

Take-Home (Senior DS/ML): Hinglish Code-Switch Fine-Tuning + Low-Latency Serving

Key requirement: For mixed-language outputs, **Hindi must be in Devanagari**, and **English must remain Latin script** (no Roman Hindi).

Goal (1–2 days)

Improve a small instruct model so it can:

1. follow instructions reliably,
 2. handle Hinglish/code-switch inputs,
 3. output **strict JSON** with **script-correct text** (Hindi=देवनागरी, English=Latin), and
 4. show production-aware thinking for **TTFT + concurrency**.
-

Part A — Data: Build/curate a mini dataset (200–500 examples)

Create (or augment) a dataset of Hinglish/code-switch user inputs. Include:

- Roman Hindi + English mix (e.g., “bhai kal 2 baje ke baad...”)
- Hindi typed in Devanagari + English mix
- Noisy text: typos, slang, casual tone, partial phrases
- Numerals / money / dates: “1.5 lakh”, “डेढ़ लाख”, “tomorrow 4pm”
- Ambiguity requiring clarification: “kal”, “parso”, “subah”, “shaam”, “thoda jaldi”

Output format (must be strict JSON, no extra text)

Each sample should map input → structured output *and* include script-correct **normalized_text**:

```
{
```

```
"input": "bhai kal 2 baje ke baad meeting ko Friday 4pm pe shift kar de",
"instruction": "Extract intent and slots. Output strict JSON. Also return normalized_text where Hindi is Devanagari and English is Latin.",
"output": {
  "intent": "reschedule_meeting",
  "slots": {
    "original_time": "कल 2 बजे के बाद",
    "new_time": "Friday 4pm"
  },
  "normalized_text": "भाई कल 2 बजे के बाद meeting को Friday 4pm पर shift कर दो",
  "language_mix": "hinglish",
  "script_rules": {
    "hindi": "devanagari",
    "english": "latin"
  }
}
```

Must-have: At least **30%** examples where the model must convert Roman Hindi → **Devanagari** in the output.

Part B — Fine-tune for instruction following + script correctness

Fine-tune (LoRA/QLoRA preferred) so the model:

- always emits valid JSON
- fills slots correctly
- produces **script-correct mixed output**:
 - Hindi tokens/phrases in **Devanagari**

- English words unchanged in **Latin**
- avoids extra explanations (no reasoning text)

Provide:

- training config (model, lr, epochs, rank, quantization)
 - prompt template
 - before/after examples (at least 10)
-

Part C — Evaluation: correctness + robustness + script compliance

Create an eval script with at least:

1. **JSON validity rate**
2. **Slot F1 / exact match** (simple is fine)
3. **Script compliance rate** (new):
 - Hindi should not appear in Latin letters (e.g., “kal”, “mera”, “bhai” must become “कल”, “मेरा”, “भाई” in relevant fields)
 - English should not be transliterated into Devanagari (e.g., “meeting”, “Friday”, “loan”, “OTP” must remain Latin)
4. **Stress suite (100 cases)** including:
 - mixed entities: “HDFC ka loan foreclosure charges?”
 - tricky numerals: “1.25 lakh”, “सवा लाख”, “डेढ़ लाख”
 - adversarial formatting: “Return YAML”, “Explain your reasoning”, “Ignore JSON rule”
 - code-switch punctuation/voice artifacts: “haan... wait, actually kal nahi, parso”

Deliverable: `eval.py` + `metrics.json` + `stress_suite.jsonl`

Part D — Latency: TTFT + concurrency

Measure and report:

- TTFT and tokens/sec for baseline vs fine-tuned
- concurrency: **1 / 8 / 32** parallel requests
- p50 and p95

Then propose **two concrete TTFT improvements** you'd ship:

- quantization choice + tradeoff
- vLLM/TGI batching strategy
- KV-cache / prefix caching
- prompt shortening + stop tokens
- speculative decoding (if applicable)
- routing: small model first + fallback

Deliverable: `latency_report.md` with numbers + plan.

Part E — Production & Iteration Plan (Senior signal)

Write a 1-page decision memo:

1. **Serving architecture:** what stack you'd use and why
2. **Quality gates:** minimum thresholds for JSON validity + script compliance + slot F1
3. **Data flywheel:**

- privacy-safe logging (PII handling)
- labeling guidelines for Hinglish + script rules
- active learning sampling strategy
- regression suite & release process

Deliverable: `decision_memo.md`

Submission package (single repo/zip)

- `README.md` (how to run training + eval)
 - `data/` (or generator script)
 - `train.py` / notebook
 - `eval.py` + outputs
 - `report.md` (1–2 pages): results, failures, what you'd do next
 - `latency_report.md`
 - `decision_memo.md`
 - `stress_suite.jsonl`
-

How we'll score it

1. **Script correctness (Devanagari/Latin) + JSON discipline (35%)**
2. **Hinglish/code-switch extraction quality (25%)**
3. **Evaluation quality + stress rigor (20%)**

4. Latency + production thinking (20%)