# LEARNING TO PREDICT DISTRIBUTIONS OVER WEIGHT UPDATES FOR TEST-TIME ADAPTATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Hypernetworks have recently shown success in dynamically adapting the parameters of Large Language Models (LLMs) at runtime based on signals such as task descriptions or additional demonstrations. Here we ask: how much adaptation signal can be obtained using only the input query to an LLM? To answer this, we study query-conditioned hypernetworks for LoRA estimation. Further, we introduce distributional hypernetworks, able to produce not only point estimates of parameter adapters, but also a distribution over possible LoRAs. For this we propose a simple end-to-end loss using a differentiable Monte Carlo approximation and explore multiple distribution parameterizations including regression and convex combination variants. Results show that even using the mean of the learned distribution can outperform deterministic hypernetworks. Crucially, the learned distribution enables a different form of test-time scaling: instead of spending additional compute only by sampling more token sequences from a fixed model, we sample weight updates, yielding multiple adapted models for the same query. Performance improves as more weight samples are considered and remains stronger than corresponding token-sampling adaptation baselines. Finally, we find that generated updates can transfer across queries, suggesting that the hypernetwork learns reusable structure in how the model should adapt. Together, these results show that query-conditioned distributions over weight updates can support both adaptation and test-time scaling.

## 1 INTRODUCTION

Updating parameters in a Large Language Model (LLM) requires gradient-based optimization over multiple rounds with sufficient data. This is a brittle process that can lead to catastrophic forgetting, even when using parameter efficient adaptation such as LoRA [1]. Gradient-descent is therefore particularly unsuitable for altering parameters of models at test-time, where there is little or no supervision data available. Hypernetworks offer a promising alternative way to do this. A hypernetwork is a neural network that generates the weights of another network [2]. Hypernetworks are trained offline using ordinary backpropagation, but at inference time, they directly predict weight updates for a target network. The learned parameters of the hypernetwork can therefore be viewed as slow weights that learn how to produce fast weights for each new input [3, 4]. Unlike learned optimizers [5], the hypernetwork does not require gradients at inference time. It turns parameter adaptation from a repeated optimization process into a direct prediction.

Recent work has shown success using hypernetworks to generate LoRA adapters from natural-language task descriptions [6], or generate adapters from multiple demonstrations of related examples [7]. These methods allow gradient-free adaptation, but still require such auxiliary information to be available. Here, we propose to predict a LoRA directly from the language model input. The key intuition is that different queries may reveal regularities in how the model should adapt, allowing the hypernetwork to learn an implicit mapping from input structure to useful parameter updates rather than relying on an explicit task description. This leads to a broader question: how much adaptation signal is contained in a single query? Is it sufficient to determine a useful weight update? And if a single update is underdetermined, can the query instead constrain a useful distribution over plausible updates? To address this, we propose a simple differentiable Monte Carlo approximation of the marginal log-likelihood using two types of updates: Gaussian LoRA regression and convex combinations of LoRA exemplars, depicted in Figure 1. We explore multiple parameterization and



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

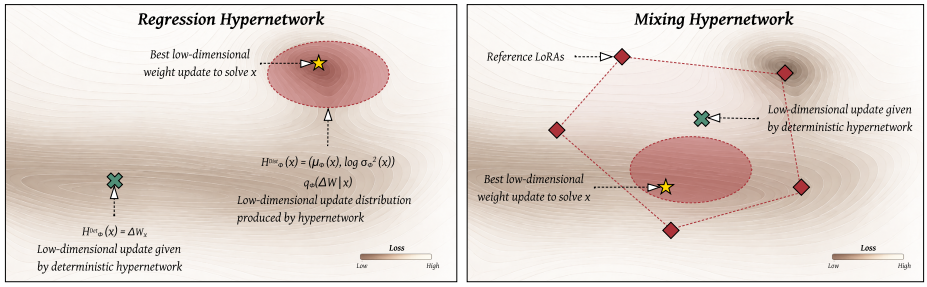



Figure 1: **Distributional hypernetworks predict a query-conditioned distribution over LoRA updates.** The regression hypernetworks model the distribution directly in LoRA weight space, while the mixing hypernetwork models a distribution over combinations of reference LoRAs.

optimization variants, such as weight correlations and loss aggregation, and compare our approach with a point-estimate hypernetwork.

Our experiments show a single query does indeed contain substantial adaptation signal. It suffices to generate useful LoRA updates without task descriptions or demonstrations, and can further support a distribution over plausible updates. The mean of the learned distribution outperforms deterministic hypernetworks by 4.6–11.7% accuracy across 9 benchmarks. More importantly, this distribution becomes a new axis for test-time scaling: sampling different adapted models is competitive with, and often stronger than, sampling additional outputs from fixed weights. The generated updates also exhibit reusable structure across queries, as distributions produced from one subset of inputs can be aggregated and successfully applied to held-out queries without re-invoking the hypernetwork.

Our contributions can be summarized as follows:

- We study how much adaptation signal can be extracted from a single query, showing that it is sufficient to generate useful query-conditioned LoRA updates without test-time optimization or task-level auxiliary information.
- We introduce **distributional hypernetworks** that predict a distribution over query-specific weight updates based on a simple differentiable Monte Carlo approximation.
- We introduce test-time scaling in weight space, where additional inference compute is used to sample alternative model updates rather than additional token sequences.
- We study the structure of the generated updates, including *when* query-specific generation becomes useful and whether generated adaptations *generalize* to other in-domain queries.

## 2 Related Work

**Hypernetworks.** Hypernetworks learn to generate the parameters of another network and provide a way to amortize adaptation without optimizing the target model at inference [2]. In language models, HyperTuning [7] and HINT [8] generate parameter-efficient adaptations from demonstrations or instructions, while Text-to-LoRA (T2L; [6]) generates LoRA weights from a task description. More recent methods generate adaptations from longer contexts or related examples [9–11]. HyperDecoders move closer to instance-level adaptation by generating a separate adaptation from each input [12]. These methods differ in what information is used to condition the hypernetwork, but predict a single adaptation. We show that a single query can provide enough information to generate useful query-conditioned LoRA updates and a distribution over possible LoRA updates. This contrasts with test-time training methods, which adapt the model through gradient-based optimization at inference time [13–18].

**Distributions over Weights.** Learning distributions over neural network parameters has been studied in Bayesian neural networks and generative weight models. Bayesian Hypernetworks [19], implicit weight uncertainty [20], and HyperGAN [21] use learned generators to represent distributions over model parameters. These methods show that useful variation can be represented directly in weight space, but the resulting distributions are typically used for uncertainty estimation, ensembling, or representing a global model distribution.

More recent work studies distributions specifically in the low-rank adaptation space. ICM-LoRA [22] and DiffLoRA [23] learn conditional generative models from collections of previously optimized adapters, while Bayesian LoRA methods model uncertainty around task-level adaptations [24–26]. LoRA ensembles similarly obtain multiple task-level adaptations through independent training [27]. Our distribution is different in both how it is produced and how it is used. It is conditioned on each query and trained directly through the frozen LM's response loss, rather than by reproducing previously optimized adapters or approximating a task-level posterior. Samples from the distribution are treated as alternative adaptations for the current query, allowing the hypernetwork to represent several useful updates rather than committing to a single one.

Further discussion of related work is deferred to Appendix A.

# 3 Method: Distributional Hypernetworks

## 3.1 Hypernetwork for LoRA Estimation

We start with a pretrained autoregressive LM $p_\theta(y \mid x)$, parameterized by neural network weights $\theta$. Rather than modifying $\theta$ directly, a hypernetwork $\mathrm{H}_\Phi$, with parameters $\Phi$, generates an input dependent weight update:

$$\theta_x = \theta + \Delta\theta_x \qquad \text{with} \qquad \Delta\theta_x = \mathrm{H}_\Phi(x), \tag{1}$$

The hypernetwork is trained offline on a dataset $\mathcal{D}$ of query–response pairs, by maximizing the likelihood the adapted model assigns to the observed response $y$. We train the hypernetwork by

$$\Phi^\star = \arg\min_\Phi \qquad \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[-\log p_{\theta+\mathrm{H}_\Phi(x)}(y \mid x)\right] \tag{2}$$

During training, the loss is backpropagated through the LM and the generated update $\Delta\theta_x = \mathrm{H}_\Phi(x)$, but only the hypernetwork parameters $\Phi$ are optimized, the pretrained parameters $\theta$ remain fixed. The resulting optimization problem is no different from conventional cross-entropy gradient descent over weights. The frozen LM provides the learning signal that teaches the hypernetwork which updates are useful for different queries. At test-time, only the query is available, and the trained hypernetwork directly produces $\Delta\theta_x$ in a single forward pass, after which the adapted model $p_{\theta+\Delta\theta_x}(y \mid x)$ generates the response. This procedure at test-time requires no ground-truth response, objective, backward pass, or iterative optimization. Consequently, the additional cost of adaptation is a single forward pass through $\mathrm{H}_\Phi$, replacing per-query optimization with a learned mapping from queries to weight updates. The hypernetwork parameters $\Phi$ can be viewed as slow weights that learn to produce the query-specific fast weights $\Delta\theta_x$.

Generating a dense update for every adapted weight matrix is impractical. For a matrix $\theta^{(\ell)} \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$, a dense update contains $d_{\text{out}} \times d_{\text{in}}$ values. In a large transformer, this would require the hypernetwork to produce billions of values per query, making its output space prohibitively large. Previous work has shown that useful updates are low-dimensional [1, 28] such as LoRA. For a rank $r \ll \min(d_{\text{in}}, d_{\text{out}})$ and scaling constant $\alpha$, the hypernetwork generates two query-specific factors, $\mathrm{A}_x^{(\ell)} \in \mathbb{R}^{r\times d_{\text{in}}}$ and $\mathrm{B}_x^{(\ell)} \in \mathbb{R}^{d_{\text{out}}\times r}$. Together, these factors induce a low-rank update. The number of values generated for each adapted matrix is therefore reduced from $d_{\text{out}}d_{\text{in}}$ to $r(d_{\text{in}} + d_{\text{out}})$. Let $\mathcal{M}$ denote the set of LM matrices adapted with LoRA. The hypernetwork outputs the LoRA factors for all matrices in $\mathcal{M}$, and these factors collectively define the effective update:

$$\Delta\theta_x^{(\ell)} = \frac{\alpha}{r}\mathrm{B}_x^{(\ell)}\mathrm{A}_x^{(\ell)}. \tag{3}$$

Given a query $x$, a hypernetwork $\mathrm{H}_\Phi(x)$ runs the input through $p_\theta$ to obtain the sequence of final-layer hidden states $\mathrm{Z}_x = \mathrm{F}_\theta^{\text{hid}}(x)$ that are fed to the network. For each target matrix $s = (\ell, m) \in \mathcal{M}$, the cross-attention-based hypernetwork is conditioned on $\mathrm{Z}_x$, a learned embedding for layer $\ell$, and a learned embedding for module type $m$. It produces the corresponding LoRA factors and thereby defines a deterministic query-specific update:

$$\Delta\theta_x^{(\ell)} = \mathrm{H}_\Phi\left(\mathrm{Z}_x, e_{\text{layer}}^{(\ell)}, e_{\text{module}}^{(m)}\right) \tag{4}$$

The hypernetwork is trained using the offline LM objective in Equation 2. Unlike prior methods that condition on a task description, demonstrations, or an explicit task identifier [6, 7, 12], our model

derives its semantic conditioning entirely from the query. Furthermore, it uses the full sequence rather than a mean-pooled representation, since pooling can discard token-level information that may be useful for constructing the update. The layer and module embeddings identify where an update will be applied, but provide no information about the task itself. Consequently, the model requires no task-level metadata and may generate a different LoRA update for every query. This deterministic formulation produces a single point in the low-rank update space, which we generalize to a query-conditioned distribution in the following subsection.

### 3.2 Distributional Hypernetwork

Predicting a single useful update from a query alone is an underdetermined problem. Several different adaptations may lead to a good response, and the query may not contain enough information to identify which one is best. A deterministic hypernetwork must nevertheless commit to one point in the LoRA parameter space, and when several solutions are plausible this point may be an average that resembles none of them. Recent evidence further suggests that the neighborhoods around large pretrained models are densely populated by diverse task-improving specialists [29]. Here we propose to train a hypernetwork to predict a distribution over query-specific adapters.

$$\Phi^\star = \arg\min_{\Phi} \quad \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathbb{E}_{\Delta\theta\sim q_\Phi(\omega|x)}\left[-\log p_{\theta+\Delta\theta}(y \mid x)\right]\right] \tag{5}$$

Since $\Delta\theta$ is continuous, we can optimize this loss directly using the reparameterization trick [30, 31]:

$$\Phi^\star = \arg\min_{\Phi} \quad \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathbb{E}_{\epsilon\sim\mathcal{N}_n(0,I_n)}\left[-\log p_{\theta+f_\Phi(x,\epsilon)}(y \mid x)\right]\right] \tag{6}$$

where $f_\Phi(x,\epsilon)$ transforms an initial parameterless $n$-dimensional Gaussian into some other distribution parametrized by $\Phi$.

At inference time, one option is to decode using the expected LoRA update:

$$y^\star = \arg\max_{y} \quad \log p_{\theta+\mathbb{E}_{q_\Phi}[\Delta\theta|x]}(y \mid x). \tag{7}$$

Alternatively, we can perform minimum Bayes risk decoding (MBR) by sampling weights rather than tokens. We first sample $K$ adapters and greedily decode from each adapted model:

$$\begin{aligned} &\epsilon^{(k)} \sim \mathcal{N}_n(0, I_n), \qquad \Delta\theta_x^{(k)} = f_\Phi(x, \epsilon^{(k)}), \\ &\widehat{y}_x^{(k)} = \arg\max_y \log p_{\theta+\Delta\theta_x^{(k)}}(y \mid x), \qquad k = 1, \ldots, K. \end{aligned} \tag{8}$$

Given the candidate set $\mathcal{Y}_x = \{\widehat{y}_x^{(1)}, \ldots, \widehat{y}_x^{(K)}\}$ and a task-specific risk $R$, the MBR prediction is

$$y^\star_{\mathrm{MBR}} = \arg\min_{\widehat{y}\in\mathcal{Y}_x} \frac{1}{K}\sum_{k=1}^{K} R\left(\widehat{y}, \widehat{y}_x^{(k)}\right). \tag{9}$$

For example, majority voting is recovered by choosing the zero-one disagreement risk $R(\widehat{y}, \widehat{y}') = \mathbb{I}[\widehat{y} \neq \widehat{y}']$. Each adapted model can therefore be decoded greedily, since predictive diversity comes from sampling model weights rather than tokens.

**Gaussian LoRA Distribution** Our first parameterization assumes Gaussian distributed LoRA factors. For a general correlated $n$-dimensional Gaussian, the reparameterized sample is

$$f_\Phi(x,\epsilon) = \mu_\Phi(x) + L_\Phi(x)\epsilon, \qquad \epsilon \sim \mathcal{N}_n(0, I_n), \qquad \Sigma_\Phi(x) = L_\Phi(x)L_\Phi(x)^\top. \tag{10}$$

For each target matrix $s \in \mathcal{M}$, the hypernetwork of Equation 4 uses one output head to predict the mean $\mu_\Phi^{(s)}(x)$ and another to predict the entries of the lower-triangular Cholesky factor $L_\Phi^{(s)}(x)$. The strictly lower-triangular entries are unconstrained, while positivity of the diagonal is enforced by applying a softplus transformation to the corresponding unconstrained outputs. This construction ensures that $f_\Phi(x,\epsilon)$ has mean $\mu_\Phi(x)$ and covariance $\Sigma_\Phi(x)$. A full covariance requires the hypernetwork to generate $n(n+1)/2$ Cholesky parameters, which is prohibitively expensive for the high-dimensional LoRA factor space. We therefore explore low-rank factorizations of the covariance as well as a diagonal covariance.

$$f_\Phi(x,\epsilon) = \mu_\Phi(x) + \sigma_\Phi(x) \odot \epsilon, \qquad \epsilon \sim \mathcal{N}_n(0, I_n). \tag{11}$$

The above yields the following distribution over LoRA weights:

$$q_\Phi(\omega \mid x) = \mathcal{N}\big(\omega; \mu_\Phi(x), \Sigma_\Phi(x)\big), \qquad \Sigma_\Phi(x) = L_\Phi(x) L_\Phi(x)^\top. \tag{12}$$

For the diagonal specialization, the number of covariance outputs is linear rather than quadratic in $d_\Delta$. In this case the noise on A and B is independent and zero-mean, so the induced update satisfies $\mathbb{E}\big[\Delta\theta_x^{(s)}\big] = \frac{\alpha}{r}\mu_{\mathrm{B},x}^{(s)}\mu_{\mathrm{A},x}^{(s)}$. Thus, the product of the mean factors recovers a single deterministic adapter, while sampling explores the region around it.

**Convex Hull LoRA Distribution** A Gaussian over millions of LoRA parameters can place samples far from regions of weight space containing useful adapters. Our second parameterization instead constrains the search to combinations of a bank of R trained reference adapters $\{\widehat{\Delta\theta}_j\}_{j=1}^{\mathrm{R}}$. The hypernetwork predicts the mean $\mu_{z,\Phi}(x)$ and Cholesky factor $L_{z,\Phi}(x)$ of a Gaussian over mixing logits. Applying the softmax maps these logits to mixing weights on the simplex:

$$\begin{aligned} \pi_\Phi(x, \epsilon) &= \mathrm{softmax}(\mu_{z,\Phi}(x) + L_{z,\Phi}(x)\epsilon)\,, \qquad \epsilon \sim \mathcal{N}_{\mathrm{R}}(0, I_{\mathrm{R}}), \\ f_\Phi(x, \epsilon) &= \sum_{j=1}^{\mathrm{R}} \pi_{\Phi,j}(x, \epsilon)\widehat{\Delta\theta}_j. \end{aligned} \tag{13}$$

We denote the resulting distribution over LoRA weights by

$$q_\Phi(\omega \mid x) = \mathrm{CH}(\omega; \mu_{z,\Phi}(x), \Sigma_{z,\Phi}(x))\,, \qquad \Sigma_{z,\Phi}(x) = L_{z,\Phi}(x) L_{z,\Phi}(x)^\top. \tag{14}$$

Here CH is notation we introduce rather than a standard distribution family. The mixing weights $\pi_\Phi(x, \epsilon)$ follow a logistic-normal distribution, and CH is its linear pushforward through the map $\pi \mapsto \sum_{j=1}^{\mathrm{R}} \pi_j \widehat{\Delta\theta}_j$. Its support is therefore contained in the convex hull of the reference adapters, restricting the model to a low-dimensional region of weight space. The two parameterizations are not directly comparable in scale because the first operates in the $d_\Delta$-dimensional factor space, while the second operates in an R-dimensional logit space with $\mathrm{R} - 1$ effective degrees of freedom.

## 4 Distributional Hypernetwork Training

For training the Distributional Hypernetwork, we use the Monte Carlo gradient estimate of Equation 6. For a single example $(x_i, y_i)$ this is

$$\nabla_\Phi \mathcal{L}_i(\Phi) \approx -\frac{1}{K}\sum_{k=1}^{K} \nabla_\Phi \log p_{\theta + f_\Phi(x_i, \epsilon_{ik})}(y_i \mid x_i). \tag{15}$$

Here $\epsilon_{i1}, \ldots, \epsilon_{iK} \overset{\text{iid}}{\sim} \mathcal{N}_n(0, I_n)$. In practice, we train with mini-batches $\mathcal{B}$ of query–response pairs and average these per-example gradients across the mini-batch.

In addition to approximating the reparameterized expectation through an average of $K$ samples, we also explore the $\max$ and $\min$ operators. Writing the sampled negative log-likelihood as $\ell_k = -\log p_{\theta + f_\Phi(x, \epsilon_k)}(y \mid x)$, these reductions are $\frac{1}{K}\sum_k \ell_k$, $\min_k \ell_k$, and $\max_k \ell_k$, respectively. The mean is the principled default: it is an unbiased Monte Carlo estimator of the expected-risk objective in Equation 6, gives every sample a gradient, and connects to established training methods based on stochastic weight distributions [32, 33].

The minimum and maximum instead optimize $K$-dependent order statistics and are therefore biased with respect to that expected-risk objective. The minimum is an optimistic best-of-$K$ or hard-assignment objective: it selects the highest-likelihood sampled adapter and resembles multiple-choice learning, where specializing different hypotheses can improve coverage of ambiguous outputs [34, 35]. This can align well with test-time sampling and selection, but its gradient depends on a single winning sample and can favor lucky samples or whichever mode wins early. The maximum is a pessimistic worst-of-$K$ objective related to robust optimization and adversarial weight perturbation [36, 37]. It can suppress catastrophic samples and favor locally robust solutions, but because the covariance is learned, the model can reduce the worst-case loss simply by collapsing its distribution.

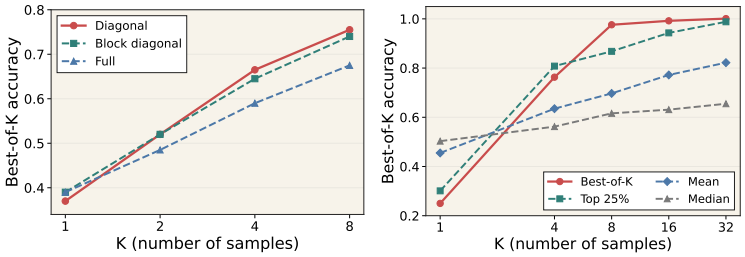



Figure 2: **Design choices for learning the weight distribution.** (a) Diagonal covariance performs comparably to more expressive covariance parameterizations. (b) Best-of-$K$ training preserves a more searchable distribution as the number of sampled updates increases.

All three reductions serve different goals: the mean targets average sampled performance, the minimum rewards search coverage, and the maximum rewards robustness. Behavior of minimum and maximum changes strongly with $K$. Without an explicit entropy reward, variance constraint, or prior, the mean and especially the maximum tend to shrink the learned covariance, while the minimum may preserve spread without guaranteeing diversity. Further details are in Appendix B.2.

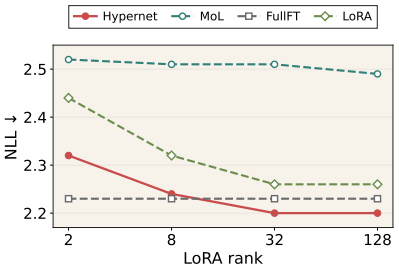
# 5 Experiments

## 5.1 Experimental Setup

**Models, training and evaluation data.** We evaluate our method on five language models, Qwen3-1.7B, Qwen3-4B, Gemma3-1B-Instruct, Llama3.2-1B-Instruct, and Llama3.2-3B-Instruct. We use a diverse set of benchmarks, covering standard LLM tasks GSM8K, MATH500, HumanEval, ARC-Challenge, MMLU-STEM, MedQA and GPQA-Diamond. Hypernetwork and baselines are trained using another collection of tasks NuminaMath CoT, MATH train, MathInstruct, MedMCQA (using only its training split), Magicoder OSS Instruct 75K, and CAMEL AI. These datasets cover a broad set of problems from mathematics, coding, science, and medicine. The data results in approximately 1.3 million training examples containing around 530 million tokens. We will report macro accuracy, as the unweighted mean of accuracy across the seven evaluation benchmarks.

**Distributional Hypernetwork architecture.** We start by constructing a deterministic version of the hypernetwork from Section 3.1, by modifying the T2L model [6]. This model receives as input a layer embedding, and a module embedding and a task embedding produced with a text encoder. We replace the last embedding by a learned pooled representation of the query, implemented with an additional attention mechanism over the last layer of the language model after it encodes the query. This network is trained with the loss in Equation 5. We construct a sampler for the distributional version of this network as explained in Section 3. This uses the same backbone but separately predicts mean and standard deviation. It then applies the Gaussian or convex Hull transformations to obtain samples from the LoRA distribution. The regression hypernetwork predicts LoRA factors directly, while the mixing hypernetwork predicts logits over a bank of reference LoRAs, which are softmaxed to obtain the mixture weights. The reference LoRAs are first trained for three epochs on the same training data, after which the hypernetwork is trained for one additional epoch. The mixing setting, therefore, receives more training exposure than the regression setting.

Training follows Equation 2. We explore training variants including Gaussian covariance learning as in Equation 10, modeled with a low-rank matrix product for tractability, as well as average, best-of-K and worst-of-K losses. We observe empirically that diagonal covariance and best-of-K losses provide the best results (Figure 2). Additional details about these analyses can be found in Appendix F. The resulting hypernetworks have approximately 300M parameters. We find that larger hypernetworks mainly improve single-update performance, while the gap narrows substantially as the weight-sampling budget increases (Appendix D.2). Inference is done either using the sampler or the predicted mean LoRA, as detailed. We provide the full architecture and additional implementation details in Appendix B.




| Method | Gemma3 1B | Llama3.2 1B | Qwen3 1.7B | Llama3.2 3B | Qwen3 4B |
|---|---|---|---|---|---|
| FullFT | 22.4 | 34.0 | 45.4 | 50.4 | 64.8 |
| LoRA | 18.7 | 25.2 | 46.0 | 46.7 | 60.6 |
| MoL | 21.0 | 22.6 | 42.2 | 44.1 | 60.6 |
| Hypernet (reg.) | 17.7 | 23.3 | 41.3 | 47.4 | 61.4 |
| Dist. Hypernet mean (reg.) | 21.1 | 27.1 | 48.4 | 50.5 | 61.5 |
| Hypernet (mix.) | 19.5 | 36.3 | 50.1 | **56.6** | **67.1** |
| Dist. Hypernet mean (mix.) | **31.3** | **38.9** | **55.7** | 56.0 | 65.3 |

Figure 3: **(a)** Effect of LoRA rank on adaptation performance. Increasing the rank improves the Hypernetwork performance over FullFT. **(b)** Macro accuracy over seven benchmarks across five base models. Distributional training generally improves the predicted update.

**Baselines.** We compare the proposed method with Full FineTuning (FullFT) and standard LoRA. We also include Mixture-of-LoRA (MoL), which uses a learned router to select one LoRA from a fixed set using top-1 routing [38, 39]. MoL is also conditioned on the input for the routing operation. This allows us to separate the benefit of generating an update from the benefit of simply choosing among several learned updates. We also consider the deterministic version of the distributional hypernetwork above. All methods use the same rank 8 except for the rank sensitivity analysis.

### 5.2 Predicting Weight Updates from Query

We first compare FullFT, LoRA, and MoL against the deterministic and distributional hypernetworks using the same training data, with the mean estimate used for the distributional hypernetwork.

We find that the LoRAs generated by the hypernetwork show trends similar to standard LoRAs but generally perform better. Figure 3 (left) shows average negative log-likelihood for the Qwen3-1.7B and Qwen3-4B models on held-out test splits of the FLAN tasks. Both standard and hypernetwork-generated LoRAs (regression) improve with rank. Hypernetwork-generated LoRAs even outperform full fine-tuning at rank 32 or higher, while the generated LoRA updates remain smaller than full-model updates. Figure 3 (right) shows average performance over the evaluation benchmarks for different models. As a general trend, hypernetworks outperform all other methods. The mixing hypernetwork performs better, which may reflect its larger size and the additional training of a bank of LoRAs. Parameter counts, including those of the hypernetwork itself, are reported in Appendix B.3.

**Finding 1:** ***A Single Query Contains Useful Adaptation Signal***

A single query contains enough adaptation signal to generate effective LoRA updates without task descriptions or demonstrations. The resulting updates match or outperform standard low-rank adaptation baselines.

Another relevant trend is that the distributional hypernetwork outperforms its deterministic counterpart in all but the largest model sizes. We attribute this improvement in part to regularization introduced by sampling during training. A deterministic hypernetwork is optimized only at the update it predicts, while a distributional hypernetwork receives gradients through perturbed updates sampled around its prediction. The resulting objective therefore depends not only on the loss at one point, but also on nearby points in weight space. For zero mean Gaussian perturbations, the expected loss can be viewed locally as the loss at the mean together with a curvature dependent term, favoring solutions whose nearby perturbations also have low loss. In this sense, distributional training encourages the predicted update to lie in a more robust region of the adaptation space rather than optimizing a single point. An extended set of examples can be found in Appendix G.1.

**Finding 2:** ***Distributional Training Improves the Predicted Update***

Training the hypernetwork as a distribution produces a stronger single update than directly training a point estimate. Even without weight sampling at test time, the mean of the learned distribution consistently outperforms the deterministic hypernet.

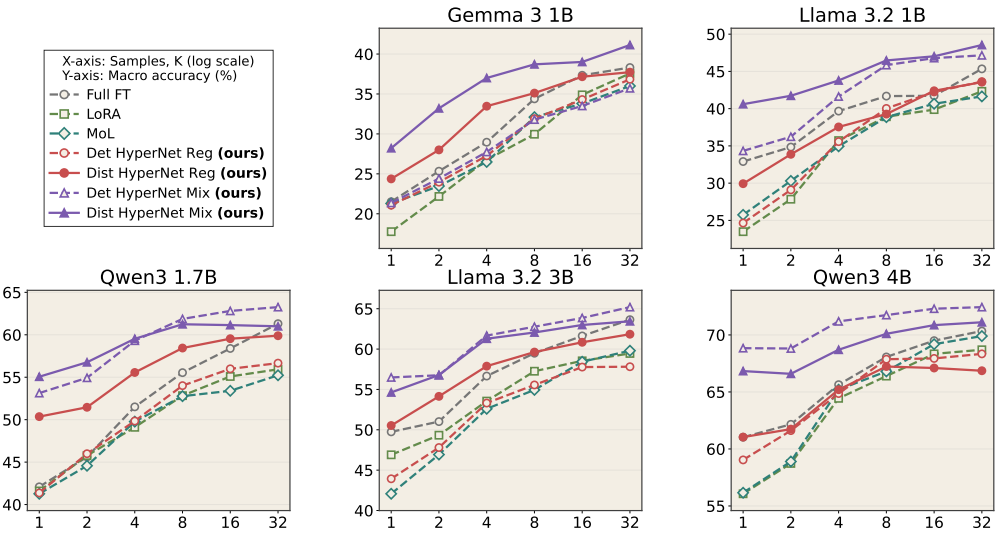



Figure 4: **Scaling test-time compute by sampling weights.** We report macro accuracy over seven benchmarks as the sampling budget increases. For each query, we generate $K$ candidate answers and use majority voting to obtain the final prediction. Dashed lines correspond to token sampling from a fixed model, including baselines and deterministic hypernetworks. Solid lines correspond to distributional hypernetworks, where each candidate is generated using a sampled weight update.

### 5.3 Predicting Weight Distribution from Query

The previous section showed that a single query contains enough adaptation signal to predict a useful update. Here we ask whether the query can also enable learning a distribution over good LoRA updates. A distributional hypernetwork provides more than a stronger single update. It also allows us to sample a set of alternative adaptations at test time, thus giving an additional way to spend test-time compute. Most test-time scaling methods keep the model weights fixed and spend additional compute on sampling more output sequences [40–42]. With the distributional hypernet, we can instead sample multiple LoRA updates, producing a different adapted model for each generation.

We compare these two ways of using additional compute through majority voting. For FullFT, LoRA, MoL, and deterministic hypernetworks, all $K$ generations are sampled from the same adapted weights. These methods therefore scale only through token sampling. For distributional hypernetworks, each generation uses a separately sampled weight update from the distribution. Increasing $K$ explores different adapted models in addition to output sequences. We use temperature $0.8$ for token sampling, while generations from sampled weight updates are decoded greedily ($T = 0$).

Figure 4 shows the resulting scaling behavior across five language models using macro accuracy over seven benchmarks. Weight sampling remains competitive with token sampling as the test-time budget increases, and is often stronger than the corresponding deterministic hypernet. This is especially clear for Qwen3 1.7B and Gemma 3 1B, where sampling from the learned weight distribution provides substantial gains over repeatedly decoding from a single predicted update. Even when the advantage is smaller, as for Qwen3 4B, weight sampling remains competitive with strong adaptation baselines. These results suggest that useful test-time diversity can come not only from sampling different outputs, but also from sampling different versions of the adapted model itself. We report the full results in Appendix G.2. We consider regression and mixing hypernetworks separately and do not compare their performances against each other. Our comparison instead focuses on the effect of deterministic token sampling and distributional weight sampling within each parameterization.

**Finding 3:** ***Test-Time Compute Can Be Spent in Weight Space***

Sampling weight updates provides a new way for test-time scaling. Across model families, weight sampling remains competitive with token sampling and often performs better, showing that useful diversity can come from changing the adapter rather than only its generated tokens.

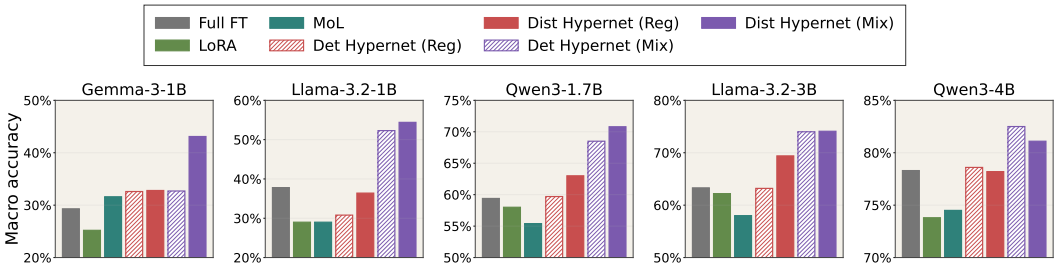



Figure 5: Generated update distributions transfer across queries. We aggregate hypernetwork outputs from one half of each benchmark and apply the resulting adaptation to the held-out half.

### 5.4 Reusing Generated Updates Across Queries

So far, we have generated a separate update for every test query. We next ask whether the updates produced for different queries contain structure that can be accumulated and reused. For each benchmark, we split the examples into two disjoint halves. We use only the prompts from the first half to generate hypernetwork outputs and combine them into a single shared LoRA. This adapter is then kept fixed and used for every query in the second half. Importantly, the hypernetwork is never called on the held-out queries and no labels from the first half are used. Although the hypernetwork observes the first-half queries as conditioning inputs, neither it nor the baselines receives labels or undergoes optimization on these examples. The experiment therefore measures whether the updates generated from one set of queries can transfer to unseen queries from the same benchmark.

For the regression hypernet, we obtain LoRA factors for each query in the first half. The deterministic hypernetwork contributes its prediction, while for the distributional hypernetwork we use the predicted mean without sampling. We average the $A$ factors across queries and separately average the $B$ factors, producing one rank-$r$ LoRA reused on the held-out half. For the mixing hypernet, each query produces mixing weights over the reference LoRA bank. We average these coefficients across the first half and use them to construct one shared adapter for the second half.

Figure 5 shows that the generated updates retain useful information across queries. Although the shared adapter is constructed entirely from a disjoint set of prompts, it remains effective on the held-out half, and the distributionally trained hypernetworks generally produce stronger reusable adaptations than their deterministic counterparts. This suggests that the hypernetwork is learning more than isolated per-query corrections: its outputs capture regularities that can be accumulated across examples and reused without calling the hypernetwork on every new query. Thus, the same mechanism can support both query-specific adaptation and a persistent shared adapter built from previously observed inputs.

## 6 Conclusion

We study how much adaptation signal can be extracted from a single query. Our results show that this is sufficient to generate useful query-conditioned LoRA updates and distributions over plausible updates. Distributional hypernetworks further improve the predicted update and enable test-time scaling through weight sampling. Sampling different adapted models is competitive with, and often stronger than, sampling more outputs from fixed weights. Finally, the generated updates exhibit reusable structure across queries, supporting both per-query and shared task-level adaptation.

## 7 Limitations

Our experiments focus on 1–4B parameter models and LoRA-based adaptation, and it remains to be seen how the behavior scales to substantially larger models. The hypernetwork must also be kept in memory and adds one forward pass per query, but inference remains considerably lighter than optimization-based adaptation since it requires no backward pass, optimizer state, or iterative updates. Given the consistent gains from generated and sampled updates, scaling this approach to larger models and more efficient hypernetworks is a promising direction for future work.

## Reproducibility Statement

We provide the information necessary to reproduce our experiments throughout the main paper and appendix. We specify the models and benchmarks used for evaluation, the datasets used for training, and the important training and implementation hyperparameters. Additional experimental and implementation details are provided in the appendix.

## AI Use Statement

We used AI tools to assist with writing and editing code and for editing portions of the manuscript. All AI-assisted code and text were reviewed by the authors. The authors take full responsibility for the final content of the paper, including all code, text, claims, and results.

## References


[1] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. 1, 3

[2] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016. 1, 2, 14

[3] Jurgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–138, 1992. 1, 14

[4] Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. *Advances in neural information processing systems*, 29, 2016. 1, 14

[5] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016. 1

[6] Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-lora: Instant transformer adaption. *arXiv preprint arXiv:2506.06105*, 2025. 1, 2, 3, 6, 14

[7] Jason Phang, Yi Mao, Pengcheng He, and Weizhu Chen. Hypertuning: Toward adapting large language models without back-propagation. In *International Conference on Machine Learning*, pages 27854–27875. pmlr, 2023. 1, 2, 3, 14

[8] Hamish Ivison, Akshita Bhagia, Yizhong Wang, Hannaneh Hajishirzi, and Matthew E Peters. Hint: hypernetwork instruction tuning for efficient zero-and few-shot generalisation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11272–11288, 2023. 2, 14

[9] Yewei Liu, Xiyuan Wang, Yansheng Mao, Yoav Gelbery, Haggai Maron, and Muhan Zhang. SHINE: A scalable in-context hypernetwork for mapping context to LoRA in a single pass. *arXiv preprint arXiv:2602.06358*, 2026. 2, 14

[10] Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-LoRA: Learning to instantly internalize contexts. *arXiv preprint arXiv:2602.15902*, 2026. 14

[11] Tomer Volk, Eyal Ben-David, Ohad Amosy, Gal Chechik, and Roi Reichart. Example-based hypernetworks for multi-source adaptation to unseen domains. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9097–9121, 2023. 2, 14

[12] Hamish Ivison and Matthew E Peters. Hyperdecoders: Instance-specific decoders for multi-task nlp. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 1715–1730, 2022. 2, 3, 14

[13] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International conference on machine learning*, pages 9229–9248. PMLR, 2020. 2, 14

[14] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. *arXiv preprint arXiv:2006.10726*, 2020. 14

[15] Jinwu Hu, Zhitian Zhang, Guohao Chen, Xutao Wen, Chao Shuai, Wei Luo, Bin Xiao, Yuanqing Li, and Mingkui Tan. Test-time learning for large language models. *arXiv preprint arXiv:2505.20633*, 2025. 14

[16] Yang Hu, Xingyu Zhang, Xueji Fang, Zhiyang Chen, Xiao Wang, Huatian Zhang, and Guojun Qi. Slot: Sample-specific language model optimization at test-time. *arXiv preprint arXiv:2505.12392*, 2025.

[17] Arnuv Tandon, Karan Dalal, Xinhao Li, Daniel Koceja, Marcel Rød, Sam Buchanan, Xiaolong Wang, Jure Leskovec, Sanmi Koyejo, Tatsunori Hashimoto, et al. End-to-end test-time training for long context. *arXiv preprint arXiv:2512.23675*, 2025.

[18] Rachit Bansal, Aston Zhang, Rishabh Tiwari, Lovish Madaan, Venkata Sai Surya Subramanyam Duvvuri, Devvrit Khatri, David Brandfonbrener, David Alvarez-Melis, Prajjwal Bhargava, Mihir Kale, et al. Let's (not) just put things in context: Test-time training for long-context llms. In *International Conference on Learning Representations*, 2026. 2, 14

[19] David Krueger, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron Courville. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*, 2017. 2, 14

[20] Nick Pawlowski, Andrew Brock, Matthew C. H. Lee, Martin Rajchl, and Ben Glocker. Implicit weight uncertainty in neural networks. *arXiv preprint arXiv:1711.01297*, 2017. 2

[21] Neale Ratzlaff and Fuxin Li. HyperGAN: A generative model for diverse, performant neural networks. *arXiv preprint arXiv:1901.11058*, 2019. 2, 14

[22] Yihua Shao, Minxi Yan, Yang Liu, Siyu Chen, Wenjie Chen, Xinwei Long, Ziyang Yan, Lei Li, Chenyu Zhang, Nicu Sebe, et al. In-context meta lora generation. *arXiv preprint arXiv:2501.17635*, 2025. 3, 14

[23] Yujia Wu, Yiming Shi, Jiwei Wei, Chengwei Sun, Yang Yang, and Heng Tao Shen. DiffLoRA: Generating personalized low-rank adaptation weights with diffusion. *arXiv preprint arXiv:2408.06740*, 2024. 3, 14

[24] Adam X. Yang, Maxime Robeyns, Xi Wang, and Laurence Aitchison. Bayesian low-rank adaptation for large language models. *arXiv preprint arXiv:2308.13111*, 2023. 3, 14

[25] Yibin Wang, Haizhou Shi, Ligong Han, Dimitris Metaxas, and Hao Wang. BLoB: Bayesian low-rank adaptation by backpropagation for large language models. *arXiv preprint arXiv:2406.11675*, 2024. 14

[26] Liyi Zhang, Jake C. Snell, and Thomas L. Griffiths. Amortized bayesian meta-learning for low-rank adaptation of large language models. In *Proceedings of the Workshop on Uncertainty-Aware NLP*, 2025. doi: 10.18653/v1/2025.uncertainlp-main.17. 3, 14

[27] Xi Wang, Laurence Aitchison, and Maja Rudolph. LoRA ensembles for large language model fine-tuning. *arXiv preprint arXiv:2310.00035*, 2023. 3, 14

[28] Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing (volume 1: long papers)*, pages 7319–7328, 2021. 3

[29] Yulu Gan and Phillip Isola. Neural thickets: Diverse task experts are dense around pretrained weights. *arXiv preprint arXiv:2603.12228*, 2026. 4

[30] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 4

[31] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014. 4

[32] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622. PMLR, 2015. 5, 14

[33] Durk P. Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, volume 28, 2015. 5

[34] Abner Guzman-Rivera, Dhruv Batra, and Pushmeet Kohli. Multiple choice learning: Learning to produce multiple structured outputs. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2942–2949, 2012. 5

[35] Stefan Lee, Senthil Purushwalkam Shiva Prakash, Michael Cogswell, Viresh Ranjan, David Crandall, and Dhruv Batra. Stochastic multiple choice learning for training diverse deep ensembles. *Advances in Neural Information Processing Systems*, 29, 2016. 5

[36] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. 5

[37] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021. 5

[38] Wenfeng Feng, Chuzhan Hao, Yuewei Zhang, Yu Han, and Hao Wang. Mixture-of-loras: An efficient multitask tuning method for large language models. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 11371–11380, 2024. 7

[39] Ziyu Zhao, Leilei Gan, Guoyin Wang, Wangchunshu Zhou, Hongxia Yang, Kun Kuang, and Fei Wu. LoRAretriever: Input-aware LoRA retrieval and composition for mixed tasks in the wild. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. doi: 10.18653/v1/2024.findings-acl.263. 7, 14

[40] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. 8, 14

[41] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[42] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20286–20332, 2025. 8, 14

[43] Alexandra Chronopoulou, Matthew E. Peters, Alexander Fraser, and Jesse Dodge. Adaptersoup: Weight averaging to improve generalization of pretrained language models. *arXiv preprint arXiv:2302.07027*, 2023. 14

[44] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. *arXiv preprint arXiv:2203.05482*, 2022. 14

[45] Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. LoRAHub: Efficient cross-task generalization via dynamic LoRA composition. *arXiv preprint arXiv:2307.13269*, 2023. 14

[46] Hanqing Wang, Bowen Ping, Shuo Wang, Xu Han, Yun Chen, Zhiyuan Liu, and Maosong Sun. LoRA-flow: Dynamic LoRA fusion for large language models in generative tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. doi: 10.18653/v1/2024.acl-long.695. 14

[47] Shangbin Feng, Zifeng Wang, Yike Wang, Sayna Ebrahimi, Hamid Palangi, Lesly Miculicich, Achin Kulshrestha, Nathalie Rauschmayr, Yejin Choi, Yulia Tsvetkov, Chen-Yu Lee, and Tomas Pfister. Model swarms: Collaborative search to adapt LLM experts via swarm intelligence. *arXiv preprint arXiv:2410.11163*, 2024. 14

[48] Conor F. Hayes, Elliot Meyerson, Kajetan Schweighofer, Roberto Dailey, Babak Hodjat, Risto Miikkulainen, and Xin Qiu. Beyond the best guess: Improving LLM solution coverage with evolution strategies. *arXiv preprint arXiv:2608.12679*, 2026. 14

[49] Roman Soletskyi, Marylou Gabrié, and Bruno Loureiro. A theoretical perspective on mode collapse in variational inference. *Machine Learning: Science and Technology*, 6(2):025056, 2025. 20

# A Extended Related Work

**Test-Time Training.** Test-time training adapts a model to the current input by updating its parameters during inference. Early methods construct a self-supervised objective from the test input and take one or more gradient steps to adapt to distribution shift [13, 14]. More recent work applies the same idea to language models, including sample-specific optimization, test-time learning, and adaptation to long contexts [15–18]. These methods differ in the objective and the parameters being updated, but all perform optimization after seeing the test input. Our goal is complementary. We train the adaptation mechanism beforehand so that a useful update can be produced directly from the query without gradient-based optimization at test time.

**Gradient-Free Adaptation.** Hypernetworks provide a natural way to amortize parameter adaptation by learning a network that generates the parameters of another network [2]. This idea is closely related to fast weights, where a set of learned parameters controls another set of parameters that can change more rapidly [3, 4]. In language models, HyperTuning and HINT generate parameter-efficient adaptations from demonstrations or instructions [7, 8], while Text-to-LoRA generates a LoRA directly from a natural-language task description [6]. More recent work such as SHINE and Doc-to-LoRA maps longer contexts into LoRA weights in a single pass [9, 10], and example-based hypernetworks generate adaptations from examples from related domains [11]. HyperDecoders move closer to instance-level adaptation by generating a separate decoder adaptation from each input [12]. These approaches show that useful parameter updates can be generated without test-time backpropagation. Our setting differs in that the semantic information comes only from the current query and the hypernetwork predicts a distribution over possible LoRA updates rather than a single adaptation.

**Distributions over Weights.** Learning distributions over neural network parameters has a long history in Bayesian neural networks and generative weight models. Bayes by Backpropagation learns distributions over weights through variational inference [32], while Bayesian Hypernetworks, implicit weight uncertainty, and HyperGAN use learned generators to represent richer distributions over neural network parameters [19–21]. In these approaches, samples typically represent uncertainty or different models drawn from a global learned weight distribution.

Several recent methods study distributions specifically in the low-rank adaptation space. ICM-LoRA learns a conditional generative model from a collection of LoRAs that were optimized beforehand [22]. DiffLoRA similarly learns to generate LoRA weights from previously constructed personalization adapters using a diffusion model [23]. These methods provide important evidence that LoRA weights can be modeled with a conditional generative distribution, but the distribution is learned by reproducing previously optimized adapters. Bayesian LoRA methods take a different view. Laplace-LoRA constructs a distribution around a fitted LoRA after adaptation, while BLoB learns a variational distribution over LoRA parameters during fine-tuning [24, 25]. Amortized Bayesian meta-learning similarly represents uncertainty over task-specific low-rank adaptations [26]. LoRA ensembles obtain weight diversity by independently optimizing several adapters for the same task [27].

Our distribution is different in both how it is produced and how it is used. It is conditioned on each query and trained directly through the frozen language model's response loss rather than by reconstructing a set of adapters or approximating a task-level posterior. Samples from the distribution are treated as alternative adaptations for the current query rather than only as uncertainty estimates. This also means that the predicted distribution may change from one query to the next, even when the queries come from the same benchmark or task.

**Test-Time Scaling.** Most test-time scaling methods keep the model parameters fixed and spend additional compute by generating and selecting among more output sequences [40–42]. Other work shows that useful variation can also exist in weight space. AdapterSoup and Model Soups average independently trained parameters [43, 44], while LoRAHub, LoRARetriever, and LoRA-Flow retrieve or combine adapters for a new input or task [39, 45, 46]. Model Swarms searches among and modifies model experts during inference, and evolutionary weight perturbations provide another way to search for diverse solutions in parameter space [47, 48]. These results motivate weight space itself as a useful source of test-time diversity.

Our approach learns this search distribution during training. At test time, the hypernetwork directly produces a distribution from which multiple adapted models can be sampled. We study this in two settings. The regression hypernetwork predicts a distribution directly over LoRA weights, while the mixing hypernetwork predicts a distribution over combinations of reference LoRAs. The latter is related to adapter retrieval and composition, but instead of returning one composition, it defines several possible adaptations that can be explored with additional test-time compute.

# B Hypernetwork Details

## B.1 Hypernetwork Architecture

Figure 6 shows the architecture used for the hypernetwork. The hypernetwork is conditioned on three inputs: the token-level representation of the input query, a learned layer embedding, and a learned module embedding. The query is first passed through the frozen language model to obtain a sequence of hidden states. The hidden states are projected to the internal dimension of the hypernetwork before being used as the context representation.

We use this pair representation as the *query* to a cross-attention layer, while the sequence of query token representations acts as the keys and values. Thus, each $(\text{layer}, \text{module})$ pair can attend to a different part of the input when constructing its LoRA parameters. The cross-attention output is combined with the pair representation through a residual connection and then processed by eight residual MLP blocks. A final head projector consisting of LayerNorm, a linear layer, and a SiLU activation produces the representation used by the output heads.

The output layer contains separate heads for the two LoRA factors A and B. For a distributional version of the hypernetwork with diagonal covariance, the same features are used to predict both the mean and element-wise standard deviation of each factor, $\mu_{\mathrm{A}}(x), \sigma_{\mathrm{A}}(x), \mu_{\mathrm{B}}(x)$, and $\sigma_{\mathrm{B}}(x)$. A sampled LoRA is obtained through the reparameterization

$$\mathrm{A} = \mu_{\mathrm{A}}(x) + \sigma_{\mathrm{A}}(x) \odot \epsilon_{\mathrm{A}}, \qquad \epsilon_{\mathrm{A}} \sim \mathcal{N}(0, I), \tag{16}$$

$$\mathrm{B} = \mu_{\mathrm{B}}(x) + \sigma_{\mathrm{B}}(x) \odot \epsilon_{\mathrm{B}}, \qquad \epsilon_{\mathrm{B}} \sim \mathcal{N}(0, I), \tag{17}$$

where the noise for A and B is sampled independently. The deterministic hypernetwork uses the same architecture but predicts only the corresponding A and B point estimates.

## B.2 Training Details

For the distributional hypernetwork, we sample $K$ candidate LoRA updates for each query and evaluate the language modeling loss for each corresponding adapted model. Under the best-of-$K$ objective, the candidate with the lowest loss is selected, and the backward pass is performed through that sampled update. Figure 7 illustrates this gradient flow using the colored arrows. The gradient passes from the language modeling loss through the adapted language model, through the generated LoRA update, and finally into the hypernetwork that produced it. Although gradients are propagated through the language model to obtain the learning signal, its pretrained weights remain frozen throughout training. Only the hypernetwork parameters are updated. Because the sampled LoRA is produced through the reparameterized distribution, the same gradient also trains the parameters controlling both the center and spread of the predicted update distribution.

## B.3 Hypernetwork Parameter Count

The size of the hypernetwork depends on the base language model. Although we use rank-8 LoRA updates throughout the main experiments, different base models have different hidden dimensions and numbers and shapes of the target matrices. Consequently, the number of LoRA parameters that the hypernetwork must generate varies across models. Since the output heads of the hypernetwork scale with the dimensionality of these generated LoRA factors, the total hypernetwork parameter count varies accordingly.

Table 1 reports the parameter counts for each base model. The deterministic hypernetwork predicts one set of LoRA factors, whereas the distributional hypernetwork additionally predicts the parameters controlling the spread of the distribution. For the diagonal Gaussian parameterization

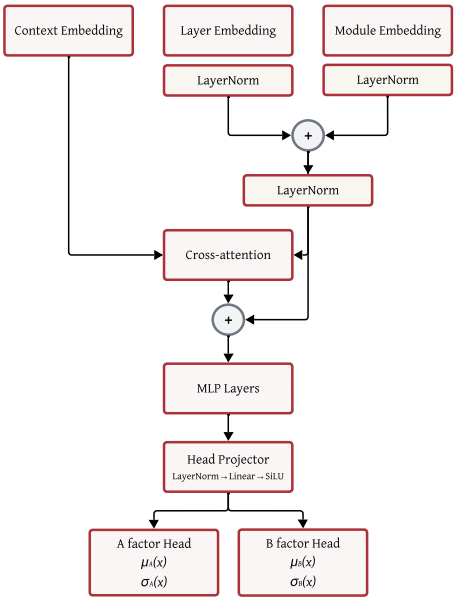



Figure 6: **Hypernetwork architecture.** Each layer–module pair attends over the token-level query representation and produces module-specific LoRA factor parameters.

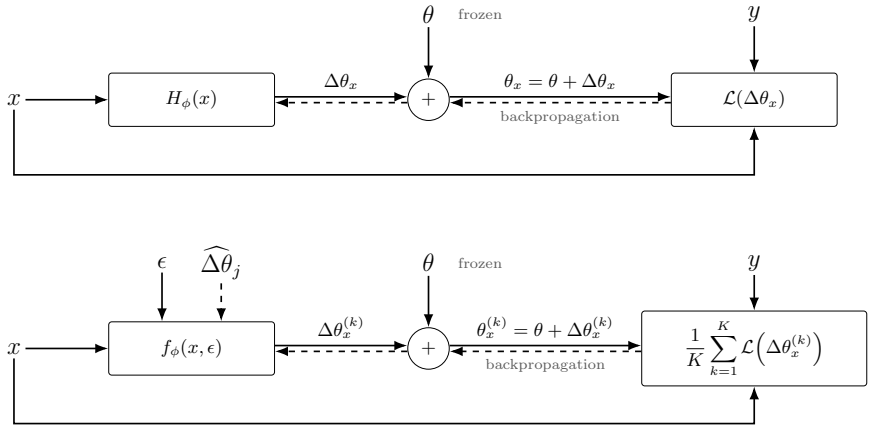



Figure 7: **Training the hypernetwork.** The hypernetwork generates an update $\Delta W_x$, which is added to the frozen language model weights $W_0$. The language modeling loss is backpropagated through the adapted model and the generated update to train only the hypernetwork parameters.

used in our main experiments, this introduces additional $\sigma$ heads for the LoRA factors and therefore increases the size of the hypernetwork.

We also examine how the parameter count changes with the LoRA rank. Table 2 shows this scaling for Qwen3-1.7B and Qwen3-4B, comparing FullFT, standard LoRA, and the deterministic hypernetwork. As the rank increases, both the generated LoRA update and the hypernetwork output heads grow accordingly.

## C Implementation Details

**Sampling.** During training, we draw Gaussian noise in antithetic pairs, $\epsilon$ and $-\epsilon$, to reduce the variance introduced by sampling.

Table 1: Parameter counts across base models and adaptation methods. FullFT reports the number of trainable LM parameters, while LoRA reports the trainable parameters of a single rank-8 adapter. For mixing hypernetworks, the reported count includes both the hypernetwork and the fixed bank of 50 rank-8 reference LoRAs.

| | | | **Regression Hypernet** | | **Mixing Hypernet + Bank** | |
|---|---|---|---|---|---|---|
| **Base model** | **FullFT** | **LoRA** $r = 8$ | **Det.** | **Dist.** | **Det.** | **Dist.** |
| Qwen3-1.7B | 1,700M | 5.5M | 194M | 295M | 368.0M | 368.1M |
| Qwen3-4B | 4,000M | 10.6M | 245M | 396M | 624.1M | 624.2M |
| Gemma-3-1B | 1,000M | 5.0M | 192M | 291M | 343.3M | 343.4M |
| Llama-3.2-1B | 1,000M | 3.9M | 219M | 345M | 289.3M | 289.4M |
| Llama-3.2-3B | 3,000M | 7.6M | 233M | 372M | 472.4M | 472.5M |

Table 2: Parameter scaling with LoRA rank for Qwen3-1.7B and Qwen3-4B. The LoRA row gives the number of trainable parameters for standard LoRA, which is also the size of the update generated by the hypernetwork.

| **Qwen3-1.7B** | | | | |
|---|---|---|---|---|
| **Approach** | **Rank 2** | **Rank 8** | **Rank 32** | **Rank 128** |
| FullFT trainable | ∼1,700M | ∼1,700M | ∼1,700M | ∼1,700M |
| LoRA trainable / generated update | 1.4M | 5.5M | 22.0M | 88.1M |
| Deterministic hypernet trainable | 118M | 194M | 497M | 1,707M |
| **Qwen3-4B** | | | | |
| **Approach** | **Rank 2** | **Rank 8** | **Rank 32** | **Rank 128** |
| FullFT trainable | ∼4,000M | ∼4,000M | ∼4,000M | ∼4,000M |
| LoRA trainable / generated update | 2.7M | 10.6M | 42.5M | 169.9M |
| Deterministic hypernet trainable | 132M | 245M | 699M | 2,514M |

**Implementation of the mixture parameterization.** Equation 13 sums R full-size updates, which would be wasteful to form explicitly. We instead evaluate it as a single low-rank product. For each target matrix $s \in \mathcal{M}$, write the reference adapters in factored form $\widehat{\Delta\theta}_j^{(s)} = \frac{\alpha}{r}\mathrm{B}_j^{(s)}\mathrm{A}_j^{(s)}$ with $\mathrm{A}_j^{(s)} \in \mathbb{R}^{r\times d_{\text{in}}}$ and $\mathrm{B}_j^{(s)} \in \mathbb{R}^{d_{\text{out}}\times r}$, and stack them, scaling only the A side by the sampled mixing weights:

$$\mathrm{A}_{\text{cat}}^{(s)} = \begin{bmatrix} \pi_{x,1}^{(k)}\mathrm{A}_1^{(s)} \\ \vdots \\ \pi_{x,\mathrm{R}}^{(k)}\mathrm{A}_{\mathrm{R}}^{(s)} \end{bmatrix}, \qquad \mathrm{B}_{\text{cat}}^{(s)} = \begin{bmatrix} \mathrm{B}_1^{(s)} & \cdots & \mathrm{B}_{\mathrm{R}}^{(s)} \end{bmatrix}, \qquad \mathrm{B}_{\text{cat}}^{(s)}\mathrm{A}_{\text{cat}}^{(s)} = \sum_{j=1}^{\mathrm{R}} \pi_{x,j}^{(k)}\,\mathrm{B}_j^{(s)}\mathrm{A}_j^{(s)} \tag{18}$$

The right-hand identity is exact, not an approximation, and holds for any $\pi$, we verify it numerically to floating-point tolerance. The sampled adaptation is therefore applied as one LoRA of rank R$r$ rather than as R separate updates, and by linearity it is also equivalent to applying each reference adapter to the input and summing the outputs with weights $\pi_x^{(k)}$, as mixture-of-LoRA architectures do. Two consequences are worth stating. First, the effective rank of a mixture sample is up to R$r$, whereas averaging the factors instead of the updates would keep rank $r$; the latter is the convention used for *merging* adapters into a single deployable module (`peft`'s `linear` mode, LoRAhub), but it computes $(\sum_j \pi_j \mathrm{B}_j)(\sum_j \pi_j \mathrm{A}_j)$, which differs from Equation 13 by cross terms $\mathrm{B}_i\mathrm{A}_j$, $i \neq j$, that carry $1 - \sum_j \pi_j^2$ of the total weight and were never trained. Since we require samples to be convex combinations of the reference updates, we use the exact form throughout. Second, this costs no additional parameters: $\mathrm{A}_{\text{cat}}$ and $\mathrm{B}_{\text{cat}}$ are views of the frozen bank, and only the adapter-path compute grows with R.

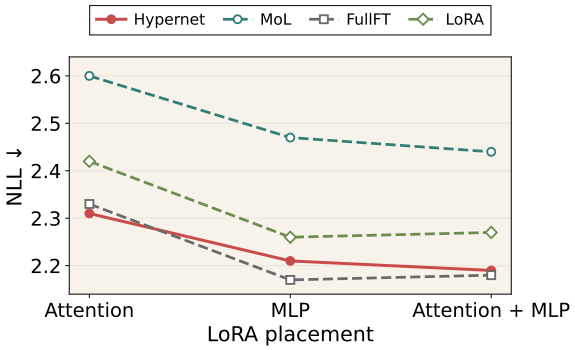



Figure 8: Effect of LoRA placement. Hypernetwork-generated updates follow the same trend as directly optimized LoRAs: adapting the MLP is more effective than adapting attention alone, while adapting both provides only a small additional improvement.

Table 3: Effect of hypernetwork capacity on distributional weight sampling. We report accuracy for different hypernetwork sizes and sampling budgets. Increasing hypernetwork capacity generally improves single-sample performance, while the gap between small and large hypernetworks becomes much smaller as the number of sampled weight updates increases.

| **Base model** | **Hypernet size** | $K = 1$ | $K = 8$ | $K = 64$ |
|---|---|---|---|---|
| Qwen3-1.7B | 25M | 44.7 | 51.9 | 53.3 |
| | 100M | 45.3 | 53.0 | 53.7 |
| | 500M | 44.5 | 53.2 | 53.9 |
| | 1B | 46.7 | 53.5 | 54.2 |
| Qwen3-4B | 25M | 51.3 | 60.1 | 61.2 |
| | 100M | 50.1 | 60.5 | 61.3 |
| | 500M | 54.1 | 61.4 | 61.8 |
| | 1B | 55.8 | 61.4 | 62.1 |

# D Additional Experiments

## D.1 Effect of Hypernetwork-generated LoRA Placement

We additionally study where the hypernetwork-generated LoRA update should be applied. Figure 8 compares adapting the attention projections, the MLP projections, or both. The generated updates follow the same qualitative behavior as directly optimized LoRAs. Updating the MLP gives a larger improvement than updating attention alone, while adapting both attention and MLP provides only a small additional gain. This suggests that the behavior of the generated updates is consistent with ordinary gradient-based LoRA adaptation rather than relying on a particular choice of target module.

## D.2 Effect of Hypernetwork Capacity.

We vary the size of the distributional hypernetwork from approximately 25M to 1B parameters while keeping the base model and training objective fixed. Larger hypernetworks generally improve performance, but the benefit is much more pronounced when using a single predicted update. For Qwen3-4B, for example, increasing the hypernetwork size from 25M to 1B improves accuracy by 4.5 points at $K = 1$, but by only 0.9 points at $K = 64$. A similar pattern appears for Qwen3-1.7B.

These results suggest that hypernetwork capacity is most important for producing a strong individual update, while sampling multiple weight updates reduces the dependence on having a very large hypernetwork. Since our main setting uses the distribution specifically for test-time weight sampling, we use moderately sized hypernetworks in the main experiments rather than scaling the hypernetwork aggressively.

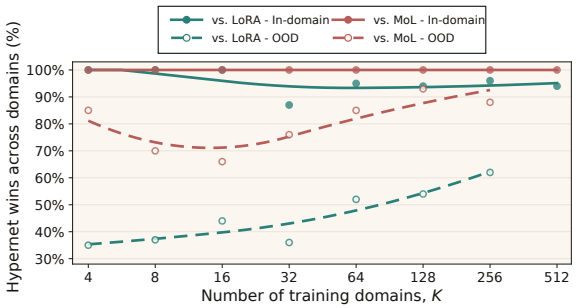



Figure 9: **The benefit of generating updates grows with task diversity.** We report the percentage of domains where the hypernetwork outperforms each baseline. Higher values indicate a larger advantage for the hypernetwork.

## E When Does Generating an Update Become Useful?

Generating a different update for every query may not always be necessary. If different inputs require similar adaptations, a single shared LoRA may be sufficient. We therefore vary the diversity of the training distribution by training on $K$ domains, with $K$ ranging from 4 to 512, and compare the deterministic hypernetwork with a single LoRA and MoL.

Figure 9 shows that the benefit of generation increases with task diversity. On in-domain data, the hypernetwork consistently outperforms the shared LoRA. On unseen domains, the shared LoRA remains competitive at low diversity, but becomes less effective as $K$ increases and a single update becomes less representative of what different queries require. MoL provides a stronger query-conditioned baseline, but remains limited to selecting from a fixed set of updates, while the hypernetwork can generate a new update for each query.

**Finding 4:** ***Task Diversity Determines When Updates Should Be Generated***

A shared update works well when tasks are similar, but becomes less representative as task diversity grows. At high diversity, generating input-conditioned updates scales better than relying on a fixed update or a fixed bank of updates.

## F Learning a Distribution over Weight Updates

### F.1 Choosing the Covariance

A distributional hypernetwork also requires a choice of how to represent the covariance over its generated updates. We compare diagonal, block diagonal, and full covariance parameterizations. Since directly predicting the latter two would be expensive, we represent them using low-rank factorizations.

Figure 2(left) shows that a more expressive covariance provides little benefit. The three parameterizations perform similarly at different sampling budgets, while the diagonal covariance performs slightly better as the number of samples increases. At $K = 8$, for example, the diagonal covariance reaches approximately 0.75 mean accuracy, compared with 0.74 for block diagonal and 0.68 for full covariance. We therefore use a diagonal covariance throughout the remaining experiments, since it is both simpler and cheaper to parameterize. The results shown are mean accuracy across 8 benchmarks.

### F.2 Learning a Searchable Distribution

The distribution is useful at test-time only if different samples continue to provide meaningfully different candidate updates. This creates a second requirement on the training objective. If the loss requires a large fraction of the sampled updates to perform well, the hypernetwork can reduce the loss by shrinking the predicted variance and concentrating its samples around a single update. We

Table 4: Benchmark-level comparison of deterministic and distributional training for regression and mixing hypernetworks across five base models on the seven benchmarks used in our main evaluation. Accuracy is reported in percentage points. Higher is better.

| | Qwen3 1.7B | | | | Qwen3 4B | | | | Gemma 3 1B | | | | Llama 3.2 1B | | | | Llama 3.2 3B | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Regression | | Mixing | | Regression | | Mixing | | Regression | | Mixing | | Regression | | Mixing | | Regression | | Mixing | |
| Benchmark | Det. | Dist. | Det. | Dist. | Det. | Dist. | Det. | Dist. | Det. | Dist. | Det. | Dist. | Det. | Dist. | Det. | Dist. | Det. | Dist. | Det. | Dist. |
| GSM8K | 56.8 | **64.0** | **78.0** | 69.2 | **64.4** | 62.4 | **88.0** | 82.4 | **29.2** | 28.8 | 30.4 | **41.6** | **44.0** | 40.8 | 49.2 | **52.0** | 68.8 | **72.4** | 82.8 | **84.4** |
| MATH500 | 34.4 | **42.0** | 46.0 | **47.2** | 52.8 | **56.0** | **56.4** | 45.2 | 20.0 | **21.2** | 18.4 | **26.4** | 22.0 | **23.6** | **24.8** | 23.2 | **39.2** | 38.0 | **40.4** | 37.6 |
| HumanEval | 45.1 | **47.6** | 48.2 | **59.2** | 59.2 | **61.0** | **66.5** | 65.2 | 7.9 | **12.8** | 15.8 | **24.4** | 30.5 | **31.7** | 36.0 | **38.4** | 45.7 | **48.8** | 48.8 | **51.2** |
| ARC-C | 59.2 | **68.0** | 71.6 | **76.0** | **86.0** | 85.2 | 83.6 | **86.4** | 23.6 | **30.4** | 22.0 | **43.6** | 19.6 | **27.2** | 50.0 | **54.8** | 65.2 | **72.8** | **76.0** | 74.4 |
| MMLU-STEM | 52.0 | **64.8** | 58.8 | **66.8** | **76.4** | 74.4 | **82.0** | 81.2 | 23.6 | **25.2** | 20.0 | **40.0** | 17.2 | **28.8** | 34.8 | **39.6** | 47.2 | **48.8** | **61.2** | 58.8 |
| MedQA | 23.6 | **32.4** | 27.6 | **44.0** | 55.2 | **60.4** | 57.2 | **60.4** | 9.2 | **15.2** | 15.2 | **21.6** | 18.0 | **23.2** | 40.8 | **41.6** | 49.2 | **59.6** | **62.0** | 60.0 |
| GPQA-Diamond | 17.7 | **19.7** | 20.7 | **27.8** | **35.9** | 31.3 | 35.9 | **36.4** | 10.6 | **14.1** | 14.6 | **21.7** | 11.6 | **14.1** | 18.2 | **22.7** | **16.7** | 13.1 | **25.2** | **25.2** |
| **Macro** | 41.3 | **48.4** | 50.1 | **55.7** | 61.4 | **61.5** | **67.1** | 65.3 | 17.7 | **21.1** | 19.5 | **31.3** | 23.3 | **27.1** | 36.3 | **38.9** | 47.4 | **50.5** | **56.6** | 56.0 |

refer to this behavior as *distribution collapse*, where the predicted variance shrinks and repeated samples approach the same update.

We therefore compare objectives that place different amounts of pressure on the sampled distribution. The mean objective optimizes all sampled updates, while the median and top fraction objectives require progressively smaller portions of the distribution to perform well. Best-of-$K$ places the weakest constraint on the full distribution, since only the best sampled update receives the optimization signal.

The choice of $K$ also controls the balance between guiding the center of the distribution and preserving its spread. With a small finite $K$, the best sample remains closely coupled to the mean, providing a useful signal for where the mean should move. As $K$ increases, the objective becomes more tolerant of a wider distribution, since a good sample can be found farther from the mean. Top fraction objectives provide an intermediate tradeoff, applying more pressure on the mean and favoring a smaller variance than best-of-$K$, while still avoiding the stronger contraction encouraged by the mean objective.

We measure the resulting coverage using oracle performance as the number of weight samples increases. As shown in Figure 2(right), the difference becomes clear as the sampling budget grows. Thus, objectives that require more of the distribution to perform well leave substantially less room for search, while best-of-$K$ retains the largest searchable region.

Mode collapse is a distinct failure mode in generative modeling, where a learned distribution fails to represent multiple valid modes [49]. Our concern here is different: the predicted variance may shrink toward zero. Mode collapse concerns whether a model covers several distinct valid target modes. Distribution collapse instead refers to the predicted variance shrinking toward zero, so that repeated samples produce the same update. Our goal here is not to guarantee coverage of every possible mode, but to retain variation in weight space for sampling to remain useful at test-time.

**Finding 5: *Best-of-$K$ Loss Preserves a Searchable Weight Distribution***

Objectives that require many sampled updates to perform well encourage the learned distribution to contract. Best-of-$K$ places less pressure on the full distribution and retains substantially more coverage as the weight sampling budget increases.

# G Full Experimental Results

## G.1 Benchmark-wise Comparison Between Deterministic and Distributional Hypernetworks

Table 4 reports the full benchmark-wise results corresponding to the comparison in Section 5.2. Across the five models, distributional training generally improves macro performance over a directly trained deterministic hypernet.

Table 5: **Benchmark-level test-time scaling summary.** Each cell reports accuracy at $K = 1/8/32$. T denotes token sampling from fixed adapted weights and W denotes sampling a new weight update for each generation. Regression and mixing hypernetwork should be interpreted separately because their training exposure is not matched.

| Model | Method | Samp. | GSM8K | MATH | HumanEval | ARC-C | MMLU | MedQA | GPQA | Macro |
|---|---|---|---|---|---|---|---|---|---|---|
| Qwen3 1.7B | Full FT | T | .508/.700/.768 | .316/.444/.540 | .529/.643/.714 | .608/.740/.788 | .496/.664/.700 | .272/.424/.428 | .217/.273/.354 | .421/.555/.613 |
| | LoRA | T | .568/.620/.632 | .316/.428/.500 | .667/.607/.635 | .520/.760/.800 | .492/.664/.708 | .172/.336/.372 | .177/.283/.268 | .416/.528/.559 |
| | MoL | T | .468/.560/.608 | .292/.428/.480 | .707/.701/.739 | .596/.704/.740 | .512/.688/.684 | .112/.320/.352 | .202/.293/.263 | .413/.528/.552 |
| | Reg. Det. | T | .524/.588/.612 | .312/.420/.504 | .633/.702/.738 | .528/.736/.748 | .496/.676/.688 | .196/.356/.368 | .207/.303/.308 | .414/.540/.567 |
| | **Reg. Dist.** | W | .620/.684/.740 | .396/.528/.556 | .688/.705/.706 | .708/.760/.784 | .608/.692/.720 | .308/.444/.428 | .197/.278/.258 | .504/.584/.599 |
| | Mix. Det. | T | .776/.844/.864 | .456/.528/.552 | .739/.781/.791 | .648/.748/.752 | .608/.744/.760 | .280/.388/.416 | .212/.298/.293 | .531/.619/.633 |
| | **Mix. Dist.** | W | .624/.720/.736 | .416/.540/.536 | .737/.765/.771 | .752/.788/.776 | .668/.744/.744 | .404/.432/.424 | .253/.298/.283 | .551/.612/.610 |
| Qwen3 4B | Full FT | T | .680/.812/.856 | .496/.620/.656 | .706/.731/.754 | .832/.884/.912 | .736/.796/.808 | .504/.584/.600 | .318/.338/.338 | .610/.681/.703 |
| | LoRA | T | .668/.704/.692 | .400/.564/.620 | .772/.786/.829 | .800/.904/.904 | .652/.804/.812 | .340/.552/.624 | .293/.333/.328 | .561/.664/.687 |
| | MoL | T | .660/.720/.720 | .428/.564/.624 | .846/.828/.843 | .744/.876/.904 | .652/.780/.816 | .384/.560/.612 | .217/.348/.374 | .562/.668/.699 |
| | Reg. Det. | T | .660/.700/.708 | .436/.548/.600 | .755/.815/.841 | .764/.884/.884 | .692/.816/.792 | .508/.604/.600 | .318/.384/.359 | .590/.679/.683 |
| | **Reg. Dist.** | W | .628/.660/.660 | .496/.604/.624 | .792/.839/.820 | .804/.876/.868 | .724/.788/.772 | .536/.580/.588 | .293/.359/.348 | .610/.672/.669 |
| | Mix. Det. | T | .880/.928/.928 | .568/.628/.664 | .808/.846/.841 | .860/.860/.864 | .788/.816/.828 | .572/.580/.608 | .343/.364/.338 | .688/.717/.724 |
| | **Mix. Dist.** | W | .832/.916/.928 | .424/.512/.552 | .860/.833/.845 | .888/.892/.888 | .752/.796/.804 | .568/.584/.592 | .354/.374/.369 | .668/.701/.711 |
| Gemma 3 1B | Full FT | T | .240/.420/.516 | .120/.228/.276 | .568/.543/.600 | .220/.400/.392 | .192/.336/.364 | .060/.248/.276 | .106/.232/.258 | .215/.344/.383 |
| | LoRA | T | .192/.388/.512 | .100/.184/.272 | .524/.491/.507 | .128/.336/.464 | .152/.300/.356 | .076/.216/.272 | .071/.182/.247 | .178/.300/.376 |
| | MoL | T | .188/.300/.396 | .116/.212/.304 | .548/.463/.486 | .240/.428/.488 | .180/.336/.356 | .072/.240/.268 | .146/.268/.222 | .213/.321/.360 |
| | Reg. Det. | T | .212/.368/.460 | .132/.220/.292 | .633/.523/.493 | .152/.352/.472 | .144/.300/.328 | .096/.264/.292 | .106/.207/.242 | .211/.319/.368 |
| | **Reg. Dist.** | W | .284/.428/.440 | .164/.228/.284 | .433/.500/.551 | .304/.464/.472 | .224/.356/.372 | .120/.260/.264 | .177/.222/.258 | .244/.351/.377 |
| | Mix. Det. | T | .300/.436/.500 | .116/.212/.260 | .386/.377/.443 | .148/.320/.396 | .188/.304/.356 | .140/.308/.292 | .217/.268/.253 | .214/.318/.357 |
| | **Mix. Dist.** | W | .364/.480/.532 | .244/.308/.360 | .413/.500/.529 | .336/.492/.500 | .292/.388/.380 | .144/.300/.300 | .182/.242/.278 | .282/.387/.411 |

### G.2 Full Benchmark-Level Test-Time Scaling Results

Table 5 summarizes the benchmark-level results underlying Figure 4. To keep the appendix compact, we report representative low, intermediate, and high sampling budgets, $K \in \{1, 8, 32\}$, while the main figure shows the full scaling curves. For deterministic methods, additional samples are generated from fixed adapted weights. For distributional hypernet, each generation uses a newly sampled weight update. Results are reported on GSM8K, MATH500, HumanEval, ARC-Challenge, MMLU-STEM, MedQA, and GPQA-Diamond, together with their macro average.

As in the main text, regression and mixing hypernetwork are not compared directly. The reference LoRAs used by the mixing hypernetwork are trained for three epochs before one epoch of hypernetwork training, whereas the regression hypernetwork is trained for one epoch on the same data.