Generative AI-based Translation tools Suite

by

Ahbab Hossain
24141201

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
BRAC University
January 2026.

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Ahbab Hossain

24141201

Approval

The thesis/project titled "Generative AI-based Translation tools Suite" submitted by

1. Ahbab Hossain (24141201)

of [Fall], [2020] has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science in [Fall] 2025.

Examining Committee:

Supervisor:
(Member)

Rafeed Rahman

Senior Lecturer

Brac University

Thesis Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam

Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Dr. Sadia Hamid Kazi

Chairperson and Professor
Department of Computer Science and Engineering
Brac University

Abstract

This project presents the design and development of an intelligent document translation and preview system capable of handling multi-format files including Word, Excel, text, and images. The system integrates Optical Character Recognition (OCR), automated translation, and interactive preview functionalities within a unified interface. A dynamic tab-based architecture allows users to manage multiple translation sessions simultaneously, with real-time state management ensuring synchronization between the client and server.

The system leverages modern large language models (LLMs), such as the Gemini API, to perform context-aware translations. To enhance efficiency, the backend implements asynchronous batching and batched API calls, minimizing latency during large-scale translation tasks. The chat assistant module provides contextual editing support and instruction-based translation refinement, allowing users to iteratively improve outputs without manual re-uploading.

Advanced preview mechanisms have been implemented for each file type: formatted text display for Word, table parsing for Excel, OCR-based text extraction and alignment for images, and editable text containers for plain files. Together, these modules form a cohesive, scalable, and extensible framework for intelligent multilingual document processing. The result is a technically robust system that demonstrates a practical balance between automation, user control, and performance optimization.

Keywords: Document Translation, Large Language Models, LLMs, Gemini API, OCR, API Batching, State Management, Multilingual Systems, Web Application, Intelligent Document Preview

Acknowledgement

I wish to thank my supervisor, Mr. Rafeed Rahman, for his unwavering support, patience, and supervision throughout this journey.

I also wish to thank all the faculty members, administrative officers, support staff, security staff, library support staff, and to all my seniors, juniors and fellow batch mates who have indirectly or directly been a support in this life-changing leap called university.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Language remains one of the most persistent barriers in global communication and data processing. While translation tools and multilingual systems have made tremendous progress, real-world scenarios still involve handling a mix of documents, spreadsheets, images, and text files, often containing multilingual content that needs to be processed accurately and efficiently.
Traditional translation platforms such as Google Translate or DeepL are powerful for single-sentence translation but tend to lack workflow support for batch or document-level operations. They also offer limited transparency—users cannot easily review intermediate text extraction steps or selectively re-translate specific portions.
To overcome these challenges, this project introduces a Generative AI–based Multi-Format Translator, a web application capable of handling:

- File-type-specific workflows: Doc, Spreadsheet, image, and txt pipelines for efficient text extraction and translation.
- Interactive previews: Users can see raw text alongside translations for intuitive verification.
- Chat-based assistance: Users can suggest refinements in or alternative translations for individual text segments.
- Export and copy functionality: Translated content can be exported or copied in convenient formats.

This work demonstrates how combining state management, batch API calls, and dynamic previews can significantly improve user experience and productivity, creating a robust tool for handling multilingual document processing tasks.

1.2 Problem Statement

Although numerous translation tools exist, few support multi-format, interactive translation workflows that combine accuracy, speed, and flexibility. The following limitations were identified during research and development:

- **Limited File Type Support**

  - Most AI translation APIs focus on plain text, lacking integrated support for Word, Excel, image, or text file structures. Users must manually extract and reformat data before translation.

- **Error Propagation Between OCR and Translation**
  - OCR errors from images and formatting inconsistencies from Word or Excel often lead to translation inaccuracies. Once these errors occur, users cannot correct them without re-running the entire process.

- **High Latency from and Inefficiency**
  - Sequential translation calls per text segment create performance bottlenecks. Large spreadsheets or multi-paragraph documents significantly slow down processing times.

- **Lack of Real-Time Interactivity**
  - Existing systems generally produce static outputs. Users have no way to chat with the system, request alternate phrasing, or modify a single translated unit in real-time.

- **Context and Coherence Loss**
  - Segment-wise translation frequently leads to inconsistency in terminology and tone across documents. Traditional APIs translate sentences independently, losing broader semantic context.

- **Minimal State Awareness**
  - Many translation platforms lack state management logic — meaning the system forgets user corrections, preferences, or partial progress during a session.

This project addresses these problems through an interactive, state-aware translation engine that uses Gemini's contextual generative AI, batch API requests, and dynamic previews to deliver fast, editable, and coherent translations across multiple file types.

Chapter 2

Literature Review

2.1 Evolution of OCR: from rules to transformers

Optical Character Recognition (OCR) has evolved from rule-based pipelines (binarization, connected-component analysis, hand-crafted feature extraction) to deep-learning driven systems. Early modern pipelines combined Convolutional Neural Networks (CNNs) for visual feature extraction with Recurrent Neural Networks (RNNs) or LSTMs for sequential character modeling. More recently, transformer-based architectures [3] have been adapted for OCR,[5] yielding significant gains in robustness and generalization by leveraging large-scale pretraining and sequence-to-sequence decoding. TrOCR (Transformer-based OCR), for example, replaces classical CNN–RNN stacks with a vision transformer [2] encoder and a language transformer decoder, enabling end-to-end recognition that benefits from pretraining on large synthetic and real datasets. This shift improves recognition on complex layouts and diverse fonts, and it reduces the need for language-specific heuristics.

2.2 Surveys and empirical comparisons of OCR methods

Comprehensive surveys of text detection and recognition emphasize two core sub-problems—detection of text regions and recognition of the text within them—and document persistent challenges such as multi-lingual scripts, irregular layouts, and low-quality images. Recent comparative studies show that while transformer-based OCR models often lead for clean, printed text, classical engines like Tesseract [1] and specialized models continue to be competitive on resource-constrained settings or when heavy pre- and post-processing [5] is applied. These surveys underline the importance of preprocessing (binarization, denoising, deskewing) and layout-aware post-processing to improve downstream translation quality.

2.3 Text-Image Machine Translation (TIMT) and multimodal approaches

Text Image Machine Translation (TIMT) refers to translating textual content that appears inside images (e.g., signs, screenshots, scanned documents). Traditional

TIMT follows a cascade architecture—OCR followed by an MT system—which is simple and modular but susceptible to error propagation (OCR errors degrade MT output). Recent research is moving toward multimodal and end-to-end approaches [7] that jointly model visual and textual cues to reduce this propagation and improve disambiguation (for example, when visual context resolves lexical ambiguity). Surveys and new models in 2023–2025 highlight modal-contrastive learning, codebook-style multimodal representations, and joint training strategies that close the modality gap between visual inputs and translation decoders. These trends indicate that combining visual context with linguistic modeling improves translation accuracy in many challenging cases.

2.4 Integrated OCR + translation systems and practical prototypes

Beyond academic models, several prototypes and applied systems have explored integrating OCR with translation in practical pipelines. Works such as "TransDocs" [4] explore OCR-to-translation chains and word-to-word mapping to keep alignment between source and translated tokens, illustrating the engineering trade-offs between modularity and end-to-end optimization. Practical systems typically favor a hybrid approach: robust OCR for text extraction, a batch translation stage for efficiency, and UI-level controls to let users inspect and correct OCR outputs before translation. These studies demonstrate the value of preserving alignment metadata (bounding boxes, token indices) so the translation stage and UI overlays can maintain correct positional correspondence with the source image.

2.5 Lightweight, multilingual OCR tools

For applied systems that must handle Korean and other languages, the Pororo ecosystem (and community ports) provide practical OCR models [5] that are easy to integrate into Python pipelines. Several community repositories and packaged versions (e.g., pororo-ocr on PyPI or Hugging Face ports) demonstrate that Pororo-based models can be used as a drop-in OCR engine for Korean and English text extraction, including support for bounding box output and reasonably fast inference on GPU-enabled environments. These tools are useful when building prototypes that require reliable extraction for specific languages without the overhead of retraining base OCR models.

2.6 Error propagation, UI-centred correction, and interactive pipelines

A consistent theme across the literature is that purely automatic cascaded pipelines are fragile in the face of OCR errors, ambiguous contexts, or idiomatic expressions. Several studies and applied projects therefore incorporate human-in-the-loop [7] mechanisms: preview of raw OCR output, editable translations, and "refresh" features to request alternative phrasings for single segments. This design reduces the

need for full reprocessing, improves perceived latency, and increases final accuracy by letting users correct OCR mistakes before MT. PolyglotAI's design choices (per-segment refresh, preview of raw text alongside translations, and a chat assistant) follow directly from this body of work and the documented benefits of interactive correction.

2.7 Generative models (Gemini) as translation and reasoning engines

The newest generation of large generative models—exemplified by Google's Gemini family—go beyond classical neural MT by offering broad reasoning, instruction-following, and (in some models) multimodal capabilities[6]. Gemini models support large context windows, structured output modes, and multimodal inputs, making them suitable for batch translation of multiple segments with contextual instructions, or for on-demand refinement queries (e.g., "rephrase this segment for formal tone"). Developers commonly call Gemini via generateContent-style endpoints and include carefully crafted prompts that provide the model with the source text, translation instructions, and/or sample context. However, usage considerations include rate limits, prompt engineering to reduce hallucination, and robust parsing of the model's structured or semi-structured responses. For prototype systems, Gemini serves well as a flexible translation + reasoning backend, provided that engineering controls (batching, validation, fallback logic) are in place.

Chapter 3

System Flow and UX Design

This chapter details the programming logic behind the Word and Excel translation flows, the dynamic previews, and the integrated chat assistant. It explains how data moves through the system, the structure of the backend and frontend operations, and how interactive features are implemented.

3.1 Excel Translation Flow

The Excel flow is designed to handle multi-cell structured data while preserving original layout:

1. **File Upload and Parsing:**
    - The backend receives the uploaded Excel file via Flask.
    - `pandas` reads the spreadsheet into a DataFrame.
    - All cell values are converted to strings to standardize processing and avoid issues with formulas or numeric types.
2. **Translation Data Structure:**
    - A parallel dictionary structure stores translated content: {`column: [translated_row_values]`}
    - This allows per-cell updates and refresh without overwriting original content.
3. **Preview Generation:**
    - Jinja2 templates dynamically render a table with original values and translation cells.
    - Each cell includes:
        - Editable translation field
        - A refresh button to request alternative translations
        - Tooltip icons displaying context notes
    - JavaScript maintains a global state mapping the DataFrame and translations for frontend updates.

4. **Per-Cell Refresh Logic:**
    - When a user clicks the refresh button, JavaScript sends an AJAX POST request to `/update_translation/<id>` with:
        - Cell row and column
        - Original text
        - Target language
    - The backend calls the Gemini API with this information, updates the stored translation, and returns the new value for in-place UI update.

3.2 Word Document Translation Flow

Word documents require a sequential paragraph-based approach:

1. **Paragraph Extraction:**
    - The backend uses `python-docx` to read the document.
    - Empty paragraphs are filtered out, and a list of text segments is created.
2. **Translation Mapping:**
    - Translations are stored in a list matching paragraph indices.
    - This ensures alignment during export and preview.
3. **Dynamic Preview:**
    - Jinja2 renders each paragraph along with:
        - Editable translation box
        - Refresh button
        - Tooltip for context notes
    - JavaScript manages the frontend state to allow seamless updates without full page reloads.
4. **Interactive Translation Refresh:**
    - Similar to Excel, clicking refresh triggers a backend call for a single paragraph.
    - Gemini API receives the original text and optional translation history to maintain coherence.
    - Updated translation and notes are returned and applied directly in the browser.

3.3 Chat Assistant Logic

The chat assistant allows users to query the document or image content interactively:

1. **Frontend Interaction:**
   - A chat widget captures user input.
   - JavaScript appends the query to the chat history and sends it via AJAX to the backend endpoint `/chat/<id>`.
2. **Backend Handling:**
   - The backend retrieves the stored document/image content for context.
   - It constructs a prompt combining:
     - Original text (Excel cells or Word paragraphs)
     - Existing translations
     - User query
   - This prompt is sent to the Gemini API for reasoning and response generation.
3. **Response Rendering:**
   - Gemini returns a text response, optionally including suggested translations or notes.
   - The backend appends the response to the chat history.
   - The frontend updates the chat window dynamically without reloading the page.
4. **State Management:**
   - Both frontend and backend maintain a consistent chat history.
   - Each response can optionally trigger partial updates to the previewed translation tables or paragraphs.

3.4 Data Flow Overview

1. **Upload:** User uploads Excel, Word, or image file.
2. **Parsing and Storage:** Backend parses the content and initializes translation structures in memory.
3. **Preview Rendering:** Frontend displays editable previews with per-segment controls.
4. **Batch Translation:** Initial translation is performed for all segments via Gemini API.
5. **Interactive Updates:** Per-cell/paragraph refresh or chat queries trigger isolated API calls for partial updates.
6. **Export:** Users can export the fully translated document while preserving original structure.

3.5 Programming Considerations

- Asynchronous requests prevent UI blocking during translation.
- Global frontend state ensures efficient mapping of originals to translations and notes.
- Error handling is implemented to manage API latency, malformed responses, and partial failures.
- Modular design allows adding support for new document types or languages with minimal changes.

Chapter 4

System Architecture

4.1 Overview

The PolyglotAI system has a modular architecture designed to handle multiple file types and provide interactive translation capabilities. The system separates concerns into three main layers:

- **Frontend (Client):**
  - Built with HTML, CSS, and JavaScript (Jinja2 templates).
  - Provides drag-and-drop and click-to-upload interfaces for Excel, Word, Image, and TXT files.
  - Displays previews of raw content and corresponding translations in an aligned, intuitive layout.
  - Supports per-segment translation refresh and contextual tooltips via Gemini API.
- **Backend (Server):**
  - Built on Python 3.12 with Flask (or FastAPI).
  - Handles file parsing, translation, chat requests, and session-based state management.
  - Maintains a session dictionary (`STORE`) mapping file IDs to extracted content, translations, and chat history.
  - Integrates with Gemini API for translation, rephrasing, and contextual notes.
  - Supports batch API calls to reduce network overhead and latency.
- **Data Flow and State Management:**
  - Each uploaded file is assigned a unique session ID.
  - Extracted text segments are stored along with translations in structured arrays or dictionaries.
  - Frontend maintains a `window.appData` object to track the current file's content, translations, and translation direction.
  - Batch calls to Gemini API minimize latency, with per-segment refresh endpoints allowing selective updates without reprocessing the entire file.

4.2 Functional Flow

4.2.1 Upload and Parsing

- Users upload Excel, Word, TXT, or Image files.
- Backend extracts content:
  - Excel: cell values per column and row
  - Word: paragraphs
  - TXT: lines of text
  - Image: text extracted via Gemini API request (base64 input)

4.2.2 Preview

- Frontend shows raw content aligned with the source structure.
- For images, the uploaded image is displayed alongside extracted text and translations.

4.2.3 Translation

- Batch API calls to Gemini API for all segments.
- Returns translated text and optional contextual notes.
- Translations are stored in `STORE` and updated in the frontend preview.

4.2.4 Chat Assistant

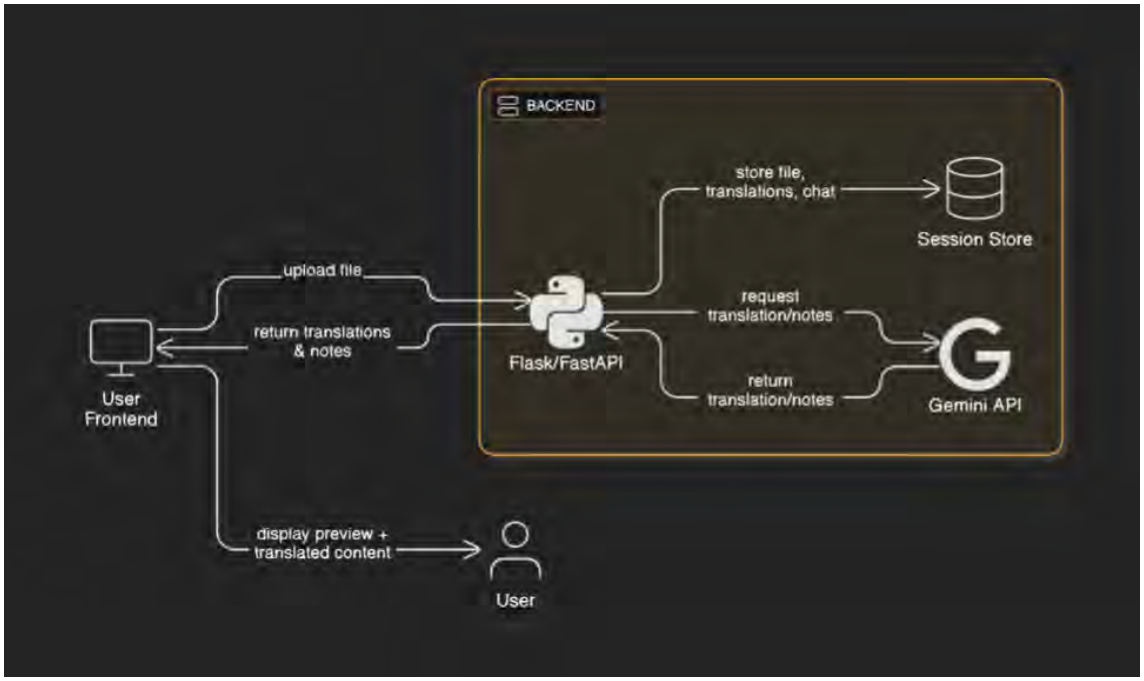
- Users can ask the chat assistant per file or per segment.
- Chat queries use Gemini API with structured prompts.
- Responses update the UI alongside original translations for reference.

4.2.5 Export and Copy

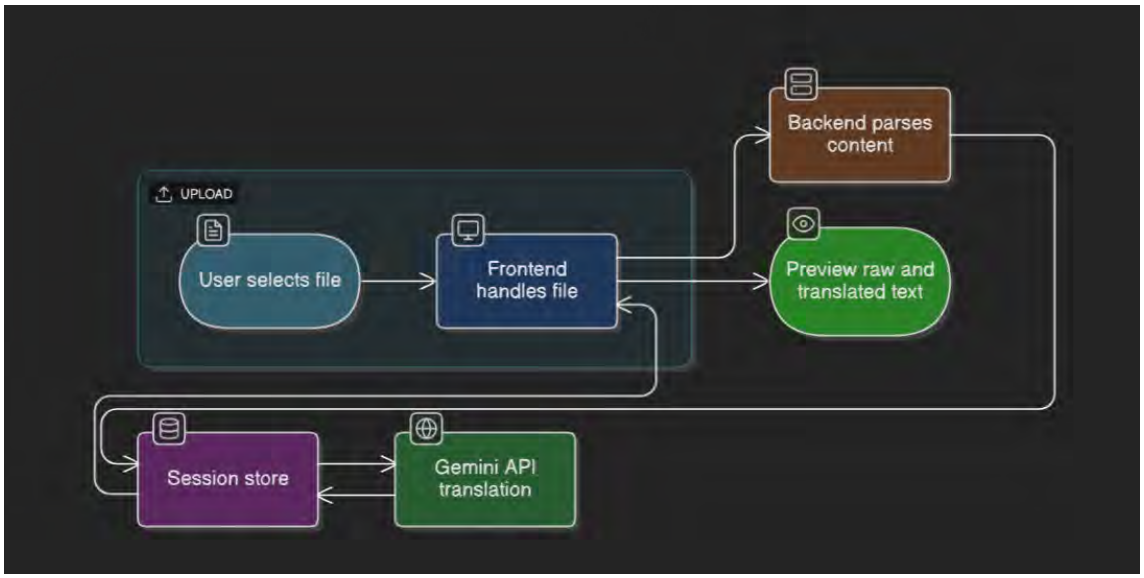
- Users can copy translations or export files in:
  - Excel (`.xlsx`)
  - Word (`.docx`)
  - TXT (`.txt`)
- Images export translations as plain text aligned to detected lines.

4.3 Diagrams

4.3.1 High-Level Architecture



4.3.2 Translation Flow



Chapter 5

Features Implemented

The system implements a variety of features to support interactive, multilingual translation for multiple file types, including Excel, Word, TXT, and images. The key features include:

5.1 Web Application Architecture

- Built using Python 3.12 with Flask (or FastAPI) backend.
- Jinja2 templates are used for dynamic HTML generation.
- Static assets (CSS, JavaScript) are served to support interactive previews and state management.
- Supports session-based state management via a unique file ID per upload, tracked in the backend `STORE`.

5.2 User Interface: Drag-and-Drop & File Upload

- Intuitive drag-and-drop interface and click-to-upload option.
- Spinner overlay indicates processing progress.
- Supports multiple file types: Excel (`.xlsx`), Word (`.docx`), TXT (`.txt`), and image files (`.png`, `.jpg`, `.jpeg`).

5.3 File Parsing and Preview

- **Excel:** Extracts cell values per row and column, stores in arrays for translation.
- **Word:** Extracts paragraphs into arrays for translation.
- **TXT:** Reads text line-by-line for translation.
- **Images:** Sends base64-encoded image to Gemini API to extract text segments.
- Previews display raw content aligned with the source structure, allowing users to easily relate original text to translations.

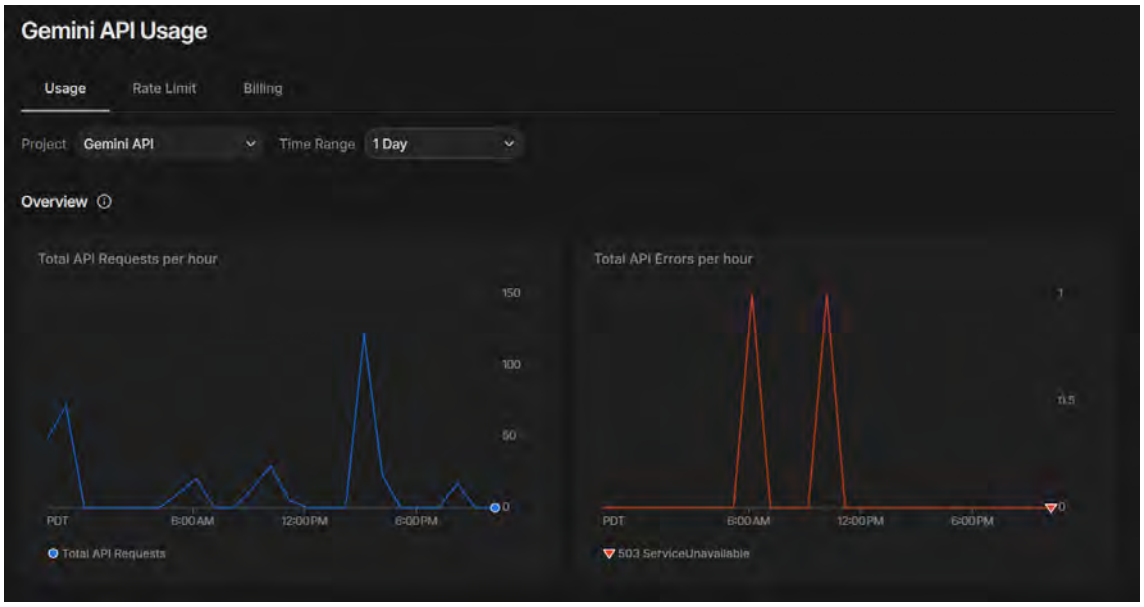


Figure 5.1: The spreadsheet after translation

5.4 Batch Translation with Gemini API

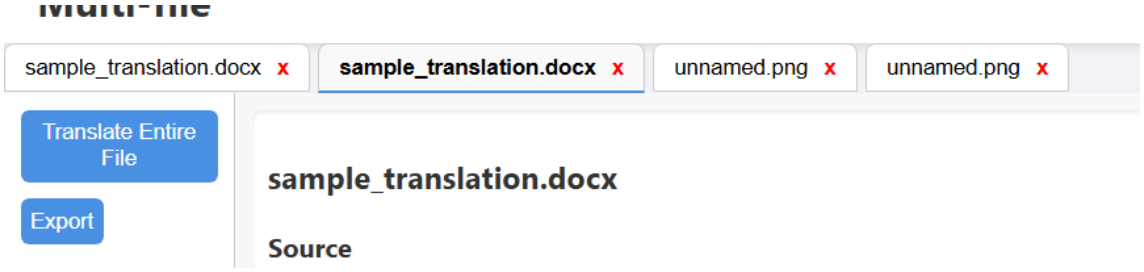
- Text segments from all file types are sent in a single batch call to Gemini API.
- Gemini API provides context-aware translations and optional notes for each segment.
- Responses are structured to allow mapping of translations back to the source content.

5.5 Interactive Preview

- Each text segment displays both original content and translated text side-by-side.
- Users can copy translations individually or export the entire file.

5.6 State Management

- Backend stores all file-specific data in `STORE` with a unique file ID.
- Frontend maintains a `window.appData` object for each open file:
  - Stores raw content, translation segments, and translation direction.
  - Tracks user interactions for per-segment refreshes.

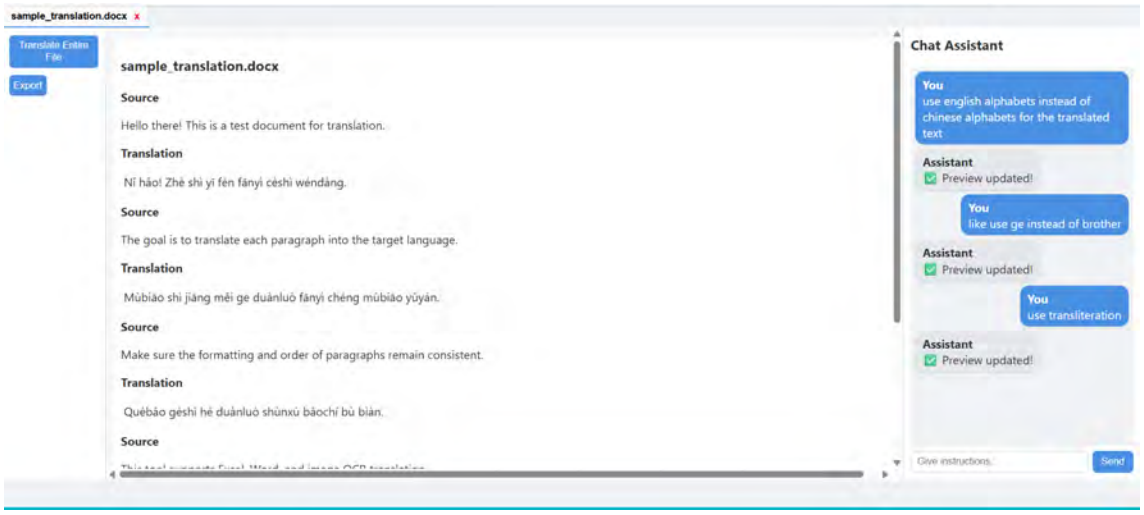


5.7 Export and Copy options

- Users can export translated files in multiple formats:
  - Excel (`.xlsx`) with translated cells.
  - Word (`.docx`) with translated paragraphs.
  - TXT (`.txt`) with translated lines.
  - Images export translations as plain text aligned with detected text lines.
- Copy buttons allow easy extraction of translations for use elsewhere.

5.8 Chat Assistant

- File-level chat assistant allows querying about translations, context, or language nuances.
- Chat messages are handled via Gemini API, leveraging structured prompts to maintain relevance.
- Responses appear in the preview alongside the source content and translations.



5.9 Optimized API Usage

- Batch translation reduces the number of API calls and minimizes latency.
- Individual segment refreshes only re-send the specific text, keeping the workflow efficient.

5.10 Supported Languages

- Initial support includes English, Spanish, French, German, Hindi, Bengali, Chinese (Simplified), Japanese, Arabic, and Portuguese.
- Translation direction can be selected per file to accommodate different workflows.

Chapter 6

Challenges faced

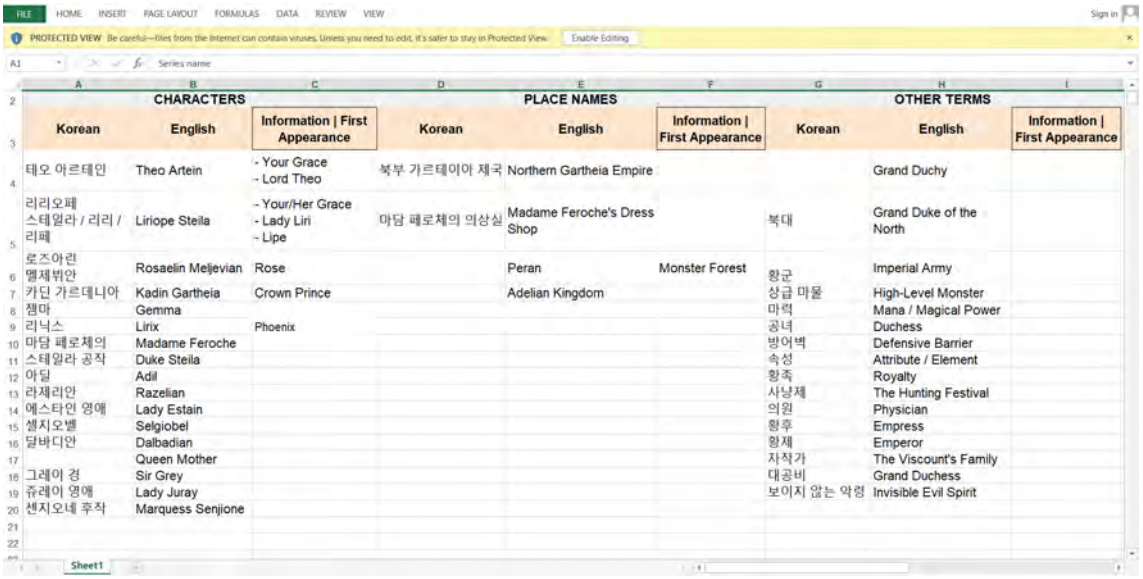
Building the Excel and Word translation features, along with interactive per-segment refresh and chat functionality, introduced several challenges:

6.1 Structured Document Handling

- **Excel Cell Mapping:** Preserving the correspondence between original and translated cells proved non-trivial, especially with merged cells or formulas. We had to convert all cell content to strings and maintain a parallel translation structure to allow per-cell updates without overwriting the original layout.
- **Word Paragraph Indexing:** Documents often contained empty or non-text paragraphs, which required filtering and careful indexing to ensure that translated paragraphs aligned correctly when exported back to Word.
- **Batch Translation Constraints:** Large spreadsheets could contain hundreds of cells. Sending all text in a single API request risked exceeding payload limits and slowing response times, necessitating chunked or batched API calls.

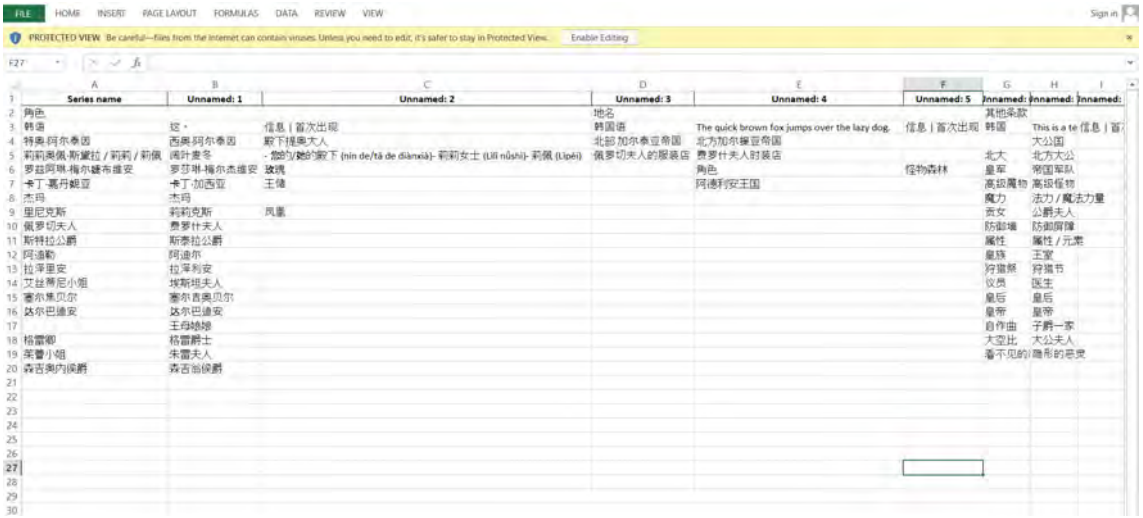
6.2 Interactive Translation Refresh and Chat

- **Frontend State Management:** Implementing per-cell or per-paragraph refresh required maintaining global state on the frontend, mapping originals to translations, and updating only the relevant segments in-place. Ensuring that multiple simultaneous updates did not interfere with each other was a key challenge.
- **API Latency:** Real-time interaction with the translation API introduced noticeable latency for large documents. Careful batching and asynchronous request handling were needed to prevent UI blocking.
- **Consistency Across Updates:** When a user requested a refreshed translation for a segment, it was critical to maintain contextual consistency with previously translated content. This involved passing the original text and translation history to the API in a structured manner.



CHARACTERS			PLACE NAMES			OTHER TERMS		
Korean	English	Information | First Appearance	Korean	English	Information | First Appearance	Korean	English	Information | First Appearance
테오 아르테인	Theo Artein	- Your Grace - Lord Theo	북부 가르데이아 제국	Northern Gartheia Empire			Grand Duchy	
리리오페 스테일라 / 리리 / 리페	Liriope Steila	- Your/Her Grace - Lady Liri - Lipe	마담 페로체의 의상실	Madame Feroche's Dress Shop		북대	Grand Duke of the North	
로즈아린 멜제뷔안	Rosaelin Meljevian	Rose		Peran	Monster Forest	황군	Imperial Army	
카딘 가르데니아	Kadin Gartheia	Crown Prince		Adelian Kingdom		상급 마물	High-Level Monster	
젬마	Gemma					마력	Mana / Magical Power	
리닉스	Lirix	Phoenix				공녀	Duchess	
마담 페로체의	Madame Feroche					방어벽	Defensive Barrier	
스테일라 공작	Duke Steila					속성	Attribute / Element	
아딜	Adil					황족	Royalty	
라제리안	Razelian					사냥제	The Hunting Festival	
에스타인 영애	Lady Estain					의원	Physician	
셀지오벨	Selgiobel					황후	Empress	
달바디안	Dalbadian					황제	Emperor	
	Queen Mother					자작가	The Viscount's Family	
그레이 경	Sir Grey					대공비	Grand Duchess	
쥬레이 영애	Lady Juray					보이지 않는 악령	Invisible Evil Spirit	
센지오네 후작	Marquess Senjione							

Figure 6.1: The format from the original is largely maintained after translation (see figure 6.2)



Series name	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8
角色			地名			其他条款		
韩语	这 -	信息 | 首次出现	韩国语	The quick brown fox jumps over the lazy dog.	信息 | 首次出现	韩国	This is a te	信息 | 首
特奥·阿尔泰因	西奥·阿尔泰因	殿下提奥大人	北部加尔泰亚帝国	北方加尔提亚帝国			大公国	
莉莉奥佩·斯蒙拉 / 莉莉 / 莉佩	阎叶皇冬	- 您的/她的殿下 (nín de/tā de diànxià)- 莉莉女士 (Lìlì nǚshì)- 莉佩 (Lìpèi)	佩罗切夫人的服装店	费罗什夫人的服装店		北大	北方大公	
罗兹阿琅·梅尔德布维安	罗兹琳·梅尔杰维安	玫瑰		角色	怪物森林	皇军	帝国军队	
卡丁·慕丹顿亚	卡丁·加西亚	王储		阿德利安王国		高级魔物	高级怪物	
杰玛	杰玛					魔力	法力 / 魔法力量	
里尼克斯	莉莉克斯	凤凰				贵女	公爵夫人	
佩罗切夫人	费罗什夫人					防御墙	防御屏障	
斯特拉公爵	斯泰拉公爵					属性	属性 / 元素	
阿迪勒	阿迪尔					皇族	王室	
拉泽里安	拉泽利安					狩猎祭	狩猎节	
艾丝蒂尼小姐	埃斯坦夫人					议员	医生	
塞尔集贝尔	塞尔吉奥贝尔					皇后	皇后	
达尔巴迪安	达尔巴迪安					皇帝	皇帝	
	王母娘娘					自作曲	子爵一家	
格雷卿	格雷爵士					大空比	大公夫人	
朱雷小姐	朱雷夫人					看不见的	隐形的恶灵	
森吉奥内侯爵	森吉翁侯爵							

Figure 6.2: The spreadsheet after translation

Figure 6.3: Code snippet of the API call

6.3 Gemini API Integration

The Gemini API is a generative AI model that performs **text generation, translation, and reasoning tasks**. In PolyglotAI, it is used to:

- Translate text from one language to another
- Provide alternative phrasings on demand
- Supply contextual notes, including cultural, idiomatic, or semantic explanations

6.3.1 Technical Workflow with Gemini API

1. The backend constructs a structured prompt containing the text to translate and the target language.
2. All text segments from an image, Excel sheet, or Word document are batched to minimize API calls.
3. The prompt is sent via a POST request with the API key and content body. Gemini returns a structured JSON response containing the translated text.
4. For interactive refresh, the API is called with a single segment and optional context to generate an alternative translation.

6.3.2 Bottlenecks Observed

- **Rate Limits and Quotas:** Large batches could hit API request limits, requiring intelligent throttling.
- **Response Parsing:** The API returns structured JSON, but malformed responses occasionally occurred due to unexpected characters or complex text formatting, requiring robust error handling and fallback mechanisms.
- **Latency:** The translation process, especially for long documents or multiple languages, could take several seconds per batch, making asynchronous handling and user feedback (spinner overlays) essential for usability.

6.4 Summary of Development Bottlenecks

The main challenges in implementing the new features were:

- Ensuring data integrity and alignment between original and translated content for Excel and Word files
- Implementing responsive per-segment translation refresh without blocking the UI
- Handling API latency, rate limits, and occasional malformed responses
- Maintaining contextual coherence across multiple translations

Despite these bottlenecks, the system successfully supports multimodal translation with per-segment interactivity, batch processing, and contextual notes, demonstrating the feasibility of an end-to-end generative AI translation framework.

Chapter 7

Conclusion

This project presents a unified, interactive platform for multilingual text translation across images, Word documents, and Excel spreadsheets. By combining OCR, document parsing, and Gemini's generative AI translation, the system enables users to see raw and translated text side by side, request alternative phrasings, and export content efficiently.
Key outcomes include:

- Significant reduction in processing time through batch API calls.
- Enhanced translation accuracy and context awareness via Gemini API.
- Improved usability with interactive previews, chat assistants, and export options.

While the current implementation demonstrates a functional MVP, it lays the foundation for future improvements such as multi-language expansion, advanced preprocessing for OCR, scalable backend architecture, and more sophisticated context-aware translation options. This system showcases a practical approach to integrating generative AI with document and image workflows, bridging gaps in existing translation tools while providing an interactive user experience.

Bibliography

[1] R. Smith, "An overview of the tesseract ocr engine," in *International Conference on Document Analysis and Recognition (ICDAR)*, 2007, pp. 629–633.

[2] A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[3] B. Shi, X. Bai, and C. Yao, "An overview of scene text detection and recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 11, pp. 2298–2320, 2018.

[4] A. Bamotra and P. Uppala, "Transdocs: Ocr with word-to-word translation," in *arXiv preprint arXiv:2304.07637*, 2023.

[5] X. Wang et al., "A survey of text detection and recognition algorithms for ocr," in *Pattern Recognition Journal*, vol. 56, 2023, pp. 163–177.

[6] G. DeepMind, "Gemini 1.5 technical report," Google Research, Tech. Rep., 2024. [Online]. Available: https://storage.googleapis.com/deepmind-media/gemini/gemini_1_5_report.pdf.

[7] W. Hu, Z. Li, and L. Zhang, "Advances in text-image machine translation: A multimodal survey," *Journal of Machine Learning Research*, vol. 25, pp. 1–43, 2024.