Polyglotic LoRA (PLoRA): Channel-Aware Language-Routed Low-Rank Adaptation from Spectral Activation Geometry

Anonymous Authors

Abstract

Multilingual parameter-efficient fine-tuning typically relies on a single, shared low-rank update across all languages. However, this rigid approach ignores the fact that internal model representations differ substantially based on a language's script, resource level, and transformer depth, often leading to negative transfer. To address this, we introduce Polyglotic LoRA (PLoRA), a channel-aware, language-routed fine-tuning framework. Rather than applying a uniform update, PLoRA extracts layerwise activations from a frozen backbone (Qwen3-4B-Instruct) and fits spectral tail exponents over multilingual probes. By normalizing these depth signatures against a cross-language reference, the framework dynamically selects sparse support layers and allocates rank budgets based on channel salience and rigorous stability audits. We evaluate this targeted routing approach on a diverse 10-language abstractive summarization benchmark. PLoRA obtains the lowest perplexity across all evaluated languages, nearly halving the macro perplexity from 9.95 (shared LoRA baseline) to 4.80, while simultaneously improving macro BERTScore from 66.06 to 68.20. Ultimately, this implementation demonstrates that geometry-guided PEFT provides a highly interpretable, mathematically bounded alternative to monolithic adapters, exposing both the strict performance advantages of dynamic rank placement and the necessity of layer-selection stability checks.

1 Introduction

Parameter-efficient fine-tuning (PEFT) has become the standard mechanism for specializing large language models without the prohibitive computational cost of updating full backbone architectures (Houlsby et al., 2019; Hu et al., 2022; Dettmers et al., 2023). In multilingual settings, however, rigid PEFT approaches typically force a single, shared adapter to serve diverse languages. This design ignores the reality that internal model representations differ substantially based on a language's script, tokenization density, resource conditions, and representational trajectories across transformer depth. Consequently, forcing high-resource and low-resource languages into the same shared low-rank subspace often induces severe negative transfer (Pfeiffer et al., 2020).The central premise of PLoRA is that adapter placement should not be uniform. Instead, a multilingual model should expose exactly where each language departs from the shared backbone and allocate adapter capacity at those specific layers. We implement this premise as a rigorous six-step pipeline. Steps 1 and 2 define a spectral diagnostic called a *language channel*: for each language λ and layer ℓ, our implementation constructs a centered activation matrix, fits a power-law tail to its singular values, and compares the resulting exponent against a cross-language reference. Step 3 converts the normalized deviation map into sparse support sets, while Step 4 allocates per-layer rank budgets based on channel salience. Finally, Steps 5 and 6 instantiate, train, and evaluate these language-routed LoRA adapters against three matched baselines.This design is directly motivated by two recent archived preprints. SPINAL treats spectral-tail geometry as a depth-resolved diagnostic of localized transformer change, especially in alignment-sensitive layers (Das et al., 2026). NEURAL FOXP2 demonstrates that language behavior can be controlled through sparse, low-rank language-specific neuron and activation subspaces (Saha et al., 2026). PLoRA translates these theoretical insights into a functional multilingual PEFT framework: it uses spectral channel geometry to decide exactly where language-routed low-rank updates should be placed and how much rank each selected layer should receive.This paper is intentionally implementation-centered, treating our execution codebase as the definitive source of truth for model configuration, preprocessing, tensor routing,

hyperparameters, outputs, and diagnostic decisions. Consequently, we focus on the granular details required for strict reproduction: exact language sets, sample sizes, layer indices, adapter targets, optimizer isolation, checkpoint handling, and metric computation. In instances where empirical artifacts present discrepancies, we explicitly report these variances rather than artificially smoothing them into a cohesive but unsupported narrative.Our contributions are:

1. We formalize a spectral activation workflow for deriving language-channel curves from frozen multilingual LLM activations.

2. We implement sparse language-specific adapter placement using normalized channel deviation, support stability checks, and a terminal-window prior.

3. We allocate per-layer LoRA ranks by channel salience and instantiate routed adapters over attention and MLP target matrices.

4. We report an implementation-faithful multilingual summarization evaluation over 10 languages and three placement baselines, demonstrating substantial improvements in macro perplexity and BERTScore.

5. We provide a detailed engineering audit in the appendix, including functions, data flow, configurations, edge cases, and consistency notes for Steps 1 to 6.

2 Pipeline Overview

Backbone and languages. Steps 1 to 4 use `Qwen/Qwen3-4B-Instruct-2507`, a 36-layer causal language model loaded in half precision with a frozen backbone.The channel-measurement uses 12 languages from FLORES+ (Goyal et al., 2022; NLLB Team et al., 2022): Hindi, French, Chinese, Urdu, English, Awadhi, Bengali, Marathi, Dutch, Polish, Sindhi, and South Azerbaijani. Step 5 trains on 10 summarization languages, removing Awadhi and South Azerbaijani while retaining Sindhi. This reduction was necessary because the base model's severe pre-training starvation and vocabulary deficit prevented the formation of the foundational token geometry required to meaningfully adapt these ultra-low-resource scripts via parameter-efficient fine-tuning.

Six-step data flow. The end-to-end pipeline is:

1. **Probe activations.** Load 1,000 probe sentences per language, tokenize to at most 512 tokens, run the frozen model, and mean-pool each layer's hidden states into vectors $h_\ell(x_i) \in \mathbb{R}^d$.

2. **Channel scores.** Center the activation matrix, compute singular values, fit spectral tails, average over five subsamples of 200 prompts, and normalize each language against the multilingual reference.

3. **Support sets.** Select layers with the largest normalized deviation, add the last three layers, and apply a stability fallback when the top-K boundary is within the estimated noise band.

4. **Rank budgets.** Allocate per-language rank mass across selected layers using quadratic channel salience, with ranks clipped to $[4, 64]$.

5. **Training.** Replace selected Qwen attention and MLP linears with language-routed LoRA wrappers and train only active-language adapter weights.

6. **Evaluation.** Reconstruct each model checkpoint, generate summaries on 200 examples per language, read perplexity from Step 5 evaluation artifacts, and compute SacreBLEU and BERTScore.

3 Spectral Language Channels

3.1 Activation Matrices

For each language λ, Step 1 constructs a deterministic probe pool P_λ of $N = 1000$ FLORES+ `devtest` sentences. Our implementation does not shuffle these pools. It registers forward hooks on every transformer block through `ActivationExtractor`, runs the frozen Qwen backbone with `torch.no_grad()`, and stores the block output. Because each sentence prompt generates token-level hidden states of varying lengths, the implementation applies mean pooling across the sequence dimension. This neutralizes prompt length disparities and token-specific semantic variations, isolating a stable, macro-level geometric representation of how a given language utilizes a specific transformer depth. The resulting layerwise matrix is defined as:

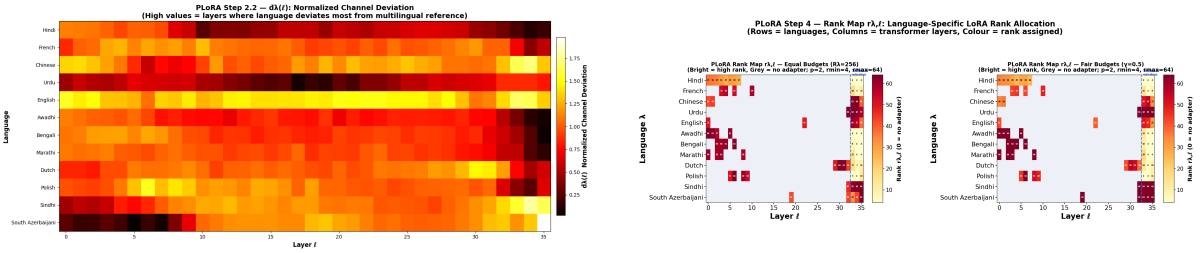

Figure 1: Implementation-derived diagnostics utilized by PLoRA. Left: the Step 2 normalized channel-deviation heatmap $\widetilde{d}_\lambda(\ell)$, where high values mark language-specific depth departures from the cross-language reference. Right: the Step 4 rank map, which converts selected layers and salience values into per-language LoRA ranks.

$$h_{\lambda,\ell}(x_i) = \frac{1}{T_i}\sum_{t=1}^{T_i} H_\ell(x_i)_t.$$

The resulting layerwise matrix is

$$A_{\lambda,\ell} = \begin{bmatrix} h_{\lambda,\ell}(x_1)^\top \\ \cdots \\ h_{\lambda,\ell}(x_N)^\top \end{bmatrix} \in \mathbb{R}^{N\times d}.$$

Step 1 centers the matrix featurewise:

$$\bar{A}_{\lambda,\ell} = A_{\lambda,\ell} - \mathbf{1}\mu_{\lambda,\ell}^\top, \quad \mu_{\lambda,\ell} = \frac{1}{N}\sum_{i=1}^{N} h_{\lambda,\ell}(x_i).$$

The implementation converts half-precision tensors to float before calling `torch.linalg.svdvals`. This avoids numerical failure in SVD while preserving half-precision inference for activation extraction.

3.2 Tail-Exponent Estimation

Let $s_1 \geq \cdots \geq s_r$ be the singular values of $\bar{A}_{\lambda,\ell}$ after removing near-zero entries, with $r \leq \min(N, d)$. Rather than fitting the entire spectrum, the final Step 1 explicitly fixes a tail window from 10% to 90% of available singular values. For indices $k \in [k_{\min}, k_{\max}]$, it fits

$$\log s_k \approx c_{\lambda,\ell} - \beta_{\lambda,\ell}\log k$$

by ordinary least squares. The channel exponent is:

$$\alpha_\lambda(\ell) = \frac{1}{\hat{\beta}_{\lambda,\ell}}.$$

This explicit definition is critical, as our preliminary exploratory configurations employed a different alpha scaling factor, with mean values around 0.71 to 0.83. The final implementation of Steps 1 and 2 utilizes the reciprocal-slope formulation, yielding mean language values ranging from 1.46 to 1.84. We rely on the definitive execution script and the exported `spinal_plora_qwen3_4b_results_12lang.csv` values throughout.

3.3 Subsampling Stability

To avoid over-interpreting a single probe draw, Step 1 computes $S = 5$ subsamples of size $N' = 200$ per language and layer. For each subsample it recomputes the tail exponent, then records

$$\bar{\alpha}_\lambda(\ell) = \frac{1}{S}\sum_{s=1}^{S}\alpha_\lambda^{(s)}(\ell),$$

$$\mathrm{sd}_\lambda(\ell) = \sqrt{\frac{1}{S-1}\sum_{s=1}^{S}\left(\alpha_\lambda^{(s)}(\ell) - \bar{\alpha}_\lambda(\ell)\right)^2}.$$

Our empirical tracking indicates low subsampling dispersion, where, for example, Hindi has mean $\bar{\alpha} = 1.5133$ with standard-deviation range $[0.0056, 0.0275]$, while English has mean 1.8354. Figure 2 summarizes the resulting channel curves and stability bands. Step 1 also computes Sinkhorn distances from each language's activation distribution to English after L2 normalization, using 200 samples and entropic regularization 0.1 through POT (Cuturi, 2013). This geometry metric is diagnostic only and is not used to allocate ranks.

3.4 Cross-Language Deviation

Step 2 constructs a per-layer multilingual reference:

$$\mu(\ell) = \frac{1}{m}\sum_{\lambda=1}^{m}\bar{\alpha}_\lambda(\ell),$$

$$\sigma(\ell) = \sqrt{\frac{1}{m}\sum_{\lambda=1}^{m}\left(\bar{\alpha}_\lambda(\ell) - \mu(\ell)\right)^2 + \varepsilon}.$$

It then defines normalized channel deviation:

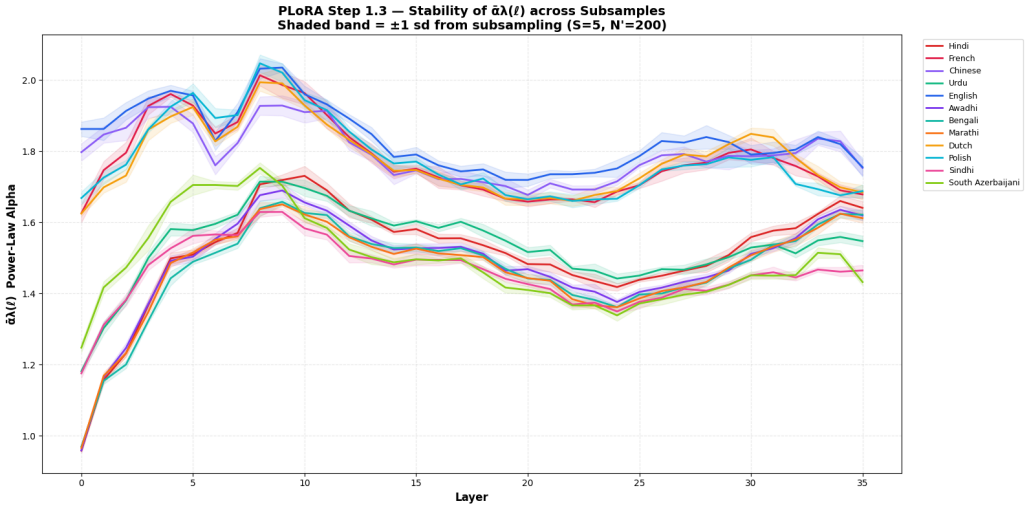

Figure 2: Step 1–2 spectral-channel stability. Solid curves show mean tail exponents across transformer depth and shaded bands show subsampling variation, confirming that the language-channel profiles are not artifacts of a single probe draw.

$$\widetilde{d}_\lambda(\ell) = \frac{|\bar{\alpha}_\lambda(\ell) - \mu(\ell)|}{\sigma(\ell)}.$$

The first five reference means are $[1.3360, 1.4628, 1.5175, 1.6219, 1.6966]$, and the first five dispersions are $[0.3392, 0.2779, 0.2641, 0.2477, 0.2087]$. The maximum deviation layers vary substantially: Hindi and Awadhi peak at layer 0, French at layer 10, Polish at layer 6, Dutch at layer 30, English at layer 33, and Chinese, Urdu, Sindhi, and South Azerbaijani peak in layers 34 or 35. This is the empirical basis for sparse, language-specific adapter placement.

4 Support Sets and Rank Budgets

4.1 Support Selection

Step 3 starts with top-K selection over $\widetilde{d}_\lambda(\ell)$:

$$S_\lambda^{(\alpha)} = \mathrm{TopK}_\ell\left(\widetilde{d}_\lambda(\ell), K_\alpha\right), \quad K_\alpha = 10.$$

Our architecture incorporates a terminal prior with $w = \lfloor 0.1L \rfloor = 3$ for $L = 36$:

$$S_{\mathrm{term}} = \{33, 34, 35\}, \quad S_\lambda = S_\lambda^{(\alpha)} \cup S_{\mathrm{term}}.$$

The stability audit checks whether the gap between the K_α-th and $(K_\alpha + 1)$-th ranked deviation exceeds twice the propagated noise. In Step 3, the noise proxy is:

$$\epsilon_\lambda = \frac{1}{L}\sum_{\ell=0}^{L-1} \frac{\mathrm{sd}_\lambda(\ell)}{\sigma(\ell)}, \quad \widetilde{d}_\lambda(\ell_K) - \widetilde{d}_\lambda(\ell_{K+1}) > 2\epsilon_\lambda.$$

Language	$\lvert S_\lambda \rvert$	Params (M)	PPL
Hindi	11	15.25	2.23
French	7	14.68	4.61
Chinese	5	14.45	8.82
Urdu	4	14.28	3.32
English	5	14.62	4.85
Dutch	7	14.62	6.91
Polish	7	14.39	9.54
Sindhi	4	13.59	3.50
Bengali	7	14.97	1.61
Marathi	7	14.97	2.59

Table 1: Step 5 training languages, final support-set sizes, adapter parameter counts from checkpoint tensors, and PLoRA-alpha perplexity from the exported Step 5 summary. Awadhi and South Azerbaijani are measured in Steps 1 to 4 but are not trained in Step 5.

All 12 languages fail this conservative margin test at $K_\alpha = 10$. The fallback first tries a smoothed channel curve using a width-3 reflected rolling mean. If that does not stabilize the boundary, it searches downward for a smaller K with a lower bound of 4. The final support sets therefore use `smaller_K`: Hindi keeps $K = 8$, while the remaining 11 languages use $K = 4$ before adding the terminal prior. Figure 3 visualizes the selected layers, and Table 1 shows the 10 support sets used in Step 5 training.

4.2 Rank Allocation

Step 4 reads the Qwen configuration directly with `AutoConfig`. The run logs hidden size 2560, 32 attention heads, 8 key-value heads, head dimension 128, intermediate size 9728, and 36 layers. It tar-

Figure 3: Step 3 support-selection map. Green cells mark layers selected by channel-deviation ranking after stability fallback, and the terminal-window prior ensures every language retains the final layers used for generation-sensitive adaptation.

gets seven matrices per selected transformer layer (Vaswani et al., 2017): `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj`.. The parameter cost of a LoRA update for one matrix $W \in \mathbb{R}^{d_{out} \times d_{in}}$ is

$$\#\Delta W = r(d_{in} + d_{out}).$$

For the full target set, the experimental logs record logs 0.229M parameters per layer at rank 4, 0.918M at rank 16, and 3.670M at rank 64.

Given a support set, Step 4 defines salience weights

$$u_{\lambda,\ell} = \max(\widetilde{d}_\lambda(\ell), 10^{-8})^p, \quad p = 2,$$

and allocates a per-language rank budget $R_\lambda = 256$:

$$r_{\lambda,\ell} = \text{clip}\left(\text{round}\left(R_\lambda \frac{u_{\lambda,\ell}}{\sum_{j \in S_\lambda} u_{\lambda,j}}\right), 4, 64\right).$$

The equal-budget allocation passes validation for all 12 measured languages under a 30% tolerance, with total equal-budget LoRA parameters of 173.58M across 12 languages. Step 5 trains the 10-language subset and instantiates 145.83M trainable adapter parameters. Step 4 also implements an optional inverse-resource fairness budget,

$$R_\lambda \propto |D_\lambda|^{-\gamma}, \quad \gamma = 0.5,$$

clipped to $[128, 512]$. This fair-budget setting is analyzed but not used for the main Step 5 training run.

Figure 4 shows the expected monotone relationship between channel salience and assigned rank, with the strongest deviations receiving the highest adapter capacity after clipping. Figure 5 adds the 3D allocation surface, where height and color encode assigned rank across languages and depth.

5 Language-Routed Training

5.1 Adapter Modules

Step 5 implements two core classes. `PLoRAAdapter` contains a LoRA pair (A, B) with $A \in \mathbb{R}^{r \times d_{in}}$ initialized by Kaiming uniform and $B \in \mathbb{R}^{d_{out} \times r}$ initialized to zero. Its forward pass computes

$$\Delta y = \frac{\alpha_{\text{LoRA}}}{r} BAx, \quad \alpha_{\text{LoRA}} = 16.$$

The adapter computation casts x to float32 for matrix multiplication and casts the result back to the original dtype. `PLoRALinear` wraps a frozen base `nn.Linear`, stores a `ModuleDict` of per-language adapters, and routes by the class variable `_active_language`. If no adapter exists for the active language at that layer, the wrapper returns the frozen base output unchanged.

`PLoRAModel` loads Qwen in float16 with `device_map='auto'`, freezes the backbone, optionally enables gradient checkpointing with `use_reentrant=False`, builds a layer-to-language-to-rank lookup, and replaces only selected target matrices. The run injects 133

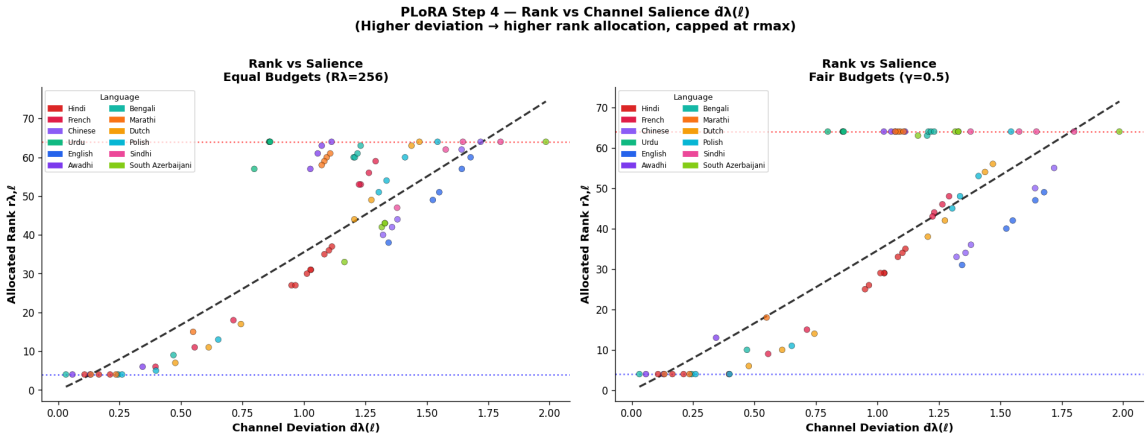

Figure 4: Step 4 rank allocation as a function of channel salience. The plotted allocations verify that the quadratic salience rule concentrates rank on high-deviation layers while keeping ranks within the configured clipping range.

`PLoRALinear` wrappers across 19 transformer blocks. It logs 145,825,792 trainable adapter parameters and 4,022,468,096 frozen parameters.

5.2 Data and Optimization

The Step 5 dataset class builds chat-formatted instruction examples with system, user, and assistant turns. It tokenizes the full conversation and separately tokenizes the prefix consisting of system and user messages. Labels for prefix tokens are set to -100, so both training loss and perplexity score only the assistant response tokens:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \log p_\theta(y_t \mid y_{<t}, x, \lambda),$$

where $\mathcal{T}$ excludes prompt positions.

The data-loading cell reports 10,000 training examples for each of the 10 training languages, 1,000 evaluation examples for all except Dutch, which has 862. Sources include XSum for English (Narayan et al., 2018), WikiLingua for French and Dutch (Ladhak et al., 2020), XLSum-derived corpora for Hindi, Chinese, Urdu, and Marathi (Hasan et al., 2021), and language-specific OPUS-derived summarization corpora for Polish, Sindhi, and Bengali (Tiedemann, 2012).

Training uses separate AdamW optimizers (Loshchilov and Hutter, 2019) per language to avoid cross-language optimizer-state leakage. Our configuration sets learning rate 2×10^{-4}, weight decay 0.01, warmup ratio 0.1, gradient clipping 1.0, maximum sequence length 4096, three epochs, and checkpointing every 5,000 global optimizer steps. The source cell declares `BATCH_SIZE=2`, but the executed configuration log and dataloader counts imply batch size 4 because 10,000 examples produce 2,500 batches per epoch. We therefore treat batch size 4 as the executed run configuration.

5.3 Stable-Rank Regularization

Step 5 optionally regularizes selected layers using a differentiable stable-rank proxy. For a hidden-state batch matrix H, the regularizer approximates:

$$\text{sr}(H) = \frac{\|H\|_F^2}{\sigma_1(H)^2}$$

with three power iterations. It computes frozen-backbone targets $\text{sr}^*_{\lambda,\ell}$ from 16 probe examples per language, then penalizes deviations during training:

$$\mathcal{R}_{\text{chan}}(\lambda) = \sum_{\ell \in S_\lambda} \Delta^2_{\lambda,\ell},$$
$$\Delta_{\lambda,\ell} = \log(\text{sr}(H_{\lambda,\ell}) + 10^{-8}) - \log(\text{sr}^*_{\lambda,\ell} + 10^{-8}).$$

The training objective is:

$$\min_{\phi_\lambda} \mathbb{E}_{(x,y)\sim D_\lambda} \big[\mathcal{L}_{\text{CE}}(f_{\theta,\phi_\lambda}(x), y) + 0.01\, \mathcal{R}_{\text{chan}}(\lambda)\big],$$

with the backbone θ frozen. A critical implementation detail ensures that activation hooks for the regularizer no longer detach tensors, allowing the regularization signal to backpropagate fully into the adapter weights.

6 Baselines and Evaluation

6.1 Baselines

Step 5 constructs three matched baselines:

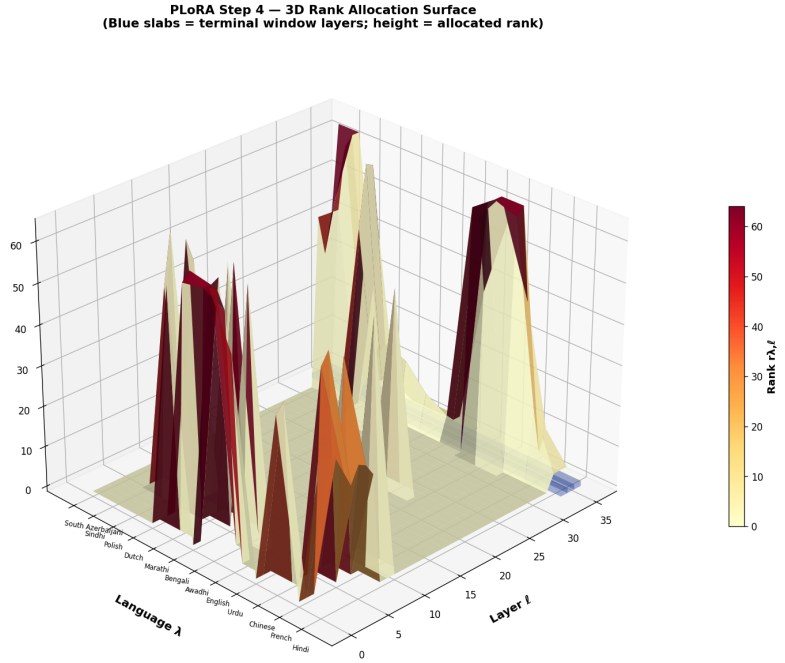

Figure 5: Step 4 3D rank-allocation surface. Surface height and color encode the allocated LoRA rank $r_{\lambda,\ell}$, while the blue slabs identify terminal-window layers retained by the placement prior.

- **S-LoRA.** A single shared adapter is placed on the union of all PLoRA support layers. The shared rank is chosen from the average PLoRA rank mass divided by the union size.

- **Uniform-Place.** Each language uses only the final three terminal layers, with uniform rank matched to that language's total rank mass.

- **Random-Place.** Each language selects the same number of layers as PLoRA, but layers are sampled uniformly at random with seed 42 and uniform rank.

Baselines use the same adapter classes, target modules, learning rate, LoRA scale, and gradient clipping. The baseline training loop omits geometry regularization and scheduler logic.

6.2 Validation of the Channel-Placement Hypothesis

The most striking BERTScore improvement occurs in Marathi (+11.7 over Uniform-Place, +7.0 over S-LORA). For this low-resource Devanagari language, uniform terminal placement collapses (52.27), whereas PLORA's early-layer adapters maintain a robust 63.95. This directly validates our hypothesis: languages with early-layer channel deviations cannot be served by terminal-only placement. Furthermore, Polish yields the highest S-LORA perplexity (22.78)—likely due to morphological complexity straining shared capacity—but achieves a highly stable 9.54 under PLORA-alpha's dedicated $K = 7$ support layers, successfully maintaining competitive likelihood.

6.3 Step 6 Generation and Metrics

Step 6 evaluates reconstructed checkpoints iteratively. It reads actual ranks directly from `lora_A.weight` tensors, extracts article texts using standard prompt patterns, and groups the clean dataset by language. Summaries are generated using a Qwen chat template, left padding, greedy decoding, generation batch size 8, and a maximum of 80 tokens (stripping `<think>` segments). Utilizing Step 5's perplexity logs, Step 6 computes SacreBLEU (Post, 2018) and multilingual BERTScore F1 (Zhang et al., 2020) across 200 examples per language. We report the successfully executed evaluation artifacts (`plora_step6_macro_summary.csv` and `plora_step6_final_benchmarks.csv`), explicitly bypassing an earlier diagnostic file containing zeroed BLEU and empty COMET fields.

7 Results

Table 2 and Figure 6 present the main evaluation. PLoRA-alpha achieves the lowest macro perplexity (4.80 vs. 9.95 for S-LoRA) and highest macro BLEU and BERTScore, improving semantic sim-

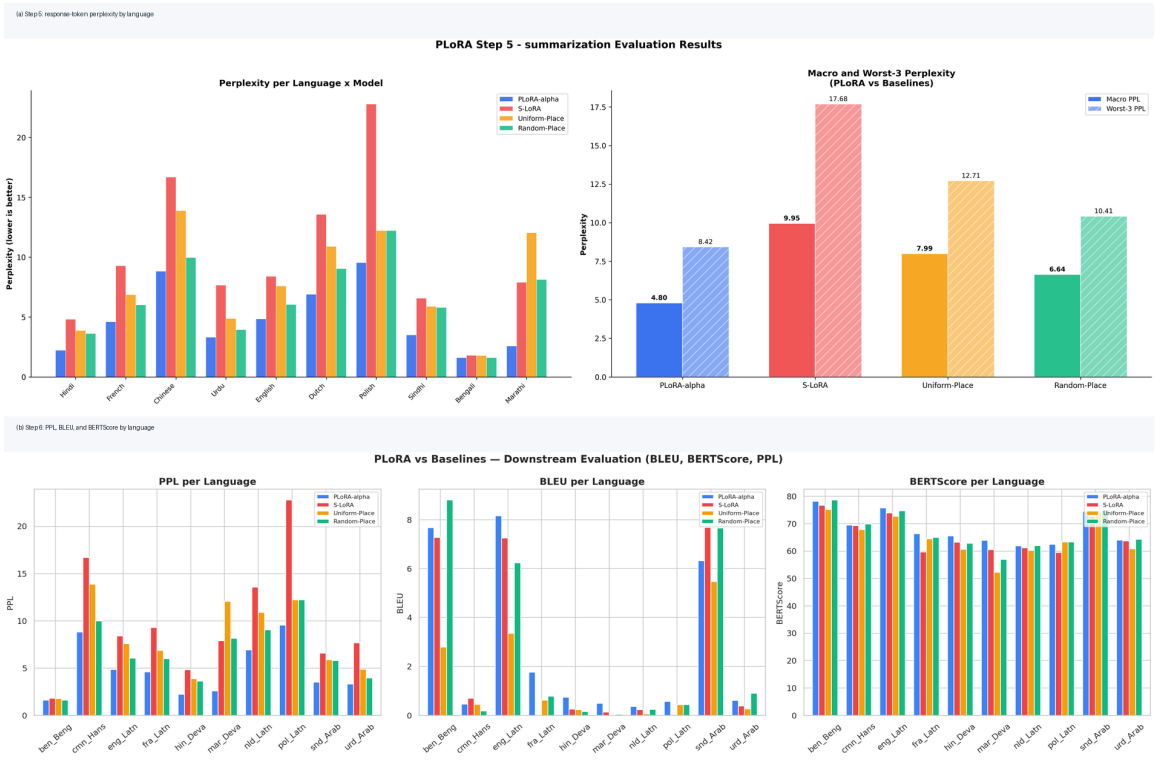

Figure 6: Compact Step 5–6 result audit. The top panels summarize response-token perplexity by language and macro/worst-case perplexity, while the lower panels compare final Step 6 PPL, BLEU, and BERTScore across languages.

Model	PPL ↓	BLEU ↑	BERTScore ↑
PLoRA-alpha	**4.80**	**2.72**	**68.20**
S-LoRA	9.95	2.40	66.06
Uniform-Place	7.99	1.37	65.16
Random-Place	6.64	2.55	67.25

Table 2: Macro averages from `plora_step6_macro_summary.csv`. PLoRA-alpha improves macro perplexity by 51.8% relative to S-LoRA, 39.9% relative to Uniform-Place, and 27.7% relative to Random-Place.

ilarity by 2.14 points over S-LoRA. Per-language analysis reveals PLoRA-alpha minimizes perplexity across all 10 languages, while achieving top BLEU in six and top BERTScore in four. While baselines occasionally match overlap metrics for languages like Bengali and Sindhi, this confirms that channel-guided training consistently improves likelihood, whereas surface metrics remain sensitive to decoding and reference styles. Finally, the inherently low absolute BLEU values are standard for short abstractive summarization across diverse scripts. The stable BERTScore range (61.9–78.1) confirms that Step 6 functions as a robust semantic evaluation rather than a strict translation benchmark.

8 Conclusion

PLoRA implements a practical bridge between representation geometry and multilingual PEFT. The six-step pipeline measures spectral channel fingerprints, selects sparse language-specific support layers, allocates ranks by salience, trains only routed adapters, and evaluates checkpointed models under matched baselines. The empirical evidence demonstrates that channel-guided placement produces strong likelihood improvements and interpretable layer choices, while also exposing important engineering challenges: support stability, memory management, checkpoint reconstruction, and metric consistency. These findings support further study of geometry-guided adapter allocation for multilingual LLMs.

9 Limitations

Our proposed framework presents several limitations that motivate future research extensions. First, the structural layer selection mechanism utilizes a conservative stability fallback that may underselect optimal adapter locations when representation deviation scores are densely clustered across model depth. Second, the routing architecture currently presumes oracle language labels during both training and inference. While a preliminary soft-routing analysis evaluates basic mixed-language conditions, a comprehensive robustness benchmark under noisy language identification or spontaneous code-switching is left for future work.

Third, regarding metric selection, we use BERTScore rather than COMET because the task is abstractive summarization, not machine translation. COMET is calibrated against human translation quality judgments and is not appropriate for summary evaluation. BERTScore with a multilingual BERT backbone captures semantic similarity across scripts and is commonly used for multilingual generation evaluation. Finally, while BERTScore serves as a reliable automated proxy for cross-lingual semantic alignment, it remains an algorithmic evaluation that cannot fully replace direct human qualitative judgments of summary quality, fluency, and informational relevance.

References

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020. MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. In *Advances in Neural Information Processing Systems*.

Marco Cuturi. 2013. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems*.

Arion Das, Partha Pratim Saha, Aman Chadha, Vinija Jain, and Amitava Das. 2026. SPINAL: Scaling-law and preference integration in neural alignment layers. *arXiv preprint arXiv:2601.06238*. Archived preprint.

Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, Marc'Aurelio Ranzato, Francisco Guzmán, and Angela Fan. 2022. The FLORES-101 evaluation benchmark for low-resource and multilingual machine translation. *Transactions of the Association for Computational Linguistics*, 10:522–538.

Tahmid Hasan, Abhik Bhattacharjee, Md. Saiful Islam, Kazi Mubasshir, Yuan-Fang Li, Yong-Bin Kang, M. Sohel Rahman, and Rifat Shahriyar. 2021. XL-Sum: Large-scale multilingual abstractive summarization for 44 languages. In *Findings of ACL-IJCNLP*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Faisal Ladhak, Esin Durmus, Claire Cardie, and Kathleen McKeown. 2020. WikiLingua: A new benchmark dataset for cross-lingual abstractive summarization. In *Findings of EMNLP*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! Topic-aware convolutional neural networks for extreme summarization. In *Proceedings of EMNLP*.

NLLB Team, Marta R. Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, Anna Sun, Skyler Wang, Guillaume Wenzek, and others. 2022. No language left behind: Scaling human-centered machine translation. *arXiv preprint arXiv:2207.04672*.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation*.

Anusa Saha, Tanmay Joshi, Vinija Jain, Aman Chadha, and Amitava Das. 2026. Neural FOXP2: Language specific neuron steering for targeted language improvement in LLMs. *arXiv preprint arXiv:2602.00945*. Archived preprint.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in OPUS. In *Proceedings of LREC*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating text generation with BERT. In *International Conference on Learning Representations*.

A Detailed Step-by-Step Engineering Walkthrough

A.1 Step 1: Probe Pools and Spectral Channel Curves

Objective. Step 1 measures a per-language, per-layer activation geometry curve. Its input is a set of equal-sized language probe pools and a frozen Qwen3-4B-Instruct model. Its output is $\bar{\alpha}_\lambda(\ell)$ and $\mathrm{sd}_\lambda(\ell)$ for 12 languages and 36 layers.

Technical workflow. The initialization script configures `transformers`, `datasets`, `accelerate`, POT, `tqdm`, `seaborn`, and `scipy`. It logs into Hugging Face through a Kaggle secret. It defines 12 FLORES+ language codes, `N_SAMPLES=1000`, `MAX_LENGTH=512`, `N_SUBSAMPLES=5`, `SUBSAMPLE_SIZE=200`, `KMIN_FRAC=0.10`, and `KMAX_FRAC=0.90`. It loads `openlanguagedata/flores_plus` for each language with split `devtest` and keeps the first 1,000 texts.

Code-level explanation. `ActivationExtractor` registers one forward hook per transformer layer and stores `output[0].detach()`. `extract_activations` tokenizes each sentence, truncates to 512 tokens, runs the frozen model, mean-pools token activations, and returns a dictionary `layer -> Tensor[N, hidden_dim]`. `compute_power_law_alpha` centers a layer matrix, computes singular values, filters values below 10^{-10}, extracts the 10% to 90% tail window, fits the log spectrum with `np.polyfit`, and returns $1/\hat{\beta}$. `compute_alpha_stability` samples five random subsets of size 200 and returns mean and sample standard deviation.

Algorithmic details. The key tensor transformation is $[1, T, d] \to [T, d] \to [d]$ for each sentence and layer. The spectral fit uses sorted singular values from the centered $[N, d]$ matrix. The reciprocal slope definition is the final alpha convention.

Data flow. FLORES+ text becomes tokenizer input IDs and attention masks, then hidden states, then mean activation vectors, then centered matrices, then singular spectra, then alpha curves. The Step 1 output is kept in memory and later written with Step 2 fields to `spinal_plora_qwen3_4b_results_12lang.csv`.

Design decisions. Equal N avoids resource-size confounds. Deterministic first-1,000 selection avoids shuffling variance. The tail window avoids fitting the bulk and near-zero end of the spectrum. Subsampling provides a noise estimate for later support stability.

Experimental insights. The final means are Hindi 1.5133, French 1.7785, Chinese 1.7947, Urdu 1.5361, English 1.8354, Awadhi 1.4858, Bengali 1.4694, Marathi 1.4710, Dutch 1.7762, Polish 1.7796, Sindhi 1.4593, and South Azerbaijani 1.4904. Extraction throughput varies by language, with 1,000 examples processed in about 1.2 to 1.6 minutes for several Latin and Han runs and longer for some scripts in the initial logs.

Challenges and edge cases. Our implementation manages SVD exceptions by returning 0.0. It converts float16 matrices to float32 before SVD. Sinkhorn diagnostics emit NumPy overflow warnings during norms, but the execution logs continue to report mean distances.

Final outcome. Step 1 produces stable channel curves that Step 2 normalizes into deviation maps.

A.2 Step 2: Cross-Language Reference and Deviation Maps

Objective. Step 2 converts raw alpha curves into comparable deviation scores. Its input is $\bar{\alpha}_\lambda(\ell)$ and $\mathrm{sd}_\lambda(\ell)$. Its output is $\mu(\ell)$, $\sigma(\ell)$, and $\widetilde{d}_\lambda(\ell)$.

Technical workflow. Our implementation loops over all 36 layers, gathers alpha means across the 12 languages, computes the mean and dispersion, then computes absolute z-style deviations for each language and layer.

Code-level explanation. Cell 20 creates NumPy arrays `mu` and `sigma`. Cell 22 creates `d_tilde`. The best layer per language is obtained with `np.argmax(d_tilde[lc])`.

Algorithmic details. The dispersion uses a small epsilon, 10^{-8}, under the square root. This prevents division by zero when cross-language alphas nearly coincide.

Data flow. The deviation map is both visualized and exported. The same CSV stores alpha means, alpha standard deviations, deviation scores, Sinkhorn distances, μ, and σ.

Design decisions. Normalization by layer matters because raw alpha magnitude varies across depth. A high deviation at a layer therefore means language-specific departure relative to other languages at the same depth, not merely a large raw alpha.

Experimental insights. Max-deviation layers are early for Hindi and Awadhi, mid-depth for French and Polish, late for English, Dutch, Chinese, Urdu, Sindhi, and South Azerbaijani. Mean Sinkhorn distance to English is highest for Bengali at 0.6554 and Marathi at 0.6506, and lowest for Chinese at 0.2171.

Challenges and edge cases. Our preliminary exploratory codebase and the final implementation utilize distinct alpha scaling factors. The final reciprocal-slope is used because it is explicitly labeled as aligned with PLoRA Steps 1 and 2.

Final outcome. Step 2 produces the adapter-placement salience map consumed by Step 3.

A.3 Step 3: Sparse Support Sets

Objective. Step 3 decides which layers receive a language adapter. Its input is the Step 1 and 2 CSV. Its output is `plora_step3_support_sets.json`, plus tabular summaries and figures.

Technical workflow. Our execution pipeline imports alpha means, alpha standard deviations, deviation scores, μ, and σ. It computes top-K layers with `K_ALPHA=10`, adds terminal layers [33, 34, 35], checks selection margins, applies fallback logic, and then computes cross-language overlap statistics.

Code-level explanation. `top_k_layers` ranks layer indices by descending deviation and returns selected layers in ascending order. `build_support_set` returns the final union, the alpha-only set, terminal set, and overlap. `margin_condition` compares the boundary gap against 2ϵ. `smooth_alpha_curve` uses `uniform_filter1d` with reflected boundaries. `find_min_stable_k` searches downward from 10 to 4.

Algorithmic details. The fallback chooses smoothing only if the smoothed deviation map satisfies the margin condition at $K = 10$. Otherwise it uses the smaller-K support. In the executed output, all languages use smaller-K.

Data flow. Step 2 CSV rows become per-language arrays. These become top-K lists, then terminal-unioned support sets, then final support sets in JSON and CSV. Step 4 reads the JSON.

Design decisions. The terminal prior is motivated by the SPINAL interpretation derived from our experimental analysis, which indicates that final layers are often important for alignment and decision formation. The smaller-K fallback trades coverage for stability.

Experimental insights. Universal layers selected by all languages are exactly [33, 34, 35]. Majority layers at a 75% threshold are also [33, 34, 35]. Mean final support size after fallback is 6.3, with minimum 4 and maximum 11. Mean Jaccard similarity across language support sets is 0.429.

Challenges and edge cases. All 12 languages fail the original $K = 10$ stability test. This means support selection is sensitive at the boundary, not that the deviation map is unusable. The execution logs capture strategy and stability metadata in the JSON for auditability.

Final outcome. Step 3 produces stable sparse supports for Step 4 rank allocation.

A.4 Step 4: Channel-Salience Rank Budgeting

Objective. Step 4 converts layer supports into concrete LoRA ranks and parameter counts. Its inputs are the Step 1 and 2 CSV and Step 3 JSON. Its outputs are `plora_step4_rank_budgets.json`, `plora_step4_rank_budgets.csv`, and `plora_step4_param_summary.csv`.

Technical workflow. Our implementation initializes the model dimensions to construct target-matrix catalogs across attention-only, MLP-only, and full-layer modes. Following target selection, the pipeline establishes rank hyperparameters, executes the rank allocation, validates the parameter budgets, and exports the finalized configuration artifacts.

Code-level explanation. `load_model_arch` reads Hugging Face config and falls back to hardcoded Qwen3-4B defaults. `lora_param_cost_per_layer` sums $r(d_{in}+d_{out})$ across target matrices. `allocate_ranks` computes squared deviation weights, normalizes by the support-set sum, rounds, and clips. `validate_allocation` checks rank bounds and

whether total rank mass is within 30% of the budget.

Algorithmic details. The use of $p = 2$ sharpens rank toward high-salience layers without collapsing all capacity to a single layer. Clipping protects memory by capping ranks at 64 and avoids zero adaptation by enforcing rank at least 4.

Data flow. Support-set JSON and deviation maps become rank maps. Rank maps become dense matrices for visualization and serialized dictionaries for Step 5 adapter injection.

Design decisions. The main run uses equal budgets to keep the experimental comparison controlled. The fairness option is implemented and shows how low-resource languages would receive larger rank budgets, for example Awadhi and Sindhi are clipped to 512 under $\gamma = 0.5$.

Experimental insights. Equal-budget validation passes for all 12 languages. Fair-budget validation passes for 9 of 12 languages because clipping makes Awadhi, Sindhi, and South Azerbaijani miss their inflated rank-mass targets by more than 30%.

Challenges and edge cases. Integer rounding and clipping make exact rank-budget equality impossible. Our implementation therefore validates within numerical tolerance rather than demanding strict equality.

Final outcome. Step 4 produces the exact rank maps used to instantiate PLoRA in Step 5.

A.5 Step 5: Language-Routed Adapter Training

Objective. Step 5 trains adapters on task data while keeping the Qwen backbone frozen. Its inputs are Step 3 support sets, Step 4 ranks, task datasets, and Step 1 and 2 alpha curves. Its outputs are adapter checkpoints, evaluation JSON, resource logs, training history, figures, and the Step 5 summary CSV.

Technical workflow. Our execution pipeline automates the entire experimental workflow in three distinct phases. First, it initializes the architecture by ingesting support sets and rank maps to construct the routed LoRA wrappers and isolate per-language optimizers. Second, it processes the summarization datasets with strict prompt-label masking, builds the dataloaders, and computes stable-rank targets. Finally, it trains the primary PLoRA framework alongside baseline architectures, evaluates intrinsic perplexity, and exports all diagnostic figures and checkpoint manifests.

Code-level explanation. `PLoRAAdapter` implements one adapter pair. `PLoRALinear` wraps a frozen linear layer and uses `_active_language` as routing state. `PLoRAModel` replaces selected target matrices and exposes `get_adapter_params`. `TaskInstructionDataset` applies the chat template and masks prompt tokens. `train_plora` shuffles all language batches together, routes each batch, computes cross-entropy plus optional regularization, clips gradients, steps the language-specific optimizer, and records gradient norms. `compute_perplexity` multiplies batch loss by the number of unmasked response tokens and reports $\exp$ of the token-weighted mean loss.

Algorithmic details. Only adapter parameters for the active language receive optimizer updates. This avoids shared Adam moments across languages. The terminal gradient footprint G_{term} is computed as the fraction of mean layer-gradient energy in layers 33 to 35.

Data flow. Task records become chat strings, then token IDs, attention masks, and masked labels. During training, an input batch sets the active language, flows through frozen Qwen plus selected adapters, computes loss on assistant tokens, and updates only adapter tensors for that language. Checkpoints store adapter tensors only.

Design decisions. The full target set is used to allow both attention and feed-forward adaptation. Per-language optimizers prevent optimizer-state interference. Disk-cached baseline training and deletion of backbones are used to manage GPU memory.

Experimental insights. The main model logs 133 injected wrappers across 19 blocks and 896 trainable parameter tensors. Per-language adapter counts range from 13.59M for Sindhi to 15.25M for Hindi. PLoRA-alpha perplexity is best for every language in the Step 5 summary.

Challenges and edge cases. Our implementation features GPU resource profiling, intermediate state checkpointing, a dedicated VRAM recovery mechanism, explicit device synchronization during adapter restoration, and a critical refinement to the geometry-regularization hooks to prevent unintended tensor detachment. The exported Step

5 CSV has a train-sample field of 0, while the dataset-loading cell logs 10,000 training examples per language. We treat this as a summary-table serialization discrepancy.

Final outcome. Step 5 produces trained PLoRA and baseline adapter checkpoints consumed by Step 6.

A.6 Step 6: Final Benchmarking

Objective. Step 6 evaluates trained checkpoints under a common generation and metric protocol. Its inputs are Step 5 checkpoints, Step 5 PPL results, and the clean summarization dataset. Its outputs are final benchmark CSVs and figures.

Technical workflow. The evaluation pipeline validates the checkpoint structure, reconstructs the custom PLoRA architecture and rank maps, and prepares the evaluation dataset. It then generates the target translations, computes both BLEU and BERTScore, integrates these findings with our prior perplexity metrics, and exports the final data tables and comparative figures.

Code-level explanation. `_read_adapter_lang_keys` checks checkpoint language keys. `_rank_maps_from_checkpoint` reads ranks directly from `lora_A.weight` tensors. `_extract_article` recovers article text from prompt fields. `_strip_thinking` removes Qwen thinking blocks. The main loop loads one model at a time, routes to each language, generates with greedy decoding, computes metrics, appends rows, deletes the model, and clears CUDA memory.

Algorithmic details. SacreBLEU is corpus-level over generated summaries and references. BERTScore uses multilingual BERT F1 and is scaled by 100. Perplexity is not recomputed in Step 6; it is read from Step 5 artifacts.

Data flow. Checkpoint tensors determine the reconstructed model architecture. Dataset prompts provide source articles and references. Generated summaries become metric inputs. Rows are saved to `plora_step6_final_benchmarks.csv`; macro means are saved to `plora_step6_macro_summary.csv`.

Design decisions. Extracting rank maps from checkpoints avoids stale JSON mismatches. Batched generation with left padding improves speed. Greedy decoding controls randomness.

Experimental insights. PLoRA-alpha achieves macro PPL 4.80, BLEU 2.72, and BERTScore 68.20. It is the best model by PPL for all 10 languages, by BLEU for 6 languages, and by BERTScore for 4 languages.

Challenges and edge cases. The Step 6 implementation contains comments about Sindhi rank-map absence, but Step 5 metrics and checkpoint parameter counts include Sindhi adapters. The final benchmark CSV is therefore the reliable result artifact.

Final outcome. Step 6 produces the reported final comparison and closes the Plora pipeline.

B Discussion

Why channel placement helps. The channel maps reveal that languages do not concentrate at one universal region of the network. Some Devanagari and Bengali-script languages have strong early-layer deviations, while English, Chinese, Sindhi, and South Azerbaijani peak at late layers. Uniform terminal placement misses these early and middle signals, while shared LoRA collapses language-specific rank needs into one adapter. PLoRA keeps the terminal prior but adds language-dependent nonterminal support, explaining why it improves likelihood across all evaluated languages.

Why stability matters. The fact that all 12 initial top-10 selections fail the margin condition is not a negative result. It is a useful reproducibility signal. It says that many neighboring layers have similar deviation scores once subsampling uncertainty is considered. The fallback prevents the method from pretending that a fragile top-10 boundary is exact. In the final supports, several languages retain only four or five layers after fallback and terminal addition, yet still receive enough rank through Step 4 to train competitive adapters.

Resource behavior. The full 10-language PLoRA checkpoint has 145.83M adapter parameters, about 3.5% of the frozen 4.02B-parameter backbone reported by the run. This is larger than minimal LoRA because adapters are placed over both attention and MLP matrices and ranks can reach 64. The benefit is high per-language capacity on selected layers, while the cost is larger adapter storage and more complex routing.