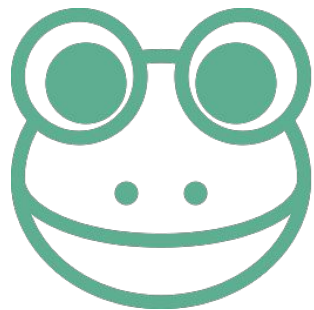


Voice-Activated Teaching Assistant in Persian and Hindi

Project Advisor: Dr Vahid Behzadan

Team Members:

- Bahareh Arghavani Nobar
- Devnath Reddy Motati



coqui TTS

- In an impressive endeavor to extend the capabilities of the Coqui TTS system to include Persian language support, Bahareh Arghavani Nobar embarked on a detailed and technical journey to modify the XTTS architecture and its tokenization process. The task began with Bahareh navigating to the development branch of the Coqui TTS GitHub repository at "<https://github.com/UNHSAILLab/TTS>" to clone the necessary codebase onto her local machine. This initial step was crucial for accessing the latest developments and tools within the Coqui TTS framework, setting the stage for the subsequent customization work.

- Following the repository setup, Bahareh's next steps were targeted towards preparing the system for Persian language support. She began by downloading essential resources to fine-tune the XTTS v2 model for Persian. This included acquiring a checkpoint file (dvae.pth) essential for model training and a tokenizer file (vocab.json) from Coqui's scarf gateway links provided. These files are pivotal in training the XTTS v2 model, with the checkpoint file containing pre-trained model parameters and the tokenizer file enabling the conversion of text into a format suitable for model processing.
- Despite Coqui TTS's support for 17 languages, Persian was not initially among them. To address this, Bahareh leveraged the flexibility of the system to incorporate Persian by modifying the vocab.json file. She introduced Byte Pair Encoding (BPE) for Persian, a technique that efficiently manages the language's unique alphabet and phonetics. In particular, Bahareh added missing Persian characters such as "پ", "گ", "ژ", and "چ" to the tokenizer's vocabulary, ensuring that the model could accurately represent Persian phonetics.
- To further customize the model for Persian, significant modifications were made to the tokenizer.py file located in /data/notebook_files/TTS/TTS/tts/layers/xtts/. Bahareh implemented custom tokenization rules for Persian, incorporating regex patterns to recognize and transform culturally significant phrases and abbreviations into their simplified forms. This customization process not only involved adding specific Persian phrases but also adapting the tokenizer to handle common symbols and figures of speech in Persian and Arabic, given their scriptural similarities. Examples of these adaptations include transforming common symbols and prepositions into their written Persian equivalents, enhancing the naturalness and accuracy of the generated speech.

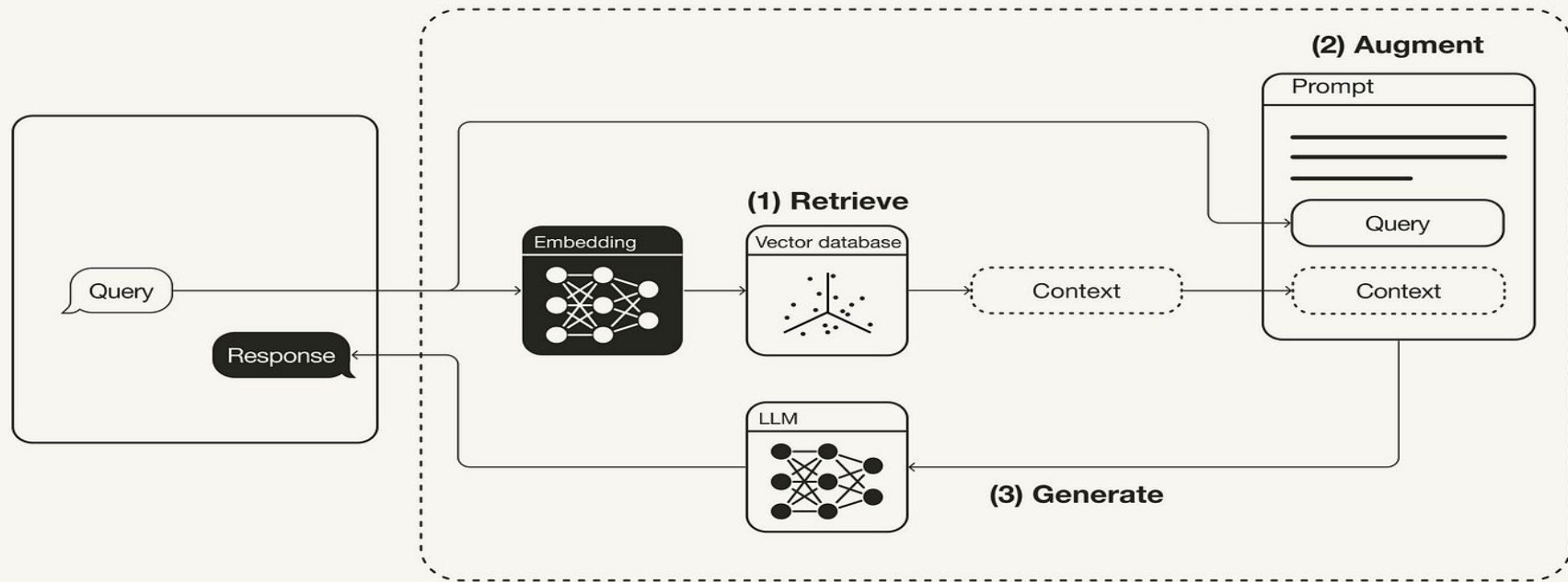
- Furthermore, Bahareh updated the XTTS configuration file (xtts_config.py) to officially include Persian in the list of supported languages, a pivotal step towards making Persian TTS a reality within the Coqui framework.
- The upcoming phase of Bahareh's project involves resuming the fine-tuning process on a curated audiobook dataset specifically prepared for this purpose. To ensure the project's progress is meticulously tracked and analyzed, Bahareh plans to utilize Weights & Biases (wandb), a platform that facilitates detailed monitoring of machine learning projects and synchronizes with TensorBoard for an integrated performance overview.
- "XTTS is trained with 16k hours of data mostly consisting of public datasets. We use all the datasets from the beginning and balance the data batches by language. In order to compute speaker latents, we used audio segments ranging from 3 to 6 seconds in length. faced problem "reference:
["https://medium.com/@erogol/xtts-v1-techincal-notes-eb83ff05bdc"](https://medium.com/@erogol/xtts-v1-techincal-notes-eb83ff05bdc)

```
        "lstrip": false,
        "rstrip": false,
        "normalized": false
    },
    "normalizer": null,
    "pre_tokenizer": {
        "type": "whitespace"
    },
    "post_processor": null,
    "decoder": null,
    "model": {
        "type": "BPE",
        "dropout": null,
        "unk_token": "[UNK]",
        "continuing_subword_prefix": null,
        "end_of_word_suffix": null,
        "fuse_unk": false,
        "vocab": {
            "[STOP]": 0,
            "[UNK]": 1,
            "[SPACE]": 2,
            "!": 3,
            "...": 4,
```

Problem on fine tune in created audiobook dataset

encountered an "AssertionError: ! len(DataLoader) returns 0. Make sure your dataset is not empty or len(dataset) > 0. " code : " "AssertionError" indicating that the DataLoader's length was zero, prompting a check to ensure the dataset was not empty and contained data. Upon further investigation, I discovered through a technical guide on XTTS fine-tuning (referenced from "<https://medium.com/@erogol/xtts-v1-technical-notes-eb83ff05bdc>") that the dataset intended for fine-tuning XTTS models should consist of audio segments ranging from 3 to 6 seconds. The challenge arose with my audiobook dataset, which was compiled using the YouTube library and Google Speech API. The segmentation of audio in this dataset was dictated by the ASR algorithm of the Google Speech API, which might not align with the recommended 3 to 6-second chunks for optimal XTTS training

Retrieval Augmented Generation(RAG)



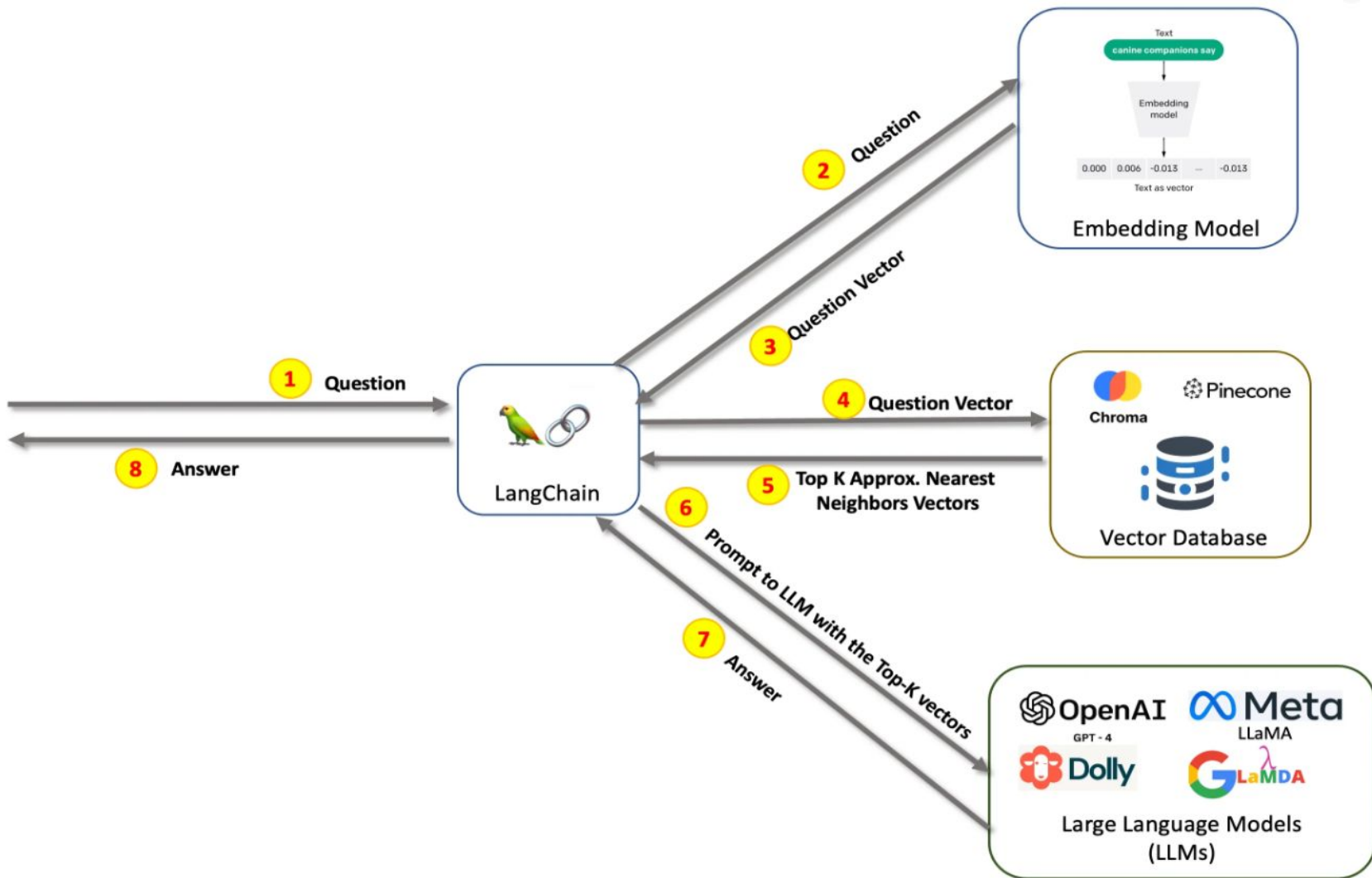


LangChain is a framework for developing applications powered by language models. It enables applications that:

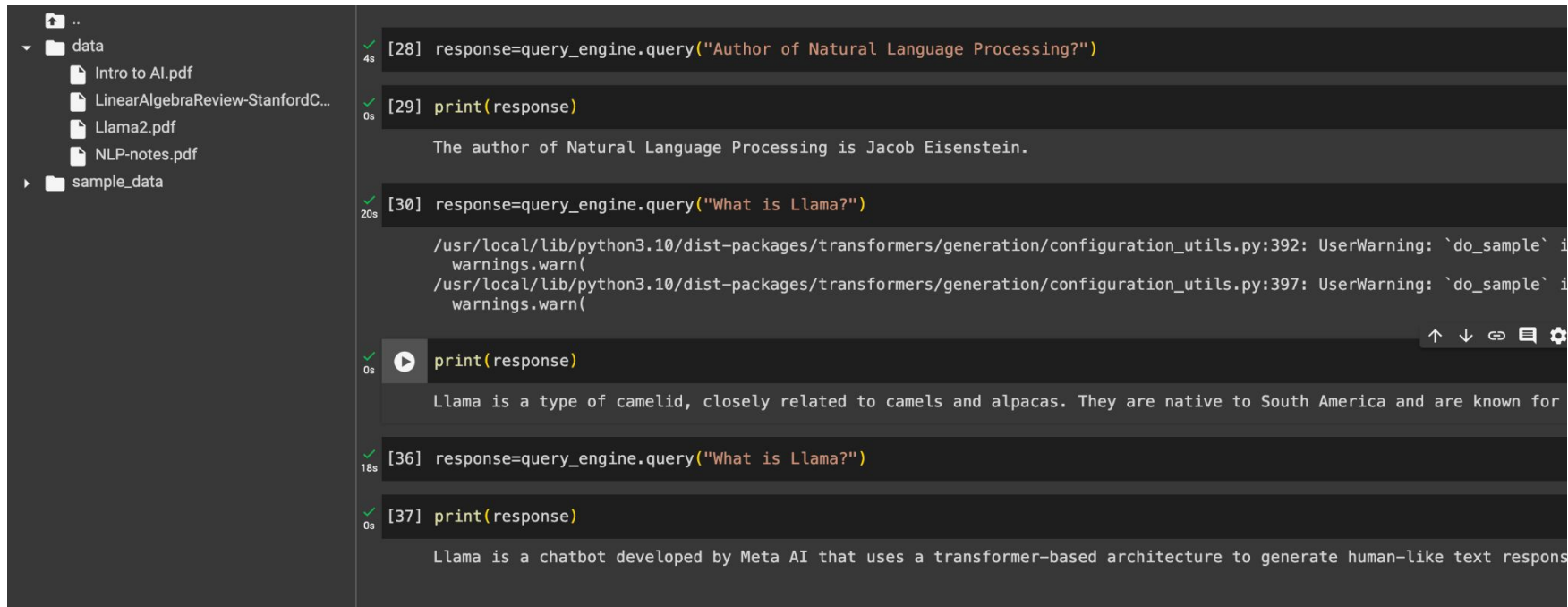
- Are context-aware: connect a language model to sources of context (prompt instructions, few shot examples, content to ground its response in, etc.)
- Reason: rely on a language model to reason (about how to answer based on provided context, what actions to take, etc.)

The framework is organized into six modules each module allows you to manage a different aspect of the interaction with the LLM. Let's see what the modules are.

- **Models:** Allows you to instantiate and use three different types of language-models, which are: Large Language Models (LLMs), Chat Models, Text Embedding Models.
- **Prompts:** The prompt is how we interact with the model to try to obtain an output from it. By now knowing how to write an effective prompt is of critical importance. This framework module allows us to better manage prompts. For example, by creating templates that we can reuse.
- **Indexes:** The best models are often those that are combined with some of your textual data, in order to add context or explain something to the model. This module helps us do just that.
- **Chains:** Many times to solve tasks a single API call to an LLM is not enough. This module allows other tools to be integrated. For example, one call can be a composed chain with the purpose of getting information from Wikipedia and then giving this information as input to the model. This module allows multiple tools to be concatenated in order to solve complex tasks.
- **Memory:** This module allows us to create a persisting state between calls of a model. Being able to use a model that remembers what has been said in the past will surely improve our application.
- **Agents:** An agent is an LLM that makes a decision, takes an action, makes an observation about what it has done, and continues in this manner until it can complete its task. This module provides a set of agents that can be used.



Llama 2's response before and after updating the document:



The screenshot shows a Google Colab notebook interface. On the left, a file explorer shows a folder named 'data' containing 'Intro to AI.pdf', 'LinearAlgebraReview-StanfordC...', 'Llama2.pdf', and 'NLP-notes.pdf', and a folder named 'sample_data'. The main area displays a series of code cells and their outputs:

- Cell [28]: `response=query_engine.query("Author of Natural Language Processing?")` (4s)
- Cell [29]: `print(response)` (0s). Output: "The author of Natural Language Processing is Jacob Eisenstein."
- Cell [30]: `response=query_engine.query("What is Llama?")` (20s). Output includes a warning: `/usr/local/lib/python3.10/dist-packages/transformers/generation/configuration_utils.py:392: UserWarning: `do_sample` is deprecated. Please use `temperature` instead. (https://huggingface.co/docs/transformers/main/en/main_classes/text_generation#transformers.generation.configuration_utils.GenerationConfig.do_sample)`
- Cell [31]: `print(response)` (0s). Output: "Llama is a type of camelid, closely related to camels and alpacas. They are native to South America and are known for"
- Cell [36]: `response=query_engine.query("What is Llama?")` (18s)
- Cell [37]: `print(response)` (0s). Output: "Llama is a chatbot developed by Meta AI that uses a transformer-based architecture to generate human-like text respons"

Google Colab link:

https://colab.research.google.com/drive/1qikFoNkFWSbO_4n7tIsCXgWsMo0SEKWm?usp=sharing

Previous week	Current week	Upcoming week
<ul style="list-style-type: none"> Data Collection and Preparation: An exhaustive approach to data gathering was outlined, targeting sources like Microsoft Azure, Kaggle, YouTube (via the "yt-dlp" library), and the Google Speech API for a diverse and high-quality dataset. A noteworthy challenge encountered was the Afghan accent in the Kaggle dataset, leading to the decision to compile a new dataset to ensure linguistic accuracy and consistency. Model Selection and Fine-Tuning: The decision to fine-tune the Couqi TTS model for enhanced Persian language support and accuracy reflects a targeted approach to model improvement. Furthermore, the project ambitiously plans to develop a pipeline integrating Automatic Speech Recognition (ASR), Retrieval-Augmented Generation (RAG) , and TTS for both Hindi and Persian, showcasing a commitment to multi-lingual support. Innovative Evaluation and Deployment: Emphasis on automating the evaluation system with an innovative Mean Opinion Score (MOS) metric underscores the project's focus on quality and user experience. Deployment strategies are carefully considered, ensuring the TTS system's applicability in real-world scenarios. Collaboration and Version Control: Effective communication within the team and with advisors through Notion, coupled with the establishment of a GitHub repository for efficient version control. Dataset Development and Challenges: The strategic use of various tools and APIs to assemble a 25-hour dataset with single-speaker audio, free of accent inconsistencies and enriched with emotional variance, demonstrates a meticulous approach to dataset quality. The decision against using the Common Voice dataset due to quality concerns further highlights the project's dedication to high standards. 	<ul style="list-style-type: none"> This week, Bahareh Arghavani Nobar made significant strides in adapting the XTTS architecture for Persian language support, showcasing a meticulous approach to enhancing the Coqui TTS system. By leveraging Byte Pair Encoding (BPE), Bahareh updated the vocab.json to incorporate unique Persian alphabets such as "پ", "ژ", "گ", and "چ", which were initially missing. This step was crucial for training the XTTS v2 model to accurately represent Persian phonetics and linguistics. Given the close similarity between the Persian and Arabic alphabets, a strategic decision was made to replace an existing supported language with Persian, further tailoring the system for accurate Persian language processing. Bahareh also innovatively modified the tokenizer.py file within the Coqui TTS architecture to include custom tokenization rules for Persian. This included regex patterns for culturally significant phrases and abbreviations, ensuring that the TTS output would reflect natural Persian speech nuances. Parallel to Bahareh's technical advancements, Devnath Reddy Motati focused on curating an NLP knowledge base, establishing a solid foundation for information retrieval. This comprehensive resource aims to enhance the project's linguistic depth, supporting the broader goal of creating a robust TTS system. Devnath also explored and implemented a basic RAG system using Llama2 from hugging face to understand the working and implementation technique. Also tried to understand the implementation of basic coqui TTS in python with simple examples. 	<ul style="list-style-type: none"> Moving forward, Bahareh plans to continue fine-tuning the model on a specially created audiobook dataset, with an eye on monitoring progress through Weights & Biases (wandb), which will also synchronize with TensorBoard. This dual tracking approach is intended to provide a granular view of the model's performance and improvements, ensuring that the Persian TTS system evolves to meet high standards of linguistic accuracy and user experience Moving forward Devnath plans to implement the RAG system using Azure Databricks and complete the Collection of NLP Subject Material for Knowledge Base.