Deploying Mistral 7B Instruct for Text Summarization and Evaluation

Introduction

Large language models have demonstrated great potential in various natural language processing applications. One of these applications that can be of great use to many people is the ability for these large language models to summarize paragraphs of text into shorter, more concise sentences while still conveying the same message. For example, news companies will sometimes provide an AI summary that will allow the reader to quickly understand what happened in the article while saving the reader time. Generative AI in general has seen massive improvements over the past 5 – 10 years with more advancements in graphics computing, and large corporations able to collect enough data to train and fine-tune these models over years.

This project aimed to develop and deploy a web application that leverages an open-source large language model to provide users with an interface to interact with an intelligent model that is capable of a specific natural language processing task, such as text summarization or idea generation. A large language model is a great choice for this application because LLMs can “understand” human written language and be trained to learn orders of words to produce coherent strings of text. Through their word embeddings, the transformer can understand the context of words and phrases (AWS, n.d.). A large language model may be an excellent way to summarize text, generate ideas, or even just get started on a writing project, but large language models are not perfect. Large language models may introduce bias into their responses if their training data is biased. Hallucinations are also a very real possibility; it is important for the user to review each response of the model to ensure that the response is accurate and not generating misleading data. On top of the web app allowing access to the LLM, the user is able to see their chat history with the model. The final application also exhibits a metrics tab that allows the user to see precalculated ROUGE and METEOR scores, as well as individual token usage and response metrics.

Model

For this project, Mistral 7B, specifically the instruct version of Mistral 7B v0.3 was the chosen model. The instruct version of the model is fine-tuned to follow more specific instructions, such as summarizing text. This model is very powerful, with 7.3 billion parameters, while also being efficient enough for the hardware in this project. Mistral 7B is also highly accessible with options to deploy it on any cloud infrastructure, download it and run it locally, or using Hugging Face's Transformers. There are also options to access the pre-trained model through a cloud service, such as the Replicate API or Hugging Face's Inference API. Mistral 7B is a relatively recently developed model that is more capable than larger models that have more parameters, is very cheap and easy to access through a cloud provider, and is performant enough to be used reliably for tasks such as text summarization and idea generation.

Architecturally, Mistral 7B enhances the Transformer decoder design by introducing several new features, including grouped-query attention, Sliding Window Attention, LayerNorm pre-normalization, and rotary positional embeddings. Grouped-query attention improves inference speed and efficiency by reducing the number of key-value heads relative to attention heads. The Sliding Window Attention is a major feature in the Mistral paper, it optimizes memory usage by restricting attention to a local window during inference. LayerNorm pre-normalization applies normalization before computation to improve stability. Finally, rotary positional embeddings help the model generalize to longer sequences. It is important to note that the instruct version of the model is the same architecturally, but the weights are fine-tuned to better handle instructions from user inputs (Jiang et al, 2023).

Project Definition and Use Case

The application concept of this project is a summarization tool that can also generate unique content. Mistral 7B is capable of answering user queries, summarizing text, or generating content based on user inputs. The goal is to use Mistral 7B to reword or summarize text based on context such as scientific or creative writing, as well as generate content for ideas for the user to get past mental blocks more quickly. The application should be simple, efficient, and fast to enable quick usability.

Mistral 7B will be integrated into the back end of the application to summarize and generate context-aware responses based on user inputs, aiming to be an aid to researchers, students, or writers. For example, a researcher may have two paragraphs and may wish to convey the same idea in a single paragraph, the researcher should be able to copy and paste the text into the input box and tell the model to summarize the same information in one paragraph and receive the generated output desired. The model is not limited to this and can handle a wide array of NLP tasks that many modern-day LLMs are capable of.

Implementation Plan

Python is the programming language used to develop the entirety of this project. In addition to Python, multiple libraries were implemented to assist in the development. Below is a list of the major libraries used to develop this project:

- SQLAlchemy
- Streamlit
- Hugging Face Hub
- Datasets

- Evaluate
- Pandas

The web framework chosen to host the project is Streamlit. The Streamlit website has a community where projects from other users can be seen and tested, as well as in-depth documentation into their API. The ease of use, aesthetic layout, and free tier are the reasons that Streamlit was chosen.

Below are the steps that were used to develop the project:

1. Gain access to Hugging Face Inference API via a read-only token.
2. Create a PostgreSQL database on AWS RDS free tier to store and retrieve data.
3. Create a Streamlit account and install the Python library.
4. Create a GitHub repository and code environment.
5. Draft test code to ensure a proper connection to the database, Inference API, and local testing of the Streamlit application.
6. Develop a project scope, features that are needed or wanted in the application.
7. Develop the needed features.
8. Develop the wanted features.
9. Deploy on Streamlit

Testing and deploying the application was the easiest part as Streamlit links to the GitHub repository and builds the Python environment needed to host the application through the requirements text file. Testing was done locally, rather than pushing to the GitHub repository for each change, the local Streamlit server detects changes and can be rerun automatically or with a single click, making testing efficient, meaning all development was done in Python files rather than Jupyter notebooks. The development of the project included research into how to access Mistral 7B through a cloud provider, as well as making sure the options are cheap and fast enough for regular use. API keys were loaded as environment variables to ensure that secret tokens were not leaked through the GitHub repository or the Streamlit application, Streamlit conveniently has a secret tokens section that loads them into the cloud environment automatically, meaning no changes to the code were necessary (Streamlit, n.d.).

Model Evaluation and Performance Metrics

The performance of the deployed application was evaluated using the inference time of each request, model accuracy using ROUGE and METEOR scores of reference material for summarizations, and user experience. The inference time was measured as the time taken between submitting a prompt through Hugging Face’s Inference API and receiving a response from the model.

Through the development of the application, inference time was tested multiple times, and it was found that the inference time of requests are sub-10 seconds, with most responses being generated and returned in under 5 seconds. Inference time is paramount to the user experience, with shorter inference time having a large positive effect on user satisfaction, long inference times would be unacceptable. Resource usage such as GPU usage and VRAM consumption, is not possible to obtain through Hugging Face’s Inference API. However, token usage is tracked allowing each user to see how many tokens they are using through the model, and allowing an administrator to query the total tokens used over time, allowing insights into cost, or even very rough estimates into resource usage if one can calculate how many tokens can be processed per unit of time on the specific hardware that the model is being ran on.

Summarization scores were calculated using ROUGE-1, ROUGE-2, ROUGE-L scores and a METEOR score. ROUGE is a set of metrics for evaluating automatic summarization and machine translation that measures the overlap between generated summaries and reference summaries based on n-gram, word sequences, and word pairs (Lin, 2004). METEOR utilizes features like synonym and paraphrase recognition to better reflect the semantic similarity between generated and reference summaries (Banerjee & Lavie, 2005). It also considers sentence-level matching, making it a valuable alternative to ROUGE for assessing text summarization performance. 100 articles were XSum dataset which include a one sentence summary of the article; each article was submitted to the Mistral-7B Instruct model and the response was saved into the database and compared to the reference summarization using Hugging Face's Evaluate library. The scores across the 100 samples are as follows:

- ROUGE-1: 0.23
- ROUGE-2: 0.05
- ROUGE-L: 0.16
- METEOR: 0.24

While the scores are lower than other fully fine-tuned summarization models, these scores are not a surprise for a zero-shot, instruction-based model such as Mistral 7B Instruct without fine-tuning to the specific dataset. Prompt refinement is included in the final version of the application, improvements were slight, and the model still had difficulty summarizing all the text in one sentence. Allowing the model to compare references that were 2, 3, or 4 sentences in length would likely improve the scores substantially.

The user experience overall is refined, smooth, and direct. The chat and history tabs are straightforward and simple, while the evaluation tab contains more in-depth metrics but still retains organization through containers, with the idea that beginners can understand what they are looking at without much thought while more advanced users can view more in-depth metrics if they wish. The user is aware of when the model is generating a response through the spinning wheel icon and typically receives a response within a few seconds.

Deployment Strategy

The application was deployed using Streamlit, which creates a Python environment directly from a requirements text file from a linked GitHub repository. The Streamlit server then runs the code from the GitHub repository, allowing a very simple deployment and update process. All remote connections such as the AWS RDS service and the Hugging Face Inference API can be called directly within the code, no changes needed when deploying. There is a wide array of options when looking into deployment, but the choice was made to include widely used, modern standards that provide cheap, simple, and reliable access. AWS provides a free tier of service, in which a PostgreSQL database on the Relational Database Service was deployed. Once the database was configured, the database URL, along with some additional parameters to access the database, were stored in a local environment file for testing, and included in the secrets section on Streamlit. To access the model through the Hugging Face Inference API, an API key is needed, which can be simply generated through their website. This API key was stored similarly in the environment file and in the secrets section on Streamlit, then any public model can be called with parameters through the Inference API in Python.

The goal during deployment was to create a user experience that is seamless, reliable, and efficient, while remaining cheap and simple on the development side. Local hosting of the model is possible but not recommended due to many factors, including resource usage on a local computer and opening ports to allow unknown network traffic to access local resources. Also, the local computer would need to be on 24/7 to ensure reliability. These choices made it clear that cloud options were superior in this use case: for a few cents or dollars, one can access many pre-trained models, compute resources, databases, and more. These choices proved to be a success as the application is reliably hosted, efficient, cheap, and fast.

UI Overview

The user interface of this application is designed to be simple and straightforward. A title is located on the top of the page, along with 3 tabs just below, "Chat with Mistral 7B Instruct," "History," and "Evaluation Metrics."

The first tab is simply an input box and a submit button to send the user prompt to the Mistral 7B model. The response is generated below the submit button, along with a dropdown box to show inference time and usage metrics if the user wants a more in-depth look into the metrics of the response, however these can also be found in the user's history tab later.

Each prompt and response are stored in the database and can be found in the user's history, under the history tab. If the user has not submitted any prompts to the model, the user will see an information label telling them to submit a prompt to show their history with the model. When there is data to be shown, the user will see a data frame with the timestamp, the prompt the user submitted, the response of the model, the latency, and the number of tokens used.

The final tab shows two different metrics: the first is the metrics on the reference text mentioned above, including the ROUGE scores, the METEOR score, as well as the entire data frame of the reference prompt, reference summarization, model response, and evaluation scores. Under the static evaluation metrics, the user's metrics are shown which include averages across their entire chat history, including the latency, token usages, and a chart the cumulative adds their token usage over each prompt and response.

Outcomes and Challenges

Outcomes:

- Fast, efficient, streamlined user interface
- Cheap and reliable cloud infrastructure
- Fast model responses
- User sessions and history
- User metrics

Challenges:

- Multiple choices of cloud providers, not sure which one is best suited
- Rate limits through Inference API, proved to not be much of an issue
- Cheap, but not many completely free options
    - Paid for pro tier of Hugging Face for one month
- Replicate API not suitable

Overall, the potential challenges got resolved through a bit of research about cloud provider options, and just making a choice proved to be sufficient. The largest challenge was trying to use the Replicate API for cloud compute resources. The Mistral 7B model was not able to be deployed to a user account, meaning, at best, a query would access the cold model on the website, which took a long time. This was unsuitable, and a change was necessary. Hugging Face was used as a backup and does host a warm version of the Mistral 7B model. Some minor challenges include figuring out how to configure the PostgreSQL database to allow remote connections and configuring a valid URL to connect to the database.

Resources Required

Below is the list of resources required to develop this application:

- Python - development
- Streamlit – application hosting and frontend
- AWS RDS PostgreSQL database – database hosting
- Hugging Face Hub/Inference - model hosting and API access
- GitHub – repository for Streamlit environment
- XSum dataset – reference dataset for calculating ROUGE and METEOR scores

Conclusion

This project successfully deployed an interactive summarization application accessing the Mistral 7B Instruct model through the Hugging Face Inference API. Utilizing Streamlit Cloud for the frontend deployment and an AWS PostgreSQL database for backend storage, this application provides users with an aesthetic interface for generating concise summaries, with more advanced features such as allowing users to view their history and their token usage, and memory of sessions using session IDs that are stored in the URL parameters, allowing users to bookmark their session and resume where they previously left off.

The summarization performance of Mistral 7B Instruct was assessed using ROUGE scores and a METEOR score on one sentence summarizations of BBC articles from the XSum dataset. While the model sometimes followed the prompt and returned a summary in one short and concise sentence, a lot of the time the model took multiple sentences to convey the summarization. This is one opportunity for improvement: comparing the model responses to multiple sentence summarizations, the model would likely perform far better in the metrics. Another area for improvement is user authentication: allowing the user to log in to view their history and metrics rather than relying on URL parameters.

References

Amazon Web Services. (n.d.). *What is LLM (Large Language Model)?* Retrieved April 8, 2025, from https://aws.amazon.com/what-is/large-language-model/

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. de las, Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., & Sayed, W. E. (2023, October 10). *Mistral 7B*. arXiv.org. https://doi.org/10.48550/arXiv.2310.06825

Streamlit. (n.d.). *Streamlit documentation*. Retrieved April 8, 2025, from https://docs.streamlit.io/

Lin, C. Y. (2004). *ROUGE: A package for automatic evaluation of summaries*. In *Text summarization branches out: Proceedings of the ACL-04 workshop* (pp. 74–81). Association for Computational Linguistics.

Banerjee, S., & Lavie, A. (2005). *METEOR: An automatic metric for MT evaluation with improved correlation with human judgments*. In *Proceedings of the ACL workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization* (pp. 65–72). Association for Computational Linguistics.