2-D Loss Function for a TinyLoRA-post-trained LLM

Bhaskar Krishnamachari

February 20, 2026

TinyLoRA shows that post-training updates for large language models can be compressed to an extremely small number of trainable scalars while preserving strong downstream behavior [1]. This report studies the supervised fine-tuning (SFT) loss induced by such a compressed parameterization in the smallest nontrivial case, where exactly two parameters are trainable. In this two-dimensional setting, the objective can be evaluated directly on a grid, which makes the geometry of the loss surface visible.

Two-Parameter TinyLoRA Formulation

For each adapted linear module ℓ, TinyLoRA freezes the base weight matrix and applies a structured low-rank perturbation. Let the frozen SVD factors be

$$W_\ell^{(0)} \approx U_\ell \operatorname{diag}(S_\ell)\, V_\ell^\top. \tag{1}$$

The TinyLoRA update in this implementation is

$$\Delta W_\ell(\theta) = \frac{\alpha}{u} \Big(U_\ell \operatorname{diag}(S_\ell)\Big) \left(\sum_{k=1}^{u} v_{g(\ell),k}\, P_{\ell,k}\right) V_\ell^\top, \tag{2}$$

where $P_{\ell,k}$ are frozen random projection tensors and $g(\ell)$ maps module ℓ to a tie-sharing group. The trainable coefficients are the scalars $v_{j,k}$, where $j \in \mathcal{G}$ indexes tie groups and $k \in \{1, \dots, u\}$ indexes coefficients within a group. Thus, u is the per-group projection dimension (not the total budget), and the full trainable vector is the concatenation

$$\theta = \operatorname{vec}\big([v_{j,k}]_{j\in\mathcal{G},\, k=1}^{u}\big) \in \mathbb{R}^b, \qquad b = |\mathcal{G}|\, u. \tag{3}$$

Equation (2) is a per-module update rule; the total number of learnable parameters appears in $b = |\mathcal{G}|u$. In this report, the scan uses budget $b = 2$, so

$$\theta = (\theta_1, \theta_2) \in \mathbb{R}^2. \tag{4}$$

All other model parameters are frozen. This reduces post-training to minimizing a scalar field over $\mathbb{R}^2$:

$$\theta^\star = \arg\min_{\theta\in\mathbb{R}^2} \mathcal{L}(\theta). \tag{5}$$

Loss Construction

The loss is computed on GSM8K training examples using the SFT convention implemented in `scripts/analysis/scan_loss_surface.py`. For each example, the prompt tokens are masked from supervision and only answer tokens contribute to next-token cross-entropy. If $\mathcal{D}$ is the selected sample set and $\mathcal{T}(x)$ denotes supervised target positions after shifting, the evaluated loss is

$$\mathcal{L}(\theta) = \frac{1}{N_{\text{tok}}} \sum_{x \in \mathcal{D}} \sum_{t \in \mathcal{T}(x)} - \log p_{\phi_0 + \Delta\phi(\theta)}(y_t \mid x_{<t}) \,, \tag{6}$$

with $N_{\text{tok}} = \sum_{x \in \mathcal{D}} |\mathcal{T}(x)|$. No gradient update is performed during the scan; instead, the two parameters are set directly at each grid point and the loss is measured.

Grid and Reproducibility Settings

The 2-D landscape is sampled on a Cartesian grid with

$$\theta_1, \theta_2 \in \{-0.05 + i\tfrac{0.10}{19}\}_{i=0}^{19}, \tag{7}$$

which yields a 20×20 scan (400 points). Table 1 records the main run settings.

Setting	Value
Model	`Qwen/Qwen2.5-7B-Instruct`
Trainable budget	$b = 2$ scalars
Grid size	20×20
Range per axis	$[-0.05, 0.05]$
Dataset slice	GSM8K train, limit 128
Batch size	4
Max sequence length	512
Dtype	bfloat16
Seed	1
Codebase	`https://github.com/anrgusc/tinylora`

Table 1: Parameters needed to reproduce the reported loss-surface scan.

Generated Plots

The plotting step pivots `loss_surface.csv` into a 20×20 matrix and produces a heatmap and contour visualization saved as `loss_surface.png`. Figure 1 shows this generated output.

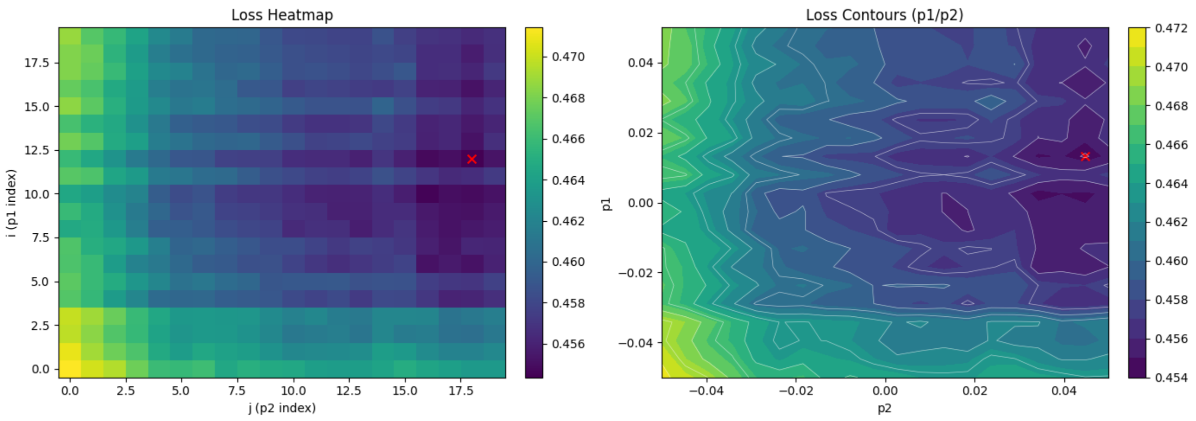

Figure 1: Loss-surface visualizations generated by the TinyLoRA 2-D scan code. Left: heatmap over grid indices. Right: contour map over parameter values (θ_1, θ_2).

Discussion

This experiment turns post-training into a fully observable two-parameter objective and makes its geometry explicit. The resulting figure provides direct evidence about smoothness, anisotropy, and basin structure under an extreme TinyLoRA budget. In practical terms, this setup is useful for diagnosing initialization ranges, validating optimizer step sizes for tiny adapters, and checking whether minima concentrate near boundaries of the scanned interval.

References

[1] John X. Morris, Niloofar Mireshghallah, Mark Ibrahim, and Saeed Mahloujifar. Learning to Reason in 13 Parameters. *arXiv preprint arXiv:2602.04118*, 2026.

[2] ANRG USC. TinyLoRA code repository. `https://github.com/anrgusc/tinylora`, 2026.