End-to-End Arabic SLM Fine-Tuning & Data Synthesis Pipeline

Project Authors:
Zakaria BOULLAM
Abderrahman YOUSFI

Supervised by:
Dr. Evan Dufraisse
Yousef Khoubrane

December 20, 2025

Abstract

This report details the engineering pipeline developed to create a specialized Arabic summarization Small Language Model (SLM). The project methodology encompasses the acquisition of raw corpora, the generation of a synthetic instruction dataset via orchestrated API calls (overcoming strict rate limits), and the parameter-efficient fine-tuning (PEFT) of the Qwen 2.5-3B model using the Unsloth framework. We demonstrate how to maximize throughput on free-tier infrastructure (Google Colab T4) while maintaining high-quality outputs, confirmed by ROUGE, BERTScore, and LLM-as-a-Judge evaluations.

1 Introduction

The objective of this project was to fine-tune a lightweight LLM capable of summarizing long-form Arabic text. Given the scarcity of high-quality Arabic instruction-response pairs, a significant portion of this project focused on constructing a robust data pipeline to generate synthetic summaries from raw articles before training the final model.

2 Data Acquisition and Analysis

2.1 Source Repository

We utilized the `RightNow-AI/rightnow-arabic-llm-corpus`, a massive open-source dataset designed for LLM training. From this repository, we extracted a subset of **5,000 documents** to serve as the foundation for our dataset.

2.2 Dataset Statistics

The source repository provides production-ready quality with comprehensive coverage. Below represent the statistics of the full corpus from which our subset was drawn:

Table 1: RightNow-Arabic-LLM-Corpus Statistics

Metric	Value
Total Articles	743,288
Total Words	244,153,780
Total Sentences	12,392,064
Unique Words	1,529,064
Vocabulary Richness	0.0063
Average Words/Article	328.5
Average Sentences/Article	16.7
Average Words/Sentence	19.7
High Quality Articles	185,351 ($\geq$70% quality score)
Dataset Size	8.7 GB (JSONL)

Key Features Justifying Selection:

- **Production-Ready:** The cleaning pipeline removed HTML artifacts and templates.
- **LLM-Optimized Format:** JSONL structure designed specifically for training.
- **Linguistic Excellence:** UTF-8 encoded with proper Arabic text normalization.
- **Scale:** One of the largest available Arabic datasets, ensuring diversity.

3 Synthetic Data Generation Pipeline

To transform raw documents into '(Text, Summary)' pairs, we utilized Large Language Models via API. This phase required complex engineering to handle the "Triple Constraint" of free-tier APIs: **Requests Per Minute (RPM)**, **Tokens Per Minute (TPM)**, and **Requests Per Day (RPD)**.

3.1 API Strategy and Model Selection

Initially, we attempted to use Groq and Gemini 2.5 Flash. However, we encountered severe rate limits (e.g., Gemini 2.5 allowed only ≈ 20 RPD). **Final Choice:** We migrated to the **Gemma-3-27b-it** model via Google AI Studio, which offered a generous limit of **14,400 RPD** and **30 RPM**.

3.2 Traffic Shaping and Optimization

To process 5,000 documents without getting banned, we implemented the following logic:

1. **Token Volume Control (TPM):** The API had a high request limit (30 RPM) but a low token limit (15k TPM). Sending full documents caused immediate crashes. *Solution:* We implemented strict truncation:

$$L_{max} = 4,500 \text{ chars} \approx 1,200 \text{ tokens} \tag{1}$$

   This allowed us to fit roughly 12 documents per minute within the 15k TPM budget.

2. **Dynamic Latency Throttling:** Instead of a fixed sleep timer, we calculated the required wait time dynamically to maximize speed:

$$T_{wait} = \max(0, T_{target} - (T_{end} - T_{start})) \tag{2}$$

   Where $T_{target} = 5.0$ seconds. This ensured we never exceeded the calculated safe velocity of 12 RPM.

3. **Fault Tolerance:** We implemented a `KeyManager` class to handle key rotation and a retry loop (3 attempts) with exponential backoff for `ResourceExhausted` errors.

4 Model Fine-Tuning Methodology

With the dataset generated, we proceeded to fine-tune a Small Language Model (SLM) on Google Colab (T4 GPU, 15GB VRAM).

4.1 Framework and Model

We selected **Unsloth** due to its optimized kernels, offering 2x faster training and 60% less memory usage compared to standard Hugging Face implementations.

- **Model:** `Qwen/Qwen2.5-3B-Instruct`. Chosen for its superior performance on Arabic benchmarks compared to Llama 3.2 3B.
- **Quantization:** 4-bit (QLoRA) to fit parameters/gradients within 15GB VRAM.

4.2 Training Hyperparameters

We utilized Low-Rank Adaptation (LoRA) with the following configuration:

Table 2: LoRA Fine-Tuning Configuration

Parameter	**Value / Choice**
LoRA Rank (r) / Alpha	16 / 16
Target Modules	All Linear Projections ($q, k, v, o, gate, up, down$)
Max Seq Length	2048 (Training), 8192 (Inference)
Batch Size	1 (per device)
Gradient Accumulation	8 (Effective batch size = 8)
Optimizer	`adamw_8bit`
Learning Rate	2×10^{-4}

Justification: We reduced the batch size to 1 and increased gradient accumulation to 8. This allowed the model to process longer sequences without Out-Of-Memory (OOM) errors during training.

5 Results and Inference Optimization

5.1 Addressing Context Limitations

During training, we limited context to 2048 tokens. At inference time, we reloaded the model with `max_seq_length = 8192`. This leveraged Qwen's RoPE scaling, allowing the SLM to summarize documents up to **25,000 characters** without retraining.

5.2 Generation Parameters

Initial tests showed repetition loops. We optimized the generation config:

- **Temperature = 0.5:** Encouraged structured summaries (bullet points).
- **Repetition Penalty = 1.2:** Strictly penalized looping behavior.
- **System Prompting:** Explicit instructions to "summarize in Arabic only" to prevent Chinese/English token leakage.

5.3 Model Evaluation

To quantify summarization quality on unseen data, we evaluated the fine-tuned model on a held-out test subset using three complementary metrics: ROUGE (lexical overlap), BERTScore (semantic similarity), and an LLM-as-a-Judge protocol (qualitative, reference-aware/faithfulness-aware scoring). This multi-metric approach is important for Arabic summarization because purely overlap-based scores (ROUGE) can underestimate quality when the model produces valid paraphrases.

5.3.1 Automatic Metrics: ROUGE

We computed ROUGE between the model-generated summaries and the reference summaries in the test set. The obtained scores are:

- **ROUGE-1:** 0.2967
- **ROUGE-2:** 0.1272
- **ROUGE-L:** 0.2828
- **ROUGE-Lsum:** 0.2852

These results indicate meaningful lexical alignment with the reference summaries, including non-trivial bigram overlap (ROUGE-2), which is typically more difficult to achieve.

5.3.2 Semantic Metrics: BERTScore

To better capture semantic equivalence beyond exact word overlap, we computed BERTScore on the same test subset. The mean scores are:

- **Precision:** 0.7791
- **Recall:** 0.7226
- **F1:** 0.7497

The BERTScore results suggest that the generated summaries preserve the meaning of the reference summaries with strong semantic similarity, even when phrasing differs.

5.3.3 LLM-as-a-Judge (Qualitative Evaluation)

Finally, we used an LLM-as-a-Judge evaluation to assess summary quality from a more human-like perspective (faithfulness, coverage, coherence, and Arabic linguistic quality). Due to cost/latency constraints, this judge-based evaluation was conducted on a sample of 30 test examples. The judge produced the following aggregate result:

Average Judge Score: 8.44 / 10

This score indicates high perceived quality, with strong fluency and coherence, and (critically) strong faithfulness to the source text when prompted to penalize hallucinations.

6 Conclusion

Across overlap-based, semantic, and judge-based evaluations, the model demonstrates solid performance on Arabic summarization. ROUGE confirms lexical alignment, BERTScore confirms semantic preservation, and the LLM-as-a-Judge score supports strong human-perceived quality, suggesting the pipeline produces reliable Arabic summaries under the long-context inference configuration.