Deep Shukla

Kendall Ahern

Mateusz Muszynski

CSCI 5253 - Project Proposal

Multilingual Text Intelligence Cloud Service

Why are we doing this?

For motivational purposes, suppose you are running an online store. One customer leaves a review in French, another sends in a support ticket in Japanese, and a third posts on social media in Spanish. Right not, making sense of all this can be a headache – you need translators, different tools, and a lot of time just to figure out what your customers are saying.

Well, in today's globally connected digital ecosystem, organizations frequently receive user-generated content in multiple languages, and across various platforms - such as customer support portals, social media, e-commerce reviews, and research submissions. The lack of automated multilingual analysis tools often requires manual translation followed by separate processing, which is both time-consuming and inefficient. Our project, to help combat this challenge, proposes a cloud-native Multilingual Text Intelligence Service that automatically detects, translates, and analyzes text in real time or in a scalable asynchronous manner.

The service will accept user-submitted text in any supported language and produce standardized outputs in English. The goal is to have our service instantly tell you 3 things: sentiment analysis (is the customer happy or sad), text summarization (what's the main point), and named entity recognition or NER (what are the key names, places, or products they're talking about). The system is designed using distributed cloud architecture to demonstrate scalability, resilience, and modularity using multiple datacenter technologies.

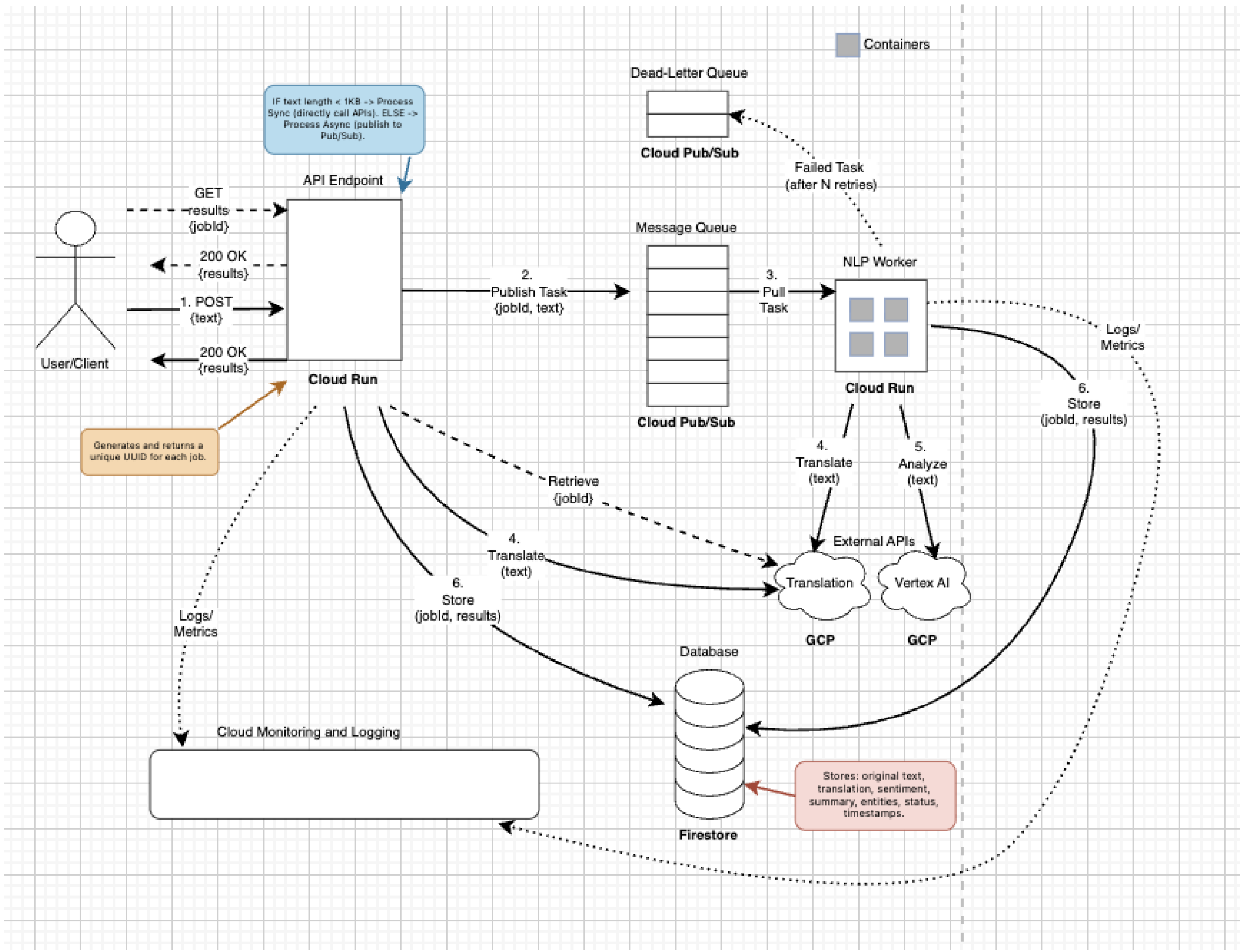
Project Goals

Our main goals for the project are pretty straightforward:

- Develop a RESTful API service that accepts and processes multilingual text inputs.
- Implement automatic language detection and English translation using cloud-based AI services.

- Integrate advanced NLP capabilities such as sentiment classification, summarization, and entity extraction.
- Support both synchronous real-time processing for short texts and asynchronous queued processing for larger or complex documents.
- Utilize distributed datacenter components to demonstrate cloud-native scalability and robustness.
- Provide a structured output and retrieval mechanism allowing users to query the status and results of submitted text.

System Architecture



Containers
Dead-Letter Queue
Cloud Pub/Sub
IF text length < 1KB -> Process Sync (directly call APIs). ELSE -> Process Async (publish to Pub/Sub).
Failed Task (after N retries)
API Endpoint
GET results {jobId}
200 OK {results}
1. POST {text}
200 OK {results}
User/Client
Cloud Run
Generates and returns a unique UUID for each job.
2. Publish Task {jobId, text}
Message Queue
Cloud Pub/Sub
3. Pull Task
NLP Worker
Cloud Run
Logs/ Metrics
6. Store (jobId, results)
4. Translate (text)
5. Analyze (text)
Retrieve {jobId}
External APIs
Translation
Vertex AI
GCP
GCP
4. Translate (text)
6. Store (jobId, results)
Logs/ Metrics
Database
Firestore
Cloud Monitoring and Logging
Stores: original text, translation, sentiment, summary, entities, status, timestamps.

https://drive.google.com/file/d/1U2kC39tSD57egR8u_rpE1e4FngQjojKq/view?usp=sharing

We are building this system with a modular, "lets let the pieces work together" approach. More concretely, this system follows a microservices-based cloud architecture with an event-driven design. The API layer will act as the entry point for user requests in basic text form. This incoming text will be evaluated to determine processing mode (sync vs. Async). For smaller inputs, the system processes text synchronously by directly invoking translation and NLP services. For larger or high-latency operations, the API publishes the task to a message queue which will enable asynchronous processing by downstream worker services.

The architecture separates responsibilities across the following modular services: translation services to convert the text into English, NLP models to generate sentiment scores, summaries, and entity labels, and a database to store both input content and output analysis. The hope is that the results are retrievable using a job identifier via the API. Cloud Monitoring will also be used to observe system health, track latency, and log failures for debugging.

Software and Hardware Components

Software Components:

- Google Cloud Run – Where our main API and background worker services – including the public API and NLP services - will live. Allows a simple way to run code without messing with the servers
- Google Cloud Pub/Sub – Provides message queue capabilities for when we want to do asynchronous processing with the larger jobs. Additionally offers fault-tolerant task distribution.
- Google Cloud Translation API – Automatically detects input language and performs machine translation into English.
- Vertex AI or Hugging Face Models (deployed via Cloud Run) – Executes sentiment analysis, summarization, and entity recognition.
- Google Firestore or Cloud SQL – Preserves translated text, NLP results, job metadata, and status for end-user retrieval – leaning towards firestore but wanted to note the possibility of both
- Cloud Logging and Monitoring – Enables system observability, error detection, and performance tracking, possibly implemented as a dashboard
- Google Cloud Storage – Used for storing large files or documents if document upload capability is incorporated (if time permits or we run into an issue with the large files, wanted to note this as a possible solution/workaround).

Hardware Components:

We shouldn't have to worry about hardware. We are using Google’s cloud which means the servers are all virtual and scale up or down automatically. If we get a sudden spike in traffic, Google will handle it for us.

Component Interaction Workflow

1. API Request Submission:
   Users submit text through an HTTP endpoint hosted on Cloud Run. Each request is assigned a unique job identifier.
2. Processing Mode Determination:
   The system evaluates the length of the text. If it is below a defined threshold, it is processed synchronously. Larger texts or batch submissions are sent to a Pub/Sub queue for asynchronous processing.
3. Language Detection and Translation:
   The submitted text is passed through the Cloud Translation API, which automatically detects the source language and translates the text to English.
4. Natural Language Processing Stage:
   The translated text undergoes further processing using either managed Vertex AI language models or containerized NLP models. This stage performs sentiment analysis, summarization, and entity recognition.
5. Data Storage:
   Outputs are stored in Firestore or Cloud SQL, enabling persistent storage of results along with metadata such as processing timestamps, language used, and model confidence scores.
6. Result Retrieval:

   Users can retrieve the results via a dedicated status endpoint by supplying the assigned job identifier. The system returns the processed output to the user in structured JSON format.

7. Monitoring and Error Handling:

   Cloud Monitoring and Logging automatically track performance and collect error data. Failed messages are automatically re-queued or routed to a dead-letter topic for investigation.

Debugging, Testing, and Evaluation Plan

- Unit Testing: API endpoints and NLP functions will be tested individually to ensure correctness.
- Integration Testing: End-to-end flow will be tested using staged test inputs that span multiple languages and document lengths.
- Load and Stress Testing: Tools such as Locust will be used to test concurrency and measure system behavior under high request loads.
- Observability: Logs and metrics will be captured via Cloud Logging and Monitoring to diagnose failures, performance bottlenecks, and latency issues.
- Model Evaluation: The accuracy of sentiment, summarization, and entity recognition will be evaluated using publicly available multilingual datasets. Metrics such as F1-score and ROUGE will be applied to evaluate performance. In the early stages of development, there will also be a component of manual verification as well using common sources like Google Translate to verify the translation and our general knowledge to determine the sentiment and evaluate how accurate the summary is.

Alignment with Course Requirements

This project directly satisfies the course requirement to implement a distributed service using at least four cloud datacenter technologies. Specifically, we chose to incorporate these 4:

1. Message Queueing: Google Cloud Pub/Sub
2. Microservices / Containers: Cloud Run
3. Databases: Firestore or Cloud SQL
4. Managed AI Services: Cloud Translation API and Vertex AI

We also will have:

Serverless Compute and Orchestration: Integration across these services will demonstrate principles of scalability, elasticity, and asynchronous execution within a real datacenter environment – many of the components that we discussed throughout this class in lectures thus far.