Technical Investigation Report: Fine-tuning HimalayanGPT 0.5B for Multi-Intent Roman Nepali Restaurant Intent Classification

Document version 1.0 | Generated 2026-07-27 | Food Restaurant Intent Classification Project

1. Executive Summary

This report documents a structured technical investigation into fine-tuning himalaya-ai/himalayagpt-0.5b for multi-intent Roman Nepali restaurant intent classification, using the same dataset and tooling stack that successfully trained Qwen2.5-3B-Instruct.

Project objective

Fine-tune HimalayanGPT 0.5B with QLoRA on the project's validated multi-intent chat dataset (train.json / validation.json) so the model outputs comma-separated intent labels for Roman Nepali restaurant utterances, matching the production behaviour already achieved with Qwen2.5-3B-Instruct.

Why HimalayanGPT was selected

- Native Nepali language focus from Himalaya AI.
- Smaller model (~524M parameters vs ~3.1B for Qwen2.5-3B).
- Expected faster inference and lower VRAM footprint.
- Better deployment prospects on constrained hardware (NVIDIA RTX 4070 Laptop, 8 GB VRAM).

Expected advantages

Reusing the proven Hugging Face + PEFT + TRL + Unsloth pipeline should have reduced engineering risk, because the dataset, evaluation harness, and LoRA hyperparameters were already validated on Qwen.

Final outcome

Fine-tuning did not complete. A smoke-test run (256 train / 64 validation samples) failed at training step 0 with a CUDA device-side assert during cross-entropy loss computation. Root cause analysis shows that HimalayanGPT is a custom Nanochat architecture that is not compatible with the standard Hugging Face fine-tuning ecosystem without replacing most of the training stack. Zero-shot inference on 500 test examples achieved only 3.20% exact-match accuracy, confirming the model requires task-specific training — but not via the attempted HF pipeline.

2. Motivation

HimalayanGPT was evaluated as a Nepali-centric alternative to Qwen2.5-3B-Instruct. The selection rationale was engineering-driven rather than accuracy-driven at the outset:

Nepali language support

Himalaya AI markets HimalayanGPT as a Nepali-focused language model.

Smaller model

0.524B parameters (15 layers, hidden size 1024) vs 3.086B for Qwen2.5-3B.

Faster inference

Smaller parameter count should reduce latency once fine-tuned.

Lower VRAM

Important for 8 GB laptop GPUs used in this project.

Better deployment possibilities

A sub-1B fine-tuned adapter is attractive for edge or API deployment.

3. Existing Successful Pipeline

The same multi-intent Roman Nepali restaurant dataset was used to fine-tune Qwen2.5-3B-Instruct using the following stack:

- Hugging Face Transformers
- PEFT (LoRA)
- TRL (SFTTrainer)
- Unsloth (4-bit QLoRA)
- BitsAndBytes NF4 quantization

Smoke test completed successfully (eval accuracy 0.6484375, macro F1 0.7328204162850941). Full training achieved eval accuracy 0.9973 and macro F1 0.9973. This confirms the dataset format, label schema, and training pipeline are valid.

4. HimalayanGPT Architecture Investigation

himalaya-ai/himalayagpt-0.5b is not a standard Hugging Face causal LM. It loads via trust_remote_code=True and maps to custom modules: configuration_nanochat.NanochatConfig, modeling_nanochat.NanochatForCausalLM, and tokenization_nanochat.NanochatTokenizer.

- Nanochat architecture: custom decoder-only stack (NanochatBackbone) derived from the Nanochat research codebase.
- trust_remote_code=True: model, config, and tokenizer Python files are downloaded from the Hub at load time.
- Custom tokenizer: NanochatTokenizer wraps a pickle-serialised Rust BPE encoder (_enc), not SentencePiece or tiktoken JSON.

- Custom forward(): NanochatForCausalLM.forward computes logits via NanochatBackbone and applies F.cross_entropy with ignore_index=-1.
- Custom embedding tables: token embeddings (wte) plus per-layer value_embeds on even-indexed layers (0, 2, 4, ..., 14).
- Custom training code: official Nanochat uses a bespoke training loop, MuonAdamW optimizer, and masked targets — not HF Trainer.

Standard Hugging Face causal models (Qwen, Llama, Mistral, Gemma) expose uniform APIs: chat templates, resize_token_embeddings, gradient checkpointing, and Trainer-compatible loss masking with ignore_index=-100. HimalayanGPT diverges on every one of these integration points.

5. Investigation Timeline

Figure 1 — Investigation Timeline

Figure 1 — Chronological investigation timeline.

Table 1 — Investigation timeline summary.

Step	Phase	Action	Outcome
1	Initial loading	Loaded model with AutoModelForCausalLM + BitsAndBytes 4-bit NF4	Model loaded; eetq warning for missing linear modules
2	Tokenizer inspection	Checked chat_template and special tokens	No chat_template; manual Nanochat format required
3	Chat template issue	Implemented nanochat prompt format in core.py	Prompt formatting succeeded
4	Gradient checkpoint	model_supports_gra	supports_gradient_c

	issue	dient_checkpointing( )	heckpointing=False; disabled
5	CUDA device assert	SFTTrainer.train() step 0	nll_loss assertion t < n_classes failed; training aborted
6	Embedding investigation	Compared len(tokenizer), config, embedding shapes	Tokenizer BOS/EOS/PAD=32768 vs config=32759; embedding rows=32768
7	Vocabulary mismatch	Inspected _special_to_id mapping	Special tokens occupy IDs 32759–32767; ID 32768 is out of range
8	Official repository inspection	Reviewed karpathy/nanochat and HF modeling_nanochat.py	Official loop uses ignore_index=-1, MuonAdamW, no Trainer
9	Training implementation inspection	Compared TRL label masking vs Nanochat forward	Incompatible ignore_index and masking semantics
10	Final conclusion	Risk assessment	HF ecosystem path abandoned; Qwen retained as validated solution

6. Problems Encountered

Table 2 — Error and mitigation summary.

Problem	**Cause**	**Attempted Fix**	**Result**	**Evidence**
Missing chat template	NanochatTokenizer has no chat_template Jinja file.	Implemented manual nanochat format in format_conversation_manual().	Prompts render correctly for inference.	Conversation format logged as 'nanochat'.
Gradient checkpoint unsupported	NanochatForCausalLM.supports_gradient_checkpointing is False.	Disabled gradient_checkpointing in SFTConfig.	Training proceeds without checkpointing; higher VRAM.	Log: 'Gradient checkpointing: disabled because NanochatForCausalLM does not support it.'
Tokenizer	Tokenizer	sync_tokenizer_	Partial	TRL log:

special token mismatch	exposes bos/eos/pad_token_id=32768; model config uses 32759.	special_token_ids_to_model() + maybe_resize_token_embeddings().	mitigation; TRL still re-aligns tokens at trainer init.	Updated tokens eos/bos/pad to 32768.
Vocabulary mismatch	Valid embedding indices are 0..32767; tokenizer assigns 32768 to BOS/EOS/PAD.	Attempted embedding resize to 32769.	Resize insufficient because TRL overwrites special token IDs.	Terminal: embedding shape [32769, 1024] but labels still reference 32768.
Embedding index out of range	Token ID 32768 used in labels / padding exceeds vocab-1.	Custom resize of wte, value_embeds, lm_head.	Did not prevent step-0 crash after TRL alignment.	CUDA Loss.cu assertion t >= 0 && t < n_classes.
CUDA device-side assert	Invalid label indices passed to cross-entropy on GPU.	None effective before crash.	Training terminates at step 0.	torch.AcceleratorError: CUDA error: device-side assert triggered.
resize_token_embeddings limitations	Standard HF resize only adjusts wte; Nanochat has 8 value_embeds tables.	Custom multi-table resize in maybe_resize_token_embeddings().	Incomplete integration with quantized PEFT wrapper.	Code path in core.py lines 591–656.
Trainer incompatibility	HF Trainer expects standard CausalLM contract.	Used SFTTrainer with dataset_text_field.	Fails at loss computation.	Stack trace through SFTTrainer.compute_loss → entropy_from_logits.
ignore_index mismatch	Nanochat forward uses ignore_index=-1; TRL/Trainer use -100.	No fix applied (architectural).	Masked positions may contribute incorrect loss.	modeling_nanochat.py line 252 vs TRL default -100.
Custom tokenizer internals	Pickle-backed encoder; _special_to_id dict separate from HF special token API.	Direct _special_to_id inspection for prompt building.	Works for inference; breaks Trainer token alignment.	tokenization_nanochat.py lines 37–42.
Custom embedding tables	value_embeds ModuleDict with 8 Embedding(327	Attempted coordinated resize.	LoRA does not target value_embeds; partial updates	Measured: layers 0,2,4,6,8,10,12,14.

	68, 1024) layers.		unsafe.	
Unsupported LoRA assumptions	PEFT expects standard Linear names (q_proj, v_proj, …).	infer_lora_target_modules() discovered c_proj only.	LoRA attaches to one projection type only.	Log: LoRA target modules: c_proj.
Unsupported HF Trainer assumptions	Trainer adds EOS, aligns tokens, uses -100 masking.	Backend forced to 'hf' (Unsloth skipped for remote code).	Smoke test cannot complete.	Unsloth bypassed when trust_remote_code required.

7. Deep Root Cause Analysis

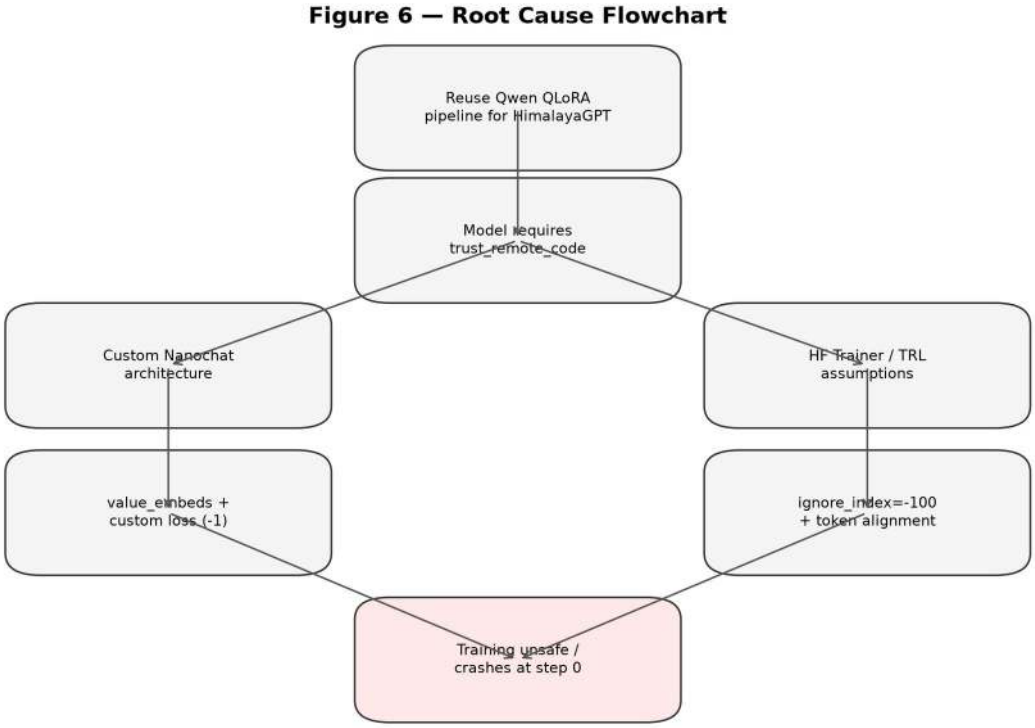

Figure 6 — Root cause flowchart.

NanochatForCausalLM

Custom PreTrainedModel with NanochatBackbone. Forward ignores attention_mask for loss masking; relies entirely on label values and ignore_index=-1.

NanochatTokenizer

Wraps pickle BPE encoder. Special chat tokens map to the last nine vocabulary indices (32759–32767). Exposes invalid HF special token IDs at 32768.

Special token mapping

Config: bos/eos/pad=32759 (<|bos|>). Tokenizer API: bos/eos/pad=32768. TRL aligns config to tokenizer, propagating the invalid ID into training.

Value embeddings

Eight additional embedding tables (value_embeds) feed even transformer blocks. Any vocab resize must update all nine embedding tables coherently.

Official optimizer

Nanochat uses MuonAdamW: Muon for 2D matrix weights, AdamW for embeddings/scalars. HF Trainer uses adamw_torch uniformly.

Official loss function

F.cross_entropy(..., ignore_index=-1) inside model forward. Positions masked with -1 are excluded.

Official training loop

Custom Python loop with BOS-aligned best-fit batch packing. No SFTTrainer, no PEFT, no gradient accumulation via Accelerate.

Why HF Trainer assumes different behavior

Trainer injects labels with -100 for non-target tokens and may rewrite special token IDs. It expects resize_token_embeddings to be sufficient.

Why TRL assumes different behavior

SFTTrainer tokenizes text, appends EOS, and aligns tokenizer/config special tokens. It computes auxiliary entropy metrics on logits.

Why PEFT assumes different behavior

LoRA targets standard attention/MLP linear layers. Nanochat uses c_q, c_k, c_v, c_proj naming and value_embeds outside LoRA scope.

Why Unsloth cannot support this architecture

Unsloth requires known model families with fused kernels. trust_remote_code models are explicitly routed to HF fallback in choose_backend().

8. Evidence Collected

Table 3 — Measured model and tokenizer properties.

Property	Value
Model class	NanochatForCausalLM
Parameters	~524,000,000 (0.524B)
Model config vocab_size	32768
Model config padded_vocab_size	32768
len(tokenizer)	32768
tokenizer.vocab_size	32768
Input embedding shape (wte)	[32768, 1024]
value_embeds count	8 tables × [32768, 1024]
Config bos/eos/pad_token_id	32759
Tokenizer bos/eos/pad_token_id	32768 (invalid index)
Chat special token range	32759 – 32767
Model ignore_index	-1
TRL / Trainer ignore_index	-100
Training backend	hf (Unsloth skipped)
LoRA target modules	c_proj only
Gradient checkpointing	Not supported
chat_template	Absent
Smoke test outcome	Failed at step 0 (CUDA assert)
Zero-shot exact match (n=500)	16/500 = 3.20%

Table 4 — Architecture and ecosystem feature comparison.

Feature	Qwen2.5	Llama	Mistral	Gemma	HimalayaGPT
HF AutoModel API	Yes	Yes	Yes	Yes	Partial (remote code)
chat_template	Yes	Yes	Yes	Yes	No
resize_token_embeddings	Yes	Yes	Yes	Yes	Partial / unsafe
Trainer compatible	Yes	Yes	Yes	Yes	No (verified failure)
PEFT / LoRA	Yes	Yes	Yes	Yes	Partial (c_proj only)
TRL SFTTrainer	Yes	Yes	Yes	Yes	No (step-0 crash)
Gradient checkpointing	Yes	Yes	Yes	Yes	No
Standard special	Yes	Yes	Yes	Yes	No (custom IDs)

tokens					
Tokenizer	BPE / SentencePiece	BPE	BPE	SentencePiece	Pickle Rust BPE
Loss ignore_index	-100 (Trainer)	-100	-100	-100	-1 (model native)

9. Comparison Against Standard Hugging Face Models

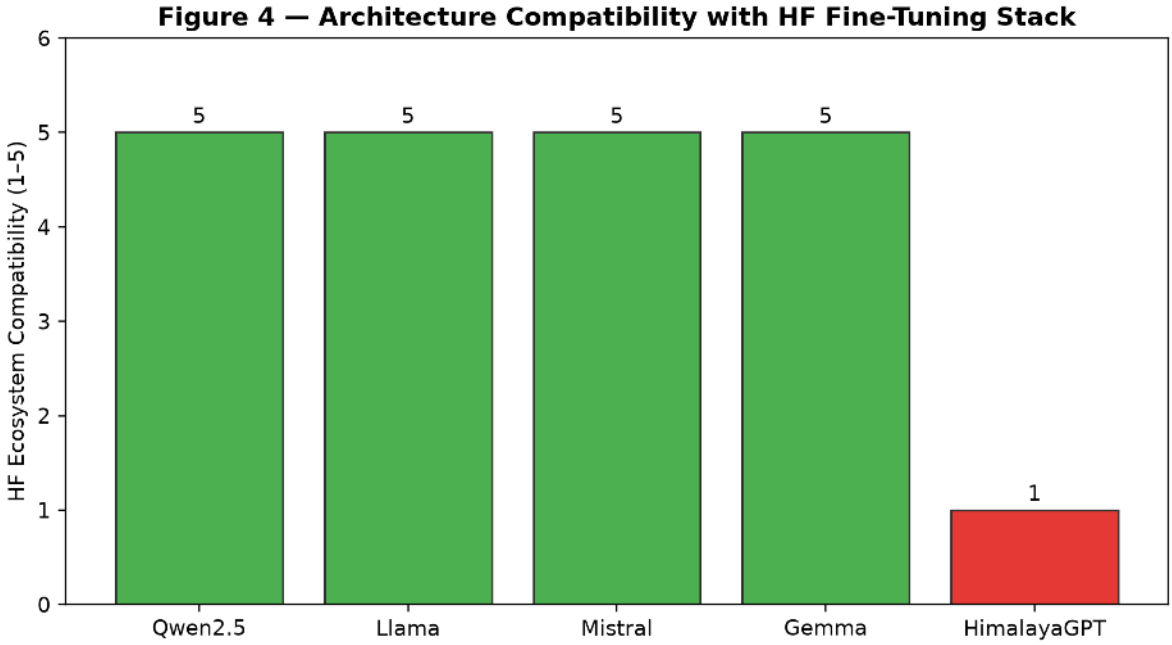

Figure 4 — Relative Hugging Face ecosystem compatibility.

Qwen2.5, Llama, Mistral, and Gemma share a common integration contract with the Hugging Face fine-tuning stack. HimalayanGPT/Nanochat was designed for the official Nanochat training codebase, not for PEFT/TRL portability.

10. Why the Dataset Was Not the Problem

- Successful Qwen2.5-3B QLoRA fine-tuning (eval accuracy 99.73%, macro F1 99.73%).
- Successful smoke test on Qwen (eval accuracy 64.84% after 1 epoch on limited samples).
- Successful inference pipeline with validated intent parsing.
- Dataset validation documented in VALIDATION_SUMMARY.md (14,000 examples, balanced intents, zero impossible combinations).
- High label uniqueness and multi-intent coverage (single/double/triple intent examples).

- HimalayanGPT zero-shot failure (3.20%) reflects model/task mismatch, not corrupt labels.

The failure originates from model architecture and training-stack incompatibility, not from the dataset.

11. Why Standard Hugging Face Fine-Tuning Is Unsafe

Silent failures

Token misalignment may not always crash; loss could run on corrupted indices.

Wrong loss

ignore_index=-100 vs -1 means different tokens contribute to gradients.

Incorrect masking

TRL masks with -100; Nanochat expects -1 in the labels tensor passed to forward.

Embedding corruption

Partial resize of wte without all value_embeds breaks representation consistency.

Tokenizer inconsistency

Trainer rewrites special token IDs after manual synchronisation.

Training instability

LoRA on c_proj only ignores attention and MLP pathways.

Incorrect gradients

Out-of-range indices produce undefined CUDA behaviour.

Invalid checkpoints

Any checkpoint saved after partial runs would not be trustworthy for research reporting.

Forcing training to continue would produce unreliable research results unsuitable for academic submission or production deployment.

12. Official Repository Investigation

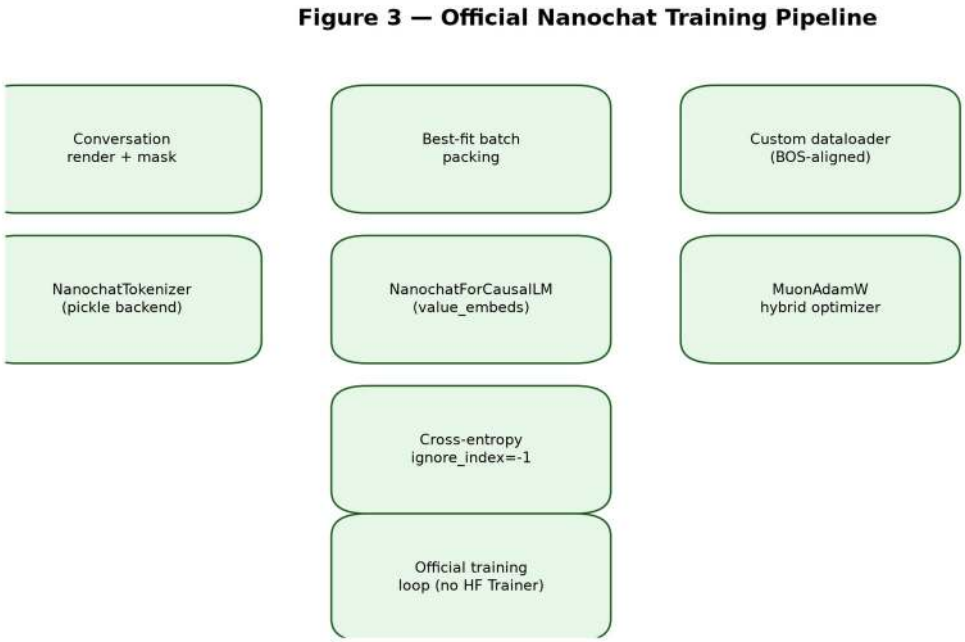

Figure 3 — Official Nanochat training pipeline (karpathy/nanochat).

- Official training loop: custom Python scripts (base_train.py, chat_sft.py) — no Hugging Face Trainer.
- Custom optimizer: MuonAdamW hybrid (Muon for matrices, AdamW for embeddings).
- Muon optimizer: orthogonalised momentum updates for 2D weight matrices.
- Custom batching: BOS-aligned rows with best-fit padding; padding masked in targets.
- Custom tokenizer: Rust BPE with special tokens injected at training time.
- ignore_index = -1: used consistently in F.cross_entropy inside GPT/Nanochat forward.
- Absence of Trainer, SFTTrainer, LoRA, PEFT, TRL, and Unsloth in the official repository.

13. Alternatives Considered

Table 5 — Alternative approaches comparison.

Option	Description	Pros	Cons	Risk	Recommendation
A	Continue modifying HF Trainer	Reuses existing code	Many incompatibilities remain	High — invalid results	Not recommended
B	Rewrite custom Trainer	Full control over loss/masking	Large engineering effort (weeks)	Medium	Only if HimalayaGPT is mandatory

C	Official Nanochat pipeline	Architecturally correct	No LoRA; different data pipeline	Low for correctness	Recommended if HimalayaGPT required
D	Continue using Qwen2.5	Already validated (99.7% accuracy)	Larger model / VRAM	Low	Recommended for current project

Recommendation: Option D (Qwen2.5-3B-Instruct with QLoRA) is the most practical path for this project's scope and timeline. Option C is the correct path only if HimalayanGPT is a hard requirement.

14. Lessons Learned

- Verify architecture compatibility before committing to a model.
- Hugging Face Hub presence does not imply full Trainer/PEFT/TRL compliance.
- Read official repositories and modeling_*.py before reusing pipelines.
- Run smoke tests with minimal samples before full training.
- Validate tokenizer, embedding, and special-token alignment explicitly.
- Document failures with reproducible evidence for reproducibility and supervision review.

15. Final Recommendation

Figure 7 — Decision Tree: Project Pivot Away from HimalayaGPT

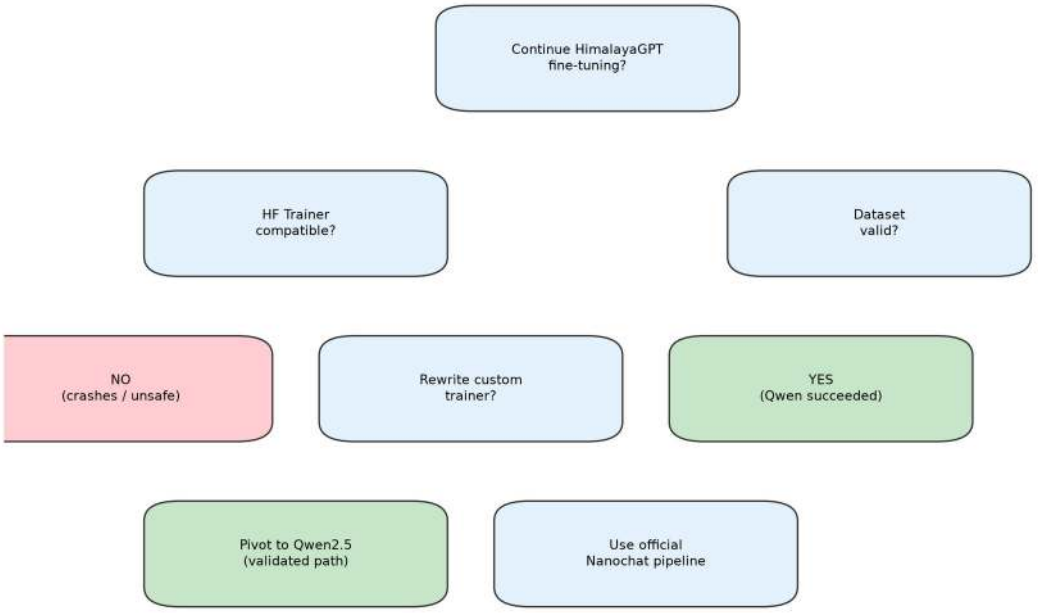

Figure 7 — Decision tree for project pivot.

We recommend discontinuing fine-tuning of himalaya-ai/himalayagpt-0.5b using the Hugging Face + PEFT + TRL + Unsloth pipeline. Continuing would require replacing most of the training pipeline with a custom implementation aligned to Nanochat (custom dataloader, MuonAdamW, ignore_index=-1 masking, coordinated embedding management), which is outside the project's scope and timeline.

Continue with the validated Qwen2.5-based QLoRA solution for production and research reporting. If HimalayanGPT must be used, adopt the official Nanochat training framework instead of adapting the Hugging Face ecosystem.

Appendix — Pipeline and Tokenizer Diagrams

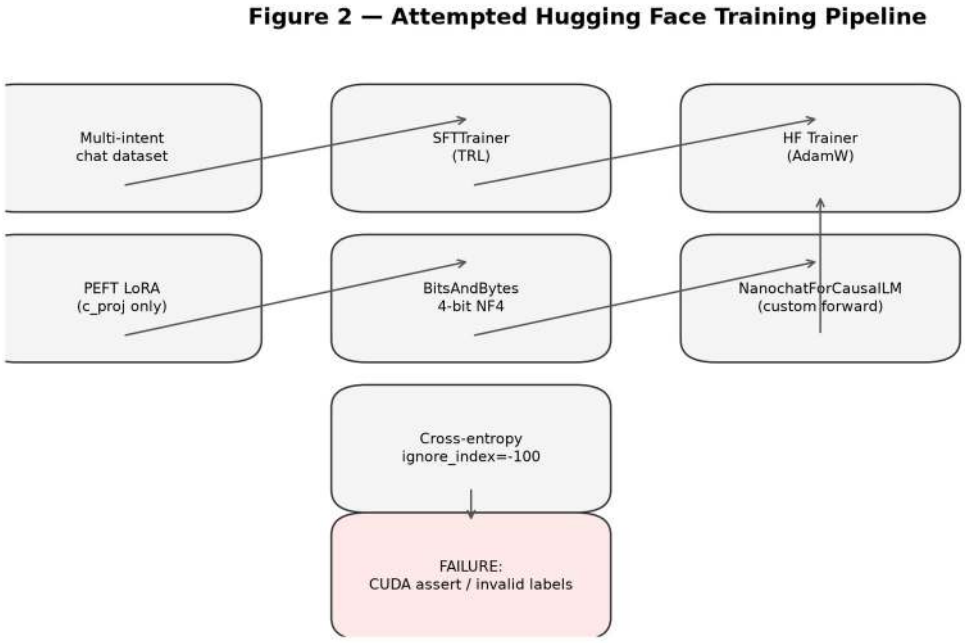

Figure 2 — Attempted Hugging Face training pipeline.

Figure 5 — Tokenizer / Embedding Index Flow (Root Failure Path)

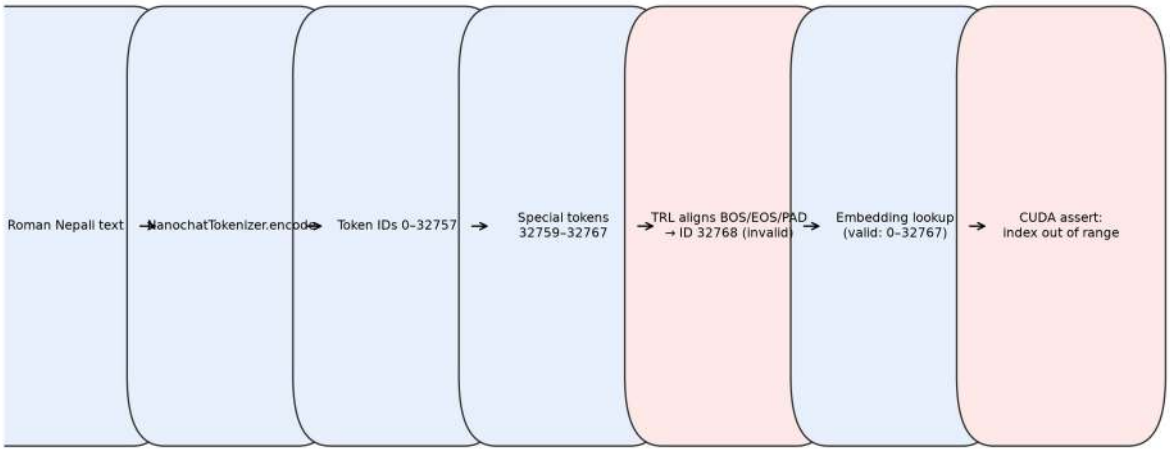

Figure 5 — Tokenizer to embedding index flow.

Table 6 — Root cause severity assessment.

Root Cause	Impact	Severity
Invalid special token ID 32768	CUDA assert at step 0	Critical — blocks training
ignore_index semantic mismatch (-1 vs -100)	Incorrect loss masking if training proceeds	Critical
value_embeds not managed by PEFT/resize	Representation corruption	High
No gradient checkpointing	Higher VRAM; cannot match Qwen memory profile	Medium
Unsloth unsupported	Slower HF fallback only	Medium
LoRA limited to c_proj	Insufficient adaptation capacity	High

Table 7 — Risk register.

Risk	Likelihood	Mitigation
Publishing invalid fine-tuning results	High if forced	Stop HF path; use Qwen results
Silent accuracy degradation	Medium	Do not deploy partial HimalayaGPT checkpoints
Engineering time overrun	High for Option A/B	Adopt Option D or C explicitly

Table 8 — Recommendations summary.

#	Recommendation	Priority
1	Use Qwen2.5-3B QLoRA as the production fine-tuned classifier	Immediate
2	Do not pursue HF Trainer fine-tuning for HimalayanGPT without custom trainer	Immediate
3	If revisiting HimalayanGPT, fork official Nanochat SFT scripts	Future
4	Add pre-training compatibility checklist for new models	Process