BEYOND UNIFORM RANKS: HIERARCHICAL RANK ALLOCATION FOR LLM FINE-TUNING IN SENTIMENT ANALYSIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Financial sentiment analysis has become crucial for market forecasting and investment decision-making, with Low-Rank Adaptation (LoRA) enabling efficient large language model (LLM) fine-tuning by introducing trainable low-rank matrices. However, LoRA typically assigns the same rank to every transformer layer, although layers respond differently to rank: across four depth groups, the rank with the lowest measured MSE ranges from 128 to 384. To address this limitation, we propose Hierarchical Rank Allocation (HiRank), a performance-based exploration framework that treats per-layer rank as a decision variable under a global parameter budget and scores each candidate allocation by the validation error of a complete fine-tuning run. HiRank searches in two phases: (1) coarse-grained joint exploration scores candidate layer groupings and narrows a rank interval for each group, and (2) fine-grained rank optimization assigns one rank per group within these intervals while keeping the total rank within the budget. The allocation is fixed before training, so the fine-tuned model keeps the inference cost of uniform LoRA. On three financial sentiment datasets and two parameter budgets, HiRank lowers MSE relative to uniform-rank LoRA in all 24 grouping, budget, and dataset settings, by 6.70% on average with eight layer groups. At the 512-rank budget and with the same number of fine-tuning evaluations, it also lowers MSE by 4.38% relative to random search over per-layer rank vectors.

1 INTRODUCTION

Sentiment analysis has become essential for understanding market dynamics and investment decisions in financial applications, leveraging LLMs to extract meaningful insights from market-related texts (Ding et al., 2024b; Lee et al., 2025). However, training LLMs from scratch requires enormous computational resources (Llama Team, 2024), driving the widespread adoption of pre-trained models followed by fine-tuning (Anisuzzaman et al., 2025). Even full fine-tuning is memory-bound on commodity hardware, which motivates parameter-efficient adaptation. The practical question is then how to spend a fixed adaptation budget well. Because transformer layers respond differently to added capacity, the answer depends on how that budget is divided across layers.

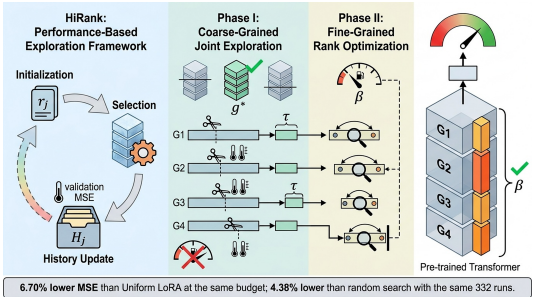


Figure 1: **Left:** each round proposes a rank vector $\mathbf{r}_j$, fine-tunes the LLM with it, and logs its validation MSE in $\mathcal{H}_j$. **Right:** Phase I selects g^* and narrows each group's interval to width τ; Phase II assigns ranks within budget β.

Parameter-Efficient Fine-Tuning (PEFT) methods answer this need across domains (Wang et al., 2025). These approaches, including adapter modules (Hu et al., 2023; Steitz & Roth, 2024), prefix tuning (Zhang et al., 2023c; Le et al., 2025), and prompt tuning (Liu et al., 2023; Jain et al., 2024), aim to achieve comparable performance while dramatically reducing trainable parameters. Among these methods, LoRA has emerged as particularly effective, offering superior performance with minimal additional trainable parameters (Dettmers et al., 2023; Jiang et al., 2024). LoRA reparameterizes each weight update as a product of two low-rank matrices $\Delta W = BA$ while freezing the pretrained

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.

backbone, reducing the trainable parameter count by one to two orders of magnitude and enabling adaptation on commodity GPUs (Hu et al., 2022). The simplicity and modularity of this design have made LoRA the de-facto standard for adapting large foundation models across NLP, vision, and multimodal tasks.

However, current LoRA implementations employ uniform rank allocation across all transformer layers (Hu et al., 2022), which is suboptimal due to the intrinsic heterogeneity of different layers (Skean et al., 2025). This uniform distribution ignores the hierarchical nature of representation learning (Geva et al., 2022), where shallow layers capture low-level features and deeper layers encode complex semantic relationships (Jin et al., 2025). Moreover, different layers exhibit varying rank requirements (Song et al., 2025), with some demanding higher ranks for expressiveness while others achieving their best measured performance at lower ranks. This limitation becomes particularly severe in resource-constrained fine-tuning scenarios, where inappropriate allocation to one layer directly compromises the capacity available for others. These observations call for treating per-layer rank as a decision variable jointly optimized under a global parameter budget.

To address these limitations, we propose Hierarchical Rank Allocation (HiRank), a performance-based exploration framework with a hierarchical allocation strategy (Figure 1). It decomposes the optimization into two phases: (1) coarse-grained exploration that scores candidate layer groupings and narrows a rank interval for each group, and (2) fine-grained optimization that assigns one rank per group within these intervals under the budget. Our main contributions are as follows:

- We identify a critical performance gap in current LoRA implementations, where uniform rank allocation ignores the hierarchical heterogeneity among transformer layers, leading to suboptimal parameter budget utilization.
- We propose HiRank, a two-phase search driven by measured validation performance rather than gradient-importance proxies, decomposing the global allocation problem into tractable sub-problems while enforcing the parameter budget exactly.
- HiRank reduces MSE by 6.70% on average over Uniform LoRA and by 4.38% over random search at a matched search budget, on the NEU, FXE, and INV financial-regression benchmarks under matched parameter budgets.

2 Motivation

Hierarchical Heterogeneity. To test whether layers differ in their sensitivity to the LoRA rank, we manually adjust the rank r for different layers while keeping others fixed. As illustrated in Figure 3, different layers exhibit fundamentally different responses: some favor higher ranks while others achieve their best measured performance at lower ranks. This finding confirms that a one-size-fits-all rank allocation is inherently suboptimal, raising Challenge #1 of *how to determine appropriate rank for each layer*. Per-group details are provided in Appendix A.2.

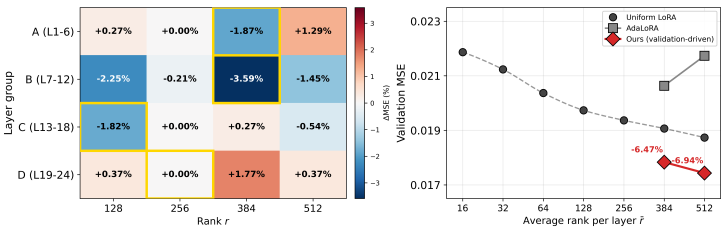


Figure 2: **Left:** MSE change vs. uniform r=256 per group. **Right:** MSE vs. average rank $\bar{r}$. Details in Appendix A.3.

Rank-Performance Trade-off. We also observe that the relationship between rank and performance is not simply linear. As shown in Figure 3, for certain layers, increasing the rank beyond a threshold actually degrades performance, likely due to overfitting from over-parameterization. This issue is exacerbated under fixed parameter budgets, where over-allocating ranks to certain layers reduces the capacity available for others, amplifying the cost of suboptimal assignments; per-group penalties reach up to 3.37% (Figure 2, left). Each layer thus possesses an optimal rank interval balancing expressiveness against overfitting risk. Consequently, this introduces Challenge #2 of *how to optimize model performance under budget constraints*.

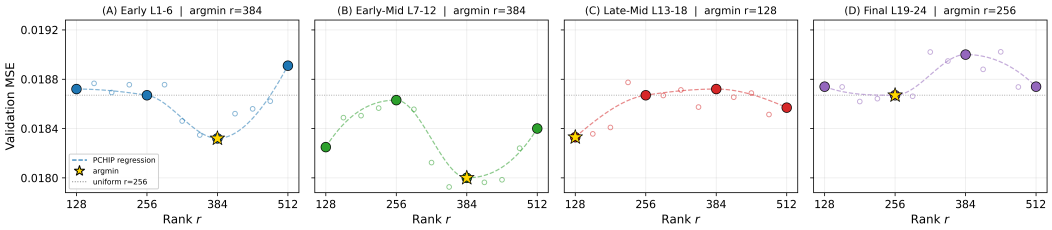


Figure 3: Layer-wise rank sensitivity at $r \in \{128, 256, 384, 512\}$, with other groups fixed at r=256. Gold stars: argmins $r^\star = (384, 384, 128, 256)$. Gray dashed: uniform-$r$=256 baseline.

3 Problem Formulation

To address the challenges identified in Section 2, we formulate layer-wise LoRA rank allocation for a pre-trained LLM with L Transformer layers as a budget-constrained discrete optimization problem:

$$\mathbf{r}^* = \arg\min_{\mathbf{r}\in\mathbb{N}^L} \mathcal{E}(f_{\mathbf{r}}; \mathcal{D}) \quad \text{s.t.} \quad \sum_{i=1}^{L} r_i \le \beta, \tag{1}$$

where r_i is the LoRA rank of layer i, $f_{\mathbf{r}}$ is the LLM fine-tuned with ranks $\mathbf{r}$, $\mathcal{E}(f_{\mathbf{r}}; \mathcal{D})$ is its error on the task data $\mathcal{D}$, and β is the total rank budget. Appendix A.4 and A.5 give the LoRA background and the extended formulation.

Challenge #1 corresponds to the decision variable: $\mathbf{r}$ gives each layer its own rank, whereas uniform LoRA fixes every r_i to β/L. Challenge #2 corresponds to the constraint: all layers share one budget, so raising one layer's rank leaves less for the others.

4 Solution

HiRank scores each candidate by full fine-tuning and searches in two phases: Phase I selects a layer grouping and narrows each group's rank interval; Phase II assigns one rank per group under β.

4.1 Performance-Based Exploration Framework

The framework (Figure 4) searches the allocations of Eq. (1) by treating fine-tuning as a black-box evaluator. It runs a closed loop of three operations: *initialization* (rank-vector generation), *selection* (LLM fine-tuning), and *history update* (validation and feedback). At iteration j, the exploration module $\mathcal{M}(\cdot)$ proposes a candidate $\mathbf{r}_j$, the LLM is fine-tuned with $\mathbf{r}_j$, and its validation error $\mathcal{E}_j$ is appended to the history $\mathcal{H}_j$, which conditions the next proposal. Each candidate is thus scored by the measured performance of a complete fine-tuning run. The loop fixes how candidates are evaluated and recorded, and $\mathcal{M}(\cdot)$ decides which candidate to evaluate next; Section 4.2 instantiates it with a hierarchical allocation strategy.

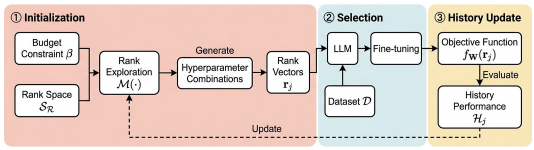


Figure 4: Performance-based exploration framework: Initialization (pink), Selection (light blue), and History Update (light yellow), with $\mathcal{H}_j$ feeding back to constrain the next rank vector.

Initialization. The exploration module $\mathcal{M}$ samples the candidate $\mathbf{r}_j$ at iteration j from the rank space $\mathcal{S}_\mathcal{R} = \{\mathbf{r} \in \mathbb{N}^L : r_i \in \mathcal{R}, \forall i\}$, where $\mathcal{R} = \{r_{\min}, \ldots, r_{\max}\}$ is the per-layer rank range, subject to the budget constraint of Eq. (1).

Selection. The candidate $\mathbf{r}_j$ instantiates a complete LoRA configuration that fine-tunes the LLM on the training set $\mathcal{D}_{\text{train}}$, yielding the adapted model $f_\mathbf{W}(\mathbf{r}_j)$ for the current rank assignment. This is the only operation in the loop that consumes gradient compute; its output is the trained model handed off for validation.

History Update. The trained model $f_\mathbf{W}(\mathbf{r}_j)$ is evaluated on the validation set $\mathcal{D}_{\text{val}}$ using the objective $\mathcal{E}(\cdot)$, producing the validation performance $\mathcal{E}_j$. The pair $(\mathbf{r}_j, \mathcal{E}_j)$ is appended to the history repository $\mathcal{H}_j = \{(\mathbf{r}_k, \mathcal{E}_k)\}_{k=1}^{j}$, which feeds back to constrain the next rank selection. The loop terminates upon convergence or budget exhaustion, returning the best-performing $\mathbf{r}^*$ in $\mathcal{H}$.

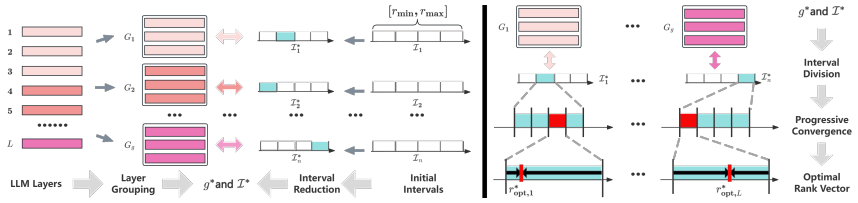


Figure 5: Hierarchical allocation strategy: **Left:** coarse-grained joint exploration of layer groupings and rank intervals; **Right:** fine-grained rank optimization within each refined interval.

Performance-Driven Selection Mechanisms. The exploration module $\mathcal{M}$ implements different selection strategies based on the phase. During initialization, the module selects an initial configuration:

$$\mathbf{r}_1 = \mathcal{M}(\mathcal{S}_\mathcal{R}) \quad \text{s.t.} \quad \sum_{i=1}^{L} r_{1,i} \leq \beta. \tag{2}$$

In subsequent iterations (Details in Appendix A.6), the selection incorporates historical performance data:

$$\mathbf{r}_{j+1} = \mathcal{M}(\mathcal{S}_\mathcal{R} | \mathcal{H}_j). \tag{3}$$

4.2 Hierarchical Allocation Strategy

This strategy instantiates the exploration module $\mathcal{M}(\cdot)$ defined in Section 4.1, and decomposes the optimization into two phases as illustrated in Figure 5.

Phase I: Coarse-Grained Joint Exploration. Phase I selects how many layer groups to use and refines each group's rank interval by iterative interval reduction until its width is at most a threshold τ (Figure 5, left).

Algorithm 1 evaluates different grouping strategies defined by candidate set $\mathcal{G} = \{g \in \mathbb{N} : 1 < g \leq L \text{ and } L \bmod g = 0\}$. It uses the binary greedy interval reduction function $\Phi(\cdot)$ to refine each group's interval, and the expansion function $\Omega(\cdot)$ to map group-level ranks to layer-level configurations, where all layers within a group share one rank. For each candidate g, it narrows every group's interval with Φ, forms the rank vector from the interval midpoints, evaluates it once on $\mathcal{D}_{\text{val}}$, and keeps the grouping with the lowest error $\mathcal{E}_g$ together with its intervals. Restricting $\mathcal{G}$ to exact divisors of L yields equal-sized groups and keeps $|\mathcal{G}|$ small enough to enumerate; Phase I omits the budget constraint β deliberately, deferring strict enforcement to Phase II so that grouping topology is selected on raw fit alone.

Algorithm 1 Coarse-Grained Joint Exploration

Input: L, $\mathcal{G}$, initial range $[r_{\min}, r_{\max}]$, interval threshold τ
Output: $(g^*, \mathcal{I}^*)$ where $\mathcal{I}^* = \{[l_i^*, u_i^*]\}_{i=1}^{g^*}$ with $u_i^* - l_i^* \leq \tau$

for each $g \in \mathcal{G}$ **do**
  $\mathcal{I}_g^{\text{opt}} \leftarrow \Phi(g, [r_{\min}, r_{\max}], \tau)$
  $\mathbf{r}_g \leftarrow \Omega(\text{midpoints}(\mathcal{I}_g^{\text{opt}}), g)$
  $\mathcal{E}_g \leftarrow \mathcal{E}(f_{\mathbf{r}_g}; \mathcal{D}_{\text{val}})$
  if $\mathcal{E}_g < \mathcal{E}^*$ **or** first iteration **then**
    $\mathcal{E}^* \leftarrow \mathcal{E}_g$, $(g^*, \mathcal{I}^*) \leftarrow (g, \mathcal{I}_g^{\text{opt}})$
  end if
end for

The function $\text{midpoints}(\mathcal{I}) = \{\lfloor (l_i + u_i)/2 \rfloor\}_{i=1}^{|\mathcal{I}|}$ computes the midpoint of each interval in $\mathcal{I}$ as representative rank values for evaluation. Phase I returns the grouping g^* that minimizes validation error together with its refined intervals $\mathcal{I}^* := \Phi(g^*, [r_{\min}, r_{\max}], \tau)$, evaluating only $|\mathcal{G}|$ candidate groupings at coarse rank resolution and thereby replacing the layer-level search space $|\mathcal{S}_\mathcal{R}| = (r_{\max} - r_{\min} + 1)^L$ with a g-dimensional surrogate.

Algorithm 2 details the binary greedy interval reduction function $\Phi(\cdot)$. The function $\textsc{ConstructVector}(v, i, \mathcal{I}_g)$ creates a complete rank vector by setting group i to value v while using midpoint values for all other groups, enabling fair performance comparison. Groups are processed independently: each group's interval is refined while other groups stay at their current midpoints. This treats group-level interactions as negligible at the coarse granularity of Phase I, a working assumption rather than a verified property; Table 4 bounds its practical cost, since the structured search retains a 4.38% average MSE advantage over unstructured random search at identical compute. Each iteration halves the active interval, so every group reaches the stopping width τ after $\lceil \log_2((r_{\max} - r_{\min})/\tau) \rceil$ halvings, independently of where the per-group optimum lies.

Algorithm 2 Binary Greedy Interval Reduction (Φ)

Input: g, $[r_{\min}, r_{\max}]$, τ
Output: Optimized intervals $\mathcal{I}_g^{\text{opt}}$ satisfying width constraint τ

Initialize $\mathcal{I}_g[i] \leftarrow [r_{\min}, r_{\max}]$ for all i
for each group $i = 1$ to g **do**
  $[l_i, u_i] \leftarrow \mathcal{I}_g[i]$
  while $u_i - l_i > \tau$ **do**
    $m \leftarrow \lfloor (l_i + u_i)/2 \rfloor$; $(m_l, m_r) \leftarrow (\lfloor (l_i + m)/2 \rfloor, \lfloor (m + u_i)/2 \rfloor)$
    $\mathbf{r}_x \leftarrow \textsc{ConstructVector}(m_x, i, \mathcal{I}_g)$, $\mathcal{E}_x \leftarrow \mathcal{E}(f_{\mathbf{r}_x}; \mathcal{D}_{\text{val}})$ for $x \in \{l, r\}$
    if $\mathcal{E}_l < \mathcal{E}_r$ **then**
      $u_i \leftarrow m$
    else
      $l_i \leftarrow m$
    end if
  end while
  $\mathcal{I}_g[i] \leftarrow [l_i, u_i]$
end for

Upon completion, Phase I produces the selected grouping g^* and corresponding rank intervals $\mathcal{I}^* = \{\mathcal{I}_1^*, \mathcal{I}_2^*, \ldots, \mathcal{I}_{g^*}^*\}$, where each interval $\mathcal{I}_i^* = [l_i, u_i]$ defines refined search boundaries for Phase II.

Phase II: Fine-Grained Rank Optimization. This phase determines the exact rank value for each group within the refined intervals $\mathcal{I}^*$ from Phase I, with strict budget enforcement to satisfy the constraint $\sum_{i=1}^{L} r_i \leq \beta$, as shown in the right part of Figure 5.

Algorithm 3 processes groups sequentially. For each group i, the algorithm computes the effective upper bound $u_{eff} = \min(u_i, \lfloor \beta_{rem}/s \rfloor)$, where β_{rem} denotes the remaining budget and $s = L/g^*$ is the number of layers per group. Sequential processing is essential: each group's chosen r_i^* reduces β_{rem}, tightening u_{eff} for subsequent groups, whereas a parallel assignment would lack this running-budget signal and could violate $\sum r_i \leq \beta$. Since β_{rem} is decremented by $s\, r_i^*$ after every group, any choice $r_i^* \in [l_i, u_{eff}]$ preserves the budget at each step by induction on i, guaranteeing $\sum_{l=1}^{L} r_l \leq \beta$ upon termination.

Algorithm 3 Fine-Grained Rank Optimization

Input: L, $(g^*, \mathcal{I}^*)$, β
Output: $\mathbf{r}_{\text{opt}}^*$

$\mathbf{r}_g \leftarrow \mathbf{0}_{g^*}$; $\beta_{rem} \leftarrow \beta$; $s \leftarrow L/g^*$
for $i = 1$ to g^* **do**
  $[l_i, u_i] \leftarrow \mathcal{I}^*[i]$
  $u_{eff} \leftarrow \min(u_i, \lfloor \beta_{rem}/s \rfloor)$
  $r_i^* \leftarrow \Psi([l_i, u_{eff}], i, \mathbf{r}_g)$
  $\mathbf{r}_g[i] \leftarrow r_i^*$; $\beta_{rem} \leftarrow \beta_{rem} - r_i^* \cdot s$
end for
return $\Omega(\mathbf{r}_g, g^*)$

Algorithm 4 details the binary greedy rank search function $\Psi(\cdot)$, which performs binary search within the constrained interval $[l_i, u_{eff}]$. The function $\textsc{BuildRankVector}(v, i, \mathbf{r}_g)$ constructs a complete rank vector by setting group i to value v while using previously determined rank values from $\mathbf{r}_g$ for other groups. Ψ contrasts the two endpoints l and u directly within the narrow interval inherited from Phase I (width $\leq \tau$), retaining the half whose endpoint scores better.

Algorithm 4 Binary Greedy Rank Search (Ψ)

Input: Interval $[l, u]$, group index i, current group config $\mathbf{r}_g$
Output: Selected rank r^*

while $u > l$ **do**
  $\mathbf{r}_x \leftarrow \textsc{BuildRankVector}(x, i, \mathbf{r}_g)$, $\mathcal{E}_x \leftarrow \mathcal{E}(f_{\mathbf{r}_x}; \mathcal{D}_{\text{val}})$ for $x \in \{l, u\}$
  if $\mathcal{E}_l < \mathcal{E}_u$ **then**
    $u \leftarrow l + \lfloor (u - l)/2 \rfloor$
  else
    $l \leftarrow l + \lceil (u - l)/2 \rceil$
  end if
end while
return l

Upon completion, $\Omega(\cdot)$ maps group-level ranks $\mathbf{r}_g$ to the layer-level vector $\mathbf{r}_{\text{opt}}^*$. The returned allocation is feasible for Eq. (1) evaluated on $\mathcal{D}_{\text{val}}$, with the budget provably enforced by the dynamic u_{eff} in Algorithm 3. Within each interval, Algorithm 4 retains the better-scoring

Table 1: Performance comparison of different LoRA rank configurations on regression tasks across three datasets. ↓ indicates lower is better. The first row is the no-LoRA baseline; the last two rows are the best uniform LoRA configurations.

Configuration	NEU Dataset			FXE Dataset			INV Dataset		
	MSE↓	MAE↓	RMSE↓	MSE↓	MAE↓	RMSE↓	MSE↓	MAE↓	RMSE↓
Baseline	0.0189	0.0981	0.1375	0.0202	0.1014	0.1421	0.0183	0.0959	0.1353
Rank=8	0.0221	0.1055	0.1487	0.0215	0.1063	0.1466	0.0199	0.1013	0.1411
Rank=16	0.0231	0.1081	0.1520	0.0227	0.1092	0.1507	0.0198	0.1012	0.1407
Rank=32	0.0211	0.1040	0.1453	0.0225	0.1078	0.1500	0.0201	0.1031	0.1418
Rank=64	0.0201	0.1010	0.1418	0.0213	0.1051	0.1460	0.0197	0.1011	0.1404
Rank=128	0.0193	0.0995	0.1389	0.0209	0.1036	0.1446	0.0190	0.0985	0.1378
Rank=256	0.0191	0.0987	0.1382	0.0205	0.1028	0.1432	0.0185	0.0974	0.1360
Rank=384	**0.0187**	**0.0974**	**0.1368**	**0.0203**	**0.1018**	**0.1425**	**0.0182**	**0.0958**	**0.1349**
Rank=512	**0.0184**	**0.0955**	**0.1356**	**0.0197**	**0.1004**	**0.1404**	**0.0181**	**0.0952**	**0.1345**

endpoint at every halving, a greedy rule that runs only on intervals already narrowed to width $\leq \tau$ by Phase I; Appendix A.21 gives its search complexity.

4.3 Accumulation-Based Strategy

An alternative rank allocation strategy draws inspiration from AdaLoRA's importance-driven approach (Zhang et al., 2023b), utilizing accumulated sensitivity scores to guide parameter budget distribution across transformer layers. For each layer i, the accumulated importance score is defined as:

$$S_i^{\text{acc}} = \frac{1}{T} \sum_{t=1}^{T} \sum_{\mathbf{W} \in \mathcal{W}_i} \|\mathbf{W} \odot \nabla_{\mathbf{W}} \mathcal{L}^{(t)}\|_F, \tag{4}$$

where T is the total number of training steps and $\mathcal{W}_i$ represents all weight matrices in layer i. The rank allocation for layer i is determined by proportionally distributing the total budget β according to normalized importance scores:

$$r_i = \max\left(1, \left\lfloor \frac{S_i^{\text{acc}}}{\sum_{k=1}^{L} S_k^{\text{acc}}} \cdot \beta \right\rfloor\right). \tag{5}$$

5 Evaluation

5.1 Implementation Details

Software and Hardware. We use Ubuntu 22.04, Python 3.9.19, PyTorch 2.3.1, PEFT 0.12.0, and CUDA 12.4, running with 32GB RAM and an NVIDIA RTX 3090Ti GPU.

Datasets. We evaluate our approach on three financial sentiment analysis datasets: NEU, FXE, and INV, containing news and analysis texts on EUR/USD exchange rates with sentiment polarity scores ranging from -1 to 1. More details are provided in Appendix A.8.

Backbone Model. We utilize Twitter-RoBERTa-Large (Loureiro et al., 2023) as the backbone with 24 transformer layers and a regression head for sentiment polarity prediction. More details are provided in Appendix A.9.

Evaluation Metrics. We employ Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE) as evaluation metrics. Metric definitions are provided in Appendix A.10.

Hyperparameters. We use a 9:1 random split for training and validation, train all models for 100 epochs, and report metrics from the epoch achieving the lowest validation MSE. All reported NEU/FXE/INV metrics are computed on this validation split, and every method compared here (uniform LoRA, AdaLoRA, the LoRA variants, and both search baselines) is evaluated under the identical protocol. LoRA configurations employ rank values from 8 to 512, with alpha values equal to the rank and dropout rate of 0.05. Complete specifications are provided in Appendix A.11.

Search Configuration. We evaluate a 384-rank budget ($\beta = 384 \times L$, $[r_{\min}, r_{\max}] = [256, 512]$) and a 512-rank budget ($\beta = 512 \times L$, $[r_{\min}, r_{\max}] = [384, 640]$). The search objective $\mathcal{E}$ is

Table 2: MSE↓ of uniform LoRA and ours.

Method	NEU	FXE	INV
384-rank budget			
LoRA	0.0187	0.0203	0.0182
Ours (Group=3)	0.0183	0.0198	0.0177
Ours (Group=4)	0.0181	0.0196	0.0177
Ours (Group=6)	0.0179	0.0195	0.0176
Ours (Group=8)	**0.0175**	**0.0190**	**0.0170**
512-rank budget			
LoRA	0.0184	0.0197	0.0181
Ours (Group=3)	0.0180	0.0191	0.0176
Ours (Group=4)	0.0178	0.0190	0.0172
Ours (Group=6)	0.0175	0.0186	0.0171
Ours (Group=8)	**0.0172**	**0.0183**	**0.0168**

Table 3: MSE↓ of AdaLoRA and of our allocation with eight groups. AdaLoRA prunes ranks during training from an initial to a target rank (initial → target), and each of its results is a single training run. Best in bold.

Configuration	NEU	FXE	INV
AdaLoRA			
$640 \rightarrow 384$	0.0211	0.0220	0.0188
$640 \rightarrow 512$	0.0221	0.0233	0.0198
$512 \rightarrow 384$	0.0231	0.0240	0.0207
Ours (Group=8)			
384-rank budget	0.0175	0.0190	0.0170
512-rank budget	**0.0172**	**0.0183**	**0.0168**

validation MSE, and Phase I uses $\mathcal{G} = \{2, 3, 4, 6, 8, 12, 24\}$ and $\tau = 64$. More details are provided in Appendix A.12 and A.13.

Search Cost. Each candidate allocation is scored by a complete fine-tuning run, so one search instance consumes 332 evaluations, or 44 to 83 GPU-hours on a single RTX 3090Ti. This is a one-time offline cost per (dataset, budget) pair: the selected allocation is trained once and served at the same inference cost as uniform LoRA. Evaluation-count details are in Appendix A.21; per-evaluation timing and GPU-hour figures are in Appendix A.11.6.

5.2 Main Results

LoRA Rank Configuration Analysis. Table 1 presents LoRA rank configurations across three datasets. Lower rank configurations (8 to 64) underperform baseline with 6.5% to 14.3% average MSE degradation. Ranks 128 and 256 narrow this gap but still remain above the no-LoRA baseline on all three datasets (1.1% to 3.8%); only the largest tested ranks recover, with r=384 matching or beating the baseline on NEU and INV and r=512 the only configuration that improves on it across all three (1.1% to 2.6%). The superior performance at ranks 384 to 512 can be attributed to enhanced parameter space coverage and reduced low-rank approximation error, enabling more comprehensive task-specific adaptations (Hu et al., 2022; Zhang et al., 2023b). This performance pattern establishes the empirical foundation by identifying 384 and 512 as the best-performing uniform ranks; these serve as the target average ranks $\bar{r}$ on which our exploration framework's search ranges are centred ($\bar{r} \pm 128$; Appendix A.12), and they provide parameter bounds for comparative methods (e.g., AdaLoRA and LoRA variants).

Hierarchical Allocation Performance. Table 2 shows substantial improvements over uniform LoRA across all settings. With 8 groups, our method improves average MSE by 6.47% under the 384-rank budget and 6.94% under the 512-rank budget, where INV improves most at 7.18%; the two budgets average 6.70%. That the ordering holds across two budgets and three datasets indicates the search recovers layer heterogeneity rather than budget-specific noise. Details in Appendix A.15.

Comparison under a Matched Search Budget. Table 4 isolates the contribution of the search procedure itself by giving random search over per-layer rank vectors the same budget of 332 fine-tuning evaluations. Random search recovers part of the gain over uniform LoRA, lowering MSE by 2.5% to 2.8% across the three datasets, and our structured search improves on it by a further 4.38% MSE on average. The advantage therefore comes from how the search space is organized, not from the number of evaluations spent. Details in Appendix A.22.

Table 4: MSE↓ under a matched budget of 332 fine-tuning evaluations at the 512-rank budget. Best in bold.

Method	NEU	FXE	INV
Uniform LoRA (r=512)	0.0184	0.0197	0.0181
Random Search (332 evals)	0.0179	0.0192	0.0176
Ours (Group=8)	**0.0172**	**0.0183**	**0.0168**

Comparison with Accumulation-Based Strategy. Table 3 compares our hierarchical allocation strategy with AdaLoRA, a distribution-based approach that adaptively prunes ranks during training. Our method outperforms all AdaLoRA configurations, achieving 15.31% to 22.71% average MSE

Table 5: Comparison of LoRA variants under the 512-rank budget: QLoRA (Dettmers et al., 2023), LoRA+ (Hayou et al., 2024), DoRA (Liu et al., 2024a), and DyLoRA (Valipour et al., 2023). Best in bold.

Method	NEU Dataset			FXE Dataset			INV Dataset		
	MSE↓	MAE↓	RMSE↓	MSE↓	MAE↓	RMSE↓	MSE↓	MAE↓	RMSE↓
LoRA	0.0184	0.0955	0.1356	0.0197	0.1004	0.1404	0.0181	0.0952	0.1345
QLoRA	0.0191	0.0982	0.1382	0.0210	0.1035	0.1449	0.0187	0.0972	0.1367
LoRA+	0.0182	0.0952	0.1349	0.0196	0.1001	0.1400	0.0179	0.0947	0.1338
DoRA	0.0181	0.0948	0.1345	0.0195	0.0997	0.1396	0.0178	0.0944	0.1334
DyLoRA	0.0182	0.0953	0.1349	0.0197	0.1002	0.1404	0.0179	0.0950	0.1338
Ours	**0.0172**	**0.0928**	**0.1311**	**0.0183**	**0.0955**	**0.1353**	**0.0168**	**0.0905**	**0.1296**

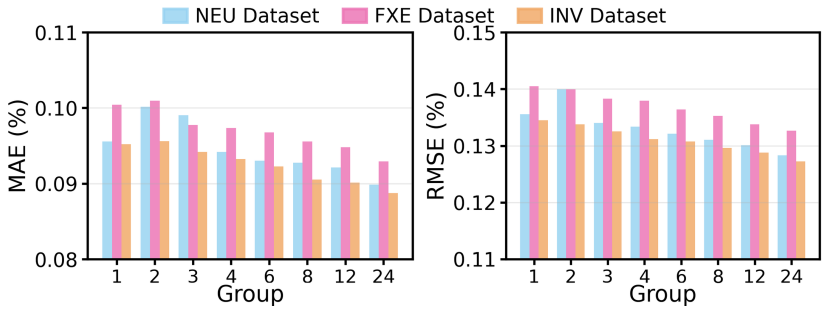


Figure 7: **Left:** MAE and **Right:** RMSE of our hierarchical allocation across grouping granularities at the 512-rank budget.

reduction depending on the AdaLoRA configuration. This suggests that structured hierarchical exploration provides more effective architecture-aware parameter allocation than sensitivity-driven adaptive methods relying on noisy gradient-based importance signals. The comparison is asymmetric in search compute: each AdaLoRA configuration is a single training run, whereas our allocation is selected by a 332-evaluation search, and the reported range spans the strongest to the weakest AdaLoRA configuration. Table 4 therefore provides the compute-matched test, which is the stricter one. Details in Appendix A.16.

Training Stability Analysis. Figure 6 compares training dynamics between AdaLoRA and our hierarchical allocation approach on the INV dataset. Both AdaLoRA configurations (640→384 and 640→512) start at high MSE (∼0.045) and oscillate before converging, while our method starts at ∼0.024 and descends smoothly. Pre-committing the rank distribution removes the mid-training rank reassignment that drives AdaLoRA's instability. Details in Appendix A.17.

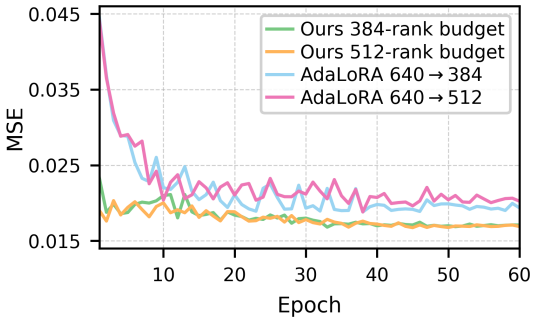


Figure 6: Training stability of AdaLoRA against our method at two rank budgets on INV.

Comparison with LoRA Variants. Our method leads Table 5 on every metric and dataset, including against DoRA and LoRA+. These methods enhance LoRA via weight decomposition or learning-rate adjustments but retain uniform rank allocation; our approach directly optimizes per-layer rank distribution, exploiting the hierarchical heterogeneity. DoRA is the strongest variant on all three datasets (MSE 0.0181, 0.0195, and 0.0178), and our allocation lowers MSE relative to it by 4.97%, 6.15%, and 5.62% on NEU, FXE, and INV. Details in Appendix A.18.

5.3 Ablation Studies

Effect of Layer Grouping. Figure 7 analyzes the impact of grouping granularity on our hierarchical allocation strategy under 512-rank budget. The results demonstrate consistent performance improvements as group count increases from 1 (standard uniform LoRA) to 24, with the finest granularity marking the accuracy ceiling of the family. We observe that Group=2 occasionally underperforms

Group=1, likely because coarse two-way partitioning creates suboptimal boundaries that fail to capture meaningful layer heterogeneity. These findings confirm that finer-grained rank allocation enables more precise parameter distribution aligned with layer-specific requirements. Phase I selects g^* by validation error alone, and error decreases monotonically with granularity here, so for the main results we fix $g{=}8$ as a cost-accuracy operating point: it trails $g{=}24$ by under 0.003 MAE while requiring one third of the Phase II evaluations (96 against 288 at $\tau{=}64$), and $g{=}L$ assigns an independent rank to every layer, dissolving the grouping that makes the search tractable.

Effect of Dataset Size. The table in Appendix A.20 (due to page limit) demonstrates that our framework exhibits data dependency across different scales. At 25% data, MSE increases by 50.8% (FXE) and 51.8% (INV), indicating insufficient samples for reliable rank optimization. The framework achieves stable improvements only at 75 to 100% data scales, showing that our hierarchical approach requires substantial data to distinguish between rank configurations.

Budget Efficiency. Our allocation under the 384-rank budget reaches MSE 0.0175, 0.0190, and 0.0170 on NEU, FXE, and INV, which is lower than uniform LoRA with rank 512 on every dataset (0.0184, 0.0197, and 0.0181) by 4.89%, 3.55%, and 6.08%. The allocation therefore reaches a lower error than a uniform configuration that uses one third more rank budget.

6 Related Work

Sentiment Analysis. Early studies employed traditional machine learning techniques for sentiment analysis, e.g., classifying insurance reviews (Hossain & Rahman, 2023) and aspect-based analysis with neutrosophic sets (Li et al., 2023). Deep learning advanced the field, enabling stock market forecasting (Liapis et al., 2023), multilingual e-commerce analysis (Savci & Das, 2023), and BERT-based hotel review classification (Wen et al., 2023). More recently, LLMs have emerged as powerful tools for financial sentiment analysis, with heterogeneous LLM agents (Xing, 2025) and preference optimization approaches (Iacovides et al., 2025) achieving state-of-the-art performance.

PEFT. PEFT methods evolved from early adapter-based approaches (Hu et al., 2023; Karimi Mahabadi et al., 2021). LoRA (Hu et al., 2022) introduced efficient low-rank matrix decomposition, spawning variants including LoraHub (Huang et al., 2023), LLaMA-Adapter (Zhang et al., 2024), and QLoRA (Dettmers et al., 2023). Recent advances focus on dynamic rank allocation, with nested LoRA (Li et al., 2025) and elastic rank adaptation (Chang et al., 2025) demonstrating that adaptive parameter distribution improves fine-tuning efficiency. Recent work also reduces computational overhead (Chen et al., 2025) and memory footprint (Zhang et al., 2023a; 2025).

Rank Allocation and Search. AdaLoRA (Zhang et al., 2023b) redistributes a global rank budget by pruning singular values during training, SoRA (Ding et al., 2023) learns a sparse set of active rank components with gates, and ALoRA (Liu et al., 2024b) reallocates ranks across modules from estimated rank importance. DyLoRA (Valipour et al., 2023) instead trains one adapter across a range of ranks so that a rank can be chosen after training. Selecting a configuration by measured validation performance is a search problem in which random search (Bergstra & Bengio, 2012) and regularized evolution (Real et al., 2019) are standard baselines. HiRank fixes the allocation before training and organizes this search around contiguous layer groups and per-group rank intervals.

7 Conclusion

Uniform rank allocation in LoRA ignores the hierarchical heterogeneity among transformer layers and wastes parameter budget. We propose HiRank, which decomposes rank optimization into coarse-grained joint exploration of layer groupings and rank intervals, followed by fine-grained rank assignment under the budget constraint. On three financial sentiment datasets it reduces MSE by 6.70% on average over uniform LoRA at an identical parameter budget. The reduction holds for every grouping and both budgets we evaluate, and at a matched budget of 332 fine-tuning evaluations the structured search lowers MSE by 4.38% relative to random search over per-layer rank vectors. The allocation is chosen once before training, so the fine-tuned model keeps the inference cost of uniform LoRA. Across four depth groups, the rank with the lowest measured MSE ranges from 128 to 384, which is the variation a single uniform rank cannot follow.

REFERENCES

D. M. Anisuzzaman, Jeffrey G. Malins, Paul A. Friedman, and Zachi I. Attia. Fine-tuning large language models for specialized use cases. *Mayo Clinic Proceedings: Digital Health*, 3(1):100184, 2025. doi: 10.1016/j.mcpdig.2024.11.005.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012. URL https://jmlr.org/papers/v13/bergstra12a.html.

Jianxin Bi. Stock market prediction based on financial news text mining and investor sentiment recognition. *Mathematical Problems in Engineering*, 2022(1):2427389, 2022.

A Bouteska, M Kabir Hassan, M Faisal Safa, et al. Riding the waves of investor sentiment: Cryptocurrency price and renewable energy volatility during the pandemic-war era. *Journal of Behavioral and Experimental Finance*, 44:101001, 2024.

Huandong Chang, Zicheng Ma, Mingyuan Ma, Zhenting Qi, Andrew Sabot, Hong Jiang, and HT Kung. Elalora: Elastic & learnable low-rank adaptation for efficient model fine-tuning. *arXiv preprint arXiv:2504.00254*, 2025.

Guanduo Chen, Yutong He, Yipeng Hu, Kun Yuan, and Binhang Yuan. Ce-lora: Computation-efficient lora fine-tuning for language models. *arXiv preprint arXiv:2502.01378*, 2025.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Hongcheng Ding, Xiangyu Shi, Ruiting Deng, Salaar Faroog, Deshinta Arrova Dewi, Shamsul Nahar Abdullah, and Bahiah A Malek. Eur/usd exchange rate forecasting incorporating text mining based on pre-trained language models and deep learning methods. *arXiv preprint arXiv:2411.07560*, 2024a.

Hongcheng Ding, Xuanze Zhao, Ruiting Deng, Shamsul Nahar Abdullah, and Deshinta Arrova Dewi. Eur-usd exchange rate forecasting based on information fusion with large language models and deep learning methods. *arXiv preprint arXiv:2408.13214*, 2024b.

Hongcheng Ding, Xuanze Zhao, Ruiting Deng, Shamsul Nahar Abdullah, Deshinta Arrova Dewi, and Zixiao Jiang. Dynamic adaptive optimization for effective sentiment analysis fine-tuning on large language models. *arXiv preprint arXiv:2408.11856*, 2024c.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. Sparse low-rank adaptation of pre-trained language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Mor Geva, Avi Caciularu, Kevin Ro Wang, and Yoav Goldberg. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 30–45, 2022.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. LoRA+: Efficient low rank adaptation of large models. In *International Conference on Machine Learning*, 2024.

Md Shamim Hossain and Mst Farjana Rahman. Customer sentiment analysis and prediction of insurance products' reviews using machine learning approaches. *FIIB Business Review*, 12(4): 386–402, 2023.

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. In *International Conference on Learning Representations*, 2022.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. LLM-Adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. doi: 10.18653/v1/2023.emnlp-main.319.

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.

Giorgos Iacovides, Wuyang Zhou, and Danilo Mandic. Findpo: Financial sentiment analysis for algorithmic trading through preference optimization of llms. In *Proceedings of the 6th ACM International Conference on AI in Finance*, pp. 647–655, 2025.

Abhinav Jain, Swarat Chaudhuri, Thomas Reps, and Chris Jermaine. Prompt tuning strikes back: Customizing foundation models with low-rank prompt adaptation. *Advances in Neural Information Processing Systems*, 37:47297–47316, 2024.

Ting Jiang, Shaohan Huang, Shengyue Luo, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, Qi Zhang, Deqing Wang, et al. Mora: High-rank updating for parameter-efficient fine-tuning. *arXiv preprint arXiv:2405.12130*, 2024.

Mingyu Jin, Qinkai Yu, Jingyuan Huang, Qingcheng Zeng, Zhenting Wang, Wenyue Hua, Haiyan Zhao, Kai Mei, Yanda Meng, Kaize Ding, et al. Exploring concept depth: How large language models acquire knowledge and concept at different layers? In *Proceedings of the 31st international conference on computational linguistics*, pp. 558–573, 2025.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035, 2021.

Minh Le, Chau Nguyen, Huy Nguyen, Quyen Tran, Trung Le, and Nhat Ho. Revisiting prefix-tuning: Statistical benefits of reparameterization among prompts. In *International Conference on Learning Representations*, 2025.

Jean Lee, Nicholas Stevens, and Soyeon Caren Han. Large language models in finance (finllms). *Neural Computing and Applications*, 37(30):24853–24867, 2025.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 367–377, 2019.

Lujun Li, Cheng Lin, Dezhi Li, You-Liang Huang, Wei Li, Tianyu Wu, Jie Zou, Wei Xue, Sirui Han, and Yike Guo. Efficient fine-tuning of large models via nested low-rank adaptation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 22252–22262, 2025.

Xihua Li, Yun Luo, Hui Wang, Jiong Lin, and Bin Deng. Doctor selection based on aspect-based sentiment analysis and neutrosophic TOPSIS method. *Engineering Applications of Artificial Intelligence*, 124:106599, 2023. doi: 10.1016/j.engappai.2023.106599.

Charalampos M Liapis, Aikaterini Karanikola, and Sotiris Kotsiantis. Investigating deep stock market forecasting with sentiment analysis. *Entropy*, 25(2):219, 2023.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM computing surveys*, 55(9):1–35, 2023.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *International Conference on Machine Learning*, 2024a.

Zequan Liu, Jiawen Lyn, Wei Zhu, Xing Tian, and Yvette Graham. ALoRA: Allocating low-rank adaptation for fine-tuning large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2024b.

Llama Team. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024.

Daniel Loureiro, Kiamehr Rezaee, Talayeh Riahi, Francesco Barbieri, Leonardo Neves, Luis Espinosa Anke, and Jose Camacho-Collados. Tweet insights: A visualization platform to extract temporal insights from twitter. *arXiv preprint arXiv:2308.02142*, 2023.

Muhammad Mujahid, EROL Kına, Furqan Rustam, Monica Gracia Villar, Eduardo Silva Alvarado, Isabel De La Torre Diez, and Imran Ashraf. Data oversampling and imbalanced datasets: an investigation of performance for machine learning and feature engineering. *Journal of Big Data*, 11(1):87, 2024.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4780–4789, 2019. doi: 10.1609/aaai.v33i01.33014780.

Pinar Savci and Bihter Das. Prediction of the customers' interests using sentiment analysis in e-commerce data for comparison of arabic, english, and turkish languages. *Journal of King Saud University-Computer and Information Sciences*, 35(3):227–237, 2023.

Oscar Skean, Md Rifat Arefin, Dan Zhao, Niket Patel, Jalal Naghiyev, Yann LeCun, and Ravid Shwartz-Ziv. Layer by layer: Uncovering hidden representations in language models. In *International Conference on Machine Learning*, 2025.

Xinyuan Song, Keyu Wang, PengXiang Li, Lu Yin, and Shiwei Liu. Demystifying the roles of llm layers in retrieval, knowledge, and reasoning. *arXiv preprint arXiv:2510.02091*, 2025.

Jan-Martin O Steitz and Stefan Roth. Adapters strike back. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 23449–23459, 2024.

Mojtaba Valipour et al. Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, 2023.

Luping Wang, Sheng Chen, Linnan Jiang, Shu Pan, Runze Cai, Sen Yang, and Fei Yang. Parameter-efficient fine-tuning in large language models: a survey of methodologies. *Artificial Intelligence Review*, 58(8):227, 2025.

Yu Wen, Yezhang Liang, and Xinhua Zhu. Sentiment analysis of hotel online reviews using the bert model and ernie model—data from china. *Plos one*, 18(3):e0275382, 2023.

Frank Xing. Designing heterogeneous llm agents for financial sentiment analysis. *ACM Transactions on Management Information Systems*, 16(1):1–24, 2025.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Yang You, Guiming Xie, Xuejian Gong, and Kunlong Zhou. Train small, infer large: Memory-efficient LoRA training for large language models. In *International Conference on Learning Representations*, 2025.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*, 2023a.

Qingru Zhang et al. Adaptive budget allocation for parameter-efficient fine-tuning. *International Conference on Learning Representations*, 2023b.

Renrui Zhang, Jiaming Han, Chris Liu, Aojun Zhou, Pan Lu, Yu Qiao, Hongsheng Li, and Peng Gao. Llama-adapter: Efficient fine-tuning of large language models with zero-initialized attention. In *The Twelfth International Conference on Learning Representations*, 2024.

Zhen-Ru Zhang, Chuanqi Tan, Haiyang Xu, Chengyu Wang, Jun Huang, and Songfang Huang. Towards adaptive prefix tuning for parameter-efficient language model fine-tuning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1239–1248, Toronto, Canada, 2023c. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-short.107. URL https://aclanthology.org/2023.acl-short.107/.

AI USE STATEMENT

The authors used ChatGPT and Claude to check grammar and polish the English of author-written text, and Gemini to refine the visual presentation of figures.

REPRODUCIBILITY STATEMENT

The details needed to reproduce the results of this paper, including the search settings, the search procedure, the evaluation metrics, and the software setup, are given in this appendix.

APPENDIX

The sections below give the limitations, the extended related work, and the measurements, derivations, and baseline details behind the main paper.

TABLE OF CONTENTS

A.0 Limitations	13
A.1 Extended Related Work	14
A.2 Hierarchical Heterogeneity	14
A.3 Rank-Performance Trade-off	15
A.4 LoRA Background	17
A.5 Problem Formulation Details	18
A.6 Performance-Based Exploration Framework Details	19
A.7 Accumulation-Based Strategy	20
A.8 Dataset	21
A.9 Backbone Model	22
A.10 Evaluation Metrics	22
A.11 Hyperparameters	23
A.12 Rank Budget Configuration Rationale	25
A.13 Solution Space Analysis	26
A.14 LoRA Rank Configuration Analysis	26
A.15 Hierarchical Allocation Performance Analysis	27
A.16 Comparison with Accumulation-Based Strategy	29
A.17 Training Stability Analysis	30
A.18 Comparison with LoRA Variants	31
A.19 Effect of Layer Grouping	32
A.20 Effect of Dataset Size	33
A.21 Algorithm Analysis	34
A.22 Random-Search Baseline	36

APPENDIX A.0: LIMITATIONS

Tasks and Backbones. The allocation that HiRank selects reflects how rank sensitivity varies across the layers of one backbone on one task, and this variation can differ for classification, generation, and other model families. Evaluations on further tasks and backbones, including decoder LLMs, would show how far the allocation strategy transfers. The experiments in this work cover sentiment regression on three financial datasets with one encoder backbone.

Other Rank-Allocation Methods. Several methods reallocate LoRA rank during training from learned importance signals, such as ALoRA (Liu et al., 2024b) and SoRA (Ding et al., 2023), and a direct comparison with them would place HiRank within this family. The comparisons in this work cover uniform LoRA, AdaLoRA, and the LoRA variants QLoRA, LoRA+, DoRA, and DyLoRA.

Stronger Search Baselines. Random search at a matched number of fine-tuning evaluations separates the effect of the search structure from the effect of the search budget, and population-based methods such as regularized evolution (Real et al., 2019) are stronger baselines for this test. This work compares HiRank with random search.

APPENDIX A.1: EXTENDED RELATED WORK

Rank Allocation for LoRA. LoRA (Hu et al., 2022) assigns the same rank to every adapted matrix. AdaLoRA (Zhang et al., 2023b) parameterizes each update in SVD form and prunes singular values during training according to an importance score, redistributing a global budget across matrices. SoRA (Ding et al., 2023) places a gate on each rank component and learns a sparse set of active components, and ALoRA (Liu et al., 2024b) estimates the importance of individual ranks and reallocates them across modules during training. DyLoRA (Valipour et al., 2023) trains an adapter over a range of ranks so that a rank can be chosen after training, and work on dynamic rank allocation includes nested LoRA (Li et al., 2025) and elastic rank adaptation (Chang et al., 2025). HiRank fixes the allocation before training and scores each candidate allocation by the validation error of a complete fine-tuning run.

Variants of the LoRA Update. Other variants keep the rank fixed and change how the update is stored or trained. QLoRA (Dettmers et al., 2023) fine-tunes adapters on a 4-bit quantized backbone, LoRA+ (Hayou et al., 2024) uses different learning rates for the two factors, and DoRA (Liu et al., 2024a) decomposes each weight into magnitude and direction. LoraHub (Huang et al., 2023) composes LoRA modules across tasks, and LLaMA-Adapter (Zhang et al., 2024) adapts a frozen LLM with learnable adaption prompts. These methods are complementary to the choice of per-layer ranks.

Search over Training Configurations. Choosing ranks by measured validation performance is a search in which every evaluation is a fine-tuning run. Random search (Bergstra & Bengio, 2012) is a strong baseline for such searches and remains competitive in architecture search (Li & Talwalkar, 2019), and regularized evolution (Real et al., 2019) improves on it in image-classifier architecture search. HiRank organizes the search around the layer structure of the model: it narrows a rank interval for each contiguous layer group and then assigns ranks within the budget.

APPENDIX A.2: HIERARCHICAL HETEROGENEITY

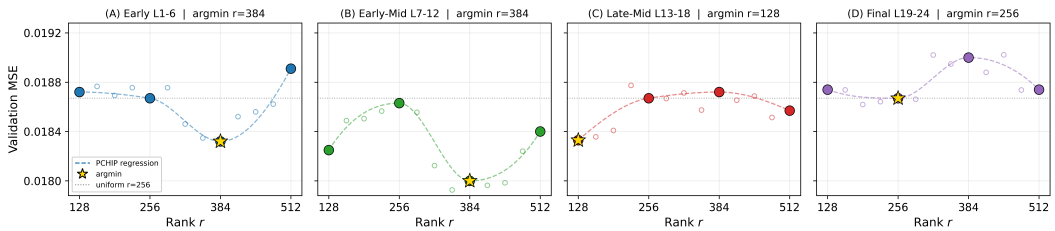


Figure 8: Layer-wise rank sensitivity for groups (A) to (D). Filled markers: measured MSE at $r \in \{128, 256, 384, 512\}$ with the other three groups held at r=256. Dashed curves: PCHIP regression fits through the four measurements per panel; small open markers visualize the regression at intermediate ranks. Gold stars: empirical argmins. Gray dashed line: uniform-r=256 baseline. Argmins differ across groups, $r^\star = (384, 384, 128, 256)$, revealing heterogeneous, non-monotone sensitivity.

Experimental Setup. We configure a baseline LLM with LoRA rank 256 across all layers and divide the transformer into four equal groups based on layer depth. The first group (A) represents early layers that capture low-level features, the second group (B) encompasses early-middle layers handling intermediate representations, the third group (C) contains late-middle layers responsible for high-level semantic processing, and the final group (D) includes the last layers performing task-specific adaptations. For each group, we systematically vary the rank values to 128, 256, 384, and 512 while keeping other groups at the baseline rank of 256. Per-group results are shown as panels (A) to (D) of Figure 8; the gray dashed line in each panel marks the uniform-r=256 baseline. For our 24-layer backbone, this yields Group (A): layers 1 to 6, Group (B): layers 7 to 12, Group (C): layers 13 to 18, and Group (D): layers 19 to 24.

Group (A) - Early Layers. Figure 8, panel (A), shows that early layers achieve optimal performance at r=384, with r=512 exhibiting clear degradation and r=128 landing slightly above baseline. These layers handle syntactic processing and low-level feature extraction, benefiting from moderate added capacity while suffering from over-parameterization at the upper end. The over-allocation penalty for Group A reaches 3.16% (Figure 9, left).

Group (B) - Early-Middle Layers. Figure 8, panel (B), exhibits the deepest dip among the four groups, with the optimum at r=384. MSE decreases substantially from r=128 to r=384, then degrades again at r=512, indicating diminishing returns past the optimum. These layers bridge low-level features and high-level semantics, requiring sufficient adaptation capacity yet vulnerable to over-parameterization. Group B's over-allocation penalty is 3.37%, the largest among the four groups (Figure 9, left).

Group (C) - Late-Middle Layers. Figure 8, panel (C), reveals the most counter-intuitive pattern: the optimum sits at the smallest tested rank (r=128), with monotonic degradation through r=384. Late-middle layers handle complex semantic composition and appear to be already well-served by pre-trained representations, so additional LoRA capacity primarily increases overfitting risk. Group C's over-allocation penalty is 2.09% (Figure 9, left), and this group is the central target for validation-driven budget redistribution.

Group (D) - Final Layers. Figure 8, panel (D), shows that Group D is rank-saturated at the uniform baseline: r=256 matches the baseline by construction, and no other tested rank improves over it; r=384 in particular yields the worst MSE among the four configurations. This indicates a specific capacity requirement well-served by the uniform-256 setting, with both insufficient (r=128) and excessive (r=384) rank being detrimental. Group D's over-allocation penalty is 1.77% (Figure 9, left).

Summary of Heterogeneity Patterns. The empirical findings reveal four distinct sensitivity patterns across layer groups: Group (A) and (B) favor higher ranks around 384, Group (C) performs best at the lowest rank 128, and Group (D) requires precise calibration at rank 256. The four groups therefore have different adaptation requirements, so a single uniform rank cannot serve all four groups: the optimal rank for one group is often harmful for another.

APPENDIX A.3: RANK-PERFORMANCE TRADE-OFF

Optimal Rank Range. Our empirical observations reveal that the relationship between rank and performance follows a pattern where performance initially improves with increasing rank, reaches an optimal point, and then degrades with further rank increases. This trade-off arises from the competing effects of model expressiveness and overfitting risk, suggesting that each layer group has an optimal rank range rather than benefiting from arbitrarily high ranks.

Evidence from Group (C). Group (C) in Figure 8, panel (C), shows the effect most clearly. As rank increases from r=128 to r=384, MSE increases from 0.01833 to 0.01872, representing a performance degradation of approximately 2.1%. This observation directly contradicts the intuition that higher ranks always improve performance. The degradation indicates that excessive parameterization in late-middle layers leads to overfitting, where the model memorizes training data patterns rather than learning generalizable representations.

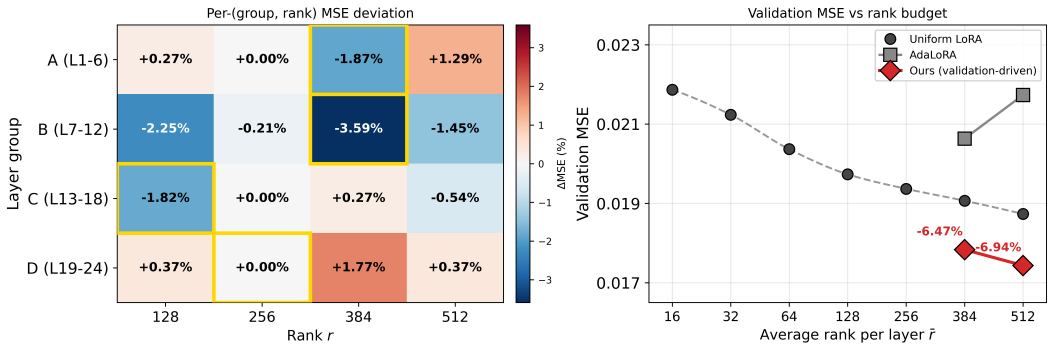


Figure 9: **Left:** Per-(group, rank) MSE deviation from the uniform-r=256 baseline, computed from the 16 measurements in Figure 8. Each cell reports $(\text{MSE} - \text{baseline})/\text{baseline} \times 100\%$; gold border marks the per-group argmin. The heterogeneity is visible cell by cell: only Group B has all four cells below baseline; Group C's argmin sits at r=128 (the smallest rank tested); Group D's argmin coincides with the baseline by construction. **Right:** MSE vs average rank per layer $\bar{r}$ (log scale), averaged across NEU/FXE/INV. Eleven measurements: Uniform LoRA's seven-point sweep $r \in \{16, \ldots, 512\}$ with PCHIP regression (dashed); AdaLoRA at the 384- and 512-rank budgets; Ours at the same two budgets. Validation-driven hierarchical search reduces MSE by 6.47% at the 384-rank budget and 6.94% at the 512-rank budget (avg 6.70%) over Uniform LoRA, and consistently beats AdaLoRA (15.31% to 22.71% depending on the AdaLoRA configuration; Table 3 of the main paper).

Evidence from Group (D). Group (D) exhibits a similar pattern with optimal performance at rank 256. Both lower ranks (128) and higher ranks (384, 512) show worse performance than the optimal point. This pattern suggests that final layers have a specific capacity requirement: insufficient rank fails to capture task-specific adaptations, while excessive rank introduces harmful overfitting. The symmetric degradation around the optimal point indicates a delicate balance between underfitting and overfitting in task-specific layers.

Evidence from Groups (A) and (B). Even groups that generally benefit from higher ranks exhibit performance degradation at the upper end. Groups (A) and (B) show improved performance from rank 128 to 384, but performance degrades at rank 512. This pattern suggests that all layer groups have an upper bound on beneficial rank allocation, beyond which additional parameters become counterproductive.

Theoretical Explanation. The rank-performance trade-off can be understood through the bias-variance decomposition. Low-rank configurations exhibit high bias (underfitting) due to insufficient model capacity, while high-rank configurations exhibit high variance (overfitting) due to excessive model complexity. The optimal rank represents the point where the combined error from bias and variance is minimized. This optimal point varies across layer groups due to their different functional roles and pre-trained representations, forming layer-specific optimal rank ranges.

Implications for Rank Allocation. Optimal rank ranges constrain how a budget can be allocated. First, it invalidates simple heuristics such as "maximize rank within budget" or "allocate rank proportionally to layer size." Second, it necessitates empirical exploration to identify the optimal rank range for each layer group, as the optimal point cannot be determined analytically. Third, it complicates budget-constrained optimization, as allocating rank to one layer group affects the budget available for others, potentially pushing multiple groups away from their optimal ranges. These implications motivate the performance-based exploration framework proposed in our methodology.

Connection to Bias-Variance Trade-off. Our observations align with the classical bias-variance trade-off in machine learning, where model performance depends on balancing underfitting and overfitting. In the context of LoRA fine-tuning, rank serves as a proxy for model complexity. Lower ranks may underfit by limiting adaptation capacity, while higher ranks may overfit by introducing

excessive parameters. This connection provides theoretical grounding for our empirical observations and suggests that layer-specific rank optimization within appropriate ranges is essential for achieving optimal fine-tuning performance.

APPENDIX A.4: LORA BACKGROUND

LoRA Decomposition Mechanism. Low-Rank Adaptation (LoRA) replaces full-parameter fine-tuning with trainable low-rank matrices that approximate the parameter update required for task adaptation. The core principle relies on the hypothesis that the intrinsic rank of task-specific parameter changes is much lower than the full parameter space dimensionality, enabling efficient adaptation through low-rank decomposition.

For a pre-trained weight matrix $W^0 \in \mathbb{R}^{m\times n}$, the adapted weight is expressed as:

$$W = W^0 + \Delta W = W^0 + UV^T, \tag{6}$$

where $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{n\times r}$ are trainable low-rank factors, and $r \ll \min(m, n)$ is the rank. The number of trainable parameters introduced by LoRA for this matrix is $(m + n) \times r$, which is significantly smaller than $m \times n$ required for full fine-tuning.

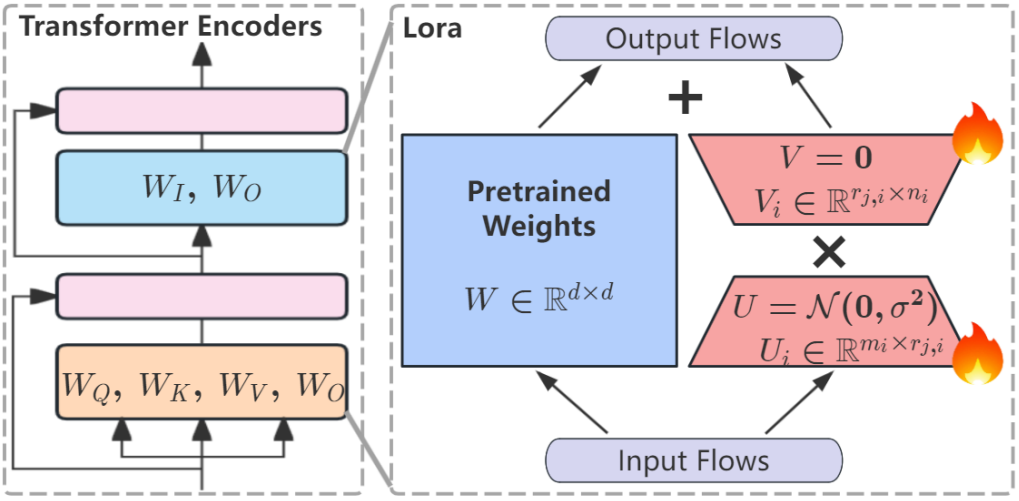


Figure 10: LoRA Adaptation in Transformer Architecture. **Left:** Transformer encoder structure with adaptable weight matrices. **Right:** LoRA decomposition mechanism showing frozen pre-trained weights (blue) and trainable low-rank factors (indicated by flame symbols).

LoRA in Transformer Architecture. Figure 10 illustrates the comprehensive application of LoRA across transformer encoder layers. The left side shows the transformer encoder structure with adaptable weight matrices, while the right side details the LoRA decomposition mechanism. The transformer encoder contains multiple adaptable components including the attention mechanism matrices W_Q, W_K, W_V, W_O and feed-forward network matrices W_{up}, W_{down}, all of which can be incorporated into the set $\mathcal{W}_i$ for layer i.

Initialization Strategy. Each adaptable weight matrix $W \in \mathbb{R}^{m\times n}$ undergoes LoRA decomposition where the original pre-trained weights remain frozen (shown in blue in Figure 10) while two trainable low-rank matrices $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{n\times r}$ are introduced (indicated by flame symbols). The matrix V is initialized to zero to ensure that the adaptation starts from the original pre-trained model behavior, while U is initialized using Gaussian distribution $\mathcal{N}(0, \sigma^2)$. This initialization strategy ensures that at the beginning of fine-tuning, $\Delta W = UV^T = 0$, preserving the pre-trained model's behavior.

Adaptable Components. The set of weight matrices $\mathcal{W}_i$ in our problem formulation represents all adaptable parameters in layer i, which varies across different transformer architectures:

- **Attention Modules**: Query projection (W_Q), Key projection (W_K), Value projection (W_V), and Output projection (W_O). These matrices are central to the self-attention mechanism and demonstrate high adaptation potential due to their role in processing contextual relationships.
- **Feed-Forward Networks**: Up-projection (W_{up}) and Down-projection (W_{down}) matrices. These provide additional capacity for task-specific feature transformations.

Parameter Efficiency Analysis. The rank parameter r directly controls the representational capacity of each adapted component. For a single weight matrix with dimensions $m \times n$, LoRA introduces $(m + n) \times r$ trainable parameters compared to $m \times n$ for full fine-tuning. The compression ratio is:

$$\text{Compression Ratio} = \frac{(m+n) \times r}{m \times n} = \frac{r(m+n)}{mn}. \tag{7}$$

For typical transformer dimensions where $m = n = d_{\text{model}}$, this simplifies to $\frac{2r}{d_{\text{model}}}$. With $d_{\text{model}} = 1024$ and $r = 8$, the compression ratio is approximately 1.6%, demonstrating the significant parameter efficiency of LoRA.

Scaling Factor. In practice, LoRA applies a scaling factor α/r to the low-rank update:

$$W = W^0 + \frac{\alpha}{r} UV^T, \tag{8}$$

where α is a hyperparameter that controls the magnitude of the adaptation. We set $\alpha = r$ throughout, so the scaling factor is 1. This scaling factor allows practitioners to adjust the adaptation strength without modifying the learning rate.

Advantages of LoRA. LoRA offers several advantages over traditional fine-tuning approaches: (1) **Parameter Efficiency**: Only a small fraction of parameters are trainable, reducing memory requirements and enabling fine-tuning on consumer hardware. (2) **No Inference Latency**: The low-rank matrices can be merged with pre-trained weights after training, introducing no additional computation during inference. (3) **Task Switching**: Multiple LoRA adapters can be trained for different tasks and switched at inference time by simply loading different low-rank matrices. (4) **Preservation of Pre-trained Knowledge**: By keeping pre-trained weights frozen, LoRA reduces the risk of catastrophic forgetting.

APPENDIX A.5: PROBLEM FORMULATION DETAILS

Model Configuration. Consider a pre-trained LLM with L Transformer layers $\mathcal{T} = \{T_1, T_2, \ldots, T_L\}$. We use $\mathcal{W}_i$ to represent the set of weight matrices in layer i that are subject to LoRA adaptation. The specific composition depends on the transformer architecture, encompassing attention modules (Query, Key, Value, Output projections) and optionally feed-forward layers. Here, m_i and n_i represent the output and input dimensions of the i-th layer's adaptable matrices, respectively.

LoRA Decomposition. Using rank r_i, each LoRA-adapted weight matrix $W_i \in \mathcal{W}_i$ in layer i is approximated as:

$$W_i \approx W_i^0 + U_i V_i^T, \tag{9}$$

where $U_i \in \mathbb{R}^{m_i \times r_i}$ and $V_i \in \mathbb{R}^{n_i \times r_i}$ are trainable low-rank factors.

Rank Allocation Vector. Let $\mathbf{r} = [r_1, r_2, \ldots, r_L] \in \mathbb{N}^L$ denote the rank allocation vector, where r_i is the rank assigned to layer i. The feasible rank space is defined as:

$$\mathcal{S}_{\mathcal{R}} = \{\mathbf{r} \in \mathbb{N}^L : r_i \in \mathcal{R}, \forall i \in \{1, \ldots, L\}\}, \tag{10}$$

where $\mathcal{R} = \{r_{\min}, r_{\min} + 1, \ldots, r_{\max}\}$ is the candidate rank set and $r_{\max}$ is the maximum allowed rank per layer.

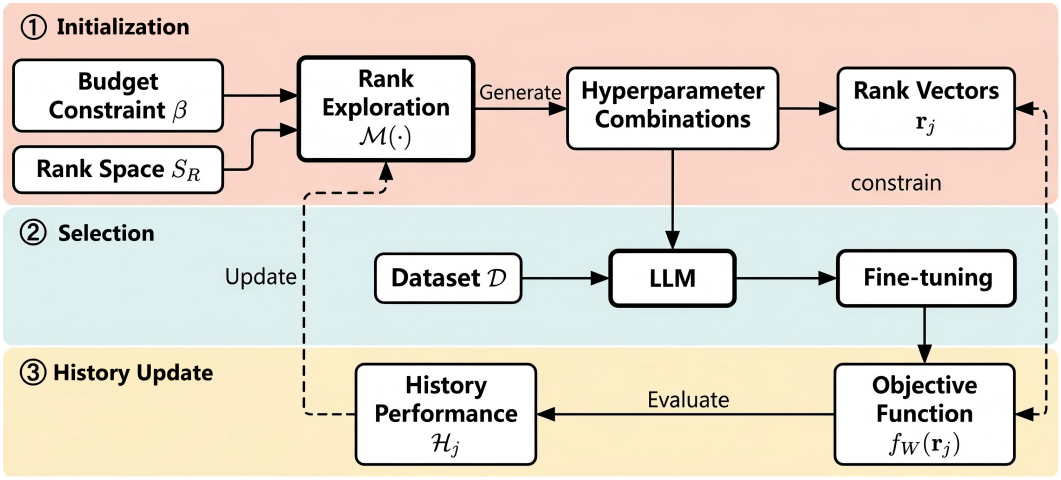


Figure 11: Performance-based exploration framework: Initialization (pink) generates a candidate rank vector, Selection (light blue) fine-tunes the model under that configuration, and History Update (light yellow) evaluates it and returns $\mathcal{H}_j$ to guide the next candidate.

Optimization Objective. The objective is to find an optimal rank allocation $\mathbf{r}^*$ that minimizes the task-specific loss subject to a total rank budget β:

$$\mathbf{r}^* = \arg\min_{\mathbf{r}\in\mathcal{S}_\mathcal{R}} \mathcal{E}(f_\mathbf{r}; \mathcal{D}) \quad \text{s.t.} \quad \sum_{i=1}^{L} r_i \leq \beta, \tag{11}$$

where $\mathcal{E}(\cdot)$ denotes the evaluation metric (MSE for regression tasks), $f_\mathbf{r}$ represents the model fine-tuned under rank configuration $\mathbf{r}$ (equivalently denoted as $f_{\mathcal{W}(\mathbf{r})}$ to emphasize the dependency on adapted weights), and $\mathcal{D}$ is the downstream task dataset.

Budget Constraint Explanation. The constraint $\sum_{i=1}^{L} r_i \leq \beta$ ensures that the total rank allocation does not exceed the budget β. This rank budget corresponds to a parameter budget of $\beta \times \omega$, where $\omega = (m+n) \times |\mathcal{W}_i|$ is the parameter scaling factor per rank unit. For architectures with uniform layer dimensions, this simplification maintains mathematical equivalence while enabling direct rank comparisons.

Search Space Complexity. The size of the rank space, before the budget constraint is imposed, is $|\mathcal{S}_\mathcal{R}| = (r_{\max} - r_{\min} + 1)^L$. For our 512-rank budget configuration with $L = 24$ layers, $r_{\min} = 384$ and $r_{\max} = 640$, this yields approximately 10^{58} possible configurations, making exhaustive search computationally infeasible and motivating efficient search strategies.

Appendix A.6: Performance-Based Exploration Framework Details

Figure 11 illustrates the performance-based exploration framework. We describe the role of each component and the transition to the hierarchical allocation strategy.

A.6.1 Component Description

As shown in Figure 11, the framework consists of the following components:

Input Components:

- **Rank Space** $\mathcal{S}_\mathcal{R}$: The complete set of possible rank vectors, where each vector $\mathbf{r} \in \mathbb{N}^L$ assigns a rank value to each of the L transformer layers.
- **Budget Constraint** β: The maximum total rank budget across all layers, ensuring $\sum_{i=1}^{L} r_i \leq \beta$.

Core Modules:

- **Rank Exploration**: Generates candidate rank vectors $\mathbf{r}_j$ guided by the exploration module $\mathcal{M}(\cdot)$.
- **Exploration Strategy**: Determines how to select the next rank configuration based on historical performance using $\mathcal{H}_j$ in Figure 4 of the main paper.
- **Hyperparameter Combinations**: Combines the selected rank vector with other fine-tuning hyperparameters (e.g., learning rate, batch size) to form a complete training configuration.
- **Objective Function**: Computes the evaluation metric $\mathcal{E}(\cdot)$ (MSE in our experiments) on the validation set $\mathcal{D}_{\text{val}}$.
- **History Performance** $\mathcal{H}_j$: Stores all evaluated configurations and their corresponding performance, enabling performance-driven selection in subsequent iterations.

A.6.2 Workflow Description

The workflow in Figure 11 proceeds as follows:

Step 1: Rank Selection. The exploration module $\mathcal{M}$ selects a rank vector $\mathbf{r}_j$ from the rank space $\mathcal{S}_{\mathcal{R}}$. In the initialization phase, this selection follows a predefined strategy (e.g., midpoint of the rank range). In subsequent iterations, the selection is guided by historical performance data $\mathcal{H}_j$ through the exploration strategy.

Step 2: Configuration Assembly. The selected rank vector $\mathbf{r}_j$ is combined with other hyperparameters to form a complete configuration for LLM fine-tuning.

Step 3: Fine-tuning. The LLM is fine-tuned on the training set $\mathcal{D}_{\text{train}}$ using the assembled configuration, producing a fine-tuned model $f_{\mathcal{W}(\mathbf{r}_j)}$.

Step 4: Evaluation. The fine-tuned model is evaluated on the validation set $\mathcal{D}_{\text{val}}$ using the objective function $\mathcal{E}(\cdot)$, yielding the performance score $\mathcal{E}_j$.

Step 5: History Update. The configuration-performance pair $(\mathbf{r}_j, \mathcal{E}_j)$ is stored in the history repository, updating $\mathcal{H}_{j+1} = \mathcal{H}_j \cup \{(\mathbf{r}_j, \mathcal{E}_j)\}$.

Step 6: Iteration Decision. The framework checks convergence criteria. If not converged, $\mathcal{H}_{j+1}$ is leveraged to guide the next rank selection, returning to Step 1.

A.6.3 Transition to Hierarchical Allocation Strategy

The performance-based exploration framework provides a general mechanism for navigating the rank configuration space. The hierarchical allocation strategy (Section 4.2) instantiates this framework by specifying concrete implementations for the exploration module $\mathcal{M}(\cdot)$:

Phase I (Coarse-Grained Exploration): The exploration module implements Algorithm 1 and Algorithm 2. The rank space $\mathcal{S}_{\mathcal{R}}$ is reduced by grouping layers, and binary greedy interval reduction iteratively narrows the promising rank intervals. Each interval reduction step corresponds to one iteration of the framework, where the history $\mathcal{H}_j$ guides the decision of which half-interval to retain.

Phase II (Fine-Grained Optimization): The exploration module implements Algorithm 3 and Algorithm 4. Starting from the refined intervals $\mathcal{I}^*$ obtained in Phase I, binary greedy rank search pinpoints the selected rank value for each group. The budget constraint β is enforced through dynamic upper bound computation u_{eff}, ensuring the final allocation satisfies resource limitations.

The two-phase instantiation reduces the number of required evaluations from exponential in L to logarithmic in the interval width, at the cost of restricting the search to grouped, greedily halved configurations.

Appendix A.7: Accumulation-Based Strategy

The accumulation-based rank allocation strategy draws on AdaLoRA (Zhang et al., 2023b) and serves as a baseline comparison method in our experiments.

SENSITIVITY ACCUMULATION

The accumulated importance score S_i^{acc} computes layer-wise importance by accumulating gradient-based sensitivity measurements during training:

$$S_i^{\text{acc}} = \frac{1}{T} \sum_{t=1}^{T} \sum_{\mathbf{W} \in \mathcal{W}_i} \|\mathbf{W} \odot \nabla_{\mathbf{W}} \mathcal{L}^{(t)}\|_F, \tag{12}$$

where T is the total number of training steps, $\mathcal{W}_i$ denotes the set of all weight matrices in layer i (including Query, Key, Value, and Output projections), $\mathcal{L}^{(t)}$ is the loss at training step t, $\odot$ denotes element-wise multiplication, and $\|\cdot\|_F$ denotes the Frobenius norm.

The term $\mathbf{W} \odot \nabla_{\mathbf{W}} \mathcal{L}$ measures the sensitivity of the loss with respect to each weight, scaled by the weight magnitude. This product captures how much each parameter contributes to the loss change, with larger values indicating that the parameter has greater influence on model performance. Accumulating this metric over training steps provides a robust estimate of each layer's overall importance for the fine-tuning task.

PROPORTIONAL BUDGET ALLOCATION

Given the accumulated importance scores, the rank allocation proportionally distributes the total budget β across layers:

$$r_i = \max\left(1, \left\lfloor \frac{S_i^{\text{acc}}}{\sum_{k=1}^{L} S_k^{\text{acc}}} \cdot \beta \right\rfloor\right), \tag{13}$$

where $\frac{S_i^{\text{acc}}}{\sum_{k=1}^{L} S_k^{\text{acc}}}$ computes the normalized importance proportion for layer i, and the floor operation ensures discrete rank values. The $\max(\cdot, 1)$ constraint guarantees that every layer retains at least rank 1 for minimal adaptation capacity. This allocation strategy assigns higher ranks to layers with greater accumulated sensitivity, under the assumption that more important layers benefit from increased representational capacity.

APPENDIX A.8: DATASET

A.8.1 FINANCIAL TEXT DATASET

Financial text is used for sentiment analysis alongside quantitative indicators. The integration of unstructured textual data with structured financial indicators has proven particularly effective in exchange rate forecasting (Ding et al., 2024a), stock market trend prediction (Bi, 2022), and cryptocurrency price analysis (Bouteska et al., 2024). However, financial texts present unique challenges including domain-specific terminology, implicit sentiments, and inherent data distribution imbalance where neutral sentiments dominate while extreme sentiments remain underrepresented (Mujahid et al., 2024).

A.8.2 NEU Dataset. The NEU dataset (Ding et al., 2024b) covers EUR/USD exchange-rate news articles and analytical content. Its negative mean score reflects the sentiment bias characteristic of financial news discourse.

A.8.3 FXE Dataset. The FXE dataset (Ding et al., 2024c) covers preprocessed EUR/USD news and analysis texts. Its near-neutral mean score indicates a relatively balanced sentiment distribution.

A.8.4 INV Dataset. The INV dataset (Ding et al., 2024a) is the largest of the three and captures market sentiment towards exchange-rate fluctuations.

A.8.5 DATASET SUMMARY

Table 6 provides a comparative summary of the three datasets used in our experiments.

Table 6: Summary statistics of the three financial sentiment datasets.

Dataset	Samples	Score Range	Mean	Std
NEU	12,780	[-0.950, 0.900]	-0.060	0.312
FXE	10,461	[-0.900, 0.900]	-0.014	0.258
INV	23,241	[-1.000, 0.900]	-0.044	0.303

A.8.6 Split Construction

One CSV row is one example. The 9:1 split is taken by shuffling the full frame under the fixed seed and cutting the validation set from the head of the shuffled frame, without stratification by sentiment score. The collection and labelling protocols are those of the cited sources (Ding et al., 2024b;c;a) and are not varied in this work.

Appendix A.9: Backbone Model

A.9.1 Model Architecture

We employ Twitter-RoBERTa-Large (Loureiro et al., 2023) as the embedding generator for text data, as illustrated in Figure 12. The training set is defined as $D = \{(x_i, y_i)\}_{i=1}^{N}$, where N denotes the total number of texts, x_i represents each text in the dataset, and $y_i \in [-1, 1]$ corresponds to the annotated sentiment polarity score.

The pre-trained RoBERTa tokenizer $\mathrm{Tokenizer}(\cdot)$ processes each text x_i, and then the tokenized text passes through 24 Transformer encoder blocks $\mathrm{Encoder}(\cdot)$ to generate the final hidden state H_i:

$$H_i = \mathrm{Encoder}(\mathrm{Tokenizer}(x_i)), \tag{14}$$

where $H_i \in \mathbb{R}^{1\times 1024}$. This hidden state serves as input to task-specific heads.

A.9.2 Regression Head

For sentiment polarity analysis, we implement a regression head comprising two linear layers and a sigmoid activation function:

$$S_i = \mathrm{LL}_2(\sigma(\mathrm{LL}_1(H_i))), \tag{15}$$

where $\mathrm{LL}_1 : \mathbb{R}^{1024} \to \mathbb{R}^{128}$ is the first linear layer, $\sigma(\cdot)$ denotes the sigmoid function, and $\mathrm{LL}_2 : \mathbb{R}^{128} \to \mathbb{R}$ produces the final sentiment polarity score $S_i \in [-1, 1]$.

A.9.3 Loss Function

For the regression task, we use MSE as the loss function:

$$L_{\mathrm{r}} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2, \tag{16}$$

where n denotes the number of texts in the training set, $\hat{y}_i$ represents the predicted polarity score, and y_i is the ground truth sentiment polarity score.

Appendix A.10: Evaluation Metrics

For the sentiment score regression task, we employ the following metrics:

A.10.1 Mean Squared Error (MSE)

MSE calculates the average squared differences between the predicted sentiment polarity scores ($\hat{y}_i$) and the annotated sentiment polarity scores (y_i):

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{17}$$

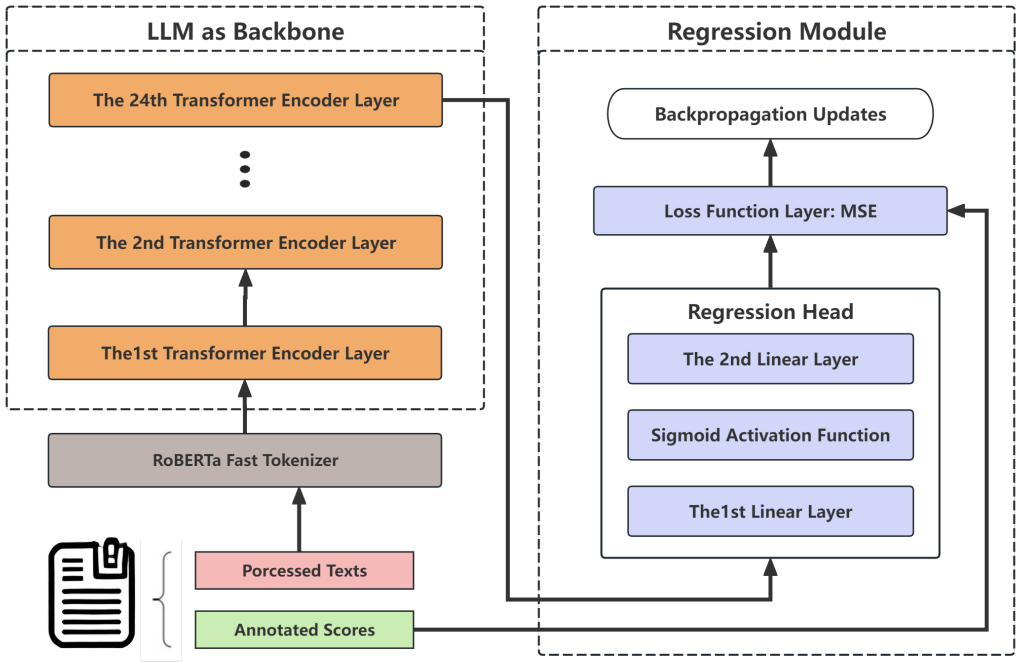


Figure 12: LLM as the backbone is connected with a regression head.

where n is the number of texts in the evaluation split.

A.10.2 Mean Absolute Error (MAE)

MAE measures the average magnitude of the absolute errors:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|\,. \tag{18}$$

A.10.3 Root Mean Squared Error (RMSE)

RMSE is the square root of MSE:

$$\text{RMSE} = \sqrt{\text{MSE}}. \tag{19}$$

Appendix A.11: Hyperparameters

A.11.1 Model and Tokenization Parameters

Base Model Configuration:

- **Architecture**: Twitter-RoBERTa-Large (Loureiro et al., 2023)
- **Model dimensions**: $d_{\text{model}} = 1024$, $d_{\text{ff}} = 4096$
- **Number of layers**: $L = 24$
- **Number of attention heads**: 16
- **Maximum sequence length**: 512 tokens

Tokenization:

- **Tokenizer**: RoBERTa tokenizer with byte-pair encoding
- **Truncation**: First 512 tokens for sequences exceeding limit
- **Padding**: Dynamic padding to maximum length in batch

A.11.2 LoRA Configuration

Standard LoRA Settings:

- **Rank values**: $r \in \{8, 16, 32, 64, 128, 256, 384, 512\}$
- **Alpha**: $\alpha = r$ (scaling factor equal to rank)
- **Dropout rate**: $p_{\text{dropout}} = 0.05$
- **Target modules**: Query (Q), Key (K), Value (V), Output (O) projections

Hierarchical Allocation Budgets:

- **384-rank budget**: $\beta = 384 \times L = 9,216$, $r_{\min} = 256$, $r_{\max} = 512$
- **512-rank budget**: $\beta = 512 \times L = 12,288$, $r_{\min} = 384$, $r_{\max} = 640$

A.11.3 Training Parameters

Optimizer:

- **Optimizer**: AdamW with weight decay
- **Learning rate**: $\eta = 1 \times 10^{-5}$
- **Schedule**: Cosine annealing with 100 warmup steps
- **Weight decay**: $\lambda = 0.01$

Training Protocol:

- **Epochs**: 100
- **Batch size**: 10 for $r \leq 128$, 8 for $r \leq 256$, 6 for $r \leq 384$, and 5 above that
- **Gradient clipping**: Maximum norm of 1.0
- **Mixed precision**: FP16 enabled
- **Random seeds**: 42 (Python, NumPy, PyTorch). The same seed shuffles the full frame before the 9:1 cut, so it also fixes which rows fall in the validation split, and it draws the reduced-scale subsample of Appendix A.20

A.11.4 Hierarchical Allocation Parameters

Phase I - Coarse-Grained Exploration:

- **Candidate groupings**: $\mathcal{G} = \{2, 3, 4, 6, 8, 12, 24\}$ for $L = 24$
- **Interval width threshold**: $\tau = 64$, balancing search granularity and computational efficiency

Phase II - Fine-Grained Optimization:

- **Budget enforcement**: Sequential allocation with strict constraint
- **Convergence threshold**: $\epsilon = 1 \times 10^{-4}$

A.11.5 Baseline Parameters

AdaLoRA Configurations:

- **Configuration 1**: $640 \rightarrow 384$ (initial $\rightarrow$ target budget)
- **Configuration 2**: $640 \rightarrow 512$
- **Configuration 3**: $512 \rightarrow 384$
- **Pruning schedule**: Cubic budget schedule (AdaLoRA default), warmup ratio 0.1
- **Importance score**: Gradient-based sensitivity with exponential moving-average smoothing (coefficient 0.9)

QLoRA Settings:

- **Quantization**: 4-bit NormalFloat (NF4)
- **LoRA parameters**: Same as standard configuration

A.11.6 Compute Budget and Wall-Clock Summary

Hardware. All main experiments were conducted on a single workstation running Ubuntu 22.04 with one NVIDIA RTX 3090Ti GPU (24 GB VRAM) and 32 GB system RAM; no distributed training was used.

Per-evaluation training time. A single fine-tuning run of Twitter-RoBERTa-Large for 100 epochs on the largest dataset (INV) takes 8 to 15 minutes per run depending on rank configuration; higher-rank configurations fall toward the upper end of this range.

Total search compute for main experiments. The hierarchical search (Appendix A.21) requires $N_{\text{total}} = 332$ evaluations per (dataset, budget) configuration. Multiplying by the per-evaluation timing above, a single search instance completes in 44 to 83 GPU-hours; the full primary suite (3 datasets $\times$ 2 budget configurations, i.e., 6 search instances) totals 266 to 498 GPU-hours on the same RTX 3090Ti.

Exploratory and pilot experiments. Preliminary hyperparameter scans, motivation-figure rank sweeps (Figures 2 and 3 of the main paper), and exploratory runs not included in the main tables contributed an additional 50 to 100 GPU-hours, reported here for completeness.

Appendix A.12: Rank Budget Configuration Rationale

A.12.1 Budget Formulation

The rank budget is defined as:

$$\beta = \bar{r} \times L, \tag{20}$$

where $\bar{r}$ denotes the target average rank per layer and L is the total number of transformer layers. This formulation ensures fair comparison with uniform LoRA configurations: a hierarchical allocation with $\beta = 384 \times L$ uses the same total rank as uniform LoRA with rank 384 across all layers.

A.12.2 Search Range Design

The search range $[r_{\min}, r_{\max}]$ is centered around the target average rank $\bar{r}$ to allow per-layer deviation while maintaining the overall budget constraint:

- **384-rank budget**: $\bar{r} = 384$, search range $[256, 512]$ (i.e., $\bar{r} \pm 128$)
- **512-rank budget**: $\bar{r} = 512$, search range $[384, 640]$ (i.e., $\bar{r} \pm 128$)

This symmetric range design enables the hierarchical allocation strategy to discover non-uniform rank distributions where some layers receive higher ranks (up to $r_{\max}$) while others receive lower ranks (down to $r_{\min}$), as long as the total satisfies $\sum_{i=1}^{L} r_i \leq \beta$.

A.12.3 Interval Threshold

The interval threshold $\tau = 64$ controls the termination condition of the coarse-grained exploration (Algorithm 2). Each refined interval $\mathcal{I}_i^* = [l_i, u_i]$ satisfies $u_i - l_i \leq \tau$, balancing search granularity and computational efficiency. A smaller τ yields finer intervals but requires more iterations, while a larger τ reduces computation at the cost of precision.

Appendix A.13: Solution Space Analysis

A.13.1 Solution Space Definition

The rank allocation optimization problem operates within a discrete, high-dimensional feasible set, obtained by intersecting the rank space $\mathcal{S}_{\mathcal{R}}$ with the budget constraint:

$$\mathcal{S}_{\text{feasible}} = \{\mathbf{r} \in \mathbb{N}^L : r_{\min} \leq r_i \leq r_{\max}, \sum_{i=1}^{L} r_i \leq \beta\}, \tag{21}$$

where each configuration vector $\mathbf{r} = [r_1, r_2, \ldots, r_L]$ represents a complete rank assignment across all L transformer layers, $r_{\min}$ and $r_{\max}$ define the per-layer rank bounds, and β represents the total rank budget constraint.

A.13.2 Cardinality Analysis

Without the budget constraint, the number of possible configurations is:

$$|\mathcal{S}_{\text{unconstrained}}| = (r_{\max} - r_{\min} + 1)^L. \tag{22}$$

A.13.3 Space Reduction via Hierarchical Allocation

Two mechanisms reduce the search space:

Layer Grouping. By partitioning L layers into g groups where layers within each group share identical ranks, the search space reduces from $(r_{\max} - r_{\min} + 1)^L$ to $(r_{\max} - r_{\min} + 1)^g$:

$$|\mathcal{S}_{\text{grouped}}| = (r_{\max} - r_{\min} + 1)^g. \tag{23}$$

Binary Search. Rather than enumerating all grouped configurations, our binary greedy search requires only $O(\log_2 \tau)$ iterations per group, where τ is the interval threshold. With two configurations evaluated per iteration, the fine-grained stage costs:

$$N_{\text{fine}} = 2g \times \log_2 \tau. \tag{24}$$

Adding the coarse-grained exploration, which costs 236 evaluations, gives a measured total of 332 evaluations per (dataset, budget) pair. The full itemisation is in Appendix A.21.

Overall Reduction. The hierarchical allocation strategy reduces the number of evaluated configurations from $O(10^{58})$ to $O(10^2)$, a a reduction factor of approximately 3×10^{55}. This factor compares evaluation counts only: it quantifies how much of the space is left unexplored, not how close the returned allocation is to the exhaustive optimum, which is unknown.

Appendix A.14: LoRA Rank Configuration Analysis

Experimental Setup. This experiment systematically evaluates standard LoRA performance across rank configurations $r \in \{8, 16, 32, 64, 128, 256, 384, 512\}$ on three financial sentiment datasets. For each configuration, the alpha value equals the rank ($\alpha = r$) and dropout rate is fixed at 0.05. The baseline represents full fine-tuning performance without LoRA adaptation. All experiments use identical training hyperparameters including learning rate 1e-5, batch size 5-10 (adaptive based on rank configuration), and 100 training epochs to ensure fair comparison.

Table 7: Performance comparison of different LoRA rank configurations on regression tasks across three datasets.

Configuration	NEU Dataset			FXE Dataset			INV Dataset		
	MSE↓	MAE↓	RMSE↓	MSE↓	MAE↓	RMSE↓	MSE↓	MAE↓	RMSE↓
Baseline	0.0189	0.0981	0.1375	0.0202	0.1014	0.1421	0.0183	0.0959	0.1353
Rank=8	0.0221	0.1055	0.1487	0.0215	0.1063	0.1466	0.0199	0.1013	0.1411
Rank=16	0.0231	0.1081	0.1520	0.0227	0.1092	0.1507	0.0198	0.1012	0.1407
Rank=32	0.0211	0.1040	0.1453	0.0225	0.1078	0.1500	0.0201	0.1031	0.1418
Rank=64	0.0201	0.1010	0.1418	0.0213	0.1051	0.1460	0.0197	0.1011	0.1404
Rank=128	0.0193	0.0995	0.1389	0.0209	0.1036	0.1446	0.0190	0.0985	0.1378
Rank=256	0.0191	0.0987	0.1382	0.0205	0.1028	0.1432	0.0185	0.0974	0.1360
Rank=384	0.0187	0.0974	0.1368	0.0203	0.1018	0.1425	0.0182	0.0958	0.1349
Rank=512	0.0184	0.0955	0.1356	0.0197	0.1004	0.1404	0.0181	0.0952	0.1345

Low-Rank Performance (r = 8-64). The low-rank configurations consistently underperform the baseline across all datasets. Rank=8 shows average MSE degradation of 10.7% compared to baseline, with NEU dataset exhibiting the largest degradation of 16.9% (0.0221 vs 0.0189), while FXE shows 6.4% degradation (0.0215 vs 0.0202) and INV shows 8.7% degradation (0.0199 vs 0.0183). Notably, Rank=16 produces the worst overall performance with 14.3% average MSE degradation, suggesting this rank falls in a particularly unfavorable region where the low-rank approximation introduces significant errors without providing sufficient capacity for task adaptation. The performance gradually improves at Rank=32 and Rank=64, with the average gap to baseline narrowing to approximately 11% and 6.5% respectively.

Medium-Rank Performance (r = 128-256). Performance stabilizes in this regime and begins to approach baseline. Rank=128 approaches baseline performance within 4% MSE across all datasets, representing the minimum viable rank for financial sentiment analysis tasks. Rank=256 remains slightly above the no-LoRA baseline on all three datasets (average 1.2% higher MSE), achieving 0.0191 (NEU), 0.0205 (FXE), and 0.0185 (INV).

High-Rank Performance (r = 384-512). The best uniform LoRA performance among tested ranks is achieved in this regime. Rank=384 shows performance comparable to baseline with slight improvement on NEU (1.1%) and INV (0.5%) but slight degradation on FXE (0.5%). Rank=512 achieves the best uniform LoRA results with 2.1% average MSE improvement over baseline, reaching MSE values of 0.0184 (NEU), 0.0197 (FXE), and 0.0181 (INV). The INV dataset shows the strongest performance at Rank=512. The progressive improvement from Rank=384 to Rank=512 demonstrates that higher ranks continue to benefit performance in this regime, though with diminishing returns.

Conclusion. These results establish the empirical foundation for our hierarchical allocation strategy. The best-performing uniform ranks, 384 and 512, serve as the target average ranks $\bar{r}$ for our performance-based exploration framework; each search range is centred on its $\bar{r}$ as $[\bar{r}-128, \bar{r}+128]$, i.e., $[256, 512]$ and $[384, 640]$ (Appendix A.12.2). Importantly, even at the best uniform configuration (Rank = 512), the improvement over baseline is limited to approximately 2.1%, while our hierarchical approach achieves 6.94% improvement with an identical total parameter budget. The two figures use different reference points: the 2.1% is measured against the no-LoRA baseline, the 6.94% against uniform LoRA at the same budget. Read separately, they show that redistributing a fixed budget across layers buys more than raising the uniform rank does.

APPENDIX A.15: HIERARCHICAL ALLOCATION PERFORMANCE ANALYSIS

Experimental Setup. Table 7 reports uniform LoRA across the tested rank values on all three datasets. This experiment evaluates our hierarchical allocation strategy under two budget configurations: 384-rank budget ($r_{\min} = 256$, $r_{\max} = 512$, $\beta = 384 \times L$) and 512-rank budget ($r_{\min} = 384$, $r_{\max} = 640$, $\beta = 512 \times L$), where $L = 24$ is the number of transformer layers. For each budget, we

Table 8: Performance comparison between LoRA rank=384 and our hierarchical allocation strategy (384-rank budget).

Dataset	Metric	Group=3		Group=4		Group=6		Group=8	
		LoRA	Ours	LoRA	Ours	LoRA	Ours	LoRA	Ours
NEU	MSE	0.0187	0.0183	0.0187	0.0181	0.0187	0.0179	0.0187	0.0175
	MAE	0.0974	0.0958	0.0974	0.0932	0.0974	0.0945	0.0974	0.0937
	RMSE	0.1368	0.1353	0.1368	0.1345	0.1368	0.1338	0.1368	0.1323
FXE	MSE	0.0203	0.0198	0.0203	0.0196	0.0203	0.0195	0.0203	0.0190
	MAE	0.1018	0.0998	0.1018	0.0992	0.1018	0.0988	0.1018	0.0975
	RMSE	0.1425	0.1407	0.1425	0.1400	0.1425	0.1396	0.1425	0.1378
INV	MSE	0.0182	0.0177	0.0182	0.0177	0.0182	0.0176	0.0182	0.0170
	MAE	0.0958	0.0941	0.0958	0.0934	0.0958	0.0935	0.0958	0.0914
	RMSE	0.1349	0.1330	0.1349	0.1330	0.1349	0.1327	0.1349	0.1304

Table 9: Performance comparison between LoRA rank=512 and our hierarchical allocation strategy (512-rank budget).

Dataset	Metric	Group=3		Group=4		Group=6		Group=8	
		LoRA	Ours	LoRA	Ours	LoRA	Ours	LoRA	Ours
NEU	MSE	0.0184	0.0180	0.0184	0.0178	0.0184	0.0175	0.0184	0.0172
	MAE	0.0955	0.0990	0.0955	0.0942	0.0955	0.0930	0.0955	0.0928
	RMSE	0.1356	0.1342	0.1356	0.1334	0.1356	0.1323	0.1356	0.1311
FXE	MSE	0.0197	0.0191	0.0197	0.0190	0.0197	0.0186	0.0197	0.0183
	MAE	0.1004	0.0977	0.1004	0.0973	0.1004	0.0968	0.1004	0.0955
	RMSE	0.1404	0.1382	0.1404	0.1378	0.1404	0.1364	0.1404	0.1353
INV	MSE	0.0181	0.0176	0.0181	0.0172	0.0181	0.0171	0.0181	0.0168
	MAE	0.0952	0.0942	0.0952	0.0932	0.0952	0.0923	0.0952	0.0905
	RMSE	0.1345	0.1327	0.1345	0.1311	0.1345	0.1308	0.1345	0.1296

evaluate four grouping strategies $g \in \{3, 4, 6, 8\}$ that satisfy the divisibility constraint $L \bmod g = 0$. The interval threshold is set to $\tau = 64$. The baseline comparison uses uniform LoRA with rank equal to the average budget (384 or 512).

Results under 384-Rank Budget. Under the 384-rank budget (Table 8), for the NEU dataset, MSE improves from 0.0187 (uniform LoRA) to 0.0183 (Group=3), 0.0181 (Group=4), 0.0179 (Group=6), and 0.0175 (Group=8), corresponding to improvement rates of 2.14%, 3.21%, 4.28%, and 6.42% respectively. The FXE dataset shows similar trends with MSE reducing from 0.0203 to 0.0190 at Group=8, achieving 6.40% improvement. The INV dataset exhibits the most substantial gains at Group=8, with MSE dropping from 0.0182 to 0.0170 (6.59% improvement).

Results under 512-Rank Budget. Under the 512-rank budget (Table 9), the NEU dataset achieves MSE=0.0172 at Group=8, representing 6.52% improvement over uniform LoRA (0.0184). The FXE dataset improves by 7.11%, with MSE reducing from 0.0197 to 0.0183. Most notably, the INV dataset achieves the best absolute result with MSE=0.0168 at Group=8, a 7.18% improvement over the baseline.

Monotonic Improvement Pattern. A consistent pattern emerges across all experimental settings: performance improves monotonically as group granularity increases. The average MSE improvement scales from 2.56% at Group=3, to 3.53% at Group=4, to 4.59% at Group=6, and finally to 6.70% at Group=8. This monotonic relationship validates our hypothesis that finer-grained rank allocation better captures the heterogeneous characteristics of transformer layers at different depths.

Table 10: Complete performance comparison between AdaLoRA and our hierarchical allocation strategy.

Method	Configuration	NEU Dataset			FXE Dataset			INV Dataset		
		MSE↓	MAE↓	RMSE↓	MSE↓	MAE↓	RMSE↓	MSE↓	MAE↓	RMSE↓
AdaLoRA	640→384	0.0211	0.1061	0.1453	0.0220	0.1057	0.1483	0.0188	0.1020	0.1371
	640→512	0.0221	0.1055	0.1487	0.0233	0.1082	0.1526	0.0198	0.1008	0.1407
	512→384	0.0231	0.1085	0.1520	0.0240	0.1117	0.1549	0.0207	0.1031	0.1439
Ours (Group=8)	384-rank	0.0175	0.0937	0.1323	0.0190	0.0975	0.1378	0.0170	0.0914	0.1304
	512-rank	**0.0172**	**0.0928**	**0.1311**	**0.0183**	**0.0955**	**0.1353**	**0.0168**	**0.0905**	**0.1296**

Cross-Dataset Consistency. Our hierarchical allocation reduces MSE in all 24 experimental configurations (2 budgets $\times$ 3 datasets $\times$ 4 groupings), demonstrating robust generalization across different data distributions. The INV dataset consistently shows the highest absolute improvements, while NEU exhibits the most stable improvement patterns across grouping configurations. FXE shows larger variance between configurations, suggesting that foreign exchange-related texts may have more complex layer-wise requirements.

Conclusion. The 8-group configuration under 512-rank budget achieves the best average MSE improvement of 6.94% over uniform allocation at an identical total parameter budget.

APPENDIX A.16: COMPARISON WITH ACCUMULATION-BASED STRATEGY

Experimental Setup. This experiment compares our hierarchical allocation strategy with AdaLoRA, a representative accumulation-based adaptive rank allocation method. AdaLoRA employs SVD-based parameterization and adaptively prunes singular values during training based on gradient-derived importance scores. We evaluate three AdaLoRA configurations: 640→384 (initial rank 640, target rank 384), 640→512 (initial rank 640, target rank 512), and 512→384 (initial rank 512, target rank 384). All AdaLoRA experiments use default hyperparameters including warmup ratio 0.1, cubic pruning schedule, and regularization coefficient $\gamma = 0.1$. Our method uses Group=8 configuration for both 384-rank and 512-rank budgets.

Performance on NEU Dataset. Table 10 gives the full AdaLoRA comparison on all three datasets. On NEU our method outperforms every AdaLoRA configuration. The best AdaLoRA configuration (640→384) achieves MSE=0.0211, while our 512-rank budget configuration achieves MSE=0.0172. This represents an 18.48% MSE improvement. Even our lower-budget configuration (384-rank) outperforms the best AdaLoRA result, achieving MSE=0.0175 (17.06% improvement). The MAE metric shows similar patterns, with our method achieving 0.0928 compared to AdaLoRA's best of 0.1055.

Performance on FXE Dataset. The FXE dataset exhibits the largest performance gap between methods. AdaLoRA configurations achieve MSE values ranging from 0.0220 to 0.0240. Our 512-rank budget configuration achieves MSE=0.0183, representing improvements of 16.82% to 23.75% in MSE depending on the AdaLoRA configuration. The substantial gap suggests that foreign exchange-related sentiment analysis particularly benefits from our structured hierarchical exploration compared to sensitivity-driven adaptive methods.

Performance on INV Dataset. On the INV dataset, our method achieves the best absolute performance across all experiments with MSE=0.0168 under the 512-rank budget. AdaLoRA's best configuration (640→384) achieves MSE=0.0188, resulting in 10.64% MSE improvement for our method. While the relative improvement on INV is smaller than other datasets, the absolute performance difference remains substantial.

Analysis of Performance Gap. The substantial performance gap between our method and AdaLoRA can be attributed to three fundamental differences. First, AdaLoRA relies on gradient-

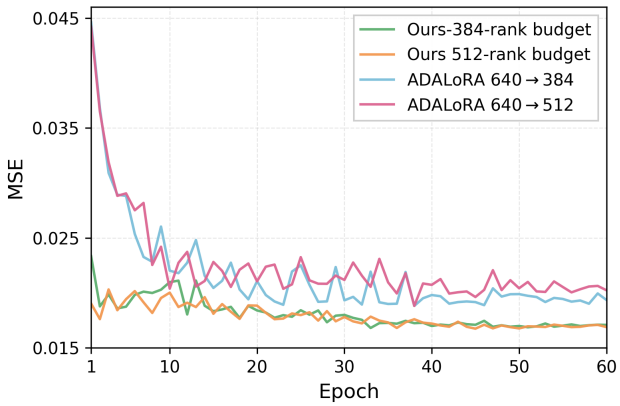


Figure 13: Training stability comparison between AdaLoRA and our hierarchical allocation method with different rank budgets on INV dataset.

based importance scores that can be noisy during early training phases, leading to suboptimal pruning decisions that propagate throughout training. Our method uses direct validation performance evaluation, providing more reliable optimization signals. Second, AdaLoRA's pruning decisions are irreversible. Once a singular value is removed, the model cannot recover if the decision was suboptimal. Our hierarchical allocation determines rank configurations before training, avoiding such irreversible errors. Third, AdaLoRA makes layer-wise decisions based on local importance without global coordination, while our two-phase strategy performs joint optimization across all layers.

Conclusion. Our hierarchical allocation strategy consistently and substantially outperforms AdaLoRA across all datasets and configurations, achieving 15.31% to 22.71% average MSE improvement depending on the AdaLoRA configuration.

APPENDIX A.17: TRAINING STABILITY ANALYSIS

Experimental Setup. This experiment analyzes training dynamics and stability characteristics by monitoring validation MSE throughout the complete training process (100 epochs) on the INV dataset. We compare four configurations: our method with 384-rank and 512-rank budgets (both using Group=8), and AdaLoRA with 640→384 and 640→512 configurations. Training curves are recorded at each epoch, and quantitative stability metrics are computed including initial MSE, convergence epoch (defined as the first epoch where MSE falls within 5% of final value), post-convergence standard deviation, and maximum oscillation amplitude.

Initialization Phase Analysis. Figure 13 shows the difference in initialization behavior. Both AdaLoRA configurations start with high MSE values around 0.045, while our method begins with significantly lower values around 0.024, representing a 47% reduction in initial error. This initialization advantage stems from our pre-optimized rank allocation that provides well-calibrated parameter distribution from the start, whereas AdaLoRA initializes with uniform high ranks before adaptive pruning begins. The lower initial MSE indicates that our rank configuration is already closer to optimal before any task-specific training occurs.

Convergence Behavior. Our method demonstrates substantially faster convergence compared to AdaLoRA. The 512-rank configuration reaches convergence at epoch 32, while the 384-rank configuration converges at epoch 35. In contrast, AdaLoRA 640→384 requires 58 epochs and AdaLoRA 640→512 requires 65 epochs to converge, representing 1.8× slower convergence. During the early training phase (epochs 1-20), AdaLoRA exhibits significant oscillations with MSE fluctuating between 0.020 and 0.030, while our method shows smooth monotonic decrease. This difference arises because AdaLoRA must simultaneously learn task-specific features and discover appropriate rank allocation, while our method focuses entirely on task optimization with fixed, pre-determined ranks.

Post-Convergence Stability. After convergence, our method maintains stable MSE with little oscillation. The standard deviation of MSE during epochs 50-100 is 0.0002 for our 512-rank configuration and 0.0003 for 384-rank, compared to 0.0015 and 0.0012 for AdaLoRA 640→512 and 640→384 respectively. This represents 4.0 to 7.5× lower post-convergence standard deviation for our method. The maximum oscillation amplitude shows similar patterns: our method exhibits maximum oscillation of 0.0015-0.0018, while AdaLoRA shows 0.0089-0.0095, approximately 5× larger. The continued instability in AdaLoRA stems from ongoing rank adjustments that perturb learned representations even after the model has nominally converged.

Table 11: Performance comparison of LoRA-based methods under 512-rank budget.

Method	NEU Dataset			FXE Dataset			INV Dataset		
	MSE↓	MAE↓	RMSE↓	MSE↓	MAE↓	RMSE↓	MSE↓	MAE↓	RMSE↓
Ours	**0.0172**	**0.0928**	**0.1311**	**0.0183**	**0.0955**	**0.1353**	**0.0168**	**0.0905**	**0.1296**
LoRA	0.0184	0.0955	0.1356	0.0197	0.1004	0.1404	0.0181	0.0952	0.1345
QLoRA	0.0191	0.0982	0.1382	0.0210	0.1035	0.1449	0.0187	0.0972	0.1367
LoRA+	0.0182	0.0952	0.1349	0.0196	0.1001	0.1400	0.0179	0.0947	0.1338
DoRA	0.0181	0.0948	0.1345	0.0195	0.0997	0.1396	0.0178	0.0944	0.1334
DyLoRA	0.0182	0.0953	0.1349	0.0197	0.1002	0.1404	0.0179	0.0950	0.1338

Conclusion. The training stability analysis reveals three key advantages of our hierarchical allocation: 47% lower initial MSE due to pre-optimized rank distribution, 1.8× faster convergence by eliminating simultaneous rank discovery, and 4.0 to 7.5× lower post-convergence standard deviation through fixed rank allocation. These stability improvements have important practical implications including better reproducibility across training runs, more reliable best-epoch selection from the validation curve, and reduced sensitivity to learning rate and other hyperparameters.

APPENDIX A.18: COMPARISON WITH LORA VARIANTS

Experimental Setup. This experiment compares our hierarchical allocation strategy against five representative LoRA variants: standard LoRA, QLoRA (4-bit quantized LoRA), LoRA+ (differentiated learning rates for A and B matrices), DoRA (magnitude-direction decomposition), and DyLoRA (dynamic rank training). All methods use rank=512 for fair comparison under identical parameter budgets. Method-specific hyperparameters follow original paper recommendations: QLoRA uses 4-bit NormalFloat quantization with double quantization enabled; LoRA+ uses learning rate ratio of 10 between matrices A and B; DoRA uses magnitude learning rate of 5e-4; DyLoRA samples ranks uniformly from [256, 512] during training.

Comparison with Standard LoRA. Our hierarchical allocation at the reported operating point achieves consistent MSE improvements over standard LoRA on all three datasets. On NEU dataset, MSE improves from 0.0184 to 0.0172 (6.52% reduction). FXE dataset shows 7.11% MSE improvement (0.0197 to 0.0183). INV dataset demonstrates the largest improvement with a 7.18% MSE reduction (0.0181 to 0.0168).

Comparison with QLoRA. QLoRA shows degraded performance compared to standard LoRA due to quantization-induced precision loss, achieving MSE values of 0.0191, 0.0210, and 0.0187 on NEU, FXE, and INV datasets respectively. Our method outperforms QLoRA by substantial margins: 9.95% on NEU, 12.86% on FXE, and 10.16% on INV, with average improvement of 10.99%.

Comparison with LoRA+ and DoRA. Table 11 lists the LoRA variants under the 512-rank budget. LoRA+ and DoRA represent recent improvements over standard LoRA through optimized learning dynamics and magnitude-direction decomposition respectively. Both methods show modest improvements over standard LoRA, with DoRA achieving slightly better results (MSE: 0.0181, 0.0195, 0.0178 across datasets). However, our method still outperforms DoRA by 4.97% on NEU, 6.15% on FXE, and 5.62% on INV. The average improvement of 5.58% over DoRA indicates that the gain does not come from learning dynamics alone.

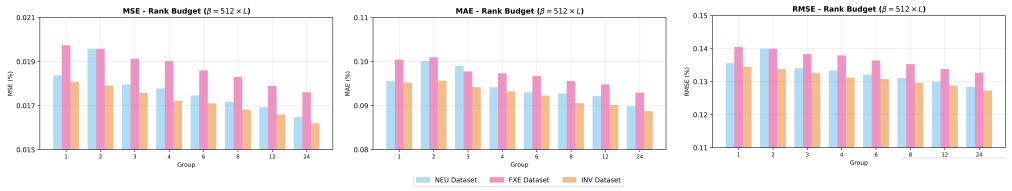


Figure 14: Performance of our hierarchical allocation strategy with different groups under 512-rank budget.

Comparison with DyLoRA. DyLoRA enables dynamic rank selection during inference by training across multiple rank values. Despite this flexibility, DyLoRA achieves similar performance to standard LoRA (MSE: 0.0182, 0.0197, 0.0179), suggesting that dynamic inference-time rank selection does not address the fundamental limitation of uniform rank allocation during training. Our method outperforms DyLoRA by 5.49% on NEU, 7.11% on FXE, and 6.15% on INV.

Conclusion. Our hierarchical allocation achieves the best MSE on all three datasets, outperforming standard LoRA by 6.94% average MSE improvement and recent variants by 5.58% to 10.99%. Importantly, our approach addresses a different aspect of LoRA optimization, namely, rank distribution across layers, compared to existing variants that focus on quantization (QLoRA), learning dynamics (LoRA+), parameterization (DoRA), or inference flexibility (DyLoRA). This orthogonality suggests potential for combining hierarchical allocation with these methods for further improvements.

APPENDIX A.19: EFFECT OF LAYER GROUPING

Experimental Setup. This ablation study investigates the effect of layer grouping granularity on our hierarchical allocation strategy. For the 24-layer Twitter-RoBERTa-Large backbone, valid grouping configurations satisfying the divisibility constraint $L \bmod g = 0$ include $\mathcal{G} = \{1, 2, 3, 4, 6, 8, 12, 24\}$. Group=1 is equivalent to standard uniform LoRA where all layers share the same rank, while Group=24 represents maximum flexibility with each layer having an independent rank. All experiments use the 512-rank budget configuration ($\beta = 512 \times L$) with interval threshold $\tau = 64$. Figure 14 presents the complete results across the three error metrics (MSE, MAE, RMSE) for all three datasets.

Performance Trends. Figure 14 reveals a clear monotonic improvement pattern as group count increases across all three datasets. At Group=1 (uniform LoRA baseline), FXE dataset exhibits the highest MSE values, followed by NEU and INV. As the number of groups increases, all datasets show consistent MSE reduction, with the improvement becoming more pronounced at higher group counts. This improvement stems from the fact that higher group counts can capture the heterogeneous characteristics of transformer layers at different depths. When layers are grouped more finely, each group can receive a rank value that is better suited to its specific representational requirements.

Dataset-Specific Behavior. The three datasets exhibit distinct response patterns. The INV dataset (orange bars) shows particularly strong improvement at higher group counts, achieving the lowest MSE at Group=12 and Group=24, suggesting that INV's sentiment patterns benefit most from fine-grained layer-specific adaptation. The FXE dataset (pink bars) consistently shows the highest MSE across all configurations, indicating greater inherent difficulty in foreign exchange-related sentiment analysis. However, FXE also demonstrates substantial relative improvement from coarse to fine grouping. The NEU dataset (blue bars) exhibits moderate and stable improvement, indicating balanced layer-wise requirements. These patterns likely reflect differences in linguistic complexity and sentiment expression styles across corpora.

Trade-off and Practical Recommendations. While higher group counts consistently yield better performance, the marginal improvement diminishes at the highest granularities. The visual gap between adjacent bars decreases as group count increases. This arises because each additional group introduces new degrees of freedom that require more exploration iterations in our binary greedy search. For practical deployment, Group=8 offers a favorable trade-off between performance gain

Table 12: Impact of dataset size on hierarchical allocation strategy performance across different data scales. Percentages quoted in Section 5.3 are computed from the 512-rank budget rows.

Configuration	Metric	FXE Dataset				INV Dataset			
		25%	50%	75%	100%	25%	50%	75%	100%
384-rank budget (Group=8)	MSE	0.0289	0.0241	0.0207	0.0190	0.0263	0.0218	0.0188	0.0170
	MAE	0.1267	0.1158	0.1042	0.0975	0.1221	0.1105	0.0986	0.0914
512-rank budget (Group=8)	MSE	0.0276	0.0228	0.0201	0.0183	0.0255	0.0212	0.0184	0.0168
	MAE	0.1245	0.1139	0.1028	0.0955	0.1198	0.1089	0.0971	0.0905

and computational cost, while applications with sufficient resources can benefit from Group=12 or Group=24 for maximum performance.

APPENDIX A.20: EFFECT OF DATASET SIZE

Experimental Setup. This ablation study investigates how dataset size affects the performance of our hierarchical allocation strategy, as shown in Table 12. We evaluate at four data scale conditions: 25%, 50%, 75%, and 100% of the original training data. The reduced-scale sets are drawn from the training split as a plain random sample under the fixed seed, without stratification by sentiment score. For each scale, we perform stratified random sampling to maintain the original label distribution. All configurations in this ablation are trained under the fixed random seed and training protocol of Appendix A.11.3. Both 384-rank and 512-rank budget configurations are evaluated with Group=8 setting. The FXE and INV datasets are used for this analysis as they represent different characteristics of financial sentiment data.

Performance at Full Data Scale (100%). At 100% data scale, our method achieves optimal performance as reported in the main experiments. Under 512-rank budget, FXE dataset achieves MSE=0.0183, while INV dataset reaches MSE=0.0168. The 384-rank budget shows slightly lower but still strong performance with FXE at MSE=0.0190 and INV at MSE=0.0170. These results serve as the baseline for evaluating degradation at reduced data scales.

Performance Degradation at Reduced Scales. At 75% data scale, performance degradation is moderate and acceptable. FXE dataset shows MSE increase from 0.0183 to 0.0201 (9.84% degradation) under 512-rank budget, while INV shows increase from 0.0168 to 0.0184 (9.52% degradation). Our method therefore maintains reliable performance with 75% of training data.

Severe Degradation at Limited Data. At 50% data scale, degradation becomes more pronounced. FXE achieves MSE=0.0228 (24.59% degradation from 100%), while INV shows MSE=0.0212 (26.19% degradation). At 25% data scale, our method suffers severe performance degradation: FXE reaches MSE=0.0276 (50.82% degradation) and INV shows MSE=0.0255 (51.79% degradation). This significant degradation indicates that insufficient training samples prevent reliable rank optimization signal generation during the exploration phase.

Root Cause Analysis. The data dependency of our method stems from the validation-guided exploration mechanism. With limited training data, three compounding factors degrade performance: the validation set becomes smaller with higher variance in performance estimates, training converges to suboptimal solutions making different rank configurations harder to distinguish, and overfitting risks increase leading to unreliable performance signals. These factors collectively undermine the effectiveness of our binary greedy search algorithms.

Conclusion. Our hierarchical allocation framework exhibits critical data dependency with distinct performance regimes. At 75-100% data scales, the method maintains stable and substantial improvements over uniform LoRA. At 50-75% data scales, moderate degradation occurs but improvements

over baseline may still be achievable with reduced grouping granularity (e.g., Group=4). Below 50% data scale, severe degradation suggests that standard uniform LoRA may be preferable.

APPENDIX A.21: ALGORITHM ANALYSIS

A.21.1 ALGORITHM EFFECTIVENESS ANALYSIS

EXPLOITATION OF ARCHITECTURAL COHERENCE

Transformer architectures exhibit natural clustering patterns where functionally related layers demonstrate similar rank preferences. Our layer grouping strategy exploits this architectural coherence:

$$\text{Performance}(\mathbf{r}) \approx \sum_{i=1}^{g} f_i(r_{G_i}), \tag{25}$$

where r_{G_i} denotes the shared rank for group i, and $f_i(\cdot)$ captures the group-specific performance contribution. We adopt this quasi-separable structure as a modeling assumption rather than a measured property: it is what licenses optimizing each group independently, and when it fails the greedy halving in Algorithms 2 and 4 can discard the half-interval containing the optimum. We therefore assess the procedure empirically rather than analytically, against unstructured search at matched compute (Appendix A.22).

BINARY SEARCH TERMINATION

The binary greedy interval reduction (Algorithm 2) terminates after a fixed, data-independent number of halvings per group. Convergence to the within-interval optimum additionally requires quasi-unimodality of the per-group objective; our own motivating measurements (Figure 8) show that this can fail, so the result below is a termination bound, not an optimality guarantee. For an interval $[l, u]$ with width $w = u - l$, the number of iterations to reach width τ is:

$$N_{\text{iter}} = \lceil \log_2(w/\tau) \rceil. \tag{26}$$

BUDGET CONSTRAINT SATISFACTION

Phase II (Algorithm 3) guarantees strict budget constraint satisfaction through dynamic upper bound computation:

$$u_{\text{eff}} = \min\left(u_i, \left\lfloor \frac{\beta_{\text{rem}}}{s} \right\rfloor\right), \tag{27}$$

where β_{rem} is the remaining budget and $s = L/g^*$ is the number of layers per group.

A.21.2 TIME COMPLEXITY ANALYSIS

PHASE I: COARSE-GRAINED EXPLORATION

Algorithm 1 iterates over all candidate groupings $\mathcal{G}$, and for each grouping, Algorithm 2 performs binary interval reduction for each group:

$$\mathcal{C}_{\text{Phase I}} = O\left(\sum_{g \in \mathcal{G}} g \times \log_2\left(\frac{r_{\max} - r_{\min}}{\tau}\right) \times T_{\text{eval}}\right). \tag{28}$$

where $\mathcal{G}$ is the set of candidate groupings and T_{eval} denotes the time for one model training and evaluation cycle.

PHASE II: FINE-GRAINED OPTIMIZATION

Algorithm 3 processes g^* groups sequentially, and Algorithm 4 performs binary search within each refined interval:

$$\mathcal{C}_{\text{Phase II}} = O\left(g^* \times \log_2(\tau) \times T_{\text{eval}}\right). \tag{29}$$

TOTAL COMPLEXITY

Summing the query counts of the two phases at the operating point used throughout this paper gives:

$$\begin{aligned} N_{\text{total}} &= N_{\text{Phase I}} + N_{\text{Phase II}} \\ &= 236 + 96 = 332 \text{ evaluations.} \end{aligned} \tag{30}$$

In closed form, the total number of validation queries is bounded by:

$$\begin{aligned} \mathcal{C}_{\text{total}} \;&=\; \mathcal{O}\!\left(|\mathcal{G}| \cdot L \cdot \log_2 \frac{r_{\max} - r_{\min}}{\tau} \;+\; g^* \log_2 \tau \right) \\ &\ll\; |\mathcal{S}_{\mathcal{R}}| = (r_{\max} - r_{\min} + 1)^L, \end{aligned} \tag{31}$$

collapsing the original $(r_{\max} - r_{\min} + 1)^L$-sized configuration space to a polylogarithmic search budget. Including per-evaluation cost, the overall time complexity is:

$$\begin{aligned} \mathcal{C}_{\text{total}} = O\big((|\mathcal{G}| \times g_{\max} + g^*) \\ \times \log_2 \left(\frac{r_{\max} - r_{\min}}{\tau} \right) \times T_{\text{eval}} \big). \end{aligned} \tag{32}$$

A.21.3 COMPARISON WITH ALTERNATIVE APPROACHES

EXHAUSTIVE SEARCH

Exhaustive search evaluates all valid configurations in the solution space:

$$\mathcal{C}_{\text{exhaustive}} = O(|\mathcal{S}_{\mathcal{R}}| \times T_{\text{eval}}) \approx O(10^{58} \times T_{\text{eval}}). \tag{33}$$

Comparison. Our method achieves a speedup factor of:

$$\frac{10^{58}}{332} \approx 3.0 \times 10^{55}. \tag{34}$$

This and the ratio below are evaluation-count ratios. They bound search cost, and say nothing about the quality of the returned solution relative to the exhaustive optimum; the empirical quality comparison at matched budget is Table 4 of the main paper.

SINGLE-PHASE GREEDY ALGORITHM

A single-phase greedy approach iteratively allocates one rank unit to the layer providing maximum performance improvement:

$$\mathcal{C}_{\text{greedy}} = O(\beta_{\text{alloc}} \times L \times T_{\text{eval}}), \tag{35}$$

where $\beta_{\text{alloc}} = \beta - L \times r_{\min}$ is the number of rank units to allocate beyond the minimum.

Comparison. Our method achieves a speedup factor of:

$$\frac{73,728}{332} \approx 222\times \text{ faster.} \tag{36}$$

RANDOM SEARCH

Random search samples configurations uniformly from the solution space and returns the best-performing configuration after a fixed number of evaluations:

$$\mathcal{C}_{\text{random}} = O(N_{\text{samples}} \times T_{\text{eval}}). \tag{37}$$

Comparison. With equal evaluation budget ($N_{\text{samples}} = 332$), random search explores a negligible fraction ($332/10^{58} \approx 10^{-56}$) of the solution space without exploiting architectural structure, and in our experiments it trails the structured search by 4.38% average MSE at the same budget (Table 4 of the main paper).

Table 13: Complexity comparison for the 512-rank budget configuration with $L = 24$, $r_{\min} = 384$, $r_{\max} = 640$, and $\tau = 64$.

Method	Evaluations	Speedup vs. Greedy	Solution Quality
Exhaustive Search	$\sim 10^{58}$	–	Global optimum (intractable)
Single-Phase Greedy	73,728	1×	Local optimum
Random Search (equal budget)	332	222×	Unstructured baseline
Ours (Hierarchical)	332	222×	Best of the three at 332 evals

Table 14: MSE↓ on the financial-regression datasets at the matched 332-evaluation budget. Random Search uniformly samples per-layer rank vectors under the $512\times L$ sum-budget constraint; the best of 332 samples is reported. RS slightly outperforms AdaLoRA's best configuration ($640\to384$), consistent with (Bergstra & Bengio, 2012; Li & Talwalkar, 2019) on Random Search as a strong high-budget baseline; our hierarchical allocation outperforms RS at identical compute on every dataset. Best configuration in bold.

Method (512-rank budget)	NEU	FXE	INV
Uniform LoRA (r=512)	0.0184	0.0197	0.0181
AdaLoRA (640→384)	0.0211	0.0220	0.0188
Random Search (332 evals)	0.0179	0.0192	0.0176
Ours (Group=8)	**0.0172**	**0.0183**	**0.0168**

A.21.4 Summary

In Table 13, our hierarchical allocation strategy achieves the same computational cost as random search while returning the better solution of the two at a matched budget of 332 evaluations, through structured exploration that exploits architectural coherence and employs efficient binary search within each layer group. The Solution Quality column reports observed outcomes for the strategies listed in this table at this budget, not proximity to the intractable global optimum, which is unknown. Appendix A.22 reports the random-search baseline under the same budget.

Appendix A.22: Random-Search Baseline

This section gives the random-search baseline of Table 4 and its full per-dataset results.

Headline-Number Disclosure. The 15.31% to 22.71% MSE reduction over AdaLoRA reported throughout this paper spans AdaLoRA configurations (the strongest $640\to384$ vs the weakest $512\to384$), averaged across NEU, FXE, and INV; per-dataset breakdowns are in Table 3 of the main paper.

A.22.1 Random Search baseline. The random-search baseline (Table 14) uniformly samples per-layer rank vectors $r \in [r_{\min}, r_{\max}]^L$ subject to the sum-budget constraint $\sum_l r_l = 512 \cdot L$ via the standard Dirichlet-rejection procedure of Bergstra & Bengio (2012). Each of 332 samples is fine-tuned under the same fixed 100-epoch protocol used in the main results, and the best validation MSE is reported.

A.22.2 Random Search: full per-dataset table. Table 14 reports Random Search across all three financial-regression datasets at the matched 332-evaluation budget. Random Search slightly outperforms AdaLoRA's best configuration ($640\to384$) on every dataset (NEU: 0.0179 vs. 0.0211; FXE: 0.0192 vs. 0.0220; INV: 0.0176 vs. 0.0188), confirming the high-budget Random-Search literature (Bergstra & Bengio, 2012; Li & Talwalkar, 2019). Our hierarchical allocation outperforms Random Search at identical compute on every dataset.