DEPLOYMENT-TIME MIXTURE OPTIMIZATION FOR LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Domain-specific fine-tuning produces language models with complementary strengths, yet an unseen deployment distribution may not be well matched by any single model. We introduce Hierarchical Routing, a mixture optimization method that uses an unlabeled batch to identify the Top-H most relevant sources and optimizes a weight-space combination of their selected models using autoregressive loss. The hierarchy reduces optimization from the full model bank to a low-dimensional combination of relevant candidates, while preserving complementary specializations. We instantiate the bank with models spanning source domains and robustness levels and evaluate on nine Pile domains and deployment streams whose distributions change across batches. The experiments show that routing is beneficial under heterogeneous and changing conditions, despite higher deployment cost, and that applying test-time adaptation after routing further improves performance. The results show that adaptation to unseen distributions can be solved as navigation and composition in a precomputed model space.

1 INTRODUCTION

Adapting a language model to a domain is cheaper and routine, knowing *which* adaptation will be needed at deployment is not. Deployment distributions are rarely known in advance and may be heterogeneous or drift over time, making repeated retraining impractical. Practitioners therefore face a trade-off, fine-tune narrowly and specialize at the expense of other domains, or fine-tune on a fixed mixture and commit in advance to a compromise that may not match deployment. Such distribution shifts are pervasive in NLP and can cause substantial and difficult-to-predict performance degradation (Ramponi & Plank, 2020; Arora et al., 2021; Yuan et al., 2023; Calderon et al., 2024; Koh et al., 2021; Liu et al., 2021). We view deployment adaptation as *navigation in a precomputed model space*. Rather than updating the language model when the distribution changes, we select and compose domain- and robustness-specialized checkpoints using only the incoming unlabeled batch. Our preliminary experiments on domains from The Pile (Gao et al., 2020) motivate this perspective. Figure 1 shows that fine-tuning GPT-2 Medium on a source domain substantially improves in-domain performance while often degrading performance elsewhere. Accordingly, the cross-domain evaluation in Figure 2 (left) shows that different fine-tuned models have distinct generalization profiles, with no single specialization uniformly dominating. The appropriate degree of robustness is similarly deployment-dependent, within a source domain, different robustness levels trade specialization against protection from rare, difficult, or corrupted inputs.

One alternative is to fine-tune a single model on a mixture of source domains. This, however, fixes a compromise before the deployment distribution is known and may dilute useful specialization. We instead retain complementary models in a structured portfolio indexed by *source domain* and *robustness level*, and defer their selection or composition until deployment. Adaptation therefore consists of routing within the portfolio rather than retraining the model, allowing the deployed model to respond to distribution changes without modifying or forgetting the stored specializations.

The remaining challenge is how to exploit this portfolio when new data arrive: which models should be selected, and how should their domain specializations and robustness levels be combined? A natural starting point is distributional similarity. Figure 2 (right) shows that token-level KL divergence is correlated with performance degradation, but the relationship is far from deterministic. Distributional proximity is therefore informative, but insufficient for identifying the best model or model combination for an incoming batch(Yuan et al., 2023; Calderon et al., 2024).

This motivates our central question: *Can specialization and robustness be chosen at deployment rather than fixed during training?* We address this as navigation in a precomputed model space. Given an unlabeled deployment batch, we select and compose checkpoints spanning different source domains and robustness levels by optimizing only routing coefficients, thereby postponing

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.

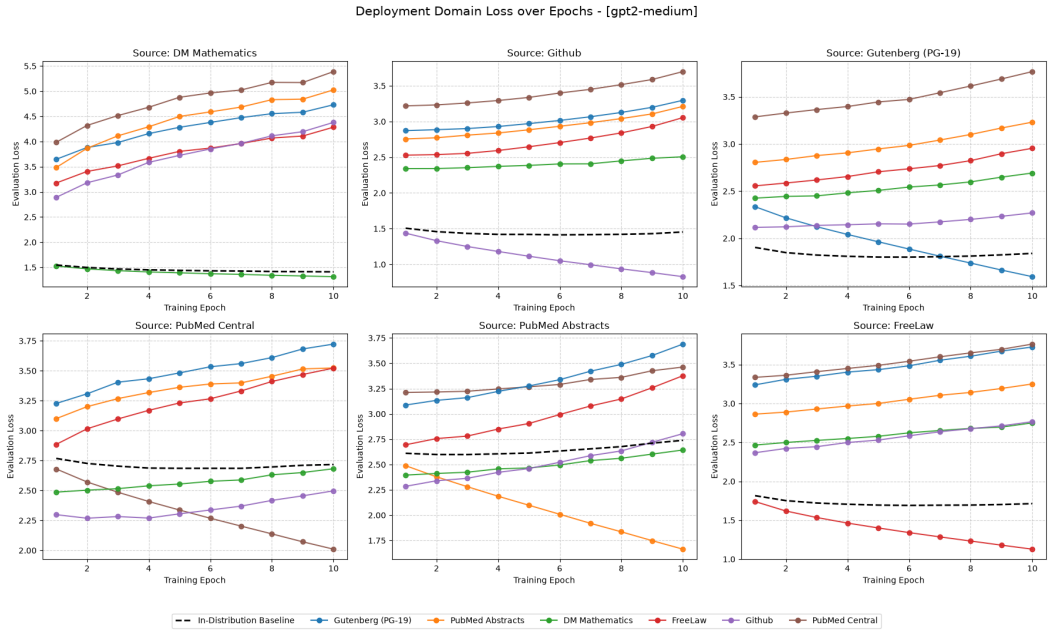


Figure 1: Evaluation loss of GPT-2 Medium during fine-tuning on six domains from The Pile (Gao et al., 2020), each panel fine-tunes on one source and evaluates on all six. The source-domain loss drops sharply, while loss on the other domains generally increases, illustrating the specialization induced by domain-specific fine-tuning.

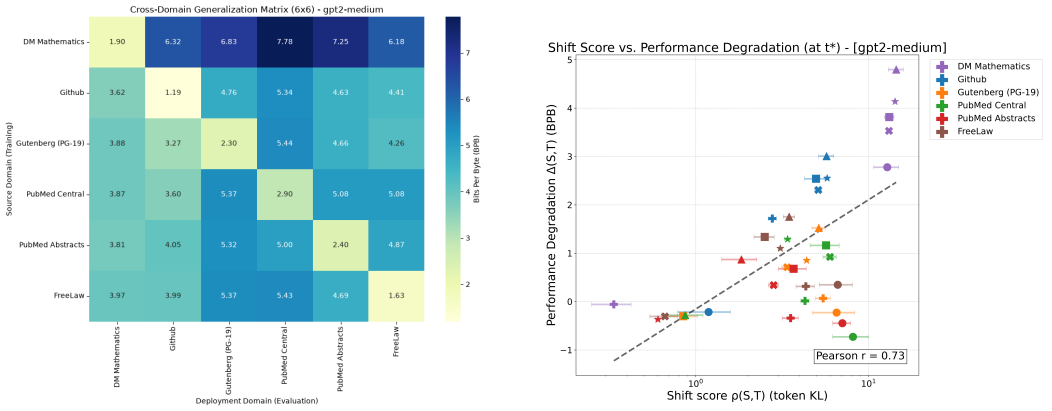


Figure 2: Experiments with GPT-2 Medium on The Pile (Gao et al., 2020). **Left:** Fine-tuned models exhibit distinct cross-domain transfer patterns. **Right:** Each point represents a source-target pair, with color indicating the fine-tuned source and marker shape the target. Token-level shift $\rho(S, T)$ correlates with performance degradation $\Delta(S, T)$ ($r = 0.73$), but the substantial dispersion shows that distributional distance alone is insufficient to predict model performance.

the specialization-robustness trade-off until the target distribution is observed. We instead study *deployment-time robustness routing*, selecting and composing models from a structured portfolio of domain specializations and robustness levels after observing an unlabeled deployment batch.

Related Work. We position our work with respect to the most closely related approaches in robust learning, test-time adaptation, model routing, and model merging.

Robust learning and data adaptation. Distributionally robust optimization (DRO) addresses distributional uncertainty by optimizing over plausible distributions (Rahimian & Mehrotra, 2019; Duchi et al., 2021), with applications to robust language modeling (Oren et al., 2019), cross-domain LLM generalization (Ma et al., 2025; Deng et al., 2025), and preference alignment (Zhu et al., 2025; Xu et al., 2025; Mandal et al., 2026). A related line adapts the training distribution through data reweighting or selection (Ren et al., 2018; Shu et al., 2019; Kang et al., 2024; Gu et al., 2025), in particular, DoReMi (Xie et al., 2023) learns domain weights to determine the data mixture used to train a single model. Both perspectives make distributional choices during training, when the

eventual deployment distribution and the appropriate degree of robustness may not yet be known. In contrast, we retain multiple domain-specialized models trained at different robustness levels and adapt their selection and composition to the observed deployment batch, making both specialization and robustness adaptive deployment-time choices rather than committing to a single model or robustness trade-off during training.

Test-time adaptation. Test-time adaptation uses unlabeled deployment data to adapt a model after training. Initially developed primarily for image classification, test-time training (Sun et al., 2020), Tent (Wang et al., 2020), and EATA (Niu et al., 2022) adapt model parameters using self-supervised or entropy-based objectives. More recently, TLM (Hu et al., 2025) extends test-time learning to LLMs, using unlabeled test inputs to adapt the model for downstream prompted tasks. Our setting instead considers distribution shift in autoregressive language modeling itself: deployment batches are evaluated directly through next-token prediction loss and used to select or compose models from a structured bank of domain and robustness specializations. Accordingly, the Tent baseline in our experiments is adapted to the language modeling setting by optimizing autoregressive cross-entropy rather than the original classification-based entropy objective.

Model routing, modular experts, and model merging. LLM routing selects among candidate models, often to balance predictive quality and inference cost (Ong et al., 2025; Jitkrittum et al., 2026), while modular approaches route or compose specialized experts (Huang et al., 2024; Ostapenko et al., 2024; Gururangan et al., 2023; Fedus et al., 2022; Jiang et al., 2024). Model merging instead combines compatible checkpoints in weight space (Wortsman et al., 2022; Ilharco et al., 2023; Rame et al., 2023; Ramé et al., 2022), including adaptive merging from unlabeled test data (Yang et al., 2024; Bertolissi et al., 2025). Our framework connects these perspectives, we construct a model bank explicitly structured by source domain and robustness level, use the incoming batch to identify relevant candidates, and optimize their composition using autoregressive loss. This allows the deployed model to adapt simultaneously to what kind of distribution is observed and how much robustness it requires, rather than committing to either choice before deployment.

Overall, prior work provides complementary mechanisms for robust training, training-data optimization, test-time parameter adaptation, expert routing, and adaptive model merging. We instead study *deployment-time robustness routing*, selecting and composing models from a structured portfolio of domain specializations and robustness levels after observing an unlabeled deployment batch, which avoids committing to a single specialization or robustness level in advance and lets the deployed model adapt to heterogeneous or changing distributions without retraining. Our contributions are:

- A **model bank that merges specialization and robustness into a single navigable space**: every model checkpoint is indexed jointly by source domain and robustness level, grown from one shared initialization, with weight interpolation filling in intermediate robustness levels without extra fine-tuning. The full bank will be released to support further routing and test-time-adaptation research.
- A **mixture-optimization framework** that reads the incoming unlabeled batch's own autoregressive loss as its only signal, selecting or composing bank models without ever touching their weights.
- **Hierarchical Routing**, which uses the bank's source-robustness structure to avoid searching over the entire model bank. Instead, it reduces the search to a much smaller set of candidates while guaranteeing performance no worse than the best individual candidate. Combined with a lightweight test-time update, **Hierarchical Routing+** achieves performance comparable to full-bank search at a fraction of the computational cost.
- An **empirical study of routing under distribution shift** spanning nine Pile domains and two model scales, showing that the preferred specialization and robustness level are genuinely deployment-dependent, that distributional proximity alone cannot predict which model will help, and that the routing-plus-adaptation ranking survives even when the target domain is entirely absent from the bank.

2 Problem Formulation

Let θ_0 be a pretrained autoregressive language model and $\mathcal{D} = \{D_1, \ldots, D_S\}$ a collection of source-domain datasets. Our objective is not to learn a single model that is uniformly robust to all possible deployment shifts. Instead, we construct offline a structured portfolio of domain- and robustness-specialized models and defer their selection or composition until an unlabeled deployment batch is observed. For a token sequence $x = (x_1, \ldots, x_T)$, let $\ell(x; \theta) = -\sum_{t=1}^{T} \log p_\theta(x_t \mid x_{<t})$ denote the autoregressive negative log-likelihood, and let P_j denote the empirical distribution induced by source dataset D_j.

Domain-robustness model bank. For each source distribution P_j, we construct models with different degrees of robustness using KL-DRO. For an ambiguity radius $\rho \geq 0$, the robust fine-tuning problem is

$$\min_{\theta} \sup_{Q':D_{\mathrm{KL}}(Q'\|P_j)\leq\rho} \mathbb{E}_{x\sim Q'}[\ell(x;\theta)]. \tag{1}$$

The KL constraint admits a Lagrangian representation in which $\lambda > 0$ controls the cost of departing from P_j. Rather than optimizing λ jointly with θ, we treat it as a controllable robustness parameter and train, for each fixed λ,

$$L_{j,\lambda}^{\mathrm{DRO}}(\theta) = \sup_{Q'} \left\{ \mathbb{E}_{x\sim Q'}[\ell(x;\theta)] - \lambda D_{\mathrm{KL}}(Q'\|P_j) \right\}. \tag{2}$$

For empirical P_j, the inner problem is solved by exponential tilting, yielding

$$L_{j,\lambda}^{\mathrm{DRO}}(\theta) = \lambda \log \mathbb{E}_{x\sim P_j}\left[\exp\left(\frac{\ell(x;\theta)}{\lambda}\right)\right]. \tag{3}$$

Small λ allows stronger adversarial reweighting toward high-loss samples, whereas large λ keeps the adversarial distribution close to P_j and approaches standard empirical-risk fine-tuning. Thus, λ provides an ordered robustness coordinate for constructing multiple adaptations of the same source domain. Importantly, this DRO radius should not be identified with an arbitrary empirical measure of deployment shift.

Starting from the common initialization θ_0, we train $\mathcal{M}_j = \{\theta_{j,\mathrm{ERM}}\} \cup \{\theta_{j,\lambda_k} : k = 1,\dots,K\}$, where $\theta_{j,\mathrm{ERM}}$ is the ordinary maximum-likelihood fine-tune and $0 < \lambda_1 < \cdots < \lambda_K$. The complete model bank is $\mathcal{M} = \{\theta_0\} \cup \bigcup_{j=1}^{S} \mathcal{M}_j$. Hence, each bank member is characterized by two complementary coordinates: its source specialization j and its robustness level λ_k[1].

Self-supervised deployment-time routing. At deployment, an unlabeled batch $B = \{x_1,\dots,x_n\} \sim Q$ arrives from an unknown distribution Q, which need not coincide with any source distribution P_j. The central problem is therefore not to choose a robustness level in advance, but to determine from B which source specializations and robustness levels are appropriate for the observed deployment condition. We do not update the stored models checkpoints. Instead, we select or compose them using their autoregressive loss on the incoming batch, $\widehat{L}_B(\theta) = \frac{1}{n}\sum_{x\in B} \ell(x;\theta)$. Let $\{\theta_m\}_{m=0}^{|\mathcal{M}|-1}$ denote the members of the bank, including θ_0. For weights w on the simplex $\Delta(\mathcal{M}) = \{w \geq 0 : \sum_m w_m = 1\}$, we define the parameter composition $\theta(w) = \sum_m w_m \theta_m$. The deployment-time routing problem is

$$w_B^\star \in \arg\min_{w\in\Delta(\mathcal{M})} \widehat{L}_B\big(\theta(w)\big). \tag{4}$$

Equation 4 uses no deployment labels, instead the autoregressive objective provides a self-supervised signal for deciding which precomputed adaptations are appropriate for the incoming batch. Restricting w to simplex vertices yields hard model selection, whereas optimizing over the simplex allows several compatible adaptations to be composed. The bank, however, contains a product structure, source domain $\times$ robustness level, that is ignored by flat optimization over all its members. We exploit this structure in Section 4 through our hierarchical router approach.

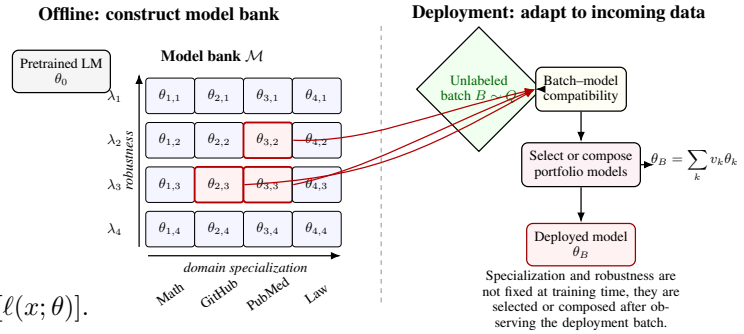
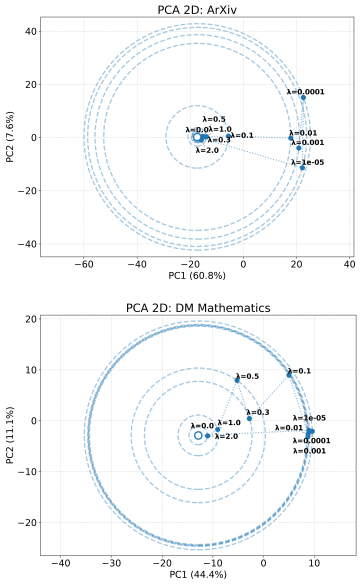


Figure 3: Starting from a common pretrained language model, we construct offline a bank spanning source-domain specializations and robustness levels. When an unlabeled, unknown distribution btach arrives, it is used to select or compose an adaptive deployed model.

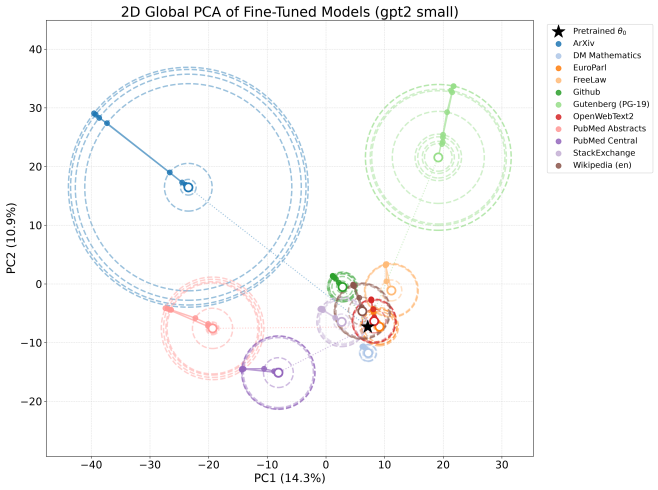
3 Robustness Paths

Geometry of the robustness paths. To examine the organization of the model bank in parameter space, we represent each model by its parameter update from the common initialization, and jointly project these update vectors into a common PCA space. Figure 4 reveals structure along both dimensions of variation in the model bank. Different source domains induce distinct parameter update directions from θ_0, reflecting domain specialization, while varying λ within a source traces

[1]Full derivations and implementation details are provided in Appendix A.1.



(a) Single-domain robustness path.



(b) Robustness paths across source domains.

Figure 4: Joint PCA projection of the adaptation vectors $\theta_{j,\lambda}$ and the common pretrained checkpoint θ_0. Each source domain defines a distinct family originating from θ_0, while points within each family correspond to different robustness levels λ. Within a family the checkpoints are ordered monotonically by λ; the ERM checkpoint ($\lambda = 0$) and the largest λ sit at the same (nominal) end of the path, and robustness increases toward the smallest positive λ. **Left:** the robustness path obtained for the ArXiv and DM Mathematics source domains. **Right:** robustness paths across multiple source domains.

an organized robustness trajectory rather than producing scattered models. The left panel isolates the ArXiv and DM Mathematics family and makes the within-domain structure particularly visible:

Table 1: Cross-entropy loss of the ArXiv robustness path across natural domains and homoglyph-corrupted[a] versions of held-out ArXiv. Column headers report Token-KL shift from ArXiv, **bold** entries indicate the best robustness level $\lambda^\star$. The preferred $\lambda^\star$ is not monotonic in shift magnitude.

	Natural domain shift					Homoglyph corruption		
λ	ArXiv D_{KL}=0.00	DM Math. 2.98	PubMed Abs. 4.87	BookCorpus 7.45	YouTube Sub. 10.03	p=0.05 0.30	p=0.30 1.41	p=0.80 2.55
0 (ERM)	2.9074	2.8079	3.5090	5.0714	3.6208	3.7915	4.8817	4.6885
10^{-5}	2.9152	2.7856	3.4076	**4.8790**	3.5283	3.7703	4.8405	4.6528
10^{-4}	2.9182	2.8072	3.4016	4.9491	3.5472	3.7864	4.8693	4.6806
10^{-3}	2.8830	**2.7700**	**3.3857**	4.9085	**3.5227**	**3.7400**	**4.8041**	4.6095
10^{-2}	2.8858	2.8104	3.3932	4.9500	3.5649	3.7546	4.8480	4.6661
0.1	**2.8799**	2.7860	3.4136	4.9886	3.6240	3.7442	4.8064	4.6267
0.3	2.9087	2.7845	3.4774	5.0501	3.5910	3.7824	4.8465	4.6390
0.5	2.9119	2.7855	3.5212	5.0022	3.6225	3.7772	4.8200	4.6239
1	2.8979	2.7908	3.4775	4.9992	3.5939	3.7674	4.8220	**4.6075**

[a] Homoglyph corruption replaces characters with visually similar alternatives (e.g., o $\to$ 0, l $\to$ 1) independently with probability p.

The right panel shows that this organization extends across source domains. When all source families are projected jointly, each source occupies a distinct region and follows a different parameter update direction from the common pretrained checkpoint θ_0. At the same time, varying λ preserves a structured within-source trajectory. The resulting geometry reflects two complementary forms of variation in the model bank: the source domain determines the main specialization direction, while λ determines the position along the corresponding robustness path. This geometric organization is consistent across model scales, as shown by the analogous PCA projections for different GPT-2 model sizes in Appendix B.6.

Functional diversity along the robustness path. We examine whether different robustness levels also induce meaningful differences at deployment. Focusing on the ArXiv source, we evaluate its robustness path across natural distribution shifts and synthetic OCR-style homoglyph corruptions of increasing strength. Table 1 shows that the preferred robustness level $\lambda^\star$ depends strongly on the deployment condition. In particular, plain ERM ($\lambda = 0$) is never optimal, while different non-zero robustness levels are preferred across shifts. Importantly, $\lambda^\star$ is not monotonic in the measured distribution shift, for example, YouTube Subtitles has a larger D_{KL} than BookCorpus but favors

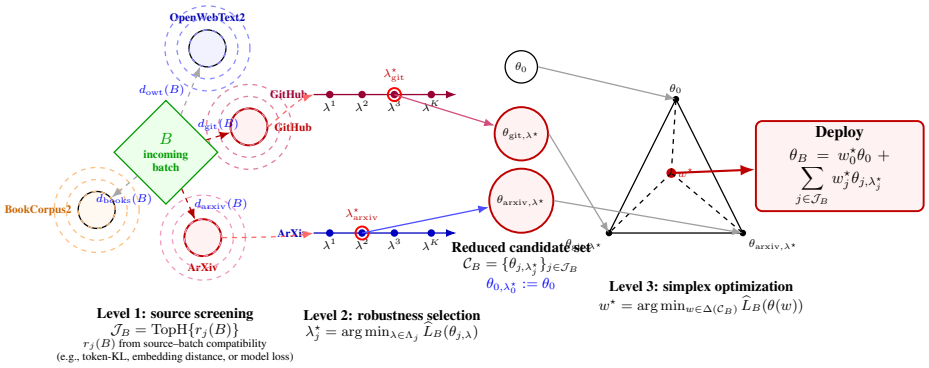


Figure 5: Given an unlabeled batch B, the router retains the H most relevant candidates (competing θ_0 against every source on its own batch loss) and selects the best-fitting robustness level within each retained source. The resulting models define a reduced candidate set $\mathcal{C}_B$ whose combination weights are optimized at deployment.

a less robust checkpoint. A similar non-monotonic pattern appears as the homoglyph corruption strength increases.

Thus, λ induces functionally distinct deployment profiles rather than merely parameterizing the training objective, and the appropriate robustness level cannot be inferred from shift magnitude alone. The model bank therefore has two complementary coordinates (source specialization j and robustness level λ) whose appropriate combination can be determined from the incoming batch.

4 Adaptive Deployment Routing

Given the model bank $\mathcal{M} = \{\theta_0\} \cup \{\theta_{j,\lambda} : j = 1, \dots, S,\ \lambda \in \Lambda_j\}$, where S is the number of source domains, the deployment problem consists of determining how the available models should be combined for an incoming unlabeled batch B. Let $\lambda_j^\star(B) \in \Lambda_j$ denote the robustness level selected for source domain j given the incoming batch B. The corresponding candidate model is therefore $\theta_{j,\lambda_j^\star(B)}$. Collecting one selected model per source together with the pretrained checkpoint gives the candidate set $\mathcal{C} = \{\theta_0\} \cup \{\theta_{j,\lambda_j^\star(B)} : j = 1, \dots, S\}$, and we route over its simplex $\Delta(\mathcal{C}) = \left\{(w_0, w_1, \dots, w_S) \in \mathbb{R}_+^{S+1} : w_0 + \sum_{j=1}^S w_j = 1\right\}$, where w_j is the deployment weight on the selected robustness model $\theta_{j,\lambda_j^\star(B)}$ of source domain j, and w_0 is the weight on the pretrained model θ_0. With every $w \in \Delta(\mathcal{C})$, we associate the interpolated model: $\theta(w) = w_0\theta_0 + \sum_{j=1}^S w_j \theta_{j,\lambda_j^\star}$.

Since the incoming batch is unlabeled, we use the autoregressive objective itself as compatibility measure. For any candidate model θ, define $\widehat{L}_B(\theta) = \frac{1}{|B|} \sum_{x \in B} \ell(x; \theta)$, where lower values indicate that the model explains the incoming batch better.[2]

We consider several deployment strategies that construct a model from the bank, and hence produce a deployed model θ_B. These strategies explore different trade-offs between deployment quality, and the computational effort required to construct the deployed model.

Exact routing formulation. Given the per-source robustness levels $\lambda_j^\star(B)$, the best interpolation weights over the candidate set are obtained by solving

$$w^\star(B) \in \arg\min_{w \in \Delta(\mathcal{C})} \widehat{L}_B\big(\theta(w)\big) = \arg\min_{w \in \Delta(\mathcal{C})} \frac{1}{|B|} \sum_{x \in B} \ell\left(x;\ w_0\theta_0 + \sum_{j=1}^S w_j \theta_{j,\lambda_j^\star(B)}\right). \tag{5}$$

The corresponding deployment model is $\theta_B^\star = \theta\big(w^\star(B)\big)$. The routing space $\Delta(\mathcal{C})$ has only S degrees of freedom (one weight per source plus w_0, minus the simplex constraint), far fewer than the $|\theta|$ parameters of the underlying models, yet the objective is nonlinear and nonconvex in w because the loss is evaluated after parameter interpolation. A globally optimal solution can therefore require many loss evaluations, so we also consider faster routing strategies that produce cheaper, suboptimal solutions to equation 5. Algorithm 1 is the shared template for the three strategies below. Note that equation 5 already fixes one robustness level $\lambda_j^\star(B)$ per source; a genuinely joint search over all (j, λ) pairs would replace $\Delta(\mathcal{C})$ by the full $\Delta(\mathcal{M})$ and is not considered here.

[2]This score can be replaced by other compatibility measures, such as distributional or representation-based distances, without changing the routing formulation. In our main implementation, however, we use the model's immediate autoregressive loss on the incoming batch.

Algorithm 1: Adaptive Model-Bank Routing

Input: Bank $\mathcal{M}$, batch B, strategy R
Output: Deployment model θ_B

Compute required losses $\widehat{L}_B(\theta)$;
if R = HARD **then**
  $\theta_B \leftarrow \arg\min_{\theta \in \mathcal{M}} \widehat{L}_B(\theta)$;
else
  // Flat and Hierarchical share this Top-H reduction
  foreach $j = 1, \ldots, S$ **do**
    $\lambda_j^\star \leftarrow \arg\min_{\lambda \in \Lambda_j} \widehat{L}_B(\theta_{j,\lambda})$;
    $r_j \leftarrow -\widehat{L}_B(\theta_{j,\lambda_j^\star})$;
    // r_j: ranking score, not a weight
  $\mathcal{J}_B \leftarrow \mathrm{TopH}_j\{r_j\}$;
  $\mathcal{C}_B \leftarrow \{\theta_0\} \cup \{\theta_{j,\lambda_j^\star} : j \in \mathcal{J}_B\}$;
  if R = FLAT **then**
    $w_k \propto \exp(-\widehat{L}_B(\theta_k)/\tau)$ for $\theta_k \in \mathcal{C}_B$;
    $\theta_B \leftarrow \sum_{\theta_k \in \mathcal{C}_B} w_k \theta_k$;
  else
    // **Hierarchical (ours)**
    Init. $w^{(0)}$ (uniform, flat-softmax, R random);
    for $t = 0, \ldots, T-1$ **do**
      $g_k \leftarrow \langle \nabla_\theta \widehat{L}_B(\theta(w^{(t)})), \theta_k \rangle$;
      $w_k^{(t+1)} \leftarrow \frac{w_k^{(t)} \exp(-\eta_t g_k^{(t)})}{\sum_r w_r^{(t)} \exp(-\eta_t g_r^{(t)})}$;
    $w^\star \leftarrow$ best iterate, over all starts, iterations, and vertices;
    $\theta_B \leftarrow \theta(w^\star)$;
return θ_B;

Hard routing. Hard Routing restricts the feasible weights to the vertices of $\Delta(\mathcal{M})$, thereby selecting a single model: $\theta_B^{\mathrm{hard}} \in \arg\min_{\theta \in \mathcal{M}} \widehat{L}_B(\theta)$.
This is the simplest deployment rule and serves as a reference for evaluating whether combining multiple bank members is beneficial.

Flat soft routing. We apply the same Top-H candidate reduction as Hierarchical Routing, yielding $\mathcal{C}_B = \{\theta_k\}_{k=1}^H$, and assign softmax weights

$$w_k(B) = \frac{\exp(-\widehat{L}_B(\theta_k)/\tau)}{\sum_{r=1}^H \exp(-\widehat{L}_B(\theta_r)/\tau)}, \qquad k = 1, \ldots, H,$$

where $\tau > 0$ is the temperature. The routed model is $\theta_B^{\mathrm{flat}} = \sum_{k=1}^H w_k(B)\theta_k$. Unlike Hierarchical Routing, no further optimization of the mixture weights is performed.

Solving the routing problem. Although problem equation 5 involves only S degrees of freedom, its objective is generally nonconvex. We solve it directly over the simplex using Exponentiated-Gradient (EG) updates. Starting from $w^{(0)} \in \Delta(\mathcal{C})$, at iteration t we construct $\theta(w^{(t)})$, evaluate the batch loss, and compute

$$g_k^{(t)} = \frac{\partial \widehat{L}_B(\theta(w^{(t)}))}{\partial w_k} = \left\langle \nabla_\theta \widehat{L}_B(\theta(w^{(t)})), \theta_k \right\rangle.$$

The routing weights are then updated as $w_k^{(t+1)} = \frac{w_k^{(t)} \exp(-\eta_t g_k^{(t)})}{\sum_r w_r^{(t)} \exp(-\eta_t g_r^{(t)})}$, which preserves the simplex constraints without explicit projection. In practice, $\theta(w^{(t)})$ is loaded as the parameters of the shared (frozen, pretrained) network architecture, and a single forward-backward pass of that network on B yields $\nabla_\theta \widehat{L}_B(\theta(w^{(t)}))$. Since $\theta(w)$ is linear in w, the chain rule then gives $g_k^{(t)}$ directly, as written above. No auxiliary model is trained to produce w. The network itself, evaluated at the interpolated weights, supplies the gradient used to update w. This is exactly the update performed by Algorithm 1's Hierarchical branch, which iterates this loop from several starting points before returning the best iterate found. The Hierarchical strategy (Algorithm 1, illustrated in Figure 5) applies this same exponentiated-gradient procedure, but first reduces $\mathcal{C}$ to the H candidates most relevant to B and runs it over $\Delta(\mathcal{C}_B) \subseteq \Delta(\mathcal{C})$ instead; the algorithm also details the multi-start initialization and the vertex comparison that together guarantee the returned solution performs no worse on B than the best individual candidate, although global optimality is not guaranteed.

5 EXPERIMENTS

5.1 BASELINES AND COMPARISON METHODS

We compare against three static baselines that ignore the incoming batch: **Pretrained**, which deploys θ_0 unchanged; **Best Single FT**, the specialist with the lowest validation loss on the first batch; and **Mixed-source FT**, trained on a uniform mixture of sources. We also consider Tent (Wang et al., 2020), adapted here to minimize autoregressive cross-entropy on the incoming batch rather than the original classification-based entropy objective, and four routing variants: **Hard Routing**, **Flat Routing**, **Hierarchical Routing** (our main method; Section 4), and **Static Hierarchical Routing**, calibrated on the first batch and then frozen.
For reference, we include a non-deployable **Target-FT Oracle**, fine-tuned directly on the target stream. It defines the 100% point of the oracle-gap-closed metric, with Pretrained defining 0%: oracle-gap-closed$(\theta) = 100\% \frac{L_Q(\theta_0) - L_Q(\theta)}{L_Q(\theta_0) - L_Q(\theta_{\mathrm{oracle}})}$, where L_Q is the loss on the target evaluation slice. When the oracle provides little or negative improvement, this metric is ill-defined or unstable.

We therefore report relative perplexity change for FreeLaw and OWT2 (marked ‡ in Table 3): the oracle is worse than Pretrained on FreeLaw and provides negligible improvement on OWT2.

On the episodic streams (Table 3) we additionally report three Tent variants applied on top of a selected or routed starting point rather than θ_0 directly: **Static TENT (Best Single-FT)**, **Online TENT (Best Single-FT)**, and **Hierarchical Routing+** (Tent applied once on top of a calibrated routing solution, then carried forward across batches), whose exact mechanisms are detailed in Appendix B.4. Full deployment-time inputs, degrees of freedom, and cost for every method are summarized in Table 4 (Appendix B.1).

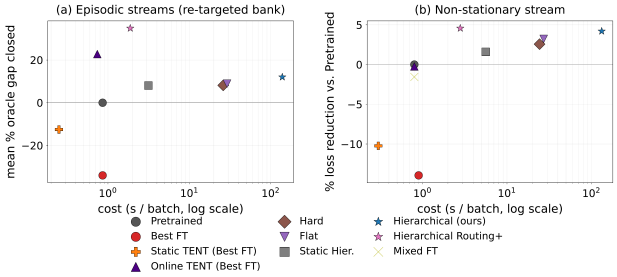


Figure 6: **Left:** Mean oracle-gap closure across the four episodic streams with a defined oracle gap (bank: PubMed Central, ArXiv, PG-19; Table 3). **Right:** Loss reduction versus Pretrained on the non-stationary stream (the same bank, and 24 pooled batches, Table 2).

5.2 Deployment protocol

At deployment, each method selects, combines, or adapts a model from each incoming batch. The same batch is used for adaptation and evaluation, following our transductive setting. Routing methods operate per batch, while static variants are calibrated on the first batch and then frozen, and online variants retain their state across the stream. All methods receive the same batches; model-bank construction is performed offline and excluded from deployment costs (Tables 2 and 3). Full details are provided in Appendix B.3.

Table 2: Non-stationary stream, 24 pooled batches from different datasets/distributions, sources and targets coherent with Table 3, model is gpt2-small. Improvement is relative to Pretrained, reported as mean ± std over the 24 batches.

Method	Loss (PPL)	ΔLoss(%)	Cost (s/batch)
Pretrained	3.037 (23.3)	$0.00 \pm 0.00\%$	0.8
Best Single FT	3.478 (40.9)	$-13.95 \pm 10.32\%$	0.9
Mixed-source FT	3.093 (25.5)	$-1.56 \pm 6.06\%$	0.8
Static TENT (Best Single-FT)	3.368 (36.7)	$-10.23 \pm 10.97\%$	0.3
Online TENT (Best Single-FT)	3.052 (25.1)	$-0.24 \pm 9.53\%$	0.8
Hard Routing	2.960 (21.9)	$+2.57 \pm 5.73\%$	24.3
Flat Routing	2.948 (22.0)	$+3.23 \pm 7.41\%$	27.2
Static Hierarchical Routing	2.999 (23.4)	$+1.60 \pm 7.99\%$	5.6
Hierarchical Routing	2.918 (21.3)	$+4.20 \pm 7.45\%$	132.2
Hierarchical Routing+	**2.907 (21.1)**	$\mathbf{+4.57 \pm 8.81\%}$	2.8

We evaluate under two stream types: a *stationary* stream, where every batch is drawn from a single condition and isolates the quality of a single routing decision, and a *non-stationary* stream, where batches from different datasets are interleaved so no single frozen model is appropriate, which is where per-batch routing is expected to separate from static alternatives.

5.3 Experimental Results

Setup. All models use a gpt2-small or gpt2-medium backbone, $K = 10$ robustness levels per source (Section 2), context length 512, batch size 16. Both the episodic streams (50 batches/target) and the non-stationary stream (24 pooled batches) use the same re-targeted $S = 3$ bank (PubMed Central, ArXiv, PG-19, all Pile subsets (Gao et al., 2020)); episodic targets are DM Mathematics, EuroParl, FreeLaw, GitHub, OWT2, and PubMed Abstracts, of which four have a well-defined oracle gap, and two (FreeLaw, OWT2) have a structurally degenerate oracle (Section 5.1). Hierarchical Routing uses $H = 3, T = 20$ EG iterations. Across banks and streams, our protocol spans 9 distinct Pile domains, the full offline bank ($S \times K$ checkpoints per setting) will be released alongside the code to support further routing and test-time-adaptation research. Reproducibility details are in Appendix B.4.

Adaptation under changing distributions. On the non-stationary stream with gpt2-small (Table 2, Figure 6b), Hierarchical Routing+ achieves the best performance among deployable methods while also being the cheapest routing variant. Plain Hierarchical Routing performs similarly but at substantially higher cost, whereas static routing provides only modest gains and committed or Tent-only baselines fall below Pretrained. This shows the value of adapting the deployed model as the distribution changes, with routing and local test-time adaptation providing complementary benefits.

Why Tent is competitive. The non-stationary bank's four sources are all several times farther (Token-KL) from its three targets than each target's true nearest neighbor outside the bank (methodology and figures: Appendix B.5), routing frequently collapses to a single vertex, identical to Hard

Table 3: Loss (oracle gap closed %) per episodic target stream, and mean deployment cost (50 batches each). Bank sources: PubMed Central, ArXiv, PG-19 ($S = 3$, $K = 10$ robustness levels each), the model is gpt2-small. 0% = Pretrained, 100% = Target-FT Oracle.

Method	DM Math.	EuroParl	FreeLaw[‡]	GitHub	OWT2[‡]	PubMed Abs.	Cost (s/batch)
Pretrained	2.545 (0.0%)	3.708 (0.0%)	2.987 (0.0%)	3.125 (0.0%)	3.293 (0.0%)	3.162 (0.0%)	0.9
Best Single FT	2.768 (−21.9%)	3.885 (−21.1%)	3.149 (−17.5%)	2.774 (10.8%)	3.745 (−57.1%)	3.495 (−103.9%)	0.9
Static TENT (Best Single-FT)	2.668 (−12.3%)	3.757 (−5.8%)	3.082 (−9.9%)	2.716 (16.8%)	3.632 (−40.4%)	3.318 (−48.9%)	0.3
Online TENT (Best Single-FT)	2.338 (19.9%)	3.395 (37.2%)	2.927 (5.9%)	2.249 (44.8%)	3.398 (−11.0%)	3.196 (−10.6%)	0.7
Hard Routing	2.545 (0.0%)	3.708 (0.0%)	2.971 (1.7%)	2.286 (32.6%)	3.293 (0.0%)	3.162 (0.0%)	26.1
Flat Routing	2.564 (−1.9%)	3.699 (1.1%)	2.954 (3.3%)	2.213 (43.5%)	3.383 (−9.4%)	3.184 (−6.9%)	29.1
Static Hierarchical Routing	2.545 (0.0%)	3.691 (2.1%)	2.987 (0.0%)	2.612 (30.0%)	3.293 (0.0%)	3.162 (0.0%)	3.1
Hierarchical Routing	2.541 (0.4%)	3.688 (2.4%)	2.921 (6.4%)	2.174 (45.0%)	3.293 (0.0%)	3.161 (0.4%)	138.8
Hierarchical Routing+	**2.305 (23.2%)**	**3.328 (45.2%)**	**2.852 (12.7%)**	**2.154 (52.4%)**	**3.271 (2.2%)**	**3.101 (18.8%)**	1.9
Target-FT Oracle[†]	*1.506 (100.0%)*	*2.866 (100.0%)*	*3.141 (−16.6%)*	*1.511 (100.0%)*	*3.226 (6.5%)*	*2.840 (100.0%)*	—

[†]Not deployable: direct fine-tuning on the target stream, shown only to anchor the 100% reference point.

Routing. This is why Tent, which adapts directly rather than selecting among irrelevant specialists, competes on stationary streams, motivating the re-targeted-bank test below.

Routing provides a better starting point for test-time adaptation. With the re-targeted bank on gpt2-small, Hierarchical Routing remains safe but provides limited gains on most episodic targets (Table 3). In contrast, **Hierarchical Routing+**, which applies online Tent to the routed model, achieves the lowest loss among deployable methods on all six targets while remaining substantially cheaper than plain Hierarchical Routing. The same trend holds for gpt2-medium, where Routing+ is best on five of six targets (Appendix B.6, Table 7). This indicates that routing and test-time adaptation play complementary roles. Routing identifies a deployment-relevant region of the model bank, from which Tent performs local adaptation.

Effective model-space adaptation need not require expensive routing. Full Hierarchical Routing shows that deployment-time optimization over the model bank can improve performance, but re-solving the complete routing problem for every batch is costly (Figure 6). Hierarchical Routing+ achieves comparable performance at substantially lower cost by combining a routed solution with local test-time adaptation. On the non-stationary stream, it is statistically indistinguishable from full Hierarchical Routing despite a considerable lower cost (Table 2). Together with the episodic results, this suggests that routing can identify a suitable region of the precomputed model space from which inexpensive local adaptation can proceed. Further reducing the cost of this model-space search remains an important direction for future work. Convergence behavior of the underlying EG search, including when its random restarts matter, is examined in Appendix B.2.

6 Conclusions and Future Works

We introduced a deployment-time adaptation framework that treats a structured bank of specialized models as a solution space navigated after the deployment distribution is observed. Organizing models by source domain and robustness level lets Hierarchical Routing exploit complementary specializations without committing in advance to a single model or training mixture. Our experiments show that the preferred specialization depends on the deployment distribution, that adaptive routing is useful under heterogeneous and changing conditions, and that routing and test-time adaptation are complementary, routing identifies a suitable region of the precomputed model space, while local adaptation refines it. While our experiments instantiate specialization through domain-specific and robust fine-tuning, the underlying question is broader. An open question is whether deployment-time navigation remains useful when complementary behaviors arise from task specialization, parameter-efficient adaptation, or prompting, and when performance is measured on downstream tasks rather than language-modeling loss. A complementary theoretical direction is to characterize when interpolation within a precomputed model space is beneficial, how its geometry and coverage affect achievable performance, and whether richer composition spaces can provide additional flexibility.

AI Assistance Generative AI tools were used as research assistants for literature discovery, clarifying ideas developed by the authors, and organizing experiments. They supported the presentation and analysis of the model bank, hierarchical routing, baselines, and ablations, but not the design or mathematical formulation of the proposed algorithms. They were also used for drafting and editing manuscript text, figures, and released code, but not to generate synthetic data or formulate or prove mathematical claims. All AI-assisted material was independently reviewed and verified by the authors, who take full responsibility for the final content.

REFERENCES

Udit Arora, William Huang, and He He. Types of out-of-distribution texts and how to detect them. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 10687–10701, 2021.

Ryo Bertolissi, Jonas Hübotter, Ido Hakimi, and Andreas Krause. Local mixtures of experts: Essentially free test-time training via model merging. In *Conference on Language Modeling (COLM)*, 2025.

Nitay Calderon, Naveh Porat, Eyal Ben-David, Alexander Chapanin, Zorik Gekhman, Nadav Oved, Vitaly Shalumov, and Roi Reichart. Measuring the robustness of nlp models to domain shifts. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 126–154, 2024.

Hexuan Deng, Wenxiang Jiao, Xuebo Liu, Jing Li, Min Zhang, and Zhaopeng Tu. DRPruning: Efficient large language model pruning through distributionally robust optimization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29152–29173. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1414.

John C Duchi, Peter W Glynn, and Hongseok Namkoong. Statistics of robust optimization: A generalized empirical likelihood approach. *Mathematics of Operations Research*, 46(3):946–969, 2021.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Yuxian Gu, Li Dong, Hongning Wang, Yaru Hao, Qingxiu Dong, Furu Wei, and Minlie Huang. Data selection via optimal control for language models. In *International Conference on Learning Representations (ICLR)*, 2025.

Suchin Gururangan, Margaret Li, Mike Lewis, Weijia Shi, Tim Althoff, Noah A. Smith, and Luke Zettlemoyer. Scaling expert language models with unsupervised domain discovery. *arXiv preprint arXiv:2303.14177*, 2023.

Jinwu Hu, Zitian Zhang, Guohao Chen, Xutao Wen, Chao Shuai, Wei Luo, Bin Xiao, Yuanqing Li, and Mingkui Tan. Test-time learning for large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 24823–24849. PMLR, 2025.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. LoraHub: Efficient cross-task generalization via dynamic LoRA composition. In *Conference on Language Modeling (COLM)*, 2024.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *International Conference on Learning Representations*, 2023.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Congchao Wang, Zifeng Wang, Alec Go, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, Aditya Krishna Menon, and Sanjiv Kumar. Universal model routing for efficient LLM inference. In *International Conference on Learning Representations*, 2026.

Feiyang Kang, Hoang Anh Just, Yifan Sun, Himanshu Jahagirdar, Yuanzhi Zhang, Rongxing Du, Anit Kumar Sahu, and Ruoxi Jia. Get more for less: Principled data selection for warming up fine-tuning in llms. *arXiv preprint arXiv:2405.02774*, 2024.

Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, et al. Wilds: A benchmark of in-the-wild distribution shifts. In *International conference on machine learning*, pp. 5637–5664. PMLR, 2021.

Jiashuo Liu, Zheyan Shen, Yue He, Xingxuan Zhang, Renzhe Xu, Han Yu, and Peng Cui. Towards out-of-distribution generalization: A survey. *arXiv preprint arXiv:2108.13624*, 2021.

Guangyuan Ma, Yongliang Ma, Xing Wu, Zhenpeng Su, Ming Zhou, and Songlin Hu. Task-level distributionally robust optimization for large language model-based dense retrieval. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 24759–24767, 2025. doi: 10.1609/aaai.v39i23.34657.

Debmalya Mandal, Paulius Sasnauskas, and Goran Radanovic. Distributionally robust reinforcement learning with human feedback. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shijian Zheng, Peilin Zhao, and Mingkui Tan. Efficient test-time model adaptation without forgetting. In *International conference on machine learning*, pp. 16888–16905. PMLR, 2022.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025.

Yonatan Oren, Shiori Sagawa, Tatsunori B Hashimoto, and Percy Liang. Distributionally robust language modeling. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4227–4237, 2019.

Oleksiy Ostapenko, Zhan Su, Edoardo Maria Ponti, Laurent Charlin, Nicolas Le Roux, Matheus Pereira, Lucas Caccia, and Alessandro Sordoni. Towards modular LLMs by building and reusing a library of LoRAs. In *International Conference on Machine Learning*, 2024.

Hamed Rahimian and Sanjay Mehrotra. Distributionally robust optimization: A review. *arXiv preprint arXiv:1908.05659*, 2019.

Alexandre Ramé, Matthieu Kirchmeyer, Thibaud Rahier, Alain Rakotomamonjy, Patrick Gallinari, and Matthieu Cord. Diverse weight averaging for out-of-distribution generalization. In *Advances in Neural Information Processing Systems*, volume 35, pp. 10821–10836, 2022.

Alexandre Rame, Guillaume Couairon, Corentin Dancette, Jean-Baptiste Gaya, Mustafa Shukor, Laure Soulier, and Matthieu Cord. Rewarded soups: towards pareto-optimal alignment by interpolating weights fine-tuned on diverse rewards. *Advances in Neural Information Processing Systems*, 36:71095–71134, 2023.

Alan Ramponi and Barbara Plank. Neural unsupervised domain adaptation in nlp—a survey. In *Proceedings of the 28th international conference on computational linguistics*, pp. 6838–6855, 2020.

Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *ICML*, 2018.

Jun Shu, Qi Xie, Longhui Yi, Qian Zhao, and Song-Chun Zhou. Meta-weight-net: Learning an explicit mapping for sample weighting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

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*, pp. 9229–9248. PMLR, 2020.

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.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pp. 23965–23998. Pmlr, 2022.

Sang Michael Xie, Shibani Santurkar, Shiori Sagawa, and et al. Doremi: Data mixture optimization for large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Zaiyan Xu, Sushil Vemuri, Kishan Panaganti, Dileep Kalathil, Rahul Jain, and Deepak Ramachandran. Robust LLM alignment via distributionally robust direct preference optimization. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0912.

Enneng Yang, Zhenyi Wang, Li Shen, Shiwei Liu, Guibing Guo, Xingwei Wang, and Dacheng Tao. AdaMerging: Adaptive model merging for multi-task learning. In *International Conference on Learning Representations*, 2024.

Lifan Yuan, Yangyi Chen, Ganqu Cui, Hongcheng Gao, Fangyuan Zou, Xingyi Cheng, Heng Ji, Zhiyuan Liu, and Maosong Sun. Revisiting out-of-distribution robustness in nlp: Benchmarks, analysis, and llms evaluations. *Advances in Neural Information Processing Systems*, 36:58478–58507, 2023.

Mingye Zhu, Yi Liu, Zheren Fu, Yongdong Zhang, and Zhendong Mao. Leveraging robust optimization for LLM alignment under distribution shifts. In *Advances in Neural Information Processing Systems*, 2025.

A Additional Derivations and Theoretical Details

This section provides the derivations underlying the KL-DRO objective used to construct the robustness paths in the main paper. We first derive the exponentially tilted adversarial distribution and the corresponding log-sum-exp objective. We then clarify the role of the robustness parameter λ, describe the mini-batch implementation used during fine-tuning, and finally provide the proof connecting an estimated deployment shift to the corresponding robustness level.

A.1 KL-DRO Dual Form and Exponential Reweighting

Consider a discrete training distribution

$$P_{\mathcal{D}}(x_i) = p_i, \qquad i = 1, \dots, N,$$

and denote

$$\ell_i(\theta) = \ell(x_i; \theta).$$

Any alternative distribution Q supported on the same samples can be represented by probabilities $q = (q_1, \dots, q_N)$ satisfying $q_i \geq 0, \qquad \sum_{i=1}^{N} q_i = 1.$

The KL-DRO problem considered in the main text (Equation 1) is

$$\sup_{q \in \Delta_N : \mathrm{KL}(q\|p) \leq \rho} \sum_{i=1}^{N} q_i \ell_i(\theta), \tag{6}$$

where

$$\mathrm{KL}(q\|p) = \sum_{i=1}^{N} q_i \log \frac{q_i}{p_i}.$$

Introducing a multiplier $\lambda > 0$ for the KL constraint gives the Lagrangian relaxation

$$\lambda\rho + \sup_{q \in \Delta_N} \left\{ \sum_{i=1}^{N} q_i \ell_i(\theta) - \lambda \sum_{i=1}^{N} q_i \log \frac{q_i}{p_i} \right\}. \tag{7}$$

For fixed λ, the first term inside the supremum favors distributions that assign more probability to high-loss samples, whereas the KL term penalizes deviations from the reference distribution. Hence λ controls the strength of this deviation.

To solve the inner problem, introduce a multiplier η for $\sum_i q_i = 1$:

$$\mathcal{L}(q, \eta) = \sum_i q_i \ell_i - \lambda \sum_i q_i \log \frac{q_i}{p_i} + \eta \left(\sum_i q_i - 1 \right).$$

The first-order condition with respect to q_i is

$$\ell_i - \lambda \left(\log \frac{q_i}{p_i} + 1 \right) + \eta = 0.$$

Therefore,

$$q_i = p_i \exp\left(\frac{\ell_i}{\lambda}\right) \exp\left(\frac{\eta - \lambda}{\lambda}\right).$$

The last factor is common to all samples and is determined by normalization. Consequently,

$$q_i^\star(\lambda) = \frac{p_i \exp(\ell_i(\theta)/\lambda)}{\sum_{k=1}^{N} p_k \exp(\ell_k(\theta)/\lambda)}. \tag{8}$$

Thus KL-DRO corresponds to an exponential tilting of the reference distribution toward high-loss observations.

Substituting equation 8 into equation 7 gives

$$\sup_{q \in \Delta_N} \left\{ \sum_i q_i \ell_i - \lambda \mathrm{KL}(q\|p) \right\} = \lambda \log \left(\sum_{i=1}^{N} p_i \exp\left(\frac{\ell_i}{\lambda}\right) \right).$$

Hence the constrained problem admits the dual representation

$$\sup_{Q : \mathrm{KL}(Q\|P_{\mathcal{D}}) \leq \rho} \mathbb{E}_Q[\ell(x;\theta)] = \inf_{\lambda > 0} \left\{ \lambda\rho + \lambda \log \mathbb{E}_{P_{\mathcal{D}}} \left[\exp\left(\frac{\ell(x;\theta)}{\lambda}\right) \right] \right\}, \tag{9}$$

under the standard regularity conditions for KL-DRO duality.

For the empirical distribution $p_i = 1/N$, this becomes

$$L_{\lambda,\rho}(\theta) = \lambda\rho + \lambda \log \left(\frac{1}{N} \sum_{i=1}^{N} \exp\left(\frac{\ell(x_i;\theta)}{\lambda}\right) \right). \tag{10}$$

When λ and ρ are fixed during fine-tuning, $\lambda\rho$ is constant with respect to θ. Therefore the models along the robustness path can equivalently be trained using

$$L_\lambda^{\mathrm{FT}}(\theta) = \lambda \log\left(\frac{1}{N}\sum_{i=1}^{N}\exp\left(\frac{\ell(x_i;\theta)}{\lambda}\right)\right). \tag{11}$$

A.2 Interpretation of the Robustness Parameter

Equation 8 makes explicit how λ determines the robustness regime. For the empirical distribution,

$$q_i^\star(\lambda) = \frac{\exp(\ell_i/\lambda)}{\sum_k \exp(\ell_k/\lambda)}.$$

As λ increases, differences between the exponential weights are attenuated. In particular,

$$\exp(\ell_i/\lambda) \to 1 \qquad \text{as} \qquad \lambda \to \infty,$$

and therefore

$$q_i^\star(\lambda) \longrightarrow p_i. \tag{12}$$

Correspondingly,

$$\lambda \log \mathbb{E}_{P_{\mathcal{D}}}\left[e^{\ell/\lambda}\right] \longrightarrow \mathbb{E}_{P_{\mathcal{D}}}[\ell], \qquad \lambda \to \infty. \tag{13}$$

Thus large λ recovers the nominal empirical-risk regime.

Conversely, as $\lambda \to 0^+$, the exponential weights increasingly concentrate on samples attaining the largest loss. Indeed, if $M = \max_i \ell_i$, then $q_i^\star(\lambda) = \frac{p_i \exp((\ell_i - M)/\lambda)}{\sum_k p_k \exp((\ell_k - M)/\lambda)}$, so samples satisfying $\ell_i < M$ receive asymptotically vanishing mass.

Therefore λ continuously controls the degree of adversarial reweighting:

$$\text{small } \lambda \quad \Longrightarrow \quad \text{strong concentration on high-loss samples,}$$

whereas

$$\text{large } \lambda \quad \Longrightarrow \quad \text{weights close to the reference distribution.}$$

This motivates the robustness paths used in the main paper: for each source distribution D_j, training models at several values of λ produces a family $\lambda \longmapsto \theta_{j,\lambda}$ that moves from strongly robust solutions toward increasingly nominal fine-tuning regimes.

A.3 Gradient and Mini-Batch Implementation

For large-scale fine-tuning, we replace the full empirical objective by its mini-batch counterpart. Given a mini-batch $\mathcal{B}$ of size n,

$$\widehat{L}_\lambda(\theta;\mathcal{B}) = \lambda \log\left[\frac{1}{n}\sum_{i\in\mathcal{B}}\exp\left(\frac{\ell_i(\theta)}{\lambda}\right)\right]. \tag{14}$$

The gradient has a particularly useful interpretation. Differentiating equation 14 gives

$$\nabla_\theta \widehat{L}_\lambda(\theta;\mathcal{B}) = \sum_{i\in\mathcal{B}} q_i^\star(\theta;\lambda)\nabla_\theta \ell_i(\theta), \tag{15}$$

where

$$q_i^\star(\theta;\lambda) = \frac{\exp(\ell_i(\theta)/\lambda)}{\sum_{k\in\mathcal{B}}\exp(\ell_k(\theta)/\lambda)}. \tag{16}$$

Thus the KL-DRO update is a weighted combination of the usual per-sample gradients. Samples with larger current losses receive larger weights, and the sharpness of this reweighting is controlled by λ. Algorithm 2 summarizes the resulting fine-tuning procedure. Repeating Algorithm 2 from the same initialization θ_0 for the grid $\lambda \in \Lambda$ produces the robustness path $\{\theta_{j,\lambda} : \lambda \in \Lambda\}$ associated with source distribution D_j.

B Additional Empirical Results

The next four subsections record exactly how every method, protocol, and dataset in Section 5.3 was implemented and measured.

B.1 Method Comparison Details

Full definitions of every method compared in Section 5.3, expanding the condensed description there.

Static baselines. These commit to a single model before any deployment batch is observed. **Pre-trained** deploys θ_0 unmodified. **Best Single FT** independently fine-tunes θ_0 on each source D_j and keeps, for the entire evaluation, the specialist with lowest validation loss. **Mixed-source FT** instead fine-tunes a single model on a uniform mixture of all source datasets.

Algorithm 2: KL-DRO Fine-Tuning at a Fixed Robustness Level

Input: Dataset $\mathcal{D}$, pretrained parameters θ_0, robustness level $\lambda > 0$, learning rate η
Output: Fine-tuned parameters θ_λ
Initialize $\theta \leftarrow \theta_0$;
while *not converged* **do**
  Sample a mini-batch $\mathcal{B} \subset \mathcal{D}$;
  Compute the normalized sequence loss $\ell_i(\theta)$ for each $x_i \in \mathcal{B}$;
  Compute

$$\widehat{L}_\lambda(\theta; \mathcal{B}) = \lambda \log \left[\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \exp\left(\frac{\ell_i(\theta)}{\lambda} \right) \right];$$

  Update

$$\theta \leftarrow \theta - \eta \nabla_\theta \widehat{L}_\lambda(\theta; \mathcal{B});$$

return θ_λ

Table 4: Comparison of the methods used in our evaluation. **Bank use** indicates which bank members are considered at deployment (**Deploy. input** = `---` means the method is calibrated once and consumes no per-batch signal; `Unlabelled` B means it is recomputed from the incoming batch); **Deploy. DOF** the free variables optimized at deployment; **Cost** is per-batch wall-clock time relative to Best Single FT (exact seconds/batch in Tables 3 and 2; hyperparameter values in Appendix B.4).

Method	Bank use	Deploy. input	Deployment rule	Deploy. Degreee of Freedom (DOF)	Cost
Static baselines					
Best Single FT	1	—	best-validation specialist, fixed	—	$1\times$
Mixed-source FT	—	—	one model, joint training on the source mixture	—	$1\times$
Routing over the bank					
Static Hierarchical Routing	Top-H	—†	Top-H + EG, frozen after calibration	fixed w	$\sim 6\times$
Hard Routing	all	Unlabelled B	$\arg\min_{\theta \in \mathcal{M}} \widehat{L}_B(\theta)$	1 discrete	$\sim 28\times$
Flat Routing	Top-H	Unlabelled B	Top-H by $r_j(B)$, then $\mathrm{softmax}(-\widehat{L}_B/\tau)$ over $\mathcal{C}_B$	H weights	$\sim 31\times$
Hierarchical Routing	Top-H	Unlabelled B	Top-H by $r_j(B)$, then EG over $\Delta(\mathcal{C}_B)$	H weights	$\sim 150\times$
Routing-then-Tent hybrids					
Static TENT (Best Single-FT)	1	—†	Tent on Best Single FT, calibrated once and frozen‡	LN params (fixed)	$\sim 0.3\times$
Online TENT (Best Single-FT)	1	Unlabelled B	Tent on Best Single FT, state carried across batches‡	LN params	$\sim 1\times$
Hierarchical Routing+	Top-H	Unlabelled B	truncated Hierarchical Routing once, then online Tent‡	H weights (fixed) + LN params	$\sim 3\times$
Oracle (not deployable)					
Target-FT Oracle	—	full target set	direct fine-tuning on the target domain	all θ	—

† Calibrated once on held-out mixture data; no per-batch signal is used at deployment. ‡ Tent adapts only LayerNorm affine parameters.

Routing strategies. Hard, Flat, and Hierarchical routing build θ_B from the bank $\mathcal{M}$ per incoming batch (Section 4, Algorithm 1). **Static Hierarchical Routing** calibrates the Hierarchical solve once on the first uncoming batch and then freezes it for the entire deployment, isolating the value of *per-batch* adaptivity from the value of the source–robustness structure alone.

External adaptation. Tent (Wang et al., 2020) applies updates to the LayerNorm affine parameters that minimize autoregressive cross-entropy on the incoming batch, rather than the original classification-based entropy objective, either directly on the pretrained model or on top of a selected or routed starting point. **Static TENT (Best Single-FT)** adapts once on B_1 from Best Single FT and then freezes, **Online TENT (Best Single-FT)** starts from Best Single FT and carries its adapted LayerNorm state forward across the stream, **Hierarchical Routing+** instead adapts on top of a once-calibrated Hierarchical Routing solution and carries the adapted state forward across the stream in the same way.

Oracle. **Target-FT Oracle** fine-tunes directly on the full target distribution and is not deployable. It defines the 100% point of the oracle-gap-closed metric used throughout Section 5.3; Pretrained defines 0%.

B.2 Exponentiated-Gradient Convergence: Trajectories and Full Comparison

This subsection validates the multi-start initialization and the vertex-comparison safety net used by Algorithm 1's Hierarchical branch (the `Init.` $w^{(0)}$ and best-iterate-selection steps): does the search actually need multiple random starts, and does it land close to the true optimum?

On the episodic GitHub stream (re-targeted bank), random restarts matter: a random-Dirichlet restart, not one of the two deterministic starts (uniform, flat-softmax), attains the lowest final-iteration loss in 47 of the 50 batches. GitHub batch_01 (Figure 7, left) is one such case: *random_5*

reaches the lowest loss (2.2890) at iteration 20, while both deterministic starts (*uniform*, *flat_soft*) plateau noticeably higher. The gap this leaves on the table if one truncated to only the deterministic starts is nonetheless small: averaged over all 50 GitHub batches, the best deterministic start is worse than the true best (any start). The vertex-comparison safety net still bounds the downside of truncation, but on this bank the random restarts are doing real, if modest, work rather than contributing nothing.

Figure 7 (right) additionally compares the routed solution against a grid-search ground truth over the same (reduced) simplex – θ_0, PG-19 (λ=10^{-5}), and PubMed Central (λ=10^{-4}) for this batch. The EG search's optimum is close to but not exactly at the true grid-search optimum. This grid search is expensive and was only computed for this one batch as a spot-check, not across the full stream, so this is illustrative of EG finding the right neighborhood rather than a general reliability claim; extending it to more batches is future work.

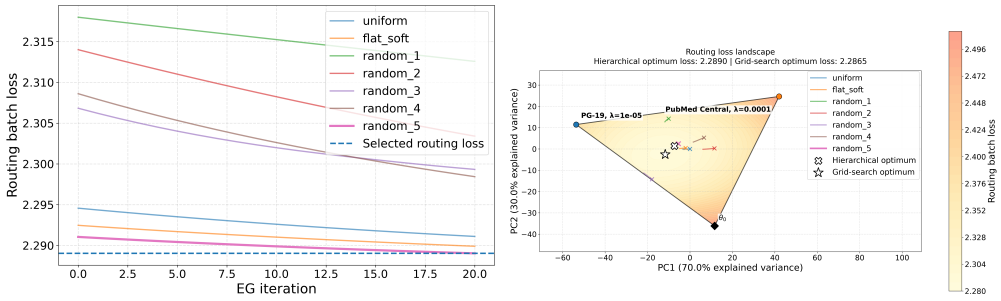


Figure 7: Episodic GitHub batch_01, re-targeted bank. **Left:** EG loss trajectories, one line per initialization; *random_5* (not a deterministic start) attains the lowest loss at iteration 20, matching the selected routing loss (dashed). **Right:** PCA loss landscape over the reduced simplex (θ_0, PG-19, PubMed Central) for the same batch, comparing the Hierarchical routing optimum ($\times$) against a grid-search ground truth ($\star$).

B.3 Deployment Protocol Details

Continual variants. *Online Tent* retains and updates its adapted parameters across successive batches, following Wang et al. (2020). This can exploit persistence in the deployment distribution, but may accumulate harmful updates when the distribution changes. In contrast, our hierarchical routing is applied independently to each incoming batch, using its autoregressive loss to select and combine models from the bank. We additionally consider *static routing* (Table 4), where the routing weights are optimised once on the first batch and then kept fixed throughout deployment.

Routing then adaptation. Routing and test-time adaptation are complementary: routing selects a starting point from the bank, test-time adaptation refines it on B. We therefore also evaluate the hybrid that routes to θ_B and then applies one online Tent update to it.

B.4 Experimental Setup and Reproducibility Details

This subsection records the exact settings needed to reproduce Section 5.3's results, beyond what is already stated there (Section 5.3's Setup paragraph). The project code is also shared for the sake of reproducibility.

Hardware and software. All model bank training and evaluation experiments were conducted on a High-Performance Computing (HPC) cluster. Jobs were allocated a single NVIDIA A100 GPU (128GB VRAM) and 8 CPU cores, utilizing PyTorch with CUDA optimizations (CUBLAS_WORKSPACE_CONFIG=:4096:8).

Model bank construction. We report results on two GPT-2 architectures, Small and Medium. Both are trained for 2 epochs with AdamW, learning rate 5.0×10^{-5}, context length 512, and a maximum token budget of 5,120,000 tokens per domain. The base batch size is 8 with 2 gradient-accumulation steps, dynamically auto-scaled (batch size halved, accumulation steps doubled) on Out-Of-Memory (OOM) errors. Each architecture incorporates multiple robustness checkpoints per domain using a KL-DRO mixed loss, trained across its own grid of 10 penalty weights λ (Table 5), where $\lambda = 0.0$ corresponds to standard Empirical Risk Minimization (ERM); $\gamma = 1.0$ and $\rho = 0.0$ throughout.

Data. Source domains were streamed directly from Hugging Face using the train split. Documents were filtered to a minimum of 10 characters, formatted, and chunked sequentially to fit the 512 context length until the token budget was reached. For training, the dataset was split internally using a 10% validation ratio with a fixed random seed of 42. For out-of-distribution (OOD) target

Table 5: Model bank λ grid, by architecture. Both architectures share the same learning rate, token budget, context length, and batch size/OOM settings (see text); only the λ grid differs.

Architecture	λ grid
GPT-2 Small	$\{0.0,\ 10^{-5},\ 10^{-4},\ 10^{-3},\ 10^{-2},\ 10^{-1},\ 0.3,\ 0.5,\ 1.0,\ 2.0\}$
GPT-2 Medium	$\{0.0,\ 0.02,\ 0.05,\ 0.10,\ 0.20,\ 0.50,\ 0.70,\ 1.00,\ 1.50,\ 2.00\}$

evaluation, we evaluated the models on exactly 2,500 samples drawn from the respective target dataset configurations.

Compute cost. All deployment-time measurements reported in Tables 3 and 2 were obtained on an NVIDIA RTX A6000 GPU (49 GB) on the server. We report wall-clock time, measured with Python's `time.time()`.For routing methods, the reported deployment time includes the scoring required by the method (including pretrained and bank scoring when applicable) together with the method-specific routing optimization and evaluation; diagnostic computations are excluded. Shared scoring operations are charged to each routing method so that the reported time reflects the cost of deploying that method independently.

Deployment evaluation. We evaluate the deployment methods under two complementary protocols: *Primary Episodic* and *Non-stationary Stream*. All experiments use a fixed random seed of 42 for reproducibility. In both protocols, the same incoming batch is used for model selection or adaptation and for evaluation. This is intentional: our setting is transductive, since the objective is to adapt directly to the current unlabelled deployment batch.

Primary Episodic. This protocol evaluates adaptation under a stationary deployment condition. Each deployment dataset is considered independently, and its batches are processed sequentially. Evaluation starts after an offset of 512k tokens, which prevents overlap with target data that may have been used to train the Target-FT Oracle. Best Single-FT is selected once on the first batch B_1. Online TENT is initialized from this checkpoint on B_1 and subsequently carries its adapted parameters from one batch to the next throughout the same deployment dataset; it is therefore not re-initialized from Best Single-FT at each batch. Static TENT performs the same initialization and adaptation on B_1, but then freezes the resulting model for all remaining batches. Similarly, Static Hierarchical Routing is optimized on B_1 and then frozen, whereas Hard, Flat, and ordinary Hierarchical Routing reroute on every incoming batch. TENT initialized from Hierarchical Routing performs a shorter Hierarchical Routing optimization only on B_1, applies its first Tent update, and then carries the resulting Tent-adapted model forward from batch to batch without any further routing. When evaluation moves to a new deployment dataset, a new independent episode begins and these initialization procedures are performed again on the new dataset's B_1.

Non-stationary Stream. This protocol evaluates whether deployment methods can track rapid distribution changes across domains. Deployment batches from disjoint target domains are interleaved in a balanced random order using the fixed random seed 42, with each target contributing the same number of batches. The resulting stream is generated and saved before evaluation, ensuring that every method is evaluated on exactly the same sequence of domain switches and batches. Since no Target-FT Oracle is used, evaluation starts at offset zero. Best Single-FT is selected on the first stream batch B_1; Static TENT and Static Hierarchical Routing are likewise adapted or optimized only on B_1 and then frozen. In contrast, online TENT retains its adaptation state across domain switches, while Hard, Flat, and ordinary Hierarchical Routing reroute on every incoming batch. TENT initialized from Hierarchical Routing performs its shorter routing optimization only on B_1 and then carries its online TENT state through the remainder of the stream, including across domain changes.

No Oracle is used in this setting. Instead, performance is reported through the per-batch Relative Loss Improvement over the pretrained model,

$$100\,\frac{L_{\text{pretrained}} - L_{\text{method}}}{L_{\text{pretrained}}},$$

which is computed independently for each batch before averaging across the stream.

Method hyperparameters. Table 6 summarizes the routing and test-time-adaptation hyperparameters, shared across all experiments. For TENT initialized from Hierarchical Routing, the initial routing optimization is deliberately truncated to $K_{\text{EG}} = 5$ iterations to reduce deployment cost, this value is a computational choice rather than an empirically optimized optimum. TENT updates only LayerNorm affine parameters.

Table 6: Routing and test-time-adaptation method hyperparameters, shared across all experiments.

Method	Hyperparameter	Value
Hierarchical Routing	Candidates retained after Top-H	$H = 3$
	EG iterations	$K_{\mathrm{EG}} = 20$
	EG learning rate	$\eta_{\mathrm{EG}} = 0.1$
	Random EG starts	5 (Dirichlet, $\alpha = 1.0$)
TENT (from Hierarchical Routing)	Truncated EG iterations	$K_{\mathrm{EG}} = 5$
TENT (all variants)	Adaptation steps	$K_{\mathrm{Tent}} = 1$
	Adaptation learning rate	$\eta_{\mathrm{Tent}} = 10^{-3}$
Flat Routing	Softmax temperature	$\tau = 1.0$

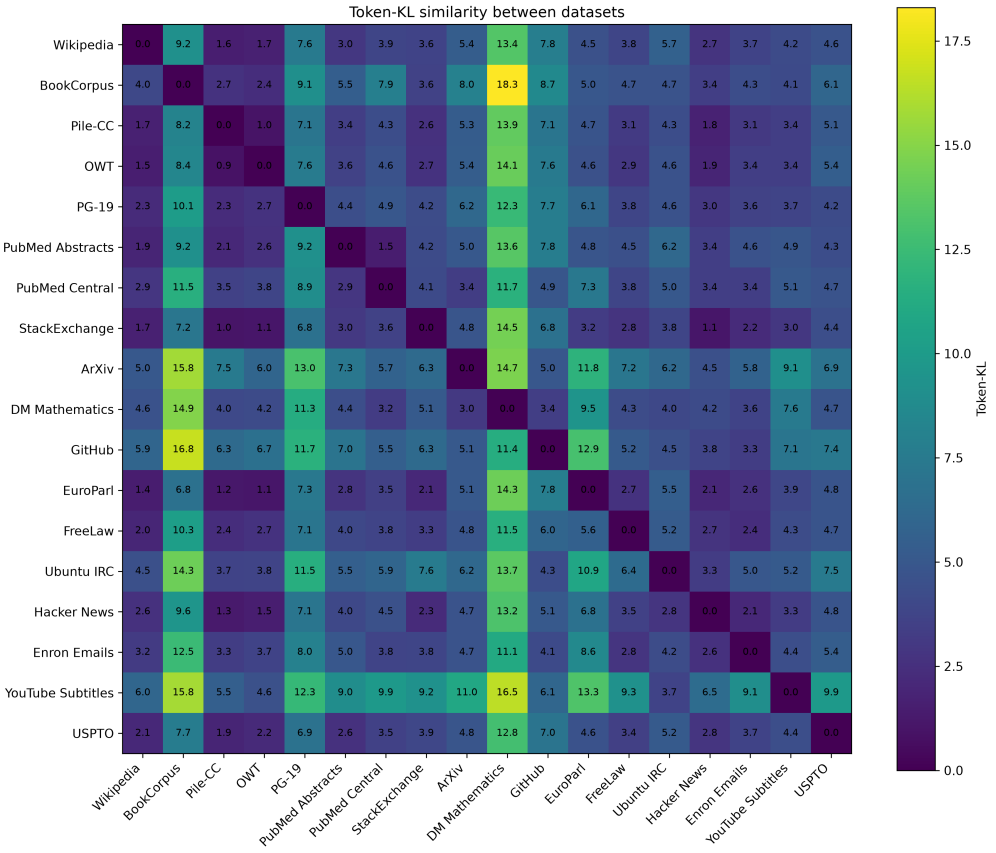


Figure 8: Pairwise Token-KL divergence between the candidate datasets. Rows correspond to datasets D_i, columns correspond to datasets D_j, and each entry reports the divergence $D_{\mathrm{KL}}(P_i \,\|\, P_j)$. Lower values indicate more similar token distributions, whereas larger values correspond to stronger distribution shifts.

B.5 Dataset Selection and Token-KL Divergence

Dataset selection is performed at two different stages of our study, with different objectives. First, to construct the cross-domain generalization experiment shown in Figure 2 (**Left**), we select six datasets: DM Mathematics, GitHub, PG-19, PubMed Central, PubMed Abstracts, and FreeLaw; based on the pairwise Token-KL divergence

$$\mathrm{TokenKL}(D_i, D_j) = D_{\mathrm{KL}}(P_i \,\|\, P_j)$$

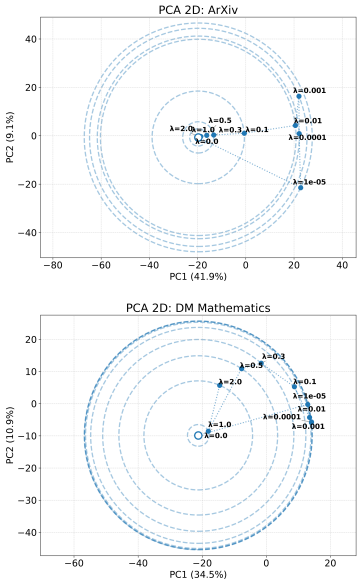
between the empirical token distributions of all 18 candidate datasets from The Pile (Figure 8). The resulting similarity matrix reveals groups of datasets with related linguistic characteristics: for instance, the biomedical corpora (PubMed Abstracts and PubMed Central) are close to each other, whereas legal, mathematical, programming, and general-text corpora occupy more distinct regions. The six datasets used in Figure 2 (**Left**) were therefore selected to cover sufficiently diverse regions of this Token-KL space and expose heterogeneous cross-domain generalization behavior.

Table 7: Scalability check: same bank sources and six targets as Table 3, $H = 3$, New Order, with the gpt2-small backbone replaced by gpt2-medium and a smaller model bank at this scale (Appendix B.6). 0% = Pretrained, 100% = Target-FT Oracle. FreeLaw‡ and OWT2‡ report relative perplexity change vs. Pretrained instead, as in Table 3.

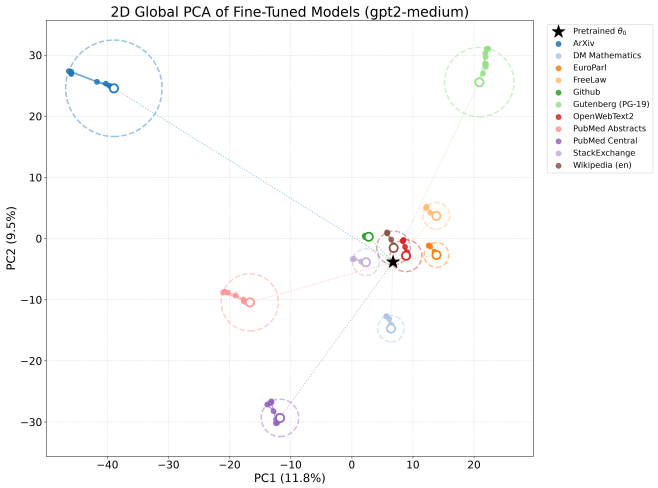
Method	DM Math.	EuroParl	FreeLaw‡	GitHub	OWT2‡	PubMed Abs.	Cost (s/batch)
Pretrained	2.399 (0.0%)	3.540 (0.0%)	2.682 (0.0%)	2.416 (0.0%)	3.004 (0.0%)	2.854 (0.0%)	1.0
Best Single FT	2.554 (−16.1%)	3.632 (−9.9%)	2.802 (−12.7%)	2.392 (0.3%)	3.275 (−31.0%)	3.065 (−58.2%)	1.2
Static TENT (Best Single-FT)	2.465 (−7.0%)	3.477 (6.7%)	2.741 (−6.0%)	2.360 (3.8%)	3.196 (−21.1%)	2.971 (−32.3%)	0.5
Episodic TENT (Best Single-FT)	2.183 (22.0%)	3.138 (43.0%)	2.622 (5.8%)	2.046 (28.9%)	3.047 (−4.4%)	2.872 (−4.9%)	1.7
Hard Routing	2.399 (0.0%)	3.540 (0.0%)	2.681 (0.2%)	2.052 (17.1%)	3.004 (0.0%)	2.854 (0.0%)	2.9
Flat Routing	2.401 (−0.2%)	3.489 (5.5%)	2.658 (2.3%)	2.001 (24.8%)	3.069 (−6.7%)	2.850 (1.1%)	4.7
Static Hierarchical Routing	2.396 (0.2%)	3.486 (5.8%)	2.636 (4.5%)	2.286 (13.4%)	3.004 (0.0%)	2.836 (4.8%)	5.2
Hierarchical Routing	2.392 (0.6%)	3.486 (5.8%)	2.630 (5.1%)	**1.950 (28.4%)**	3.004 (0.0%)	2.842 (3.2%)	241.3
Hierarchical Routing+	**2.138 (26.7%)**	**3.084 (48.7%)**	**2.557 (11.8%)**	2.005 (33.3%)	**2.961 (4.3%)**	**2.784 (19.3%)**	3.1
Target-FT Oracle†	*1.420 (100.0%)*	*2.604 (100.0%)*	*2.879 (−21.7%)*	*1.234 (100.0%)*	*2.950 (5.3%)*	*2.490 (100.0%)*	—

†Not deployable: direct fine-tuning on the target stream, shown only to anchor the 100% reference point.

This selection should be distinguished from the dataset configuration used in the experiments of Section 5. For these experiments, source and deployment domains are selected based on the global PCA representation shown in Figure 4b, with the aim of covering diverse regions of the observed domain geometry. We use PubMed Central, ArXiv, and PG-19 as source datasets for constructing the model bank, and DM Mathematics, PubMed Abstracts, OWT2, FreeLaw, EuroParl, and GitHub as deployment datasets.



(a) Single-domain robustness path.



(b) Robustness paths across source domains.

Figure 9: **Geometry of robustness paths at gpt2-medium.** Same construction and same source domains as Figure 4, at gpt2-medium instead of gpt2-small. The ERM checkpoint and the within-family ordering by λ, the source-dependent specialization directions from θ_0, and the overall organization of the bank in parameter space all match the gpt2-small case, indicating that this geometry is not specific to one model scale.

B.6 Scalability Check at GPT2-Medium

Figure 9 extends the same analysis to GPT-2 Medium using the same source domains. Table 7 similarly evaluates the six targets of Table 3, using a smaller model bank at this scale; replicating the complete bank is left for future work. The overall ranking remains essentially unchanged: Hierarchical Routing+ performs best on five of the six targets, while the FreeLaw/OWT2 oracle degeneracies persist (Section 5.1), suggesting that they are not specific to model scale. The only exception is GitHub, where plain Hierarchical Routing achieves a lower loss than Hierarchical Routing+ (1.950 vs. 2.005), whereas Routing+ performs best on GitHub with GPT-2 Small.

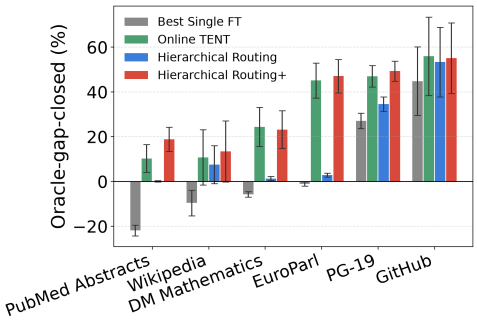


Figure 10: Leave-one-source-out ablation (gpt2-small), 6 of 10 targets with a usable Target-FT Oracle: oracle-gap-closed per target (± 1 std over 50 batches), ArXiv and OWT2 omitted (degenerate oracle, see text).

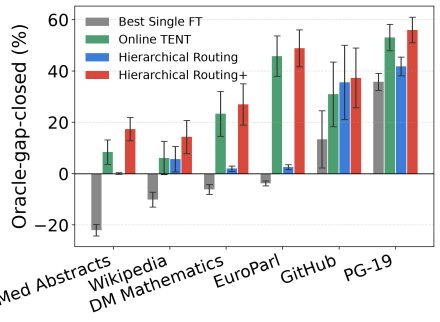


Figure 11: Leave-one-source-out ablation (gpt2-medium), 6 of 10 targets with a usable Target-FT Oracle: oracle-gap-closed per target (± 1 std over 50 batches), ArXiv and OWT2 omitted (degenerate oracle, see text).

B.7 Leave-One-Source-Out Ablation

To test whether Hierarchical Routing remains useful when a target domain is entirely absent from the bank (not merely under-represented, as in the main episodic streams), we run a leave-one-source-out ablation over 10 Pile domains (ArXiv, DM Mathematics, EuroParl, GitHub, PG-19, OWT2, PubMed Abstracts, PubMed Central, StackExchange, Wikipedia). For each domain in turn, we remove it from the bank, leaving a 9-source bank, and evaluate that reduced bank on the held-out domain as the deployment target under the same episodic protocol as Table 3 (50 batches, offset 512k tokens, $H = 3$).

Figure 10 reports gpt-small results for the six completed targets with a usable Target-FT Oracle. The results reveal a clear dependence on bank coverage. When a related source remains (PG-19, GitHub), Hierarchical Routing recovers a substantial fraction of the oracle gap, whereas gains are naturally smaller when no suitable candidate is available. Wikipedia provides an intermediate case, where frequently selected substitutes are only weakly related. Importantly, Hierarchical Routing+ is substantially less sensitive to bank coverage, combining model selection with direct adaptation to the deployment batch and performing best or close to best across all targets.

We repeat the same protocol at gpt2-medium, with all eight of the ten LOO targets completed so far having a usable Target-FT Oracle (again excluding ArXiv and OWT2). Figure 11 shows the same overall pattern as gpt2-small, Hierarchical Routing's gain tracks bank coverage, and Hierarchical Routing+ remains the most robust method regardless of coverage.

B.8 Ablation: Sensitivity to the Top-H Bank Size

Section 5.3 uses $H = 3$ throughout. To assess sensitivity to this design choice, we re-run Hierarchical Routing on DM Mathematics with $H \in \{2, 3, 4\}$ on the same fixed-seed batches. As Figure 12 shows, performance is essentially unchanged across H, while cost increases slightly with larger shortlists. This is consistent with the limited coverage of DM Mathematics in the model bank (Appendix B.7), leaving additional candidates with little useful information to contribute. While this single-target check does not imply that H is generally immaterial, it shows that the reported gains are not sensitive to its chosen value in this setting.

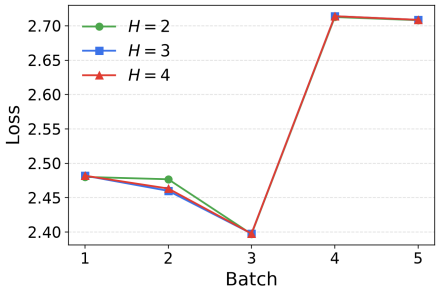


Figure 12: Per-batch loss on DM Mathematics as H varies. The three curves are visually indistinguishable.