{3}$$

where $\lambda \in \mathbb{R}_{\geq 0}$ and $\gamma \in \mathbb{R}_{>0}$ control, respectively, the maximum amplification of the gradient scaling (see Eq. 11), and the relative tolerance applied to the guide loss, which determines the onset of the amplification regime. The function $\phi_\delta : \mathbb{R} \to \mathbb{R}_{\geq 0}$ is a convex, non-decreasing, continuously differentiable linear–quadratic function [27], whose smoothness is controlled by $\delta \in \mathbb{R}_{>0}$, defined as

$$\phi_\delta(x) = \begin{cases} 0, & x \leq 0, \\ \dfrac{x^2}{2\delta}, & 0 < x < \delta \\ x - \dfrac{\delta}{2}, & x \geq \delta \end{cases} \tag{4}$$

The empirical RG risk over n samples is given by

$$R_{emp}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_i(\theta) \tag{5}$$

3.2 Gradient analysis and dynamics

3.2.1 Direction preservation and bounded scaling

Let the guide-relative error be

$$r_i(\theta) = L_i(\theta) - \gamma L_i^{\text{guide}} \tag{6}$$

Since L_i^{guide} is constant with respect to θ, by applying the chain rule to (3),

$$\begin{aligned} \nabla_\theta \mathcal{L}_i(\theta) &= \nabla_\theta L_i(\theta) + \lambda \phi_\delta'\big(r_i(\theta)\big) \nabla_\theta r_i(\theta) \\ &= \Big(1 + \lambda \phi_\delta'\big(r_i(\theta)\big)\Big) \nabla_\theta L_i(\theta) \end{aligned} \tag{7}$$

where $\phi_\delta' : \mathbb{R} \to [0, 1]$ is given by

$$\phi_\delta'(x) = \begin{cases} 0, & x \le 0 \\ \dfrac{x}{\delta}, & 0 < x < \delta \\ 1, & x \ge \delta \end{cases} \tag{8}$$

Defining

$$\omega_i(\theta) = 1 + \lambda \phi_\delta'(r_i(\theta)) \tag{9}$$

Eq. 7 can be rewritten as

$$\nabla_\theta \mathcal{L}_i(\theta) = \omega_i(\theta) \nabla_\theta L_i(\theta) \tag{10}$$

Since $0 \le \phi_\delta'(x) \le 1$ for all $x \in \mathbb{R}$, it follows that

$$\omega_i(\theta) \in [1, 1 + \lambda] \tag{11}$$

Therefore, the per-sample gradient direction is preserved since the gradient is rescaled by a positive scalar, and its magnitude is bounded by a factor of $1 + \lambda$.

3.2.2 Amplification regimes

For $\lambda > 0$, the bounded amplification induces three distinct regimes of sample behavior:

$$\omega_i(\theta) = \begin{cases} 1, & r_i(\theta) \le 0 \quad \text{(no amplification)} \\ 1 + \lambda \dfrac{r_i(\theta)}{\delta}, & 0 < r_i(\theta) < \delta \quad \text{(smooth amplification)} \\ 1 + \lambda, & r_i(\theta) \ge \delta \quad \text{(saturated amplification)} \end{cases} \tag{12}$$

When $r_i(\theta) \le 0$, the sample contributes identically to the ERM per-sample objective. For $0 < r_i(\theta) < \delta$, the scaling factor increases smoothly with the relative error. When $r_i(\theta) \ge \delta$, the gradient operates in the saturated regime, behaving like the gradient of the ERM per-sample objective scaled by a factor of $1 + \lambda$. This formulation can therefore be interpreted as an adaptive sample gradient scaling mechanism, equivalent to sample reweighting with state-dependent and bounded weights $\omega_i(\theta)$.

Beyond gradient scaling, the RG objective also modifies the local curvature of the loss landscape. Specifically, in the transition regime, it introduces a positive semidefinite update of rank at most one to the Hessian, adding nonnegative curvature along the loss gradient direction. We provide a detailed characterization of these curvature properties and prove convexity and Lipschitz gradient bounds in Appendix A.

4 Experiments

To evaluate the generality of our method across different tasks and domains, we conducted experiments on three datasets: synthetic label corruption on Fashion-MNIST [28], human annotation noise on CIFAR-10N [29, 30], and time-series forecasting on M4 [17].

All experiments were implemented using the libraries listed in Appendix F.2. General experimental details, including hardware setup and randomness control, are provided in Appendix B. Dataset-specific experimental configurations are provided in Appendices C, D, and E for Fashion-MNIST, CIFAR-10N, and M4, respectively.

4.1 Synthetic label corruption on Fashion-MNIST

4.1.1 Experimental setup

Fashion-MNIST [28] is a 10-class grayscale image classification benchmark composed of 28×28 pixel images and totaling 70,000 samples. Of these, 54,000 are used for training, 6,000 for clean validation, and 10,000 for testing.

Three target architectures are evaluated: Multilayer Perceptron (MLP) [31], Convolutional Neural Network (CNN) [32], and Vision Transformer (ViT) [33]. In the RG objective (Eq. 3), two guide architectures are considered: ResNet20 [34] and a simple MLP. Architecture search spaces and configuration details for all models are reported in Appendix C.2.

To assess robustness under corrupted supervision, synthetic symmetric label noise is applied exclusively to the training set labels at ratios $\rho \in \{0\%, 40\%, 80\%\}$. Corrupted labels are sampled uniformly from the remaining classes. A fixed random seed ensures identical corruption patterns across methods and runs, with the 40% corrupted-sample indices nested within those at 80%.

All final models are trained for 300 epochs using cross-entropy as the underlying supervised loss. Optimization is performed using Stochastic Gradient Descent (SGD) with momentum 0.9, weight decay 5×10^{-4}, a batch size of 256, and a cosine annealing learning-rate scheduler [35]. The architecture hyperparameters used for all methods are selected on clean data through 20 Optuna trials of up to 30 epochs using a fixed learning rate (LR) of 0.005. Once the architectures are fixed, we tune the LR over $\{0.01, 0.005, 0.001\}$ and method-specific hyperparameters, using 20 trials per method and noise level or until the search space is exhausted. Both searches use validation cross-entropy as the selection criterion. The search space is defined in Appendix C.4. Full experimental specifications are provided in Appendix C.

4.1.2 Results and discussion

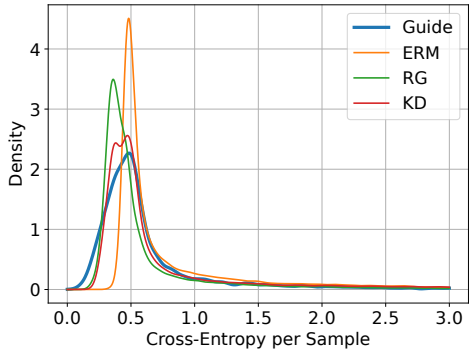
Table 1 summarizes the performance of ERM, RG, KD, SPL, and MentorNet-PD across architectures and label noise levels. All methods are trained under the same label noise conditions; RG and KD additionally use a ResNet20 guide trained under the corresponding noise conditions to ensure comparability. On the clean test set, the ResNet20 guides used by RG and KD achieve 93.68%, 90.37%, and 77.85% accuracy at 0%, 40%, and 80% label noise, respectively. Results are averaged over five runs and reported as mean test accuracy $\pm$ standard deviation (SD).

At $\rho = 0\%$, RG remains competitive, achieving the highest mean accuracy on ViT, with performance comparable to KD. At $\rho = 40\%$, RG achieves the highest mean across all three architectures, with comparable results from KD and MentorNet-PD on CNN and MLP. At $\rho = 80\%$, RG achieves the highest mean on CNN and ViT, outperforming ERM by 5.83 and 12.12 percentage points, respectively, without overlap in the reported mean $\pm$ standard deviation intervals with competing methods. The exception occurs for MLP at $\rho = 80\%$, where ERM outperforms RG (80.91% vs. 79.53%). With a clean-trained guide, however, RG reaches 86.87% (Table 18), suggesting sensitivity to guide quality.

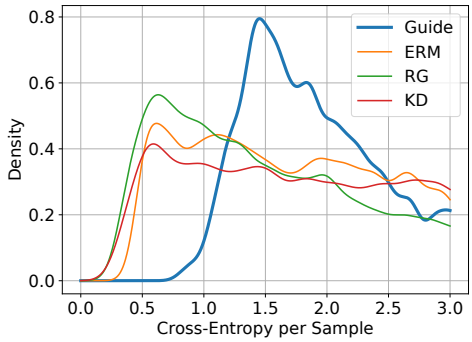
According to the results, ViT benefits most from RG relative to ERM, particularly at 40% and 80% label noise, with accuracy gains of 4.75 and 12.12 percentage points, respectively. To better understand this behavior, we compare the per-sample cross-entropy distributions of RG and KD, both trained using the same ResNet20 guide, alongside ERM as a baseline trained without a guide and the guide itself. Using ViT under 40% label noise, we analyze the distributions on the full test set and on the subset of samples misclassified by the guide, as shown in Figure 1.

Table 1: Clean-test accuracy on Fashion-MNIST (%, mean ± sample SD over five runs). Bold indicates the highest numerical mean for each setting.

Model	Noise	ERM	RG	KD	SPL	MentorNet-PD
CNN	0%	93.04 ± 0.22	92.95 ± 0.22	**93.72 ± 0.15**	93.17 ± 0.26	93.00 ± 0.13
	40%	90.18 ± 0.42	**91.37 ± 0.07**	90.95 ± 0.72	81.48 ± 7.54	91.09 ± 0.24
	80%	76.09 ± 4.48	**81.92 ± 0.28**	78.71 ± 1.93	77.11 ± 3.59	78.60 ± 1.41
MLP	0%	91.19 ± 0.07	91.25 ± 0.06	**91.41 ± 0.09**	91.20 ± 0.09	89.79 ± 0.16
	40%	88.02 ± 0.33	**89.09 ± 0.24**	89.01 ± 0.19	77.32 ± 6.17	88.71 ± 0.28
	80%	**80.91 ± 0.29**	79.53 ± 1.06	77.54 ± 0.10	80.35 ± 0.30	78.48 ± 2.04
ViT	0%	88.81 ± 1.09	**91.85 ± 0.61**	90.63 ± 0.70	88.89 ± 0.25	85.24 ± 0.39
	40%	84.87 ± 0.58	**89.63 ± 0.59**	86.60 ± 0.77	84.05 ± 0.39	83.83 ± 0.62
	80%	63.71 ± 3.44	**75.83 ± 0.60**	73.93 ± 1.01	61.76 ± 3.33	49.11 ± 4.82



(a) Global error distribution across training methods.



(b) Error distribution restricted to guide-identified difficult samples.

Figure 1: Gaussian kernel density estimates of per-sample cross-entropy for ViT under 40% label noise. (a) Global loss distributions for the guide, ERM, RG, and KD. (b) Loss distributions restricted to samples misclassified by the guide. Learner distributions are averaged over five runs.

Figure 1a presents the global error distribution, where RG and KD exhibit loss distributions closer to that of the guide than ERM.

Figure 1b presents the error distribution restricted to samples misclassified by the guide model, providing a focused view of behavior on difficult examples. In contrast with Figure 1a, RG deviates more strongly from the guide on this subset, with a larger fraction of samples exhibiting lower losses than the guide compared with ERM and KD. This behavior is consistent with the reallocation of learning focus induced by the RG objective through relative per-sample losses, allowing the learner to achieve lower losses on examples where the guide makes errors.

This contrast is quantified in Table 2, where we report rank alignment using Spearman correlation [36] on the full test set and distributional similarity using the Kolmogorov–Smirnov statistic (KS) [37, 38] and the 1-Wasserstein distance (W) [39] on both the full test set and the guide-misclassified subset, computed from learner and guide per-sample cross-entropy losses.

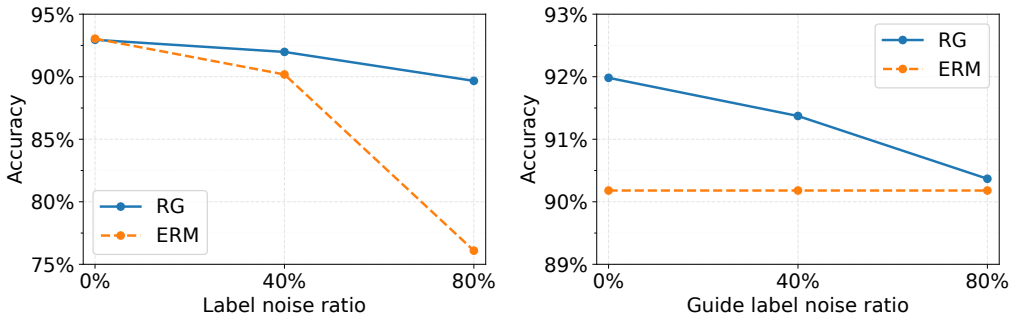
Table 2: Alignment with the guide model on the full test set and the guide-misclassified subset. Bold values indicate the strongest alignment under each metric (higher Spearman; lower KS and W).

Method	**Test set**			**Guide error subset**	
	Spearman	**KS**	**W**	**KS**	**W**
ERM	0.438 ± 0.011	0.340 ± 0.006	0.163 ± 0.010	0.277 ± 0.011	0.331 ± 0.013
RG	**0.675 ± 0.013**	0.167 ± 0.009	**0.062 ± 0.010**	0.371 ± 0.007	0.476 ± 0.015
KD	0.635 ± 0.015	**0.080 ± 0.007**	0.077 ± 0.019	**0.263 ± 0.012**	**0.289 ± 0.016**

One notable pattern in Table 2 is that both RG and KD exhibit greater global alignment with the guide than ERM, but differ substantially on samples misclassified by the guide. On this subset, RG shows the largest deviation from the guide's loss distribution, as indicated by its higher KS and 1-Wasserstein distances. KD, in contrast, exhibits the smallest distances, indicating a closer match to the guide's loss behavior. This suggests that RG can leverage the guide's information about sample difficulty without necessarily reproducing its loss behavior on difficult examples.

4.1.3 Ablation studies

We analyze the robustness of RG compared to ERM using a CNN learner under two controlled variations in label noise, as shown in Figure 2. First, in Figure 2a, we vary the label noise ratio in the learner while using a clean guide to evaluate whether a reliable guide can transfer robustness under increasing corruption. Second, in Figure 2b, we fix the learner at 40% label noise and vary the guide noise from 0% to 80% to study the effect of guide quality, covering cases where the guide is cleaner, equally noisy, or more corrupted than the learner.



(a) Performance under increasing label noise ratio. (b) Performance under varying guide noise ratio.

Figure 2: CNN robustness analysis. (a) Sensitivity to label noise in the training data with a clean guide (guide noise fixed at 0%). (b) Transfer of robustness with respect to guide quality, controlled via guide noise under 40% label noise.

Under increasing label noise, RG outperforms ERM across the evaluated CNN settings, with larger gains at higher noise levels (Figure 2a). This indicates that a guide trained on clean data can transfer robustness under corrupted supervision.

For guide noise variation, RG accuracy decreases as guide noise increases, indicating sensitivity to guide quality (Figure 2b). At 40% learner noise, RG achieves 91.98% with a clean guide and 90.37% with an 80%-noise guide. In the latter case, RG and ERM exhibit comparable performance (90.37% vs. 90.18%). Similar trends are observed for MLP and ViT (Appendix C.7.2).

Additional CNN ablations of λ, δ, γ, and guide architecture are reported in Appendix C.7.1. Increasing λ improves accuracy across the evaluated settings, while replacing the ResNet20 guide with a smaller MLP reduces performance. ERM is recovered when $\lambda = 0$.

To isolate optimization effects, ERM @ RG LR uses the LR selected for RG without changing the ERM objective. RG mean-one normalizes the per-sample gradient multipliers ω_i by their minibatch mean, retaining relative sample weights while removing average amplification. We also evaluate two-fold cross-fitting: each example is scored by a ResNet20 guide trained on the opposite half of the noisy training set. This removes in-sample guide scoring, although each guide uses only half the training examples. Complete controls and cross-fitting results appear in Appendices C.7.3 and C.7.4.

As shown in Table 3, learning-rate-matched ERM does not reproduce RG's gains on ViT, with RG improving accuracy from 83.75% to 89.63% at 40% noise and from 62.02% to 75.83% at 80%. RG mean-one retains substantial improvements, although its accuracy is lower than standard RG at 40% noise (86.97% vs. 89.63%) and nearly identical at 80% (75.75% vs. 75.83%). These results suggest that average gradient amplification is not necessary to retain RG's gains at 80% noise, although its contribution may vary with noise severity. Cross-fitting also preserves performance close to standard RG at both noise levels (89.47% vs. 89.63% and 75.18% vs. 75.83%), indicating that the observed ViT improvements do not depend on in-sample guide scoring. Complete results

Table 3: ViT optimization and cross-fitting controls under nonzero label noise (clean-test accuracy in %, mean ± sample SD over five runs). RG variants use matched-noise ResNet20 guides. Bold indicates the highest numerical mean for each setting.

Noise	ERM	ERM @ RG LR	RG	RG mean-one	RG cross-fitted
40%	84.87 ± 0.58	83.75 ± 0.42	$\mathbf{89.63 \pm 0.59}$	86.97 ± 0.29	89.47 ± 0.40
80%	63.71 ± 3.44	62.02 ± 2.10	$\mathbf{75.83 \pm 0.60}$	75.75 ± 0.80	75.18 ± 0.23

across all architectures, together with additional guide-memorization diagnostics, are provided in Appendices C.7.3, C.7.4, and C.7.5.

4.2 Human annotation noise on CIFAR-10N

To evaluate RG under real-world annotation noise, we use CIFAR-10N [30], which provides real human annotations for CIFAR-10 [29]. We consider three learner/guide annotation settings: `random1/random3` (different annotations with similar noise rates), `worst/worst` (identical high-noise annotations), and `worst/aggre` (a cleaner, aggregated guide). CNN, MLP, and ViT learners are compared with ERM using a fixed ResNet56 guide for RG. We train on 45,000 noisy-label images, select models using 5,000 clean validation examples, and evaluate on the original clean test set. Final training uses 150 epochs and five seeds. Additional experimental details are provided in Appendix D.

Table 4: CIFAR-10N clean-test accuracy (%, mean ± sample SD over five runs). Δ denotes RG minus ERM accuracy, in percentage points. Bold indicates the highest numerical mean for each setting.

Model	Learner / guide	ERM	RG	Δ (pp)
CNN	`random1/random3`	81.49 ± 0.22	$\mathbf{84.40 \pm 0.23}$	+2.91
CNN	`worst/worst`	72.53 ± 0.61	$\mathbf{76.50 \pm 0.67}$	+3.97
CNN	`worst/aggre`	72.53 ± 0.61	$\mathbf{81.53 \pm 0.29}$	+9.00
MLP	`random1/random3`	55.45 ± 0.24	$\mathbf{56.33 \pm 0.27}$	+0.88
MLP	`worst/worst`	51.88 ± 0.17	$\mathbf{53.05 \pm 0.26}$	+1.17
MLP	`worst/aggre`	51.88 ± 0.17	$\mathbf{54.43 \pm 0.33}$	+2.55
ViT	`random1/random3`	66.51 ± 0.33	$\mathbf{71.85 \pm 0.27}$	+5.35
ViT	`worst/worst`	62.69 ± 0.87	$\mathbf{70.76 \pm 0.26}$	+8.07
ViT	`worst/aggre`	62.69 ± 0.87	$\mathbf{74.01 \pm 0.34}$	+11.32

RG achieves higher mean test accuracy than ERM in all nine learner/guide settings. Gains also appear when the guide and learner use identical high-noise labels (`worst/worst`), and when they use distinct annotations with similar noise rates (`random1/random3`); thus, a cleaner guide is not required for the observed improvements. The largest gains occur with cleaner aggregated guide labels, reaching 9.00 percentage points for CNN and 11.32 for ViT under `worst/aggre`. These results corroborate the Fashion-MNIST findings that RG can improve robustness even with noise-matched guides, suggesting that reference-relative losses provide useful signals for adaptive sample weighting under noisy supervision.

4.3 Regression on M4

We evaluate the proposed RG objective in the time-series forecasting domain using the M4 dataset, one of the largest publicly available benchmarks for univariate forecasting [17]. The dataset contains 100,000 time series spanning multiple domains, including finance, demographics, industry, and macroeconomics. The series are grouped into six frequency subsets (Yearly, Quarterly, Monthly, Weekly, Daily, and Hourly), covering a wide range of temporal dynamics and seasonal patterns.

4.3.1 Setup

To control the number of generated training samples for computational feasibility, we define window sizes and strides per frequency subset. The window size specifies the length of the historical context used as input for forecasting, while the stride controls the step size of the sliding window used to generate training samples. Both are selected based on the empirical distribution of series lengths in the M4 dataset. The selection procedure is detailed in Appendix E.1.

Model performance is evaluated using OWA, the official M4 metric [17]. Model-guide loss ranking alignment is evaluated using the Spearman correlation coefficient [36] on test set data. To assess architectural generality, we evaluate three target architectures: an MLP [31], a Temporal Convolutional Network (TCN) [40], and a Gated Recurrent Unit (GRU) [41]. All neural models are trained using MSE loss, optimized with AdamW [42], together with a ReduceLROnPlateau learning-rate scheduler from PyTorch. Early stopping with a patience of 200 epochs is applied based on validation objective (RG or ERM).

We use a polynomial Nonlinear AutoRegressive (NAR) model with structure selection via Accelerated Orthogonal Least Squares (AOLS) [43, 44] as the guide model. Guide forecasts are generated using a rolling-origin scheme in which, for each sliding window, the NAR model is fitted exclusively on observations available up to that window's origin. Full experimental specifications are provided in Appendix E.

4.3.2 Results and discussion

Table 5 summarizes the performance of the proposed RG training objective compared to the ERM baseline across various architectures and M4 frequency subsets. Performance is reported in terms of OWA, where lower values indicate superior forecasting accuracy. To provide a comprehensive comparison, we also report the relative OWA improvement (percentage reduction in OWA) in the ΔOWA column. The Spearman (ERM, guide) column reports the Spearman correlation between the absolute forecast-error rankings of the guide and the ERM-trained model. For Others, the reported Spearman value is the weighted mean of the correlations computed separately for Weekly, Daily, and Hourly, using their respective numbers of series as weights. Following the M4 aggregation procedure [17], sMAPE and MASE are aggregated using the same weights before computing OWA relative to the correspondingly aggregated Naive2 benchmark.

Table 5: Comparison between baseline and RG training on the M4 dataset.

Architecture	Subset	ERM OWA	RG OWA	ΔOWA (%)	Spearman (ERM, guide)
MLP	Yearly	1.525 ± 0.008	$\mathbf{1.362 \pm 0.001}$	**10.71**	0.73
GRU	Yearly	$\mathbf{1.401 \pm 0.039}$	1.414 ± 0.006	-0.98	0.75
TCN	Yearly	1.414 ± 0.053	$\mathbf{1.395 \pm 0.012}$	**1.29**	0.76
MLP	Quarterly	1.448 ± 0.039	$\mathbf{1.124 \pm 0.010}$	**22.35**	0.29
GRU	Quarterly	1.339 ± 0.011	$\mathbf{1.209 \pm 0.010}$	**9.70**	0.35
TCN	Quarterly	1.360 ± 0.034	$\mathbf{1.132 \pm 0.007}$	**16.74**	0.34
MLP	Monthly	1.216 ± 0.015	$\mathbf{1.045 \pm 0.017}$	**14.06**	0.41
GRU	Monthly	1.127 ± 0.022	$\mathbf{1.000 \pm 0.003}$	**11.26**	0.49
TCN	Monthly	1.111 ± 0.011	$\mathbf{0.995 \pm 0.007}$	**10.47**	0.50
MLP	Others	1.125 ± 0.021	$\mathbf{1.090 \pm 0.016}$	**3.07**	0.71
GRU	Others	1.075 ± 0.005	$\mathbf{1.000 \pm 0.019}$	**6.98**	0.77
TCN	Others	1.166 ± 0.023	$\mathbf{1.124 \pm 0.004}$	**3.60**	0.73

Overall, the RG objective demonstrates superior performance compared to ERM across most subsets and architectures, with one notable exception: GRU on the Yearly subset, where RG yields a marginal degradation of -0.98% in OWA. The empirical results in Table 5, complemented by the analysis in Figure 3, suggest that this exception may be part of a broader pattern: RG provides the most

significant improvements when the learner trained under ERM and the guide exhibit divergent error rankings (i.e., a low Spearman correlation). In these scenarios, the guide may provide complementary information that the ERM model has not captured, potentially enabling more effective adaptive reweighting.

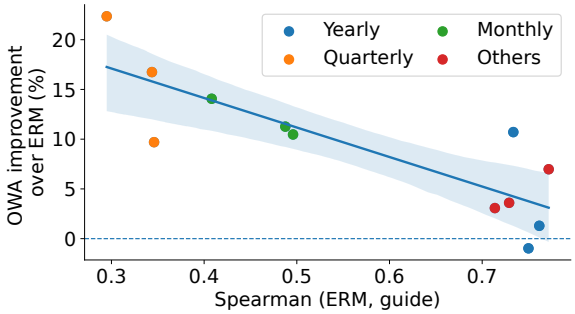


Figure 3: Relationship between ERM–guide alignment (Spearman correlation between ERM and guide errors) and relative OWA improvement of RG over ERM. Each point corresponds to a model and dataset subset.

The regression analysis in Figure 3 yields an $R^2 \approx 0.698$, indicating that ERM–guide alignment accounts for approximately 70% of the variance in observed OWA gains; the remaining variance likely reflects architecture-specific factors and subset heterogeneity not captured by alignment alone.

5 Conclusion

In this work, we introduced RG training, a novel objective that incorporates per-sample comparisons with a fixed guide model to modulate the contribution of training instances within the ERM framework. Unlike existing reference-based approaches such as knowledge distillation, our method does not rely on prediction imitation. Instead, it leverages the relative error between the learner and a guide to induce an adaptive, bounded, and direction-preserving reweighting of the gradients. Our theoretical analysis demonstrated that RG provides a principled mechanism for sample prioritization, ensuring learning focus is reallocated toward instances where the learner underperforms relative to a robust guide. Empirically, the method showed consistent improvements across diverse domains. In image classification, RG improves classification accuracy under synthetic and human annotation noise in most evaluated settings, with particularly large gains for Vision Transformers. In time-series forecasting, RG outperformed standard ERM in most settings.

Our results suggest that the effectiveness of RG is closely related to the alignment between the ERM-trained learner and the guide. The observed association between low ERM-guide agreement and high performance gains is consistent with RG acting as a corrective mechanism that exploits complementary information from the guide. When the guide's error ranking diverges from that of ERM, it may provide additional signals for adaptive sample weighting. Conversely, when alignment is already high, the guide offers limited incremental information and gains diminish, as observed in the GRU/Yearly setting of the M4 experiments. This dependence on the guide's robustness constitutes a limitation, because when the guide's signal is weak or systematically biased — e.g., if the guide assigns lower errors to noisy or corrupted samples — the resulting reweighting mechanism may amplify gradients on such samples.

Empirically validating the behavior of RG under systematically biased guides, exploring strategies to detect or mitigate such bias, and extending RG to incorporate multiple guides remain important directions for future work.

Funding and Competing Interests

This work was supported in part by JSPS KAKENHI Grant Number JP24K02944. I. R. Soares was supported by the Japanese Government (MEXT) Scholarship. P. Benedetti was supported by JST BOOST, Grant Number JPMJBS2408. The authors declare no competing interests.

References

[1] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009. ISBN 9780387848846. URL https://books.google.co.jp/books?id=eBSgoAEACAAJ.

[2] V. N. Vapnik. An overview of statistical learning theory. *Trans. Neur. Netw.*, 10(5):988–999, September 1999. ISSN 1045-9227. doi: 10.1109/72.788640. URL https://doi.org/10.1109/72.788640.

[3] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods. *Machine Learning*, 110(3):457–506, March 2021. ISSN 1573-0565. doi: 10.1007/s10994-021-05946-3. URL http://dx.doi.org/10.1007/s10994-021-05946-3.

[4] Christian Brownlees, Emilien Joly, and Gábor Lugosi. Empirical risk minimization for heavy-tailed losses. *The Annals of Statistics*, 43(6):2507–2536, 2015. ISSN 00905364. URL http://www.jstor.org/stable/43818859.

[5] Matthew Holland. Robustness and scalability under heavy tails, without strong convexity. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 865–873. PMLR, 13–15 Apr 2021. URL https://proceedings.mlr.press/v130/holland21a.html.

[6] Nagarajan Natarajan, Inderjit S Dhillon, Pradeep K Ravikumar, and Ambuj Tewari. Learning with noisy labels. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/3871bd64012152bfb53fdf04b401193f-Paper.pdf.

[7] Peter J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, March 1964. ISSN 0003-4851. doi: 10.1214/aoms/1177703732. URL http://dx.doi.org/10.1214/aoms/1177703732.

[8] Jonathan T. Barron. A general and adaptive robust loss function. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 4326–4334. IEEE, June 2019. doi: 10.1109/cvpr.2019.00446. URL http://dx.doi.org/10.1109/CVPR.2019.00446.

[9] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48. ACM, June 2009. doi: 10.1145/1553374.1553380. URL http://dx.doi.org/10.1145/1553374.1553380.

[10] M. Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. URL https://proceedings.neurips.cc/paper_files/paper/2010/file/e57c6b956a6521b28495f2886ca0977a-Paper.pdf.

[11] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

[12] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, March 2021. ISSN 1573-1405. doi: 10.1007/s11263-021-01453-z. URL http://dx.doi.org/10.1007/s11263-021-01453-z.

[13] Tae-Hoon Kim and Jonghyun Choi. Screenernet: Learning self-paced curriculum for deep neural networks, 2018. URL https://arxiv.org/abs/1801.00904.

[14] Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. Curriculum learning: A survey, 2021. URL https://arxiv.org/abs/2101.10382.

[15] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2304–2313. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/jiang18c.html.

[16] Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2006.03.001. URL https://www.sciencedirect.com/science/article/pii/S0169207006000239.

[17] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54–74, 2020. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2019.04.014. URL https://www.sciencedirect.com/science/article/pii/S0169207019301128. M4 Competition.

[18] Yunlong Feng, Xiaolin Huang, Lei Shi, Yuning Yang, and Johan A.K. Suykens. Learning with the maximum correntropy criterion induced losses for regression. *Journal of Machine Learning Research*, 16 (30):993–1034, 2015. URL http://jmlr.org/papers/v16/feng15a.html.

[19] Xiaolin Huang, Lei Shi, and Johan A.K. Suykens. Ramp loss linear programming support vector machine. *Journal of Machine Learning Research*, 15(64):2185–2211, 2014. URL http://jmlr.org/papers/v15/huang14a.html.

[20] Yitian Xu, Zhiji Yang, and Xianli Pan. A novel twin support-vector machine with pinball loss. *IEEE Transactions on Neural Networks and Learning Systems*, 28(2):359–370, 2017. doi: 10.1109/TNNLS.2015.2513006.

[21] Liming Yang and Hongwei Dong. Robust support vector machine with generalized quantile loss for classification and regression. *Applied Soft Computing*, 81:105483, 2019. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2019.105483. URL https://www.sciencedirect.com/science/article/pii/S1568494619302534.

[22] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4334–4343. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/ren18a.html.

[23] Jun Shu, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, Zongben Xu, and Deyu Meng. Meta-weight-net: Learning an explicit mapping for sample weighting. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/e58cc5ca94270acaceed13bc82dfedf7-Paper.pdf.

[24] Sören Mindermann, Jan M Brauner, Muhammed T Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Höltgen, Aidan N Gomez, Adrien Morisot, Sebastian Farquhar, and Yarin Gal. Prioritized training on points that are learnable, worth learning, and not yet learnt. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 15630–15649. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/mindermann22a.html.

[25] Zhenghao Lin, Zhibin Gou, Yeyun Gong, Xiao Liu, Yelong Shen, Ruochen Xu, Chen Lin, Yujiu Yang, Jian Jiao, Nan Duan, and Weizhu Chen. Not all tokens are what you need for pretraining. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 29029–29063. Curran Associates, Inc., 2024. doi: 10.52202/079017-0914. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/3322a9a72a1707de14badd5e552ff466-Paper-Conference.pdf.

[26] Amit Dhurandhar, Karthikeyan Shanmugam, and Ronny Luss. Enhancing simple models by exploiting what they already know. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2525–2534. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/dhurandhar20a.html.

[27] Mustafa Ç. Pinar and Stavros A. Zenios. On smoothing exact penalty functions for convex constrained optimization. *SIAM Journal on Optimization*, 4(3):486–511, 1994. doi: 10.1137/0804027. URL https://doi.org/10.1137/0804027.

[28] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. URL https://arxiv.org/abs/1708.07747.

[29] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[30] Jiaheng Wei, Zhaowei Zhu, Hao Cheng, Tongliang Liu, Gang Niu, and Yang Liu. Learning with noisy labels revisited: A study using real-world human annotations. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=TBWA6PLJZQm.

[31] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, January 1989. ISSN 0893-6080. doi: 10.1016/0893-6080(89)90020-8. URL http://dx.doi.org/10.1016/0893-6080(89)90020-8.

[32] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. ISSN 0018-9219. doi: 10.1109/5.726791. URL http://dx.doi.org/10.1109/5.726791.

[33] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 770–778. IEEE, June 2016. doi: 10.1109/cvpr.2016.90. URL http://dx.doi.org/10.1109/CVPR.2016.90.

[35] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2016. URL https://arxiv.org/abs/1608.03983.

[36] C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 100(3/4):441–471, 1987. ISSN 00029556. URL http://www.jstor.org/stable/1422689.

[37] Andrej N. Kolmogorov. Sulla determinazione empirica di una legge di distribuzione. *Giornale dell'Istituto Italiano degli Attuari*, 4:83–91, 1933.

[38] Nickolay V. Smirnov. Table for estimating the goodness of fit of empirical distributions. *The Annals of Mathematical Statistics*, pages 279–281, 1948.

[39] Cédric Villani. The wasserstein distances. In *Optimal transport: old and new*, pages 93–111. Springer, 2009.

[40] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, 2018. URL https://arxiv.org/abs/1803.01271.

[41] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014. URL https://arxiv.org/abs/1406.1078.

[42] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

[43] Stephen A Billings. *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio–Temporal Domains*. Wiley, July 2013. ISBN 9781118535561. doi: 10.1002/9781118535561. URL http://dx.doi.org/10.1002/9781118535561.

[44] Abolfazl Hashemi and Haris Vikalo. Accelerated orthogonal least-squares for large-scale sparse reconstruction. *Digital Signal Processing*, 82:91–105, November 2018. ISSN 1051-2004. doi: 10.1016/j.dsp.2018.07.010. URL http://dx.doi.org/10.1016/j.dsp.2018.07.010.

[45] Benjamin Scellier, Maxence Ernoult, Jack Kendall, and Suhas Kumar. Energy-based learning algorithms for analog computing: a comparative study. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 52705–52731. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/a52b0d191b619477cc798d544f4f0e4b-Paper-Conference.pdf.

[46] Azul Garza, Max Mergenthaler Canseco, Cristian Challú, and Kin G. Olivares. StatsForecast: Lightning fast forecasting with statistical and econometric models. PyCon US 2022, Salt Lake City, Utah, USA, 2022. URL https://github.com/Nixtla/statsforecast.

[47] Wilson Rocha Lacerda, Luan Pascoal Costa da Andrade, Samuel Carlos Pessoa Oliveira, and Samir Angelo Milani Martins. Sysidentpy: A python package for system identification using narmax models. *Journal of Open Source Software*, 5(54):2384, 2020. doi: 10.21105/joss.02384. URL https://doi.org/10.21105/joss.02384.

[48] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 929–947. ACM, April 2024. doi: 10.1145/3620665.3640366. URL `http://dx.doi.org/10.1145/3620665.3640366`.

[49] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL `https://aclanthology.org/2020.emnlp-demos.6/`.

[50] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, 2019.

[51] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. URL `http://jmlr.org/papers/v12/pedregosa11a.html`.

[52] Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández Del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E Oliphant. Array programming with NumPy. *Nature*, 585(7825): 357–362, September 2020.

[53] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J van der Walt, Matthew Brett, Joshua Wilson, K Jarrod Millman, Nikolay Mayorov, Andrew R J Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E A Quintero, Charles R Harris, Anne M Archibald, Antônio H Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: fundamental algorithms for scientific computing in python. *Nat. Methods*, 17(3):261–272, March 2020.

A Theoretical properties

This section analyzes the curvature of the RG objective and derives sufficient conditions for convexity and Lipschitz continuity of its gradient.

A.1 Curvature modification

Proposition 1 (Curvature Modification). Assume $L_i(\theta)$ is twice differentiable. For $r_i(\theta) \notin \{0, \delta\}$, the Hessian of the RG objective is obtained by differentiating the gradient expression

$$\nabla_\theta \mathcal{L}_i(\theta) = \omega_i(\theta) \nabla_\theta L_i(\theta)$$

Since $\omega_i(\theta) = 1 + \lambda \phi'_\delta(r_i(\theta))$ and $r_i(\theta) = L_i(\theta) - \gamma L_i^{\text{guide}}$, applying the product rule gives

$$\nabla^2_\theta \mathcal{L}_i(\theta) = \omega_i(\theta) \nabla^2_\theta L_i(\theta) + \lambda \phi''_\delta(r_i(\theta)) \nabla_\theta L_i(\theta) \nabla_\theta r_i(\theta)^\top. \tag{13}$$

Since L_i^{guide} is constant with respect to θ, we have $\nabla_\theta r_i(\theta) = \nabla_\theta L_i(\theta)$, yielding

$$\nabla^2_\theta \mathcal{L}_i(\theta) = \omega_i(\theta) \nabla^2_\theta L_i(\theta) + \lambda \phi''_\delta(r_i(\theta)) \nabla_\theta L_i(\theta) \nabla_\theta L_i(\theta)^\top. \tag{14}$$

The second derivative of ϕ_δ satisfies

$$\phi''_\delta(x) = \begin{cases} 0, & x < 0, \\ \dfrac{1}{\delta}, & 0 < x < \delta, \\ 0, & x > \delta. \end{cases} \tag{15}$$

Substituting $\phi''_\delta(r_i(\theta))$ and $\omega_i(\theta)$ in each regime, the Hessian admits the piecewise form

$$\nabla^2_\theta \mathcal{L}_i(\theta) = \begin{cases} \nabla^2_\theta L_i(\theta), & r_i(\theta) < 0, \\ \omega_i(\theta) \nabla^2_\theta L_i(\theta) + \dfrac{\lambda}{\delta} \nabla_\theta L_i(\theta) \nabla_\theta L_i(\theta)^\top, & 0 < r_i(\theta) < \delta, \\ (1+\lambda) \nabla^2_\theta L_i(\theta), & r_i(\theta) > \delta. \end{cases} \tag{16}$$

In the transition region $0 < r_i(\theta) < \delta$, the additional term is positive semidefinite and has rank at most one (rank one when $\lambda > 0$ and $\nabla_\theta L_i(\theta) \neq 0$). Relative to the rescaled base Hessian, it adds nonnegative curvature along directions with a component parallel to the loss gradient. Outside the transition region, the Hessian equals the ERM Hessian ($r_i < 0$) or its uniform scaling by $1 + \lambda$ ($r_i > \delta$). At $r_i \in \{0, \delta\}$, the RG objective is continuously differentiable, but its Hessian need not exist when $\lambda > 0$.

A.2 Convexity and Lipschitz properties

Proposition 2 (Convexity and Lipschitz Gradient Bound). Let $\Theta \subseteq \mathbb{R}^d$ be a convex set. Assume:

- $L_i(\theta)$ is differentiable and convex on Θ
- $\nabla_\theta L_i(\theta)$ is H_L-Lipschitz on Θ
- $\|\nabla_\theta L_i(\theta)\| \leq G$ for all $\theta \in \Theta$

Then $\mathcal{L}_i(\theta)$ is convex on Θ and its gradient is Lipschitz continuous there with an admissible constant $H_{\mathcal{L}} = (1 + \lambda) H_L + \lambda G^2 / \delta$.

Proof sketch. Convexity follows from the fact that ϕ_δ is convex and non-decreasing, and $L_i(\theta)$ is convex, so their composition $\phi_\delta(r_i(\theta))$ is convex. Since $\mathcal{L}_i(\theta) = L_i(\theta) + \lambda\, \phi_\delta(r_i(\theta))$ is a non-negative linear combination of convex functions, it is convex.

For the Lipschitz bound, we first bound $|\omega_i(\theta) - \omega_i(\theta')|$. Since $\omega_i(\theta) = 1 + \lambda\phi'_\delta(r_i(\theta))$ and ϕ'_δ is $\frac{1}{\delta}$-Lipschitz,

$$\begin{aligned}
|\omega_i(\theta) - \omega_i(\theta')| &= \lambda|\phi'_\delta(r_i(\theta)) - \phi'_\delta(r_i(\theta'))| \\
&\leq \frac{\lambda}{\delta}|r_i(\theta) - r_i(\theta')| \\
&= \frac{\lambda}{\delta}|L_i(\theta) - L_i(\theta')| \\
&\leq \frac{\lambda G}{\delta}\|\theta - \theta'\|, \qquad (17)
\end{aligned}$$

where the equality follows from the guide loss being fixed, and the last inequality follows from the mean value theorem and the uniform gradient bound on the convex set Θ.

We now bound the gradient difference. Applying the triangle inequality to $\nabla_\theta\mathcal{L}_i(\theta) = \omega_i(\theta)\nabla_\theta L_i(\theta)$,

$$\begin{aligned}
&\|\nabla_\theta\mathcal{L}_i(\theta) - \nabla_\theta\mathcal{L}_i(\theta')\| \\
&= \|\omega_i(\theta)\nabla_\theta L_i(\theta) - \omega_i(\theta')\nabla_\theta L_i(\theta')\| \\
&\leq \omega_i(\theta)\|\nabla_\theta L_i(\theta) - \nabla_\theta L_i(\theta')\| + \|\nabla_\theta L_i(\theta')\||\omega_i(\theta) - \omega_i(\theta')| \\
&\leq (1+\lambda)\cdot H_L\|\theta - \theta'\| + G\cdot\frac{\lambda G}{\delta}\|\theta - \theta'\| \\
&= \left((1+\lambda)H_L + \frac{\lambda G^2}{\delta}\right)\|\theta - \theta'\|, \qquad (18)
\end{aligned}$$

where the final inequality uses $\omega_i(\theta) \leq 1+\lambda$, the H_L-Lipschitz continuity of $\nabla_\theta L_i$, $\|\nabla_\theta L_i(\theta')\| \leq G$, and the bound derived above. Therefore, an admissible Lipschitz constant is

$$H_{\mathcal{L}} = (1+\lambda)H_L + \frac{\lambda G^2}{\delta}. \qquad (19)$$

The curvature expression characterizes how RG modifies the local loss landscape. Convexity and the Lipschitz gradient bound are guaranteed under the assumptions of Proposition 2, which are not generally satisfied by deep neural networks. The bound $H_{\mathcal{L}}$ increases linearly with λ and includes a term inversely proportional to δ, which shows how amplification strength and transition sharpness affect the upper bound on gradient variation. Under these assumptions, RG satisfies the convexity and smoothness conditions commonly used in standard gradient-based optimization analyses.

B General experiments details

B.1 Randomness control

A fixed global random seed of 42 is used. Individual experiment seeds are deterministically generated as $42 + r$, where r denotes the run index starting from 1, or the trial index in hyperparameter search experiments.

B.2 Hardware implementation details

Experiments were conducted on a workstation with:

- AMD Ryzen 9 7950X CPU
- NVIDIA RTX 5080 GPU (16GB)
- Ubuntu 24.04.4 LTS
- CUDA 12.8
- 128GB RAM

All experiments could be completed within a total budget of up to 168 GPU hours. Individual experiment runs required less than 4 GPU hours each.

C Fashion-MNIST experimental details

C.1 Dataset preparation

Fashion-MNIST [28] is split into 54,000 training samples and 6,000 validation samples using a fixed random partition with a controlled seed. The original test set of 10,000 samples is used for evaluation.

Input images are normalized using dataset statistics, with mean 0.2860 and SD of 0.3530, following [45]. Data augmentation is applied only to the training set, consisting of random cropping with padding and random horizontal flipping.

Synthetic symmetric label noise is injected exclusively into the training set to test robustness under label corruption conditions. For a given noise ratio ρ, a fraction ρ of training samples is selected uniformly at random, and their labels are replaced with a class drawn uniformly from the remaining classes. Noise generation is controlled by a fixed random seed (see B.1) and cached to ensure consistency across runs. Three noise levels are considered: $\rho \in \{0\%, 40\%, 80\%\}$.

The validation and test sets remain clean and are never corrupted. For RG training, guide model outputs are precomputed on non-augmented inputs and stored as per-sample logits, which are then used during training as auxiliary signals.

C.2 Architecture definitions

This subsection summarizes the architectures used in the classification experiments. All models were implemented using the software and libraries described in Appendix F.2. Tables report the structural components and searchable parameters, and their ranges, used during stage 1 of hyperparameter search in Appendix C.4.

C.2.1 Multilayer perceptron

The MLP processes flattened inputs and consists of a sequence of fully connected layers followed by a linear output layer.

Table 6 summarizes the MLP configuration space.

Table 6: MLP architecture configuration.

Component	Hyperparameter search range specification
Input representation	Flattened input vector
Hidden layers	1–3
Hidden dimensions	$\{128, 256, 512, 1024\}$
Dropout	$[0.0, 0.3]$
Activation	ReLU
Output layer	Fully connected (classification logits)

Each hidden layer follows the structure:

$$\text{Linear} \rightarrow \text{ReLU} \rightarrow \text{Dropout}$$

The final output layer produces 10 logits corresponding to the number of target classes in the Fashion-MNIST dataset.

C.2.2 Convolutional neural network

The CNN processes two-dimensional inputs using three stacked convolutional blocks. Each block applies convolution, ReLU activation, dropout, and max pooling to progressively reduce spatial resolution. The number of output channels is independently selected for each block.

Table 7 summarizes the CNN design space.

Each convolutional block follows:

Table 7: CNN architecture configuration.

Component	Hyperparameter search range specification
Input channels	1 (grayscale input)
Convolutional blocks	3
Block 1 channels	$\{16, 32, 64\}$
Block 2 channels	$\{32, 64, 128\}$
Block 3 channels	$\{64, 128, 256\}$
Kernel sizes	$\{3, 5\}$
Padding	Same padding
Dropout	$[0.0, 0.3]$
Pooling	MaxPool2D (stride 2)
Dense layer	$\{64, 128, 196\}$ units
Activation	ReLU

$$\text{Conv2D} \rightarrow \text{ReLU} \rightarrow \text{Dropout} \rightarrow \text{MaxPool}$$

After the convolutional stack, feature maps are flattened and processed through a fully connected classifier, which then produces 10 logits corresponding to the number of target classes in the Fashion-MNIST dataset.

C.2.3 Vision transformer

The ViT processes two-dimensional inputs by dividing them into fixed-size patches, which are linearly embedded and fed into a standard Transformer encoder. The classification token (`[CLS]`) produced by the encoder is passed to a classifier head to produce the final predictions.

Table 8 summarizes the ViT configuration space.

Table 8: ViT architecture configuration.

Component	Hyperparameter search range specification
Input size	28×28 (grayscale)
Patch size	4×4
Hidden size	$\{64, 96, 128\}$
Attention heads	$\{2, 4\}$
Encoder layers	$[2, 6]$
MLP ratio	$\{2, 4\}$
Intermediate size	Hidden size $\times$ MLP ratio
Dropout	$[0.05, 0.3]$

Each input image is divided into $7 \times 7 = 49$ non-overlapping patches of size 4×4. The patches are linearly projected into the hidden space and processed by a standard Transformer encoder consisting of multi-head self-attention and feed-forward blocks:

$$\text{Patch embedding} \rightarrow \text{Transformer encoder} \times L \rightarrow \text{CLS token} \rightarrow \text{Classifier head}$$

The classifier head processes the `[CLS]` token representation through a two-layer MLP:

$$\text{Linear}(d_h, 128) \rightarrow \text{ReLU} \rightarrow \text{Dropout} \rightarrow \text{Linear}(128, 10)$$

where d_h denotes the hidden size. The final output produces 10 logits corresponding to the number of target classes in the Fashion-MNIST dataset.

C.2.4 ResNet20 (guide)

The main guide model (reference) for the classification experiments uses a ResNet architecture adapted for grayscale inputs [34]. Specifically, we employ a 20-layer configuration corresponding to

the $6n + 2$ design with $n = 3$, commonly referred to as ResNet20. It consists of three stages with channel sizes [16, 32, 64], each composed of residual blocks with two 3×3 convolutions.

Table 9 summarizes the ResNet20 architecture.

Table 9: Reference ResNet20 architecture configuration.

Component	Specification
Input channels	1 (grayscale input)
Initial convolution	3×3, 16 channels
Residual stages	3
Blocks per stage	$\{3, 3, 3\}$
Stage channels	$\{16, 32, 64\}$
Downsampling	Stride 2 (stages 2 and 3)
Normalization	BatchNorm
Pooling	Global average pooling
Classifier	Fully connected layer
Output dimension	Number of classes
Activation	ReLU

The network follows a standard residual architecture composed of stacked residual blocks. Each residual block consists of two convolutional layers with batch normalization and ReLU activation, combined through an identity shortcut connection.

Each residual block follows:

$$\text{Conv}(3 \times 3) \rightarrow \text{BatchNorm} \rightarrow \text{ReLU} \rightarrow \text{Conv}(3 \times 3) \rightarrow \text{BatchNorm} \rightarrow \text{Shortcut} \rightarrow \text{ReLU}$$

Spatial resolution is reduced using stride-2 convolutions at the beginning of stages 2 and 3. Channel dimensionality increases across stages from 16 to 32 and then to 64 channels.

After the final residual stage, global average pooling is applied followed by a fully connected classifier that produces 10 output logits.

C.2.5 Simple multilayer perceptron (ablation guide)

To investigate the sensitivity of the RG objective to the quality and capacity of the guidance signal, we designed a simple MLP to serve as a guide in ablation studies. This model is intentionally constrained to a very low capacity to evaluate whether even a weak reference can provide informative signals for sample reweighting.

Table 10 summarizes the configuration of this simplified architecture.

Table 10: Simple MLP architecture used as a minimal guide in ablation studies.

Component	Specification
Input representation	Flattened input vector ($28 \times 28 = 784$)
Hidden layers	1
Hidden dimensions	8
Dropout	0.2
Activation	ReLU
Output layer	Fully connected (10 logits)

The hidden layer follows a fixed structure:

$$\text{Linear}(784, 8) \rightarrow \text{ReLU} \rightarrow \text{Dropout}(0.2)$$

Unlike the high-capacity ResNet20 guide, this model is designed to provide a coarse ranking of sample difficulty, allowing us to test the hypothesis that the diversity and structure of the error signal are more critical for RG performance gains than the absolute predictive accuracy of the guide.

C.3 Method definitions

In this section, we define the implementation of the methods we used to compare against RG.

C.3.1 Self paced learning

As a baseline for self-guided sample reweighting, we implement the SPL strategy proposed by Kumar et al. [10]. SPL incorporates a curriculum learning mechanism that prioritizes "easy" samples (those with low training loss) and gradually introduces "harder" samples as the training progresses.

Specifically, the weighting scheme assigns a binary weight $v_i \in \{0, 1\}$ to each training sample i based on its per-sample loss $L_i(\theta)$. The weight is determined by a pace parameter λ according to:

$$v_i = \begin{cases} 1, & \text{if } L_i(\theta) < \lambda \\ 0, & \text{if } L_i(\theta) \geq \lambda \end{cases} \tag{20}$$

The parameter λ serves as a dynamic threshold that determines the learner's "capacity" at a given epoch. To simulate the curriculum transition from simple to complex patterns, λ is increased over time following an exponential schedule:

$$\lambda_t = \lambda_0 \cdot k^t \tag{21}$$

where t denotes the epoch index, λ_0 is the initial difficulty threshold, and $k > 1$ is the step size or growth factor. In our experiments, λ_0 and k are treated as hyperparameters and optimized for each architecture. Unlike our proposed RG objective, which utilizes an external reference signal to modulate sample importance, SPL relies solely on the learner's internal loss distribution to determine the training curriculum.

C.3.2 Knowledge distillation

To evaluate the proposed method against established reference-based learning strategies, we implement KD as proposed by Hinton et al. [11]. Unlike our Reference-Guided objective, which utilizes a reference model to modulate sample importance, KD aims to transfer knowledge by encouraging the learner (student) to imitate the predictive distribution of the reference model (teacher).

The KD objective is a linear combination of the standard cross-entropy loss with the ground-truth labels and a distillation term based on the Kullback-Leibler (KL) divergence between the student and teacher distributions. To extract richer information from the teacher's logits, both distributions are softened using a temperature parameter $T > 1$. The total loss is defined as:

$$\mathcal{L}_{\text{KD}} = (1 - \alpha)\mathcal{L}_{\text{CE}}(y, \sigma(\mathbf{z}_s)) + \alpha T^2 D_{\text{KL}}\left(\sigma(\mathbf{z}_t/T) \parallel \sigma(\mathbf{z}_s/T)\right) \tag{22}$$

where $\mathbf{z}_s$ and $\mathbf{z}_t$ denote the logits produced by the student and teacher models, respectively, $\sigma(\cdot)$ is the softmax function, and $\alpha \in [0, 1]$ is a weighting hyperparameter. The term T^2 is included to ensure that the magnitude of the distillation gradient remains consistent when the temperature is scaled.

Consistent with our experimental protocol, the teacher's logits are detached from the computational graph to ensure the reference remains fixed.

C.3.3 MentorNet-PD

As an additional sample-reweighting baseline, we implement the predefined-curriculum MentorNet-PD proposed by Jiang et al. [15]. Unlike SPL, which directly determines sample weights from a predefined loss threshold, MentorNet-PD trains an auxiliary neural network (MentorNet) to approximate a predefined curriculum and subsequently uses it to assign weights during StudentNet training.

The MentorNet is pretrained on 100,000 synthetic feature vectors sampled from a grid of per-sample losses, loss-threshold differences, and training progress values. Target weights are generated by the predefined curriculum, which assigns uniform weights during the initial stage, binary weights based

on sample difficulty during the intermediate stage, and continuous weights during the final stage. Specifically, the target weighting policy is defined as

$$v_i^* = \begin{cases} 1, & t \leq 18, \\ \mathbb{I}(d_i \leq 0), & 18 < t < 90, \\ \mathrm{clip}(1 - d_i, 0, 1), & t \geq 90, \end{cases} \tag{23}$$

where d_i denotes the difference between the sample loss and its reference threshold, $\mathbb{I}(\cdot)$ is the indicator function, equal to one when its condition holds and zero otherwise, $\mathrm{clip}(x, 0, 1) = \min(1, \max(0, x))$, and $t \in \{0, \ldots, 99\}$ denotes training progress discretized into 100 levels over the StudentNet training schedule. A small recurrent MentorNet is trained for 30,000 optimization steps using mean squared error between predicted and target weights.

After pretraining, the MentorNet remains frozen throughout StudentNet optimization. For each minibatch, it receives the StudentNet's per-sample cross-entropy losses, their differences from an exponentially smoothed loss-percentile threshold, and training progress. The class-label feature is fixed to zero in our implementation. The resulting weights, bounded between zero and one by a sigmoid function, define the StudentNet objective:

$$\mathcal{L}_{\text{MentorNet-PD}} = \frac{1}{B} \sum_{i=1}^{B} v_i L_i(\theta), \tag{24}$$

where B is the minibatch size and v_i denotes the MentorNet-predicted weight after stochastic sample masking. Outside an initial burn-in period with uniform weights and no masking, random masking excludes 50% of samples during the first 17% of training progress and 5% thereafter. Following Jiang et al. [15], the weight decay coefficient is scaled by the mean sample weight in each minibatch. The loss percentile, moving-average decay, and burn-in duration are tuned jointly with the StudentNet LR.

C.4 Hyperparameter search

Hyperparameter search was conducted in two stages. In both stages, the optimizer, scheduler, and training configuration were fixed as follows: SGD with momentum 0.9, weight decay 5×10^{-4}, batch size 256, and a cosine annealing LR scheduler with $T_{\max} = 300$ (the number of scheduler updates, performed once per epoch) and a minimum LR of $\eta_{\min} = 10^{-5}$.

Stage 1: architecture hyperparameters Using clean labels and a fixed LR of 0.005, we performed 20 architecture hyperparameter optimization trials of 30 epochs (using early stopping with patience set to 15 epochs) each for MLP, CNN, and ViT under the ERM objective. Architecture search was guided by validation cross-entropy loss, using a Tree-structured Parzen Estimator (TPE) sampler with 5 initial random trials. The selected architecture hyperparameters are reported in Table 11 and were fixed for all subsequent experiments.

Stage 2: objective hyperparameters For each learner architecture, method (RG, KD, SPL, and MentorNet-PD), and label-noise level in the main experiment, an independent hyperparameter search was performed using the architecture selected in Stage 1. Each search ran 20 trials of 300 epochs, with early stopping after 30 epochs without improvement in validation cross-entropy, selecting the LR from $\{0.001, 0.005, 0.01\}$ jointly with method-specific hyperparameters and minimizing validation cross-entropy using TPE sampling with 5 startup trials. For ERM, as the search space is limited to the three learning-rate candidates, the search was performed until exhausting the search space. Final models were trained for 300 epochs, with checkpoints selected according to each method's validation objective.

For the main comparison experiment (Table 1), to ensure a fair comparison with methods that do not use a reference model, the guide model was constrained to have access to the same training data as the learner — that is, the guide label noise ratio was set equal to the learner label noise ratio. Accordingly, KD was evaluated only under matched noise conditions (0%, 40%, 80%). RG was additionally evaluated under mismatched guide noise conditions in the ablation study (Section 4.1.3), where the guide noise ratio was varied independently of the learner noise ratio.

Table 11: Selected architecture hyperparameters from Stage 1 search.

Model	Hyperparameter	Value
CNN	Channels (layer 0)	64
	Channels (layer 1)	32
	Channels (layer 2)	256
	Kernel size	5
	Dropout	0.0367
	Dense dim	128
MLP	Number of layers	3
	Hidden dim (layer 0)	1024
	Hidden dim (layer 1)	1024
	Hidden dim (layer 2)	128
	Dropout	0.0035
ViT	Hidden size	128
	Attention heads	4
	Hidden layers	3
	MLP ratio	4
	Dropout	0.0999

The search ranges for each method are as follows. For RG, the tuned hyperparameters are λ, γ, and δ:

$$\lambda \in [0.25,\ 16.0]\ (\log\text{-scale}), \quad \gamma \in [0.8,\ 1.2], \quad \delta \in [10^{-4},\ 10^{-1}]\ (\log\text{-scale}).$$

For KD, the tuned hyperparameters are the interpolation coefficient α and the temperature τ:

$$\alpha \in [0.1,\ 0.9], \quad \tau \in [1.0,\ 8.0]\ (\log\text{-scale}).$$

For SPL (hard variant), the tuned hyperparameters are the initial pacing threshold λ_0 and the growth factor k:

$$\lambda_0 \in [0.01,\ 2.0]\ (\log\text{-scale}), \quad k \in [1.05,\ 2.0].$$

For MentorNet-PD, the tuned hyperparameters are the loss percentile p, the initial uniform-weight duration b (as a percentage of training), and the moving-average decay β:

$$p \in [0.50,\ 0.95], \quad b \in \{0, \ldots, 40\}, \quad \beta \in [0.10,\ 0.90].$$

Selected values for RG, KD, SPL, and MentorNet-PD are reported in Tables 12, 13, 14, and 15, respectively.

C.5 Post-tuning training configuration

After the two search stages, five independent runs of 300 epochs were trained per condition using the selected architecture, learning rate, and method-specific hyperparameters. Training used SGD with momentum 0.9, weight decay 5×10^{-4}, batch size 256, and cosine annealing with $T_{\max} = 300$ and $\eta_{\min} = 10^{-5}$. Unlike the architecture-search stage, the selected LR varies by method and experimental condition; see Tables 12, 13, 14, and 15.

C.6 Guide models detailed results

Table 16 summarizes the standalone performance of the guide models under different noise levels.

C.7 Additional analyses

C.7.1 Hyperparameter ablations

Table 17 presents hyperparameter and guide ablations for CNN trained under 40% label noise using guides trained on clean labels. Note that ERM is a special case of the proposed formulation obtained when $\lambda = 0$, in which case $\mathcal{L}_i(\theta)$ reduces to the standard per-sample loss $L_i(\theta)$.

Table 12: Selected RG hyperparameters for evaluated learner and guide noise conditions.

Model	Noise (%)	Guide noise (%)	LR	λ	γ	δ
CNN	0	0	0.001	12.872	0.954	0.003862
CNN	40	0	0.005	4.752	1.188	0.0001153
CNN	40	40	0.005	4.752	1.188	0.0001153
CNN	40	80	0.005	4.752	1.188	0.0001153
CNN	80	0	0.001	14.394	0.839	0.0002115
CNN	80	80	0.005	12.644	1.184	0.02213
MLP	0	0	0.005	1.304	1.067	0.009997
MLP	40	0	0.001	14.394	0.839	0.0002115
MLP	40	40	0.01	6.734	1.158	0.0001875
MLP	40	80	0.01	0.269	1.191	0.01245
MLP	80	0	0.005	13.214	0.894	0.009448
MLP	80	80	0.005	12.644	1.184	0.02213
ViT	0	0	0.01	14.786	0.864	0.0003393
ViT	40	0	0.01	13.172	0.876	0.002798
ViT	40	40	0.01	6.734	1.158	0.0001875
ViT	40	80	0.005	0.301	1.187	0.002087
ViT	80	0	0.001	14.394	0.839	0.0002115
ViT	80	80	0.001	15.584	1.170	0.00203

Table 13: Selected KD hyperparameters for Fashion-MNIST.

Model	Noise (%)	LR	α	T
CNN	0	0.01	0.579	7.258
CNN	40	0.01	0.529	6.092
CNN	80	0.01	0.138	2.401
MLP	0	0.01	0.847	1.477
MLP	40	0.01	0.879	2.716
MLP	80	0.01	0.892	1.158
ViT	0	0.01	0.393	2.582
ViT	40	0.005	0.877	2.354
ViT	80	0.01	0.888	1.028

Table 14: Selected SPL hyperparameters for Fashion-MNIST.

Model	Noise (%)	LR	λ_0	k
CNN	0	0.01	1.826	1.124
CNN	40	0.01	1.765	1.444
CNN	80	0.01	0.090	1.986
MLP	0	0.01	1.146	1.618
MLP	40	0.01	0.181	1.226
MLP	80	0.01	0.370	1.813
ViT	0	0.005	0.197	1.661
ViT	40	0.001	0.021	1.997
ViT	80	0.001	0.457	1.847

Table 15: Selected MentorNet-PD hyperparameters for Fashion-MNIST.

Model	Noise (%)	LR	Percentile p	Burn-in b (%)	EMA decay β
CNN	0	0.01	0.754	34	0.122
CNN	40	0.01	0.583	12	0.520
CNN	80	0.005	0.731	24	0.137
MLP	0	0.01	0.927	39	0.747
MLP	40	0.01	0.583	12	0.520
MLP	80	0.01	0.583	12	0.520
ViT	0	0.005	0.731	24	0.137
ViT	40	0.01	0.583	12	0.520
ViT	80	0.005	0.731	24	0.137

Table 16: Guide model clean-test accuracy (mean $\pm$ sample SD over five runs).

Guide	Noise (%)	Test accuracy (%)
MLP	0	74.92 ± 0.52
ResNet20	0	93.48 ± 0.15
ResNet20	40	90.49 ± 0.35
ResNet20	80	73.02 ± 3.84

The ablation results indicate that RG exhibits limited sensitivity to variations in δ and γ across the evaluated settings. Increasing λ from 0.5 to 18 improves mean accuracy from 90.68% to 92.17%, without evidence of instability within the tested range. Replacing the ResNet20 guide with a simple MLP reduces accuracy from 91.79% to 87.89%, showing the influence of guide selection on RG performance.

C.7.2 Guide-quality analysis

To investigate the influence of guide quality on RG performance, we evaluate guides trained under different label-noise ratios for fixed learner corruption levels. Table 18 reports the results across CNN, MLP, and ViT architectures.

C.7.3 Optimization controls

RG amplifies per-sample gradients through its guide-relative weighting, raising the possibility that some of its performance gains arise from an effect similar to increasing the learning rate. To investigate this possibility, two optimization controls were designed to assess the contributions of learning-rate selection and gradient scaling to RG performance. ERM @ RG LR uses the LR selected for RG with the standard cross-entropy objective. RG mean-one preserves the guide-relative per-sample weighting but normalizes the scaling factors within each minibatch to have unit mean. Both controls

Table 17: Ablation study of RG hyperparameters. Each row reports mean accuracy (%) $\pm$ sample SD over five runs.

Ablation	**Method**	**Guide**	$\boldsymbol{\lambda}$	$\boldsymbol{\delta}$	$\boldsymbol{\gamma}$	**Accuracy $\pm$ std (%)**
λ	ERM	—	0	—	—	90.18 ± 0.42
	RG	ResNet20	0.5	10^{-3}	1.0	90.68 ± 0.44
	RG	ResNet20	3	10^{-3}	1.0	91.79 ± 0.28
	RG	ResNet20	18	10^{-3}	1.0	92.17 ± 0.08
δ	RG	ResNet20	3	10^{-1}	1.0	91.63 ± 0.17
γ	RG	ResNet20	3	10^{-3}	0.5	91.53 ± 0.29
	RG	ResNet20	3	10^{-3}	1.5	91.94 ± 0.30
Guide	RG	MLP	3	10^{-3}	1.0	87.89 ± 0.57

Table 18: RG accuracy (%, mean ± sample SD over five runs) across learner and ResNet20 guide noise ratios.

Learner	Noise	Guide 0%	Guide 40%	Guide 80%
CNN	40%	91.98 ± 0.15	91.37 ± 0.07	90.37 ± 0.13
CNN	80%	89.67 ± 0.29	—	81.92 ± 0.28
MLP	40%	90.42 ± 0.14	89.09 ± 0.24	88.06 ± 0.27
MLP	80%	86.87 ± 0.19	—	79.53 ± 1.06
ViT	40%	90.93 ± 0.49	89.63 ± 0.59	84.81 ± 0.51
ViT	80%	84.61 ± 0.80	—	75.83 ± 0.60

follow the established final training and evaluation protocol, reusing the relevant hyperparameters selected for RG (see Table 12).

Table 19: Optimization controls on Fashion-MNIST (test accuracy in %, mean ± sample SD over five runs).

Model	Noise	ERM	ERM @ RG LR	RG	RG mean-one
CNN	0%	93.04 ± 0.22	91.73 ± 0.08	92.95 ± 0.22	91.97 ± 0.14
CNN	40%	90.18 ± 0.42	90.29 ± 0.13	91.37 ± 0.07	91.64 ± 0.12
CNN	80%	76.09 ± 4.48	79.65 ± 0.95	81.92 ± 0.28	81.30 ± 0.61
MLP	0%	91.19 ± 0.07	90.89 ± 0.04	91.25 ± 0.06	90.85 ± 0.16
MLP	40%	88.02 ± 0.33	88.02 ± 0.32	89.09 ± 0.24	89.26 ± 0.37
MLP	80%	80.91 ± 0.29	79.91 ± 0.44	79.53 ± 1.06	80.09 ± 0.99
ViT	0%	88.81 ± 1.09	89.10 ± 0.44	91.85 ± 0.61	88.93 ± 0.57
ViT	40%	84.87 ± 0.58	83.75 ± 0.42	89.63 ± 0.59	86.97 ± 0.29
ViT	80%	63.71 ± 3.44	62.02 ± 2.10	75.83 ± 0.60	75.75 ± 0.80

Table 19 shows that matching RG's LR does not reproduce its improvements, particularly for ViT under noisy labels. Mean-one normalization reduces ViT accuracy at 40% noise but produces comparable results at 80%. These findings indicate that RG's benefits cannot be explained solely by a different learning-rate selection.

C.7.4 Cross-fitted guides

We evaluate two-fold cross-fitting to investigate whether RG's improvements depend on guide losses computed on examples used to train the guide. The noisy training set is partitioned into two disjoint folds, with a separate ResNet20 guide trained on each fold. Each training example receives its reference loss from the guide trained on the complementary fold, ensuring that the guidance used during learner training is computed out of fold. The learner retains access to the full training set, and each guide is trained on half of the available examples. Cross-fitted RG uses the LR and objective hyperparameters selected for the corresponding matched-noise RG configuration (Table 12), with no additional hyperparameter search.

Table 20: Effect of two-fold cross-fitted guides on Fashion-MNIST clean-test accuracy (%, mean ± sample SD).

Model	Noise	ERM	RG	RG cross-fitted
CNN	40%	90.18 ± 0.42	91.37 ± 0.07	91.08 ± 0.24
CNN	80%	76.09 ± 4.48	81.92 ± 0.28	77.84 ± 0.61
MLP	40%	88.02 ± 0.33	89.09 ± 0.24	89.05 ± 0.29
MLP	80%	80.91 ± 0.29	79.53 ± 1.06	77.60 ± 0.67
ViT	40%	84.87 ± 0.58	89.63 ± 0.59	89.47 ± 0.40
ViT	80%	63.71 ± 3.44	75.83 ± 0.60	75.18 ± 0.23

Table 20 shows that cross-fitted RG achieves performance close to standard RG at 40% noise across all three architectures and retains most of its improvement over ERM for ViT at 80% noise. Since each training sample is evaluated by a guide trained on the complementary fold, these results suggest that RG can still benefit from guide-relative loss comparisons even when the reference information is obtained out of sample.

C.7.5 Guide memorization and weight diagnostics

To assess whether the ResNet20 guides memorize corrupted labels, we examine training samples whose labels were altered by synthetic noise. We compare guide predictions with the corrupted labels used during training (observed labels) and the corresponding clean labels before corruption (original labels). Table 21 reports prediction agreement rates and cross-entropy losses on these samples.

Table 21: Guide predictions on training samples with changed labels. Agreement rates and cross-entropy are computed against the corrupted and original labels on this subset.

Guide noise	Agreement with corrupted labels (%)	Agreement with original labels (%)	CE against corrupted labels	CE against original labels
40%	2.07	90.29	3.068	0.675
80%	3.34	77.77	2.387	1.734

The guides predominantly predict the original labels, achieving 90.29% and 77.77% agreement at 40% and 80% noise, respectively, indicating limited memorization of corrupted annotations.

We also examine how RG assigns gradient scaling factors ω_i (Eq. 9) to training samples using the selected final RG checkpoints. Samples are divided into two groups according to whether their labels were changed by synthetic noise. Table 22 reports the average scaling factor for each group and the percentage of samples with $\omega_i = 1$, indicating no additional gradient amplification.

Table 22: Gradient scaling factors evaluated on training samples using selected final RG checkpoints. Corrupted and uncorrupted refer to whether synthetic noise altered the sample label.

Learner	Noise	Mean scaling factor (ω)		Samples with $\omega = 1$ (%)	
		Corrupted	Uncorrupted	Corrupted	Uncorrupted
CNN	40%	1.23	1.74	95.2	84.4
CNN	80%	1.03	1.06	99.6	99.5
MLP	40%	1.63	2.21	90.6	82.0
MLP	80%	1.25	1.37	97.7	96.8
ViT	40%	2.54	2.86	77.1	72.3
ViT	80%	1.46	1.60	97.0	96.1

Across all architectures, unchanged-label samples receive greater average gradient amplification, particularly at 40% noise (e.g., 1.74 versus 1.23 for CNN). At 80% noise, most samples in both groups receive no additional amplification.

D CIFAR-10N experimental details

D.1 Dataset preparation

CIFAR-10N [30] provides independent human annotations (`random1`, `random2`, and `random3`), aggregated majority-vote labels (`aggre`), and a `worst` label set selecting an incorrect annotation whenever available. The original CIFAR-10 training images are partitioned by their original indices: the first 45,000 use the selected noisy annotations for training, while the last 5,000 use the original clean CIFAR-10 labels for validation. The original 10,000 clean test images are reserved for final evaluation. Training images undergo random cropping with four-pixel padding and random horizontal flipping. All images are normalized using the CIFAR-10 RGB channel means

$(0.4914, 0.4822, 0.4465)$ and standard deviations $(0.2023, 0.1994, 0.2010)$, with no augmentation applied during validation or testing. Guide reference logits are computed on non-augmented, normalized images.

The three evaluated label combinations are `random1/random3`, `worst/worst`, and `worst/aggre`, where the first label set is used for learner training and the second for training the ResNet56 guide. ERM does not use a guide, so its `worst` result is shared by the two `worst`-learner comparisons. Guide logits are precomputed on non-augmented, normalized images, aligned to original image indices, and held fixed during learner training.

D.2 Hyperparameter search

The CIFAR-10N experiments use the same learner architecture families as Fashion-MNIST (CNN, MLP, and ViT), with configurations selected independently for each learner and noisy-label condition. Stage 1 selects the architectures under ERM using 20 Optuna trials of 30 epochs, a fixed LR of 0.01, early stopping patience of 15 epochs, and minimum clean validation cross-entropy.

For CNN, the search considers three convolutional layers with channel widths independently selected from $\{64, 128\}$, a fixed kernel size $k = 3$, dense-layer width $d \in \{64, 128, 256\}$, and dropout in $[0, 0.4]$. For MLP, it considers 1 to 3 hidden layers with widths selected from $\{128, 256, 384\}$ and dropout in $[0.05, 0.3]$. For ViT, the search includes hidden size $h \in \{64, 96, 128\}$, number of attention heads $a \in \{2, 4\}$, Transformer layers $L \in \{2, \ldots, 6\}$, MLP expansion ratio $r \in \{2, 4\}$, and dropout in $[0.05, 0.3]$, with a fixed patch size of 4×4. To maintain a consistent parameter budget, MLP and ViT configurations exceeding one million trainable parameters are excluded from training during Stage 1 trials.

The selected architectures, reported in Table 23, are fixed for subsequent ERM and RG runs. Separately trained ResNet56 guides use `random3`, `worst`, or `aggre` annotations.

Table 23: Selected CIFAR-10N learner architectures from Stage 1. For CNN, the tuple specifies convolutional channel widths, k denotes kernel size, and d denotes dense-layer width. For ViT, h denotes hidden size, a the number of attention heads, L the number of Transformer layers, and r the MLP expansion ratio.

Model	Learner labels	Selected architecture	Dropout
CNN	`random1`	$(64, 128, 128)$, $k = 3$, $d = 64$	0.0126
CNN	`worst`	$(128, 128, 128)$, $k = 3$, $d = 256$	0.0104
MLP	`random1`	1 layer, width 256	0.2019
MLP	`worst`	1 layer, width 256	0.0709
ViT	`random1`	$h = 128$, $a = 2$, $L = 2$, $r = 4$	0.0647
ViT	`worst`	$h = 128$, $a = 4$, $L = 3$, $r = 2$	0.0606

Stage 2 searches LR jointly with method-specific hyperparameters using 20 Optuna trials per condition, each with a 150-epoch training budget. The selected ERM learning rates are CNN (`random1`: 0.01, `worst`: 0.005); MLP (`random1`: 0.01, `worst`: 0.01); ViT (`random1`: 0.01, `worst`: 0.01). Search configurations are ranked by the minimum validation cross-entropy, using TPE sampling with five startup trials. ERM tunes only the learning rate, whereas RG also searches λ, δ, and γ. The Stage 2 search space is defined as

$$\eta \in \{0.001, 0.005, 0.01\}, \quad \lambda \in [0.25, 20] \text{ (log-scale)},$$
$$\delta \in [10^{-4}, 1] \text{ (log-scale)}, \quad \gamma \in [0.9, 1.1].$$

Table 24 summarizes the selected RG hyperparameters.

D.3 Training configuration

All CIFAR-10N experiments use SGD with momentum 0.9, weight decay 5×10^{-4}, batch size 256, and cosine learning-rate annealing with a minimum LR of 10^{-5} over 150 epochs. Final trained

Table 24: Selected RG hyperparameters on CIFAR-10N.

Model	Learner / guide	LR	λ	γ	δ
CNN	`random1/random3`	0.005	5.565	1.094	1.209×10^{-4}
CNN	`worst/worst`	0.005	5.565	1.094	1.209×10^{-4}
CNN	`worst/aggre`	0.001	15.059	1.005	1.195×10^{-2}
MLP	`random1/random3`	0.005	2.380	0.909	2.342×10^{-2}
MLP	`worst/worst`	0.001	14.920	1.090	1.677×10^{-3}
MLP	`worst/aggre`	0.005	3.924	1.091	1.901×10^{-4}
ViT	`random1/random3`	0.01	12.931	1.027	1.725×10^{-3}
ViT	`worst/worst`	0.005	19.112	1.021	3.562×10^{-3}
ViT	`worst/aggre`	0.005	14.709	0.997	7.241×10^{-3}

models are selected by each method's best validation objective (RG or ERM). Hyperparameter trials are selected using validation cross-entropy. RG uses a fixed ResNet56 guide for each label noise set.

Table 25: Clean-test accuracy of the ResNet56 guide on CIFAR-10N (%, mean $\pm$ sample SD over five runs).

Guide training labels	Test accuracy (%)
`aggre`	84.00 ± 0.39
`random3`	81.70 ± 0.30
`worst`	74.86 ± 0.33

E M4 experimental details

E.1 Dataset preparation

The regression experiments utilize the M4 benchmark dataset [17]. Each frequency subset (Hourly to Yearly) is processed independently.

E.1.1 Windowing and computational constraints

To transform the raw time series into a supervised learning format, we employ a sliding window mechanism. For each subset, the window size W is selected based on the forecast horizon H and the 21st percentile ($Q_{0.21}$) of the series length distribution, satisfying the constraint $3H \leq W < Q_{0.21} - H$.

Strides S are determined to balance dataset representativeness with computational feasibility. Starting from a minimum stride $S = H$, we increase S until the total number of windows generated across all series in a subset is up to 50,000. Table 26 summarizes the resulting parameters.

Table 26: Dataset characteristics and windowing parameters per M4 frequency subset.

Subset	Seasonality (m)	Horizon (H)	Window (W)	Stride (S)
Yearly	1	6	18	6
Quarterly	4	8	32	32
Monthly	12	18	72	112
Weekly	1	13	52	13
Daily	1	14	56	198
Hourly	24	48	192	48

Series shorter than $W + H$ are handled via seasonal padding, where observed seasonal patterns are repeated until the minimum length requirement is met. Each series is independently normalized to the [0,1] interval using a `MinMaxScaler` fitted on the pre-test segment.

E.1.2 Temporal splitting and validation

We adopt a horizon-based temporal split. For a series $y_{1:T}$, the last H observations are reserved for the test set. The preceding segment $y_{1:T-H}$ constitutes the training/validation pool.

Within this pool, sliding windows are generated to produce input-output pairs (X, y). For each series yielding $N \geq 2$ windows, the number of validation windows is defined as

$$N_{\text{val}} = \min\left\{\max\left(1, \lfloor 0.2N \rfloor\right), N-1\right\}. \tag{25}$$

The first $N - N_{\text{val}}$ windows are assigned to training, while the remaining N_{val} windows are assigned to validation. Series yielding fewer than two windows are excluded from the training and validation window sets. The test input is defined as the final available window $y_{T-H-W+1:T-H}$.

E.1.3 Reference signal integration

A reference forecasting signal b is precomputed for every sliding window using the NAR-AOLS model (see Appendix E.2.5), following a rolling-origin forecasting scheme. These forecasts are aligned with the target horizon of each sliding window.

The final training samples are represented as triplets (X, y, b), where X is the input window, y is the ground-truth target, and b is the reference forecast. All triplets are stored in memory to ensure deterministic data loading and accelerated training across all experimental runs.

E.2 Architecture definitions

This subsection summarizes the architectures used in the regression experiments. All models were implemented using the software and libraries described in Appendix F.2. Tables report the structural components and searchable parameters, and their ranges, used during stage 1 of hyperparameter search in Appendix E.3.

E.2.1 Multilayer perceptron

The multilayer perceptron processes flattened time-window inputs and consists of a sequence of fully connected layers followed by a linear output layer. Hidden layer depth and width are optimized during tuning.

Table 27 summarizes the MLP configuration space.

Table 27: MLP architecture configuration.

Component	Specification
Input representation	Flattened time window
Hidden layers	1–3
Hidden dimensions	$\{256, 512, 1024, 2048, 4096\}$
Dropout	$[0.05, 0.3]$
Activation	ReLU
Output layer	Fully connected (regression output)

Each hidden layer follows the structure:

$$\text{Linear} \rightarrow \text{ReLU} \rightarrow \text{Dropout}$$

The final output layer produces regression targets of dimension H, where H depends on the M4 subset (see Table 26).

E.2.2 Gated recurrent unit

The GRU-based model processes sequential inputs using recurrent dynamics followed by a dense projection head. The recurrent depth, hidden size, and dropout are optimized during tuning.

Table 28 summarizes the GRU configuration space.

Table 28: GRU architecture configuration.

Component	Specification
Input format	Sequential window
GRU hidden size	$\{64, 128, 256, 384, 512\}$
GRU layers	1–3
GRU dropout	$[0.05, 0.3]$
Dense projection	$\{64, 128, 256\}$
Output layer	Fully connected (regression output)

The GRU processes the sequence and the final hidden state is passed through a feed-forward head:

$$\text{GRU} \rightarrow \text{Linear} \rightarrow \text{ReLU} \rightarrow \text{Dropout} \rightarrow \text{Linear}$$

The final output layer produces regression targets of dimension H, where H depends on the M4 subset (see Table 26).

E.2.3 Temporal convolutional network

The TCN model processes sequential inputs using causal dilated convolutions, followed by a dense prediction head. Channel depth, kernel size, and network depth are optimized during tuning.

Table 29 summarizes the TCN configuration space.

Table 29: TCN architecture configuration.

Component	Specification
Input format	Sequential window
TCN channels	Variable per level
Number of levels	2–6
Kernel size	$\{2, 3, 5\}$
Dropout	$[0.05, 0.5]$
Dense projection	$[32, 256]$
Output layer	Fully connected (regression output)

The TCN applies causal convolutions over the temporal dimension:

$$\text{TCN} \rightarrow \text{Linear} \rightarrow \text{ReLU} \rightarrow \text{Dropout} \rightarrow \text{Linear}$$

The final representation of the last time step is used for regression output prediction with dimension H, where H depends on the M4 subset (see Table 26).

E.2.4 Seasonal naive model

The Seasonal naive model [17] is implemented using the StatsForecast Python library [46].

Given a seasonal period s, forecasts from origin t are defined as

$$\hat{y}_{t+h} = y_{t+h-s\lceil h/s \rceil}, \qquad h = 1, \ldots, H.$$

Here, y_t denotes the observed value at time t, s is the seasonal period, and H is the forecast horizon.

For each time series, forecasts are generated using a rolling forecasting strategy. At each time step t, the model is fitted using all observations available up to time t, and a one-step-ahead prediction is produced. After processing the full training portion, multi-step forecasts are generated for the out-of-sample horizon.

The seasonal period s is determined from the frequency of each time-series subset (e.g., yearly, quarterly, monthly).

E.2.5 Nonlinear autoregressive model

The reference forecasting signal is generated using the NAR model [43], estimated via the AOLS algorithm [44] from the `SysIdentPy` library [47].

The model employs a first-degree polynomial basis and a fixed autoregressive lag of $y_{\text{lag}} = 1$ across all M4 frequency subsets. Its autoregressive mapping is expressed as

$$y_t = f(y_{t-1}) + \varepsilon_t, \tag{26}$$

where $f(\cdot)$ denotes the polynomial mapping and ε_t is the residual error.

To ensure computational efficiency across the 100,000 time series, AOLS selects the $k = 1$ most relevant polynomial term. Forecasts are generated using a rolling-origin scheme, in which the model is refitted at intervals of S time steps, with S corresponding to the stride specified in Table 26. At each refitting step, only observations available up to the current forecast origin are used to generate multi-step-ahead predictions.

E.3 Hyperparameter search

The hyperparameter optimization for regression experiments follows a two-stage procedure. In Stage 1, we determine the optimal structural configuration for each architecture (MLP, GRU, and TCN) by minimizing the validation MSE under standard ERM training. These architectural parameters, summarized in Table 30, are then fixed for all subsequent experiments in Stage 2.

The search budget differs between stages to balance exploration and convergence. Stage 1 consists of 50 trials, each running for up to 30 epochs with an early stopping patience of 15. Stage 2 focuses on fine-tuning training dynamics over 25 trials, with a maximum of 1000 epochs and a higher patience of 100 epochs. All tuning is performed using a TPE sampler with 5 initial startup trials.

Table 30: Architecture-specific hyperparameter search spaces for Stage 1.

Architecture	Component	Search space
MLP	Hidden layers	$[1, 3]$
	Hidden dimensions	$\{256, 512, 1024, 2048, 4096\}$
	Dropout	$[0.05, 0.3]$
GRU	Hidden size	$\{64, 128, 256, 384, 512\}$
	Number of layers	$[1, 3]$
	Dropout	$[0.05, 0.3]$
	Dense projection size	$\{64, 128, 256\}$
TCN	Number of levels	$[2, 6]$
	Channels per layer	$\{32, 64, 128, 256\}$
	Kernel size	$\{2, 3, 5\}$
	Dropout	$[0.05, 0.5]$
	Dense projection size	$[32, 256]$

Table 31 details the optimization parameters for both stages. In Stage 2, the search for ERM baselines is restricted to optimizer-related parameters. For RG training, we additionally optimize the objective-specific hyperparameters λ, δ, and the guide tolerance γ. Both learning rates and weight decay values are sampled on a logarithmic scale, with continuous sampling for learning rates and eight discrete, logarithmically spaced choices for weight decay.

E.4 Post-tuning training configuration

After hyperparameter tuning, five independent training runs are performed using the selected configurations in both stages. All experiments use the AdamW optimizer [42] and a ReduceLROnPlateau learning-rate scheduler, configured in mode `min` with factor 0.95 and a minimum LR of 10^{-6}. Training is run for up to 3000 epochs. Early stopping is applied based on validation loss, with a patience of 200 epochs.

Table 31: Training and objective-specific hyperparameter search spaces.

Configuration	Component	Search space
Stage 1 (All)	Batch size	$\{256, 512, 768, 1024\}$
	LR	$[10^{-6}, 10^{-3}]$ (log)
	Weight decay	8 log-spaced values in $[10^{-6}, 10^{-2}]$
Stage 2 (ERM)	Batch size	$\{16, 32, 64\}$
	LR	$[10^{-6}, 10^{-3}]$ (log)
	Weight decay	8 log-spaced values in $[10^{-6}, 10^{-2}]$
Stage 2 (RG)	Batch size	$\{16, 32, 64\}$
	LR	$[10^{-6}, 10^{-3}]$ (log)
	Weight decay	8 log-spaced values in $[10^{-6}, 10^{-2}]$
	λ (Scaling factor)	$[0.25, 10]$
	δ (Smoothing)	$[10^{-5}, 10^{-2}]$ (log)
	γ (Guide tolerance)	$[0.9, 1.1]$

E.5 Guide models detailed results

Table 32 shows the performance of the NAR guide model used across M4 subsets.

Table 32: Performance of the NAR guide model used across M4 subsets.

Subset	Method	SMAPE	MASE	OWA
Hourly	NAR	16.11	0.97	0.641
Daily	NAR	3.13	3.26	1.012
Weekly	NAR	9.38	2.30	0.926
Monthly	NAR	13.55	1.28	1.072
Quarterly	NAR	11.12	1.26	0.963
Yearly	NAR	16.21	3.88	0.984

F Asset licenses and attribution

This section provides details regarding the licenses and terms of use for the datasets and software assets utilized in this work. All assets were employed strictly for academic research purposes.

F.1 Datasets

- **Fashion-MNIST:** The Fashion-MNIST dataset [28] was developed by Zalando SE and is released under the **MIT License**. It is a publicly available benchmark used in this study for evaluating image classification under label noise.
- **CIFAR-10/CIFAR-10N:** CIFAR-10 [29] is a publicly available image classification benchmark with no explicit license specified on its original distribution page. CIFAR-10N [30] provides human-annotated noisy labels under the **CC BY-NC 4.0** license.
- **M4 Dataset:** The M4 Competition dataset [17] was released as part of the M4 forecasting competition. The data is available for public use and research under the **Creative Commons Attribution 4.0 International (CC-BY 4.0)** license. It is a standard benchmark in the time-series community for evaluating large-scale forecasting.

F.2 Software and libraries

All experiments were implemented in Python 3.9, utilizing the following libraries and frameworks:

- **PyTorch:** Our core implementation of the Reference-Guided objective and neural architectures is built on PyTorch [48], which is released under the **BSD 3-Clause License**.

- **Transformers:** The Vision Transformer architectures were implemented using the Hugging Face Transformers library [49], licensed under the **Apache License 2.0**.
- **Optuna:** The hyperparameter optimization and Bayesian search procedures were conducted using Optuna [50], licensed under the **MIT License**.
- **SysIdentPy:** The Nonlinear AutoRegressive (NAR) model and the Accelerated Orthogonal Least Squares (AOLS) algorithm used in regression tasks are part of the SysIdentPy library [47], licensed under the **BSD 3-Clause License**.
- **StatsForecast:** The Seasonal Naive baseline implementation for the M4 dataset was obtained from the Nixtla StatsForecast library [46], licensed under the **Apache License 2.0**.
- **Scientific Stack:** Data processing, statistical analysis, and evaluation metrics were implemented using **Scikit-learn** [51], **NumPy** [52], and **SciPy** [53], all of which are licensed under the **BSD 3-Clause License**.

F.3 Architectures

The model architectures implemented follow standard research implementations described in their respective original publications and were trained from scratch using publicly reproducible procedures.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately describe the proposed RG objective, the theoretical analysis of gradient scaling, and the experimental results in both classification and regression.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The sensitivity to guide quality and the potential for amplifying noise if the guide is systematically biased, are emphasized in Section 5.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The mathematical formulation and gradient dynamics are provided in Section 3. Formal proofs regarding curvature, convexity, and Lipschitz bounds are detailed in Appendix A.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper provides comprehensive details on architectures, hyperparameter search spaces, data preprocessing, and training protocols.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: All experiment code, logs, and results are provided, along with instructions to reproduce them. Additionally, we uploaded them to https://github.com/Raelis/reference-guided-training

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Core optimization settings, data splitting strategies, and hyperparameter tuning procedures are described in Sections 4.1.1, 4.2 and 4.3.1, with additional implementation details provided in Appendices C, D, and E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Results are reported as mean $\pm$ one standard deviation over 5 independent runs across all experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Hardware specifications (CPU, GPU, RAM) and the amount of compute required are specified in Appendix B.2.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research follows standard machine learning protocols and utilizes publicly available datasets, adhering to the community's ethical standards.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: This work is foundational research focused on optimization and loss functions. It does not introduce specific high-risk social applications or direct societal consequences.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The work uses standard image and time-series benchmarks that do not pose a high risk of misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All licenses for datasets (Fashion-MNIST, M4) and software libraries are explicitly detailed in Appendix F.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.

- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The source code introduced in this work is fully documented and released with instructions for installation, training, and evaluation.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The study does not involve crowdsourcing or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The study does not involve human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs were not used as a component of the core methodology or the experimental pipeline.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.