– Project Description –

We propose to design Translite, a lightweight, modular, multilingual translation program intended for small-scale applications with low resource availability like IoT systems since that is where our world is heading. The system begins with a language classifier which will use either an RNN derivative (GRU or LSTM) or an SVM to identify the source language. Then it dynamically loads a corresponding translation model into RAM (rather than vRAM), reducing memory requirements and enhancing portability.

The structured translation process is:
User Input > Language Classification > Translation Model Selection > Translation to Language Output.

The modular design supports user-specific language selection and enables model customization using a training script. The project uses publicly available translation datasets from Kaggle (e.g., English-French dataset).

– Related Work –

Yes, similar systems exist:

- Khan, A., & Sarfaraz, A. (2019). RNN-LSTM-GRU based language transformation. Soft Computing, 23(24), 13007-13024.
    - This paper explores the efficacy of using RNN derivatives (LSTM/GRU) to provide machine translations, particularly Urdu translations. This concept is the core of our pipeline and their results and findings should prove useful to us. We are beginning exploration into this and this paper has proven crucial to helping us understand the inner workings of machine translation through these model types.
- Can, E. F., Ezen-Can, A., & Can, F. (2018). Multilingual sentiment analysis: An RNN-based framework for limited data. *arXiv preprint arXiv:1806.04511*.
    - This paper explores multilingual sentiment analysis. We hope to use their findings to train a classification model to detect languages, rather than emotions, using a similar premise. So far we have only explored using an SVM for classification but poor results have pushed us in this direction in hopes that we can experience similar accuracy.
- Singh, S. P., Kumar, A., Darbari, H., Singh, L., Rastogi, A., & Jain, S. (2017, July). Machine translation using deep learning: An overview. In *2017 international conference on computer, communications and electronics (comptelix)* (pp. 162-167). IEEE.
    - This is an overview of the fundamentals of using deep learning to translate human language. We hope to use their proposed Recursive Recurrent Neural Networks to power our translation models.
- Wang, J. H., Liu, T. W., Luo, X., & Wang, L. (2018, October). An LSTM approach to short text sentiment classification with word embeddings. In *Proceedings of the 30th conference on computational linguistics and speech processing (ROCLING 2018)* (pp. 214-223).
    - Their proposed method of using an LSTM provides an idea of how our pipeline will look. In particular their approach to using a Word2Vec Model to represent their multilingual embeddings will likely help us in planning our embedding process. Early testing of using Bag of Words and Sequential tokenization proved way too resource intensive so

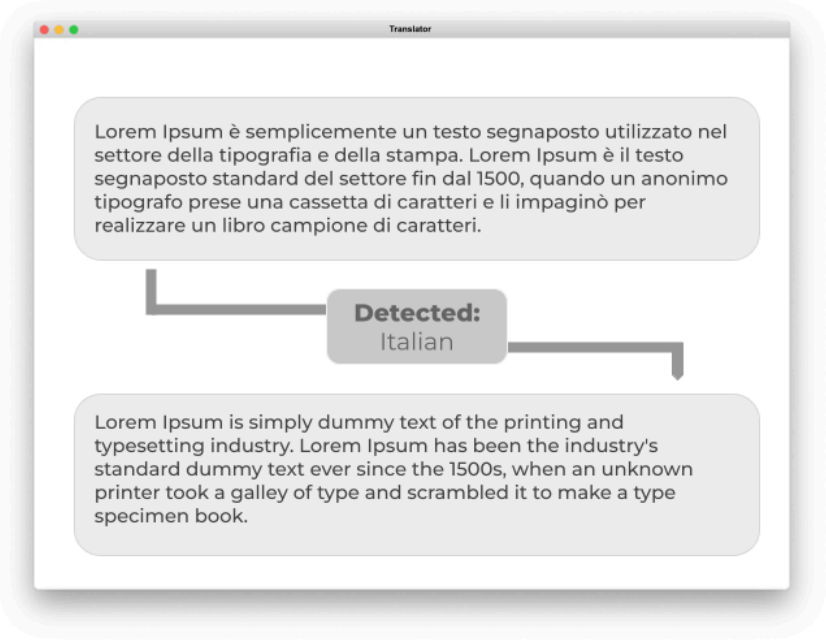
exploring training our own embedding model is the next step towards finding an efficient representation of our data, at least for the actual translation step.

- Ambikairajah, E., Li, H., Wang, L., Yin, B., & Sethu, V. (2011). Language identification: A tutorial. *IEEE Circuits and Systems Magazine*, *11*(2), 82-108.
    - While being a much older paper, this explored using an SVM to perform language classification, which may prove to be a smarter approach given our goal of minimizing resource usage. Initially this paper was what we had hoped to use as a basis for our language classification pipeline, however, due to poor results we are likely only going to use the input representation that they suggested.
- https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
    - We came across this guide that takes a similar approach we were planning to take in regards to actual translation. Our plan is to utilize this tutorial along with this (https://pytorch.org/tutorials/beginner/basics/data_tutorial.html) for the translation portion of our pipeline as it outlines how we can convert all of our Kaggle data into a format for Pytorch, and use that data to provide accurate translation. The following papers associated with machine translation will then be used to help us tune our models to be optimal.

– Code Repository –

https://github.com/Aryan-Sin/NLP_project

– Front End –



Our plan for the Frontend is to host it as a web server rather than using Tkinter as we had initially planned due to the extra complexity we feared it might bring. Since our end goal is to provide an extremely simple framework that anyone can use, our frontend is going to follow a very similar format. The only controllable portion of the frontend will be the input field at the top. Users will be able to input

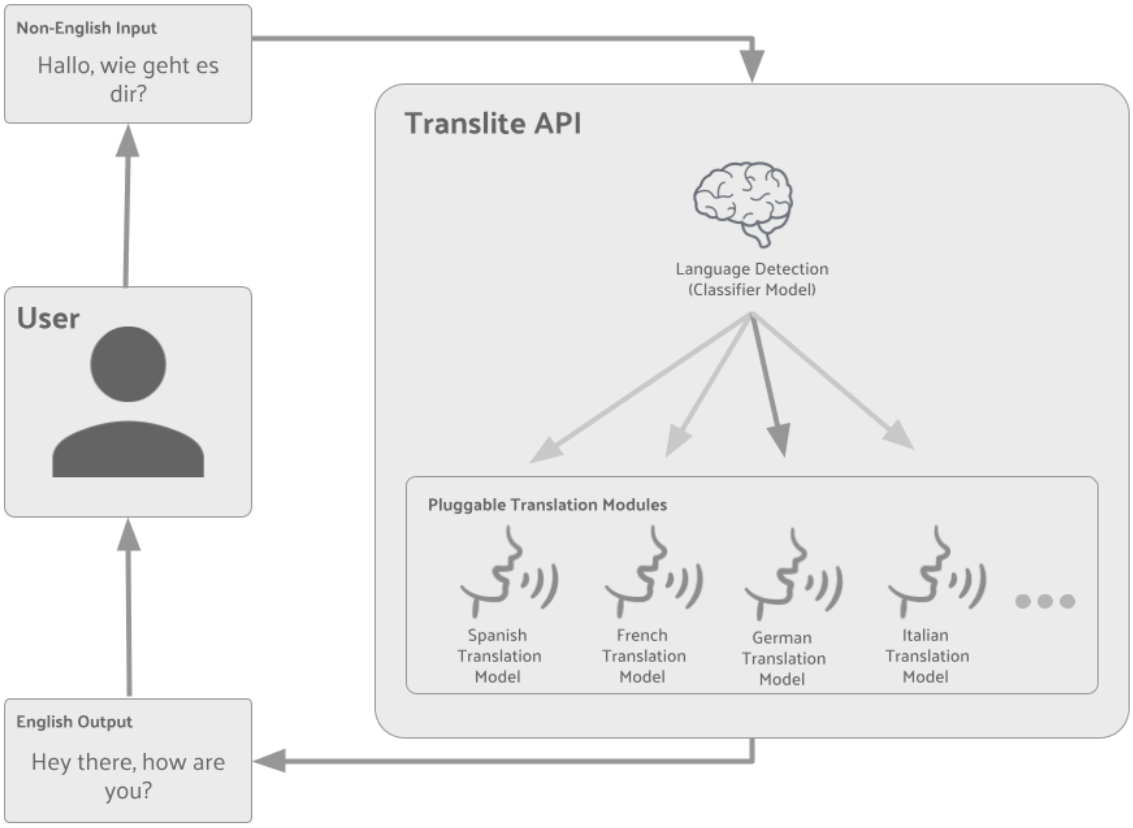
text from the language they wish to convert to English. Our hope is to then have automatic updates occur as new text is added to the field, similar to how Google Translate works. This means that our classification model will continuously check what language it thinks is coming in, and pivot towards the respective language model. On the frontend, the detected language box will change, and the actual translation will update in realtime. Since we haven't determined what latency looks like, specifically given our plan to adhere to low resource requirements, this is possibly subject to change.

– Back End Architecture –

Overall Architecture:

Our back end architecture can be structured as a four step process. User input, language detection, translation models, and finally the outputted translation. The user will enter text into the frontend interface, in any supported language, the user is not required to select a language prior to their input. For language detection, an RNN based classifier will be used to identify the input language. The classifier will be trained using a labeled dataset and multiple languages as examples. In order to align with our project goal of low resource usage, the model will be designed to be lightweight to avoid taking up to many resources. A language ID will also be assigned to any inputs to ensure that they receive the correct translation. Once the input language is properly detected, a specific translation model will be loaded into RAM. Each language has its own GRU/LSTM based translation model, which are all trained from their corresponding datasets. Finally, regarding the output, the selected model will use embeddings such as Word2Vec to translate the user input into the target language of the user. The translation is then returned to the front end, where the translated text is displayed with the detected language.

Architecture Diagram:



– Back End Details –

Components:

1. **Classifier** (GRU/LSTM or SVM) to detect language.
2. **Translation Models** trained on individual language datasets (GRU/LSTM).
3. **Training Script**: assigns language IDs, builds aggregate training set for classifier, and trains both classifier and individual translation models.
4. **Embedding Layer**: Uses Word2Vec or similar for multilingual embeddings.

Data:

- Kaggle translation datasets with English and non-English columns in CSV, JSON, Parquet, or XLSX formats
- Data is pre-existing and publicly available. It can be accessed with a free API key generated by signing up for kaggle.

- Example: English-French dataset

Prompts:

- The training is fully supervised so zero shot will not be used

Models:

- Not trained from scratch or fine tuned; pre-trained embeddings used, translation/classification models will be trained from scratch using the datasets.

– Team Members –

- **Matthew Hambrecht** – Colab Script Developer / Technical Writer
  - **Current:**
    - Dataset Retrieval and Combination Pipeline
    - Design and Testing of Classification Pipeline Techniques
    - Technical Writing on Proposal and Progress Papers
  - **Future:**
    - Technical Writing on Subsequent Assignments
    - Implementation of Classification Training Pipeline
    - Implementation of Classification Pipeline into Backend
- **Aryan Singh** – Application Developer / Technical Writer
- **Gabe Clark** – Colab Script Developer / Technical Writer
  - **Current:**
    - Technical Writing on Proposal and Progress Papers
  - **Future:**
    - Implementation of Classification Pipeline into Backend
    - Implementation of Classification Training Pipeline
- **Sai Parthasarathy** – Application Developer / Technical Writer

Completed:

- Project structure and methodology
- Dataset selection
- Initial script plans and pipeline layout

Remaining:

- Full implementation
- Integration testing

- Model training and evaluation
- UI development

– LLM Use Statement –

Some ChatGPT was used in order to structure the template.