Translation Prototype Report

Yehor KOROTENKO

26 May 2025

1 Introduction

The goal of the prototype was the developing an application that would be able to translate entirely a chosen directory to particular languages and translation of the French Python Course in order to show the efficiency of the developed application.

2 Application architecture

For the ease of creating applications of different formats such as `CLI`, `GUI` the choice of developing a library has been made. Furthermore, on top of that library a *demo* `CLI` application has been created to simplify the usage and testing process.

The library and the CLI app can be found using the next links:

1. https://github.com/DobbiKov/translate-dir-lib - for the library
2. https://github.com/DobbiKov/translate-dir-cli - for the CLI

2.1 Library architecture

The library is divided into the next modules:

- **project** - main entry to the library, the only module that "talks" to external applications
- **project_config** - designed for formatting and working with a particular project's configuration. Will be used for imitating *git* architecture in the future.
- **translator** = temporary module for translation, built to simplify the process of prompt or model choice
- **helper** = helper module to work with text providing such functions:
  - divide text into chunks
  - extract output from the model's response
  - etc.

2.2 Application logic

The *git* has been an inspiration for the application logic, thus, to create a translation logic, a directory must be chosen to be a *translation project* and be initialized using the provided command from the library.

2.2.1 Directory/language settings

Inside of such *project*, the *source directory* must be set as well as its' language, visually:

```
| project_root/
| ---- trans_conf.json
| ---- source_dir/ # with a language set
| ---- | ---- [contents of source_dir]
```

The functionality of adding *target language* as well as its removal also provided in the library. Important to note, that an addition of a target language creates a directory of the next format: `[project_name]_[language]`. Example: the project with a name: `translate_temp` and French language gives the next directory: `translate_temp_fr`. The architecture looks like:

```
| project_root/
| ---- trans_conf.json
| ---- source_dir/ # with a language set
| ---- | ---- [contents of source_dir]
| ---- translate_temp_fr/
| ---- | ---- [contents of translate_temp_fr]
```

2.2.2 Syncing and translation

All the files inside of the source directory are considered to be untranslatable by default. In order to make them translatable, the commands: `make_translatale` and `make_untranslatable` are provided.

`sync` command triggers the copy process of all the *untranslatable* files to the *target languages* directories in order to recreate the *source directory* structure to simplify the use process (makes it easy to compile when translating `latex` or `myst` project). Important: `translatable` files are not copied during the *syncing* process.

In order to translate *translatable* files, the two commands are provided:

- `translate_file` - takes a file path as an argument and a target language, translates it and copies to the *target language directory* keeping the structure (in the same way as it's done for the syncing proces).
- `translate_all` - translates all the *translatable* files using `translate_file` command.

3 Large Language Model

For the prototype testing the `Gemini 2.0 Flash` model has been chosen due to the next reasons:

- google provides free API usage with up to `1500` requests per day for the model name above
- `Gemini 2.0 Flash` is not expensive, especially in comparison to popular alternatives such as `GPT` or `Claude` (e.g: for this model the cost is `0.1$` per 1 million input tokens and `0.4$` per 1 million output tokens)
- less energy consumption in the comparison with alternatives
- ease of API usage

3.1 Energy consumption

`Gemini 2.0 Flash` is a new low-latency model from Google that works much faster then alternatives and consumes less energy as a consequence, the comparison table is provided below:

Model name	Energy consumption per request
GPT 4o	0.3 Wh
Gemini 2.0 Flash	**0.022 Wh**
Gemini 2.0 Flash-Lite	0.016 Wh
Claude Haiku 3.5	0.22 Wh
Claude Opus 3	4.05 Wh

Table 1: LLMs' energy consumption comparison table

References: [1], [2]

From the provided Table 1, we can deduce that *Gemini 2.0 Flash* is the most *energy efficient* model provided on the market giving state-of-the-art results.

4 Python Course Translation

To test the application the task to translate the python course has been established. In this section we will investigate the results.

For the course, 72 files needed to be translated that is approximately 132000 tokens. The files have been divided into chunks, each one containing at most 50 lines of the given file (it is temporary, the chunking algorithm will be refined in the future).

Each request contained 500 tokens in general to which always the prompt has been added that is 2054 tokens long. As a consequences, there were approximately 264 request each containing 2554 tokens in general.

Summing the numbers above, we get $2554 * 264 \approx 675000$ input tokens and 132000 output tokens, using [3], we get a total cost less then 0.5$.

For the energy consumption, we use Table 1, then $264 * 0.022 = 5.808 \approx 6$ Wh that is equivalent to charging a smartphone from 0% to about 15–20%.

5 Encountered problems

The syntax understanding as well as natural text extraction is a difficult process for the Large Language Models such as `Gemini 2.0 Flash` especially when it is cut out of the context as it was in this example due to the inefficient chunking.

For instance, the model doesn't always understand that a small word is a natural text and not a variable or a part of syntax. Also, in cases where the code cell is divided into parts for chunking, the model misinterprets function and variable names as text and translates them requiring the user to fix those problems.

6 Future work

Due to the problems listed in Section 5, the high importance goal is to rethink and rework the chunking algorithm.

Secondly, an important task is to refine the prompt and investigate possible solution to minimize syntax and layout changes as well as the amount of text misinterpreted as a part of code.

Bibliography

[1] Riya Bansal, "Gemini 2.0 Flash vs o4-mini Can Google Do Better Than OpenAI?." [Online]. Available: https://www.analyticsvidhya.com/blog/2025/04/gemini-2-0-flash-vs-o4-mini/?utm_source=chatgpt.com

[2] Julian Wong, "Power consumption of ChatGPT queries not as high as previously thought." [Online]. Available: https://www.linkedin.com/pulse/power-consumption-chatgpt-queries-high-previously-thought-julian-wong-ey28c

[3] "Gemini Pricing." [Online]. Available: https://ai.google.dev/gemini-api/docs/pricing