

Multilingual Health QA in Low-Resource African Languages — Solution Documentation

Zindi challenge: open-ended **generative** health question answering in 8 language-country subsets.

Winning submission (private LB): `submission_v23_judgecol.csv` = **0.694836**. This document describes only the models and pipeline that produced that entry.

1. Overview and Objectives

Problem. Given a health question (maternal health, sexual & reproductive health, adolescent SRH rights, and gender-based violence) in one of 8 subsets, generate an answer **in the same language**.

Subsets: 4 African languages — Aka_Gha (Akan/Twi), Amh_Eth (Amharic), Lug_Uga (Luganda), Swa_Ken (Kiswahili) — and 4 English-by-country splits — Eng_Eth, Eng_Gha, Eng_Ken, Eng_Uga.

Scoring metric. $0.37 \cdot \text{ROUGE-1 F1} + 0.37 \cdot \text{ROUGE-L F1} + 0.26 \cdot \text{LLM-judge}(1-5, \text{normalised})$. The submission has three answer columns (TargetR1F1, TargetRLF1, TargetLLM) and **each is scored independently** — a fact this solution exploits (Section 7.3).

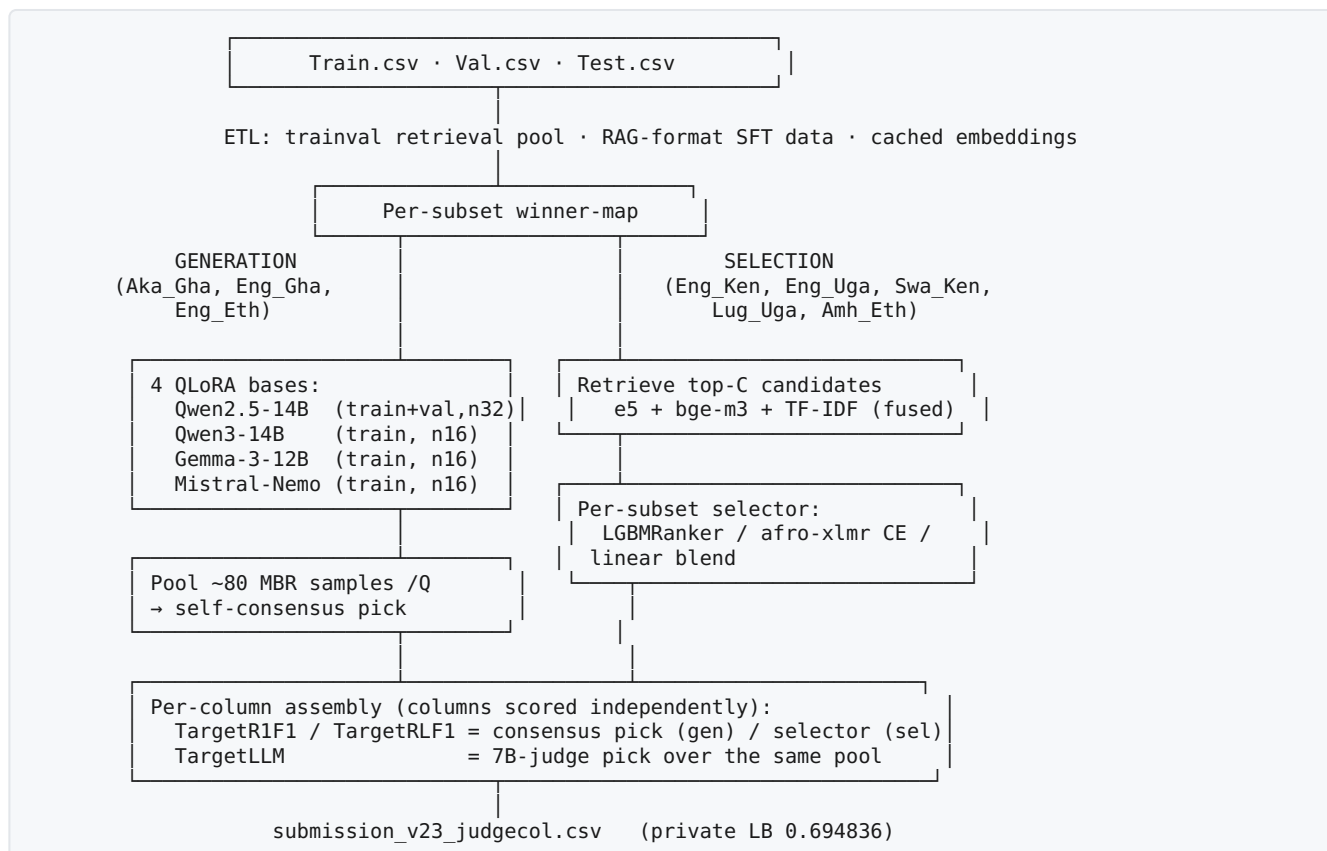
Key design insight. Because ROUGE rewards lexical overlap with the (single) reference, the task splits into two regimes, and each subset is routed to the method that wins it:

- **Duplicate-rich subsets** (Eng_Ken, Eng_Uga, Swa_Ken, Lug_Uga, Amh_Eth): near-identical questions exist in the training pool, so **retrieval + selection** of a real human answer beats generation.
- **Generation-bound subsets** (Aka_Gha, Eng_Gha, Eng_Eth): no close match exists, so the answer is **generated**, and the best results come from **cross-base ensembling**.

Outcome. This per-subset winner-map pipeline took the score from a ~0.485 retrieval baseline to **public LB 0.697241 / private LB 0.694836** (the private-best of all submissions).

2. Architecture

ETL → per-subset router → {generation branch | selection branch} → per-column assembly → submission.



3. Environment and Hardware

- **GPU:** 1× NVIDIA RTX 5090, 32 GB (Blackwell sm_120, CUDA 13, driver 580.105.08).
- **Python:** Anaconda 3.13; torch 2.9.0+cu128, transformers 4.57.3, peft 0.18.0, trl 0.24.0, accelerate 1.12.0, bitsandbytes 0.48.2, vllm 0.13.0, sentence-transformers 5.2.2, lightgbm 4.6.0, rouge-score 0.1.2, scikit-learn 1.6.1, pandas 2.2.3, numpy 2.1.3 (pinned in requirements.txt).
- **Determinism:** all training/selection scripts set seed=42 (torch / numpy / transformers.set_seed).
- **Required env vars:** MKL_THREADING_LAYER=GNU (avoids a vLLM/MKL subprocess crash under conda); HF_HUB_CACHE=/mnt/docker/hf_cache for the large base models; PYTORCH_CUDA_ALLOC_CONF=expandable_segments=True (reduces allocator fragmentation / OOM).
- **No AutoML; only challenge data + openly-licensed pretrained model weights** (Section 4).

4. Models and Data (Compliance)

Data. All supervised content comes solely from the challenge-provided Train.csv / Val.csv / Test.csv. **No external datasets are used** — the retrieval pool, the RAG few-shot examples, and every fine-tuning target are built only from challenge data.

Pretrained model weights (openly licensed; used as permitted pretrained models):

Model	Role in v23	License
Qwen/Qwen2.5-14B-Instruct	generation base (QLoRA, train+val)	Apache-2.0
Qwen/Qwen3-14B	generation base (QLoRA, train-only)	Apache-2.0
unsloth/gemma-3-12b-it	generation base (QLoRA, train-only)	Gemma Terms (commercial use permitted)
unsloth/Mistral-Nemo-Instruct-2407	generation base (QLoRA, train-only)	Apache-2.0
unsloth/qwen2.5-7b-instruct	LLM-judge selector (TargetLLM column)	Apache-2.0
intfloat/multilingual-e5-large	dense retrieval	MIT
BAAI/bge-m3	dense retrieval	MIT
BAAI/bge-reranker-v2-m3	candidate reranking	MIT
Davlan/afro-xlmr-large-76L	Luganda cross-encoder selector	MIT
sentence-transformers/LaBSE	few-shot retrieval + pseudo-reference retrieval	Apache-2.0

Every license permits the competition use; no gated or non-commercial weights are in v23.

5. ETL Process

Extract. Inputs are three CSVs (ID, input, output[, subset]): Train.csv (29,815 rows), Val.csv (6,686), Test.csv (2,618, no output). One static read; no streaming/freshness concerns. subset = the language-country code; ID = ID_<TR|VL|TS>_<lang>_<8hex> where the hex is a topic-family hash (a retrieval prior; also a ranker feature).

Transform.

- **Retrieval pool:** for test predictions, Train and Val are concatenated into a merged **trainval pool** (~35k answers) — uses all labeled data for retrieval/selection.
- **RAG-format SFT data** (prepare_qwen_rag_sft.py → outputs/sft_data/*.jsonl): each gen-subset row becomes a chat example = system instruction (answer in <lang>, match example style/length) + user (k=3 leave-one-out retrieved Q&A few-shot + word-budget hint) + assistant (gold answer). train_qwen_rag.jsonl = train-only (14,658 rows; used for the Qwen3/Gemma/Mistral bases); train_qwen_rag_final.jsonl = train+val (17,902 rows; used for the Qwen2.5-14B base). Mistral's chat template drops the system role, so a merged-system variant (train_qwen_rag_mistral.jsonl) prepends system → user.
- **Embeddings & candidate scores** are precomputed and cached (outputs/emb_cache/): e5-large + bge-m3 query/passage matrices, fused-retrieval top-C candidate indices, per-candidate ROUGE labels, and cross-encoder rerank scores.

Load. Artifacts persist to disk and are reused: outputs/emb_cache/ (*.npy/*.npz), outputs/sft_data/*.jsonl, outputs/adapters/<run-name>/final/ (LoRA adapters), outputs/mbr_samples_*.csv (decoded sample pools). No database; flat files keyed by subset and split. The embedding caches are the main optimisation — computed once and reused.

6. Data Modeling

A **per-subset winner map** — each subset is served by whichever method wins it on validation.

6.1 Generation subsets — Aka_Gha, Eng_Gha, Eng_Eth

- **Bases:** 4 instruction LLMs, each QLoRA-fine-tuned (4-bit NF4, LoRA $r=32$, $\alpha=64$, dropout 0.05, 2 epochs, effective batch 16, lr $1e-4$, max-seq 2048, seed 42) on the RAG-format SFT data. Deliberately different lineages for ensemble diversity:
- **Qwen2.5-14B** — fine-tuned on **train+val** (`train_qwen_rag_final.jsonl`), MBR-decoded $n=32$.
- **Qwen3-14B, Gemma-3-12B, Mistral-Nemo-12B** — fine-tuned on **train-only** (`train_qwen_rag.jsonl`), MBR-decoded $n=16$ each.
- **Decoding: MBR sampling** (temp 0.7, top-p 0.9) with the same RAG few-shot prompt ($k=3$ from the trainval pool) + per-question word-budget hint used at training time.
- **Ensembling (the key lever):** pool all bases' samples per question (~ 80 samples) and pick the **self-consensus** answer (max mean unigram-F1 vs the other pooled samples). The gains come from cross-base *diversity*, not from any single stronger base — see Section 9.
- **Validation:** honest val proxy uses train-only adapters (a train+val adapter cannot be scored on val without leakage). Cross-base val proxy $(R1+RL)/2$: Aka 0.354, Eng_Gha 0.428, Eng_Eth 0.720.

6.2 Selection subsets — Eng_Ken, Eng_Uga, Swa_Ken, Lug_Uga, Amh_Eth

Duplicate-rich; the pipeline **selects** a real answer from the retrieval pool rather than generating. Per-subset selector (chosen on val):

- Eng_Ken / Eng_Uga / Swa_Ken: **LGBMRanker (LambdaRank)** over top-20 fused candidates, trained on **val labels** with features = per-retriever cosines, fused-blend score, query $\leftrightarrow$ candidate unigram-F1, exact/hash match, dup-count, lengths, rerank scores. Eng_Uga additionally uses a judge-augmented per-question router that switches a question to the generation arm when it scores better there.
- Lug_Uga: **afro-xlmr-large-76L cross-encoder** (Luganda-aware) blended with retrieval/rerank scores — dense retrievers are language-blind to Luganda, so a Luganda-reading CE is essential.
- Amh_Eth: **linear blend** of retrieval + rerank scores (content is largely absent from the pool; this subset is retrieval-bound and low-weight).
- Validation proxy: Eng_Ken 0.817, Eng_Uga 0.809, Swa_Ken 0.785, Lug_Uga 0.664, Amh_Eth 0.200.

6.3 Assumptions

The topic-hash is a usable metadata prior. ROUGE's paraphrase penalty makes selecting real answers superior on duplicate-rich subsets; generation only wins where no near-duplicate exists.

7. Inference

7.1 Generation

vLLM (`exp_mbr_decode.py`), 4-bit bitsandbytes load (`--quant`), `enforce_eager`, LoRA adapter via `LoRARequest`. New test question $\rightarrow$ RAG few-shot prompt ($k=3$ retrieved from the trainval pool) $\rightarrow$ n samples $\rightarrow$ all samples written to `outputs/mbr_samples_<tag>.csv`. One decode also writes the single self-consensus pick to `outputs/test_pred_mbrdec_<tag>.csv`.

7.2 Selection

Merged-pool candidates (`build_v7_testpool.py`) $\rightarrow$ the per-subset selector $\rightarrow$ the chosen candidate answer.

7.3 Assembly & per-column optimisation

The three submission columns are scored **independently** by the LB, so each is optimised separately (`build_v23.py` then `build_v23_judgecol.py`):

- **TargetR1F1 & TargetRLF1 (ROUGE columns):** for gen subsets, the self-consensus pick over the cross-base pool; for selection subsets, the selector's pick.
- **TargetLLM (judge column):** for gen subsets, a *separate* answer chosen by a **7B pseudo-reference judge** (`unsloth/qwen2.5-7b-instruct` rates each pooled sample against the top-1 retrieved answer; `TargetLLM = argmax(rating + 0.01·consensus)`). This lifts the 26%-weighted judge component at zero ROUGE cost. Selection subsets carry no judge swap.

The build scripts assert invariants before writing (Section 10) and emit `submission_v23_judgecol.csv`.

7.4 Versioning

Adapters are versioned by run-name under `outputs/adapters/`; submission CSVs are immutable in `submissions/`. The winning submission is produced by `build_v23.py` + `build_v23_judgecol.py`; the whole pipeline is driven end-to-end by `reproduce.sh`.

8. Run Time (single RTX 5090, serial)

Stage	Script	Time
Embedding / candidate caches	retrieval/blend scripts	one-time, ~minutes/subset (cached)
QLoRA fine-tune, per base, 2 ep	<code>train_qlora.py</code>	~5.5 h (12B) / ~8 h (14B)
MBR generation, test (1,043 gen rows)	<code>exp_mbr_decode.py</code>	~1.5 h (n=16) / ~3 h (n=32)
7B judge selection over pool	<code>build_v23_judgecol.py</code>	~15–20 min
Selection ranker / CE train + apply	<code>train_select_ranker2.py</code> , <code>train_ce_reranker.py</code>	~minutes–30 min
Cross-base consensus pooling + build	<code>build_v23.py</code>	~minutes (CPU)

End-to-end total ≈ 38 h (~1.5 days), all sequential on one GPU: - QLoRA fine-tuning ≈ **27 h** (Qwen2.5-14B ~8 h + Qwen3-14B ~8 h + Gemma-3-12B ~5.5 h + Mistral-Nemo ~5.5 h). - MBR generation ≈ **7.5 h** (Qwen2.5-14B n=32 ~3 h + the three n=16 bases ~1.5 h each). - ETL + selection arm + judge passes + builds ≈ **3–4 h**.

Inference only (adapters already trained) ≈ **10 h** — useful to regenerate the submission without re-fine-tuning. **Excludes a one-time ~80–100 GB download of the open base-model weights from HuggingFace** (bandwidth-dependent, not GPU time).

9. Performance Metrics

Primary metric: composite $0.37 \cdot R1 + 0.37 \cdot RL + 0.26 \cdot \text{judge}$. Offline proxy $(R1+RL)/2$ computed with a **UnicodeTokenizer** (critical — the default ROUGE tokenizer drops Ge'ez/diacritic tokens and collapses Amharic to ~0).

LB trajectory to the winning entry (each step adds a cross-base generator to the pool):

Submission	Public LB	R1 / RL / Judge	Lever
baseline (retrieval)	~0.485	—	TF-IDF / LaBSE hybrid
v14	0.691689	0.6792 / 0.6053 / 0.8324	14B QLoRA + judge column
v17	0.692900	0.682 / 0.6076 / 0.8298	2-way cross-base
v21	0.693882	0.6843 / 0.6095 / 0.8276	3-way (+Gemma)
v23 (winning)	0.697241	0.685 / 0.6105 / 0.8381	4-way (+Mistral) + judge

Final scores: public LB 0.697241, private LB 0.694836 (the private-best of all our submissions; public→private gap ≈ 0.0024 , normal). The (R1+RL)/2 offline proxy tracked the ROUGE components robustly throughout; the 26%-weighted LLM-judge column carries $\sim \pm 0.002$ LB run-to-run variance, so judge-only deltas under ~ 0.003 were treated as noise rather than signal.

Per-subset method choice was validated on the held-out val proxy (Section 6). Notably, the Akan ceiling is **metric-bound** (~ 0.354 , oracle ~ 0.41): even a fine-tuned Twi-native model scores lower on ROUGE because of paraphrase penalty against the single reference — hence Akan is served by the diverse cross-base generator, not a single specialist.

10. Error Handling and Logging

- Every stage writes a dedicated log (`outputs/log_*.txt`); generation/training stream progress.
- The orchestration driver (`reproduce.sh`) is set `-uo pipefail` and **aborts on any step failure** (`run wrapper` → `exit 1`), so a crashed stage cannot silently corrupt downstream artifacts.
- Submission builds assert invariants before writing: **2,618 rows, IDs match SampleSubmission, no empty cells in any of the three columns**, and expected per-subset change counts.
- Empty/degenerate generations are tolerated (they score ~ 0 in consensus and are not selected).
- Known operational pitfalls are guarded: `MKL_THREADING_LAYER=GNU` (vLLM/MKL crash under conda); the decode `--model` must match the adapter's base (a silent dim mismatch otherwise crashes vLLM); the MBR decode `--tag` is the contract the build scripts read by exact filename.

11. Maintenance and Monitoring

- **Reproduce / retrain:** `bash reproduce.sh` (staged A→E; see Section 12). Caches in `emb_cache/` are reused; delete to force recompute.
- **Adding a base** (scaling the ensemble): train a new QLoRA, MBR-decode its test samples, and add the sample file to the pool list in `build_v23.py` / `build_v23_judgecol.py`. **Caveat:** a base helps only if it is *comparable quality* — a weak base drags the self-consensus pick.
- **Disk/cache:** large bases live in `/mnt/docker/hf_cache` (home disk is small). If `/mnt/docker` permissions reset to root-only, `sudo chmod o+x /mnt/docker` and re-choown the cache.
- **Monitoring quality:** the offline (R1+RL)/2 proxy tracks the LB well for ROUGE, but the LLM-judge column is **noisy ($\sim \pm 0.002$ run-to-run)** and only weakly predictable offline — do not trust judge-driven changes under ~ 0.003 without an LB read.
- **Lifecycle / scaling:** the per-subset winner map is the extension point — re-evaluate each subset's best method as new bases/data arrive; the merged-pool + cross-base structure scales by adding diverse bases (diminishing returns past ~ 4) or stronger retrievers.

12. Reproduction Steps

The driver `reproduce.sh` runs five staged, checkpointed stages (**A** ETL → **B** selection base → **C** generation → **D** assemble → **E** verify):

1. Place `Train.csv`, `Val.csv`, `Test.csv`, `SampleSubmission.csv` in the working directory.

2. Install the environment (`requirements.txt`) and set the env vars (Section 3).
3. **(A)** Build SFT data + retrieval caches: `prepare_qwen_rag_sft.py`, the embedding/blend scripts, `build_v7_testpool.py`.
4. **(B)** Selection base: `train_select_ranker2.py`, `train_ce_reranker.py`, the blend/router deploy scripts, then the v15→v17 builds → `submission_v17_judgecol.csv` (provides the selection subsets + Eng_Uga routing).
5. **(C)** Generation: QLoRA fine-tune the 4 bases (Qwen2.5-14B on train+val; Qwen3/Gemma/Mistral on train-only) with `train_qlora.py`, then MBR-decode their test samples with `exp_mbr_decode.py` (n=32 for Qwen2.5-14B, n=16 for the others).
6. **(D)** Assemble: `build_v23.py` (4-way cross-base consensus for the ROUGE columns) then `build_v23_judgecol.py` (7B-judge TargetLLM) → **`submission_v23_judgecol.csv`**.
7. **(E)** Verify: 2,618 rows, IDs match `SampleSubmission.csv`, no empty cells.

Run all at once with `bash reproduce.sh`, or a single stage with `START_AT=<X> STOP_AFTER=<X> bash reproduce.sh`; `DRY_RUN=1` prints the command plan.