# Part 1

## Overview

A clean, stateless REST API for inferring sentiment (Positive / Neutral / Negative) from customer-service call summaries in both English and Arabic, leveraging open-source, self-hosted LLMs.

**Please find code and detailed explanation of Part 1 hosted on Github here:**

https://github.com/mohamadhabash/maqsam-sentiment/tree/master

# Part 2

## Overview

This part outlines the production deployment and scaling strategy for the sentiment analysis API developed in **Part 1**. The service uses a quantized version of the JAIS-family-6p7b-chat model (4-bit INT4 precision) hosted on an NVIDIA T4 GPU (16 GB VRAM). This model delivers high sentiment classification accuracy (95% for Arabic, 98% for English) while staying within memory and latency constraints, achieving a peak memory usage of 8.1 GB and average inference latency of 1664.3 ms.

---

## 1. Infrastructure Design

We deploy the model as a RESTful FastAPI microservice, containerized using Docker and orchestrated by Kubernetes. The container image preloads the quantized JAIS model to avoid cold starts and reduce latency.

## 2. Cost-Effective Scaling Strategy

To handle Maqsam's high daily call volume and fluctuating demand between day and night hours, we adopt the following dynamic scaling plan:

- **Use NVIDIA T4 GPU nodes** with 16 GB memory to fit the 6.7B JAIS model quantized to 4-bit (~8.1 GB peak usage).

- **Configure Kubernetes Horizontal Pod Autoscaler (HPA)** to scale based on CPU/GPU utilization or request queue length. Horizontal scaling means that the response to increased load is to deploy more Pods.
- **Route traffic (API Requests) through a load balancer** to scale across multiple pods running the LLM API, ensuring high availability and throughput by evenly distributing the requests across all available pods.
- **Use time-based scaling or predictive scheduling** to use more pods during known high-traffic hours (e.g., daytime), and scale down during low-traffic periods (e.g., late night). This complements HPA by proactively aligning GPU resources with call volume patterns, reducing cold starts and controlling cost.
    - If Maqsam gets a huge volume of calls every weekday at 9 AM, we can automatically add more pods just before 9 AM (e.g., at 8:45 AM), Then scale them down again at night when traffic drops.
    - This can be rule-based, or a forecasting model trained on historical data (e.g. LSTM)

# 3. GPU Utilization & Inference Estimates

We base our throughput and memory planning on observed inference metrics of the quantized 6.7B JAIS model on a T4 GPU (these were calculated in **Part 1** using **benchmark_inference.py** script):

- **Peak VRAM during inference:** 8.1 GB
- **Inference time per request:** ~1664.3 ms
- **Expected throughput per T4 GPU:** ~35–40 requests/minute (assuming sequential execution)

# 4. Future Feature Integration

The system is designed to be easily extensible to future LLM-driven features like:

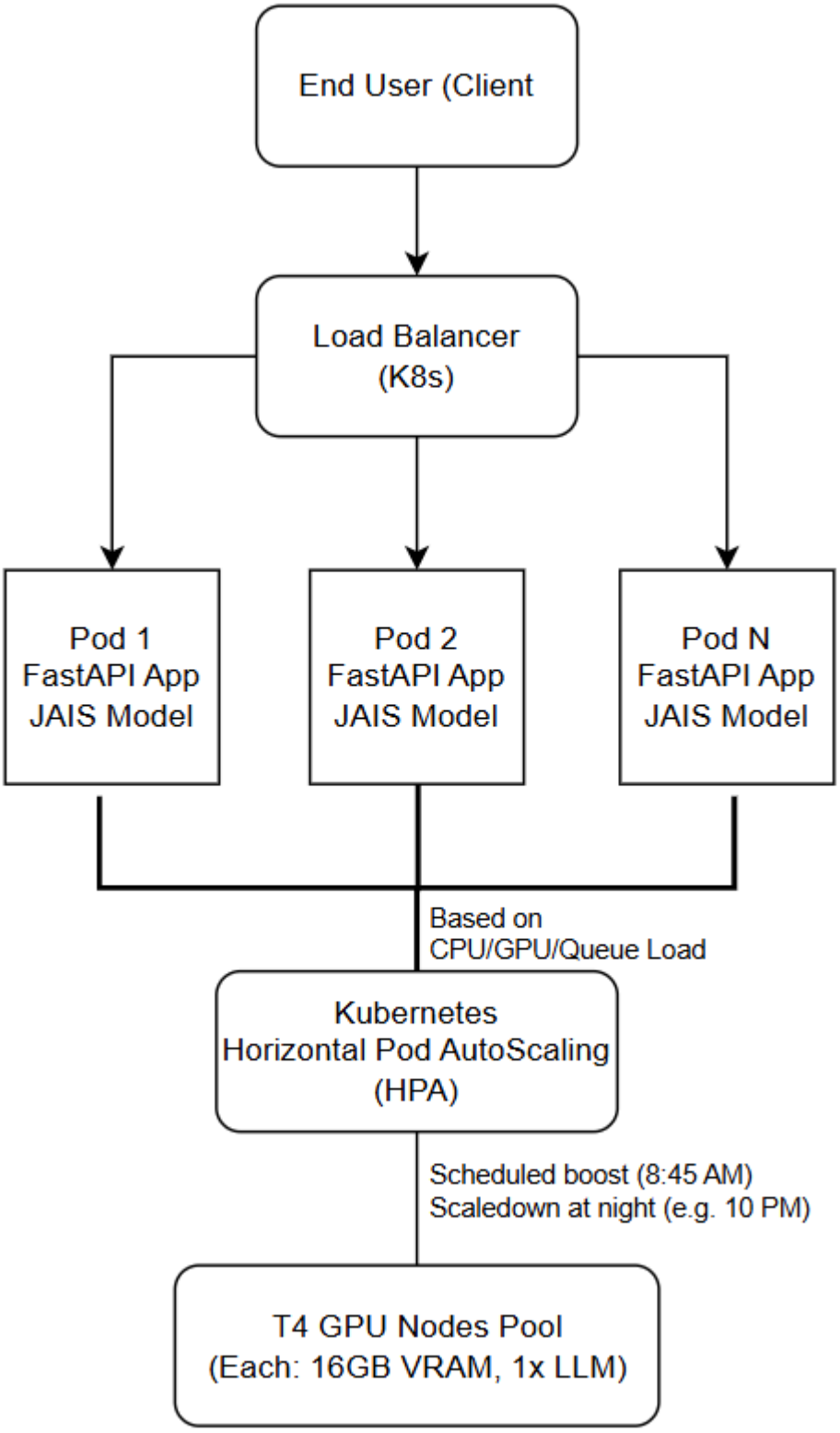
- **Call summarization – Part 3** (e.g., using the same model with modified prompt or different task adapter).
- **Keyword extraction** for quality assurance and call routing.
- **Fraud detection** using zero-shot classification capabilities of the LLM.

These features can share the same serving architecture with prompt-based routing (by adding prompts for additional tasks in **config.yaml** file in **Part 1**) or via multitask adapters (e.g., LoRA) without retraining the whole base model.

# 5. Assumptions

- Google Colab with NVIDIA T4 GPU is assumed as the available deployment environment.

- We chose the JAIS-family-6p7b-chat model for its superior performance (Accuracy: 0.95/0.98 and F1: 0.94/0.98 for Arabic/English).
- Quantization (4-bit) is used to make deployment on 16GB GPUs feasible without sacrificing accuracy.



# Part 3

## Overview

This part outlines a strategy for developing a customer service call summarization system that handles both Arabic and English languages, leveraging multimodal inputs (audio and text) and state-of-the-art models. The approach includes options such as JAIS models, LoRA adaptation, sequence-to-sequence architectures, and incorporates audio features through ASR speech encoder model.

---

## 1. Model Architecture & Selection

### 1.1 JAIS Models with Prompt Engineering

- **Model**: `jais-family-6p7b-chat` with 4-bit quantization.
- **Deployment**: Efficiently runs on T4 GPUs (16 GB), with ~7.5 GB RAM usage and ~8.1 GB peak during inference.
- **Prompting Strategy**: Utilize structured prompts to guide the model for summarization tasks.
    - **Example Prompt**:

      Summarize the following customer service call: [Transcription]
- **Advantages:**
    - Bilingual capabilities (Arabic and English).
    - High performance on sentiment analysis tasks, indicating potential high performance for summarization.
    - Can be integrated into the existing sentiment analysis deployment with different prompts, optimizing resource utilization (This can be done by adding an additional prompt under tasks in the YAML config file in **Part 1).**

### 1.2 Efficient Adaptation Techniques for Smaller LLM Models

To adapt smaller JAIS models (e.g., jais-family-590m-chat) for summarization tasks while conserving computational resources, several parameter-efficient fine-tuning methods can be employed:

- **Low-Rank Adaptation (LoRA)**: LoRA introduces trainable low-rank matrices into the model's architecture, allowing for effective fine-tuning with a significantly reduced number of parameters. This method maintains the original model weights frozen,

leading to lower memory usage and faster training times.

- **Adapter Modules**: These are lightweight, trainable components inserted into each layer of a pre-trained model. During fine-tuning, only the adapter parameters are updated, leaving the original model weights unchanged. This approach allows for task-specific adaptation with minimal additional parameters.

- **Prefix Tuning**: This technique involves prepending trainable vectors (prefixes) to the input sequence or to each layer's hidden states. The main model parameters remain frozen, and only the prefixes are optimized, enabling efficient adaptation to new tasks.

These methods offer efficient alternatives to full fine-tuning, enabling rapid adaptation to summarization tasks with reduced computational overhead and memory usage.

## 1.3 Sequence-to-Sequence Models

**Models:**

- **T5 (Text-to-Text Transfer Transformer):** T5 is an encoder-decoder model that frames all NLP tasks into a text-to-text format. It is pre-trained on a multi-task mixture of unsupervised and supervised tasks, converting each task into a text-to-text format. T5 works well on various tasks out-of-the-box by prepending a different prefix to the input corresponding to each task, such as "summarize:" for summarization tasks.
- **Long-T5:** Long-T5 is an extension of the T5 model designed to handle longer input sequences. It integrates attention mechanisms from long-input transformers and adopts pre-training strategies from summarization pre-training into the scalable T5 architecture. This model achieves state-of-the-art results on several summarization tasks.

**Training:**

- **Fine-Tuning on Domain-Specific Datasets:** Both T5 and Long-T5 can be fine-tuned on domain-specific datasets that consist of paired transcriptions and summaries. This process involves adjusting the pre-trained models to better fit the specific characteristics and requirements of the target domain, such as customer service call summaries.

- **Teacher Forcing:** During training, teacher forcing is employed, where the model is provided with the ground truth output sequence to predict the next token in the sequence. This technique helps the model learn the correct sequence generation by minimizing the discrepancy between the predicted and actual sequences.

- **Optimization Techniques:** Optimization algorithms such as AdaFactor are utilized for training these models. AdaFactor is known for its memory efficiency and

effectiveness in training large-scale models like T5.

- **Handling Long Sequences:** For tasks involving long input sequences, such as summarizing lengthy call transcriptions, Long-T5 is particularly suitable. It employs specialized attention mechanisms to efficiently process long texts without significant performance degradation.

By leveraging these sequence-to-sequence models and training methodologies, the system can effectively generate concise and coherent summaries from complex and lengthy customer service call transcriptions.

# 2. Multimodal Integration

## 2.1 Automatic Speech Recognition (ASR)

- **Function**: Converts audio calls into text transcriptions while also providing intermediate speech representations (embeddings) from its encoder. These embeddings capture rich prosodic and acoustic features such as tone, rhythm, and speaker emotion.

- **Multimodal Use**: These ASR-based speech embeddings can be integrated into the summarization pipeline in two ways:

    - **LLM Adaptation**: Adapt a large language model (e.g., JAIS) to accept speech embeddings alongside text tokens, transforming it into a speech-aware LLM.

    - **Sequence-to-Sequence Integration**: Use the speech embeddings as input to a multimodal encoder-decoder architecture like T5 or Long-T5, extending its input modality beyond text to support audio features.

## 2.2 Incorporating Audio Features

- **Prosodic Features**: Elements such as tone, pitch, and speech rate help identify emphasis, stress, and speaker emotion.

- **Silence and Overlaps**: Pauses and overlaps can signal confusion, urgency, or speaker turn-taking dynamics.

- **Integration**: These features, captured in ASR encoder outputs, can enrich the model's input to improve contextual understanding and summary quality.

## 2.3 Related Paper

The paper "Prompting Large Language Models with Audio for General-Purpose Speech Summarization" introduces an innovative framework that integrates audio processing with large language models (LLMs) for speech summarization. Key highlights include:

- **Modality-Invariant Processing:** An audio encoder is trained to convert speech into token representations compatible with LLMs, enabling the model to process speech inputs similarly to text.
- **Flexible Summarization Styles:** By varying prompts, the system can generate summaries in different styles, showcasing adaptability across various domains.
- **Performance Improvements:** The proposed method outperforms traditional cascaded approaches (ASR followed by text summarization) on benchmarks like the CNN/DailyMail dataset.

This approach demonstrates the potential of combining audio processing with LLMs to achieve versatile and efficient speech summarization.

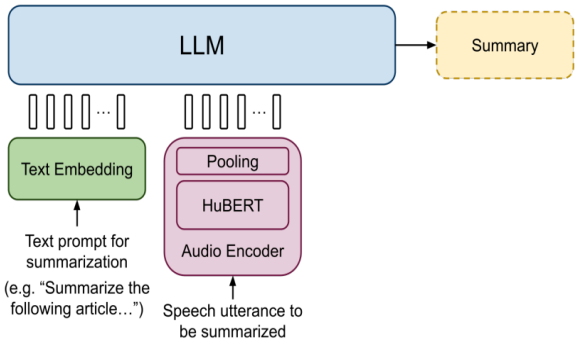



Figure 2: *Speech summarization can be performed by prompting the LLM using any combination of text and speech inputs.*

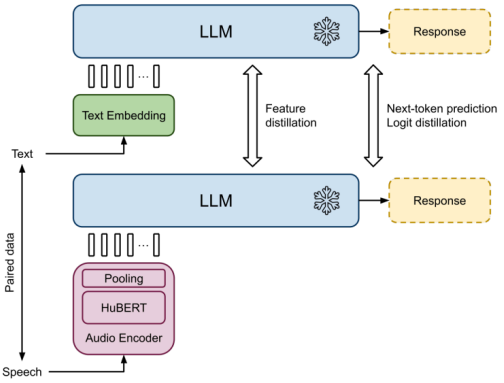



Figure 1: *The audio encoder is trained to produce token embeddings such that the LLM's response to a spoken prompt matches its response to a text prompt with the same semantic content.*

# 3. Data Sources & Preparation

## 3.1 Open-Source Datasets

- **XLSum**: A multilingual summarization dataset encompassing both Arabic and English articles.

- **SumArabic**: An Arabic abstractive text summarization dataset derived from news articles.

- **TWEETSUMM**: A dialog summarization dataset tailored for customer service interactions, featuring approximately 6,500 human-annotated summaries.

- **NatCS**: A collection of natural customer support dialogues, designed to reflect real-world conversational characteristics.

## 3.2 Data Preparation Pipeline

- **Dataset Identification**:

    - **Text-Based Datasets**: If available, utilize existing text-based customer service datasets (e.g., TWEETSUMM) for direct fine-tuning of summarization models.

    - **Audio-Based Datasets**: In the absence of suitable text datasets, we can use use audio-based datasets and proceed with the following steps:
        - **Transcription**: Apply ASR to convert audio recordings into text transcriptions.

        - **Summarization Generation**: Utilize advanced Large Language Models (LLMs), like ChatGPT-4o, Gemini or Deepseek, to generate high-quality summaries from the transcribed text.

        - **Annotation**: Use the generated summaries alongside their corresponding transcriptions to create paired datasets for supervised training.

        - **Feature Extraction (for Multimodal Training only)**: Extract prosodic and acoustic features (e.g., tone, pitch, speech rate) from the audio data to enrich the input for multimodal summarization models.

# 4. Model Evaluation, Monitoring & Drift Handling

## 4.1. Model Evaluation & Monitoring

To evaluate and ensure the summarization model remains accurate and relevant over time, we can track:

- **ROUGE Scores (ROUGE-1, ROUGE-2, ROUGE-L)**: Measure n-gram and sequence overlap between generated summaries and human references.

- **BERTScore**: Captures semantic similarity using pre-trained embeddings. Helpful when wording differs but meaning is preserved.

- **Length Ratio & Compression Ratio**: Monitor output length vs. input to detect over- or under-summarization.

- **Human Evaluation (Periodic)**: Judges coherence, fluency, and informativeness, especially important in Arabic-English dialogue settings.
- **LLM Evaluation (Periodic)**: Again ,we can utilize top-tier LLMs (Ex. ChatGPT) and prompt them to evaluate the summarization of a given call transcript (Ex. from 0 to 10)

## 4.2 Drift Detection:

- **Data Drift**: Compare new call transcription patterns (e.g. length, vocabulary, language mix) to training data via distributional checks.

- **Concept Drift**: Track drop in ROUGE/BERTScore on fresh annotated samples or via spot human/LLM evaluation.

## 4.3 Drift Handling:

- **Scheduled fine-tuning** (e.g., monthly) with new data (calls + summaries).

- **Triggered retraining** if metrics fall below thresholds.

- **This is why we NEED LoRA or adapter-based updates** that enable lightweight, fast adaptation without full retraining to quickly handle such cases.
- **Creating new Test Sets periodically** to cover any new vocabulary, patterns, etc. in the customer service calls.

# 5. Deployment

To deploy the summarization model effectively, we focus on flexibility, efficiency, and responsiveness:

- **Infrastructure**: We package the model and its dependencies using Docker and recommend Kubernetes for managing scaling and reliability as load varies.

- **Resource Management**: Since large models can be resource-heavy, we carefully choose model sizes that fit available GPU memory (e.g., T4 with 16 GB) and apply 4-bit quantization to reduce memory usage without compromising performance.

- **Latency Optimization**: We aim to strike the right balance between output quality and speed.