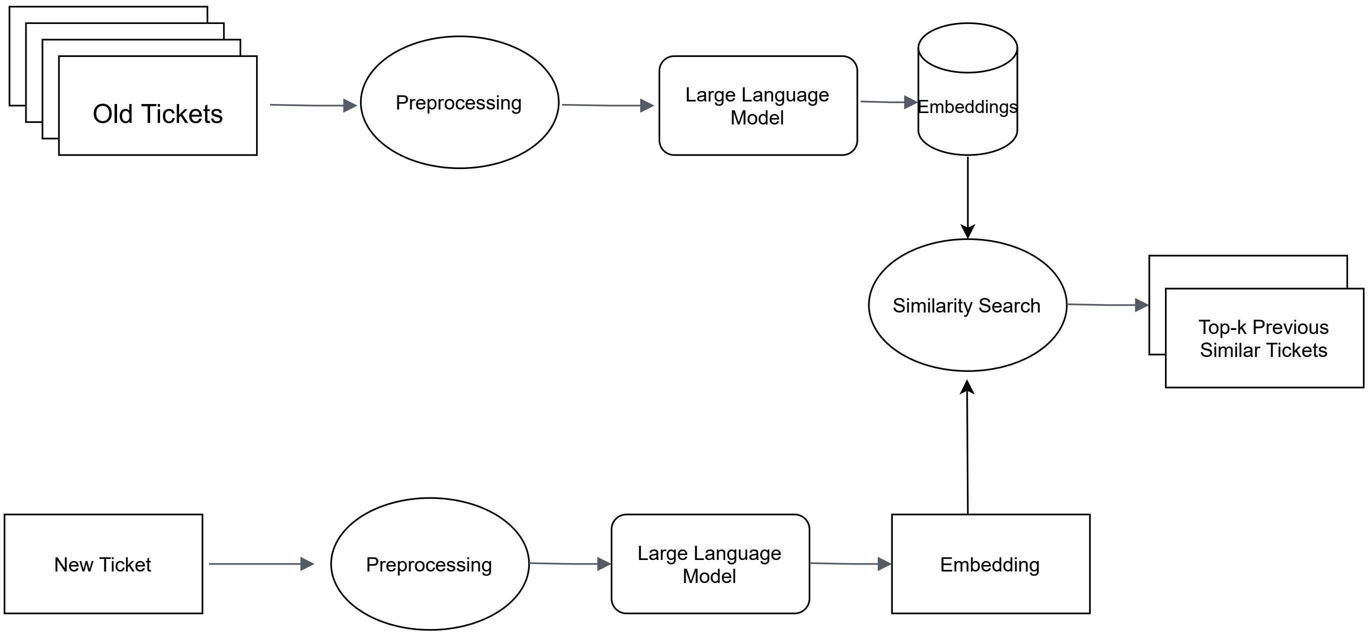
Case Study for AI Solutions Engineer

Problem

- **Goal**: Assist agents in finding a solution for incoming new tickets by utilizing information from previously resolved tickets.
- **Dataset**:
  - `old_tickets`: Multiple files which contain information about tickets that have been resolved before. The information includes fields such as:
    - `Ticket ID`: Unique identifier for the ticket
    - `Issue`: Title of the ticket
    - `Category`: Category of the ticket, e.g. Software, Hardware, Network, etc.
    - `Resolution`: Solution provided for the ticket
    - `Resolved`: Whether the ticket is resolved or not
    - `Description`: Description of the ticket
  - `new_tickets.csv`: Tickets that you can use to test and evaluate your solution.

Solution

I propose a Semantic Ticket Matching system that leverages: `Issue` and `Description` of the new ticket and the old tickets to find the most similar tickets. The system leverages a pre-trained LLM (`luminous-base`) from Aleph Alpha to encode the text data into a vector space and then uses a similarity metric to find the closest matches. The top matches can then be presented to the agent to assist them in finding a solution for the new ticket. A high level diagram of the solution can be seen below:



The developed solution is an intelligent search as it is much more than a simple text search. By analyzing the semantic meaning of ticket's `Issue` and `Description`, the system retrieves relevant historical tickets and provides valuable insights to aid in problem resolution.

I only chose the `Issue` and `Description` to embed because they are the most important fields that contain the information about the problem and the context of the ticket.

The solution can be broken down into the following components:

1. **Embedding Old Tickets Data**:
    - Load the `old_tickets` data.
    - Preprocess the data by selecting only `Issue` and `Description` information.
    - Encode the extracted information using the pre-trained LLM model to obtain embeddings.
    - Save the embeddings along with the old ticket data.
2. **Search**:
    - Encode the new ticket `Issue` and `Description` data using the same LLM model.
    - Calculate the similarity between the new ticket and the historical tickets using a similarity metric (e.g., cosine similarity).
    - Retrieve the top `k` most similar historical tickets.

You can find the code for the same in the solution.ipynb notebook. The notebook contains detailed explanations of the code and the steps involved in the solution. The structure of the files should be like this, in order to run the notebook out of the box without any changes.

```
├── README.md
├── aa-case-study-ai-solutions-engineer-main
│   ├── README.md
│   └── data
│       ├── new_tickets.csv
│       └── old_tickets
│           ├── ticket_dump_1.csv
│           ├── ticket_dump_2.xlsx
│           └── ticket_dump_3.json
├── images
│   ├── diagram.svg
│   └── spaces_screenshot.png
└── solution.ipynb
```

You can also change the path where the data is stored in the Notebook.

Note: The system returns both the `resolved` and `unresolved` tickets but it can be modified quite easily to return only the `resolved` tickets by filtering the `old_tickets` data based on the `Resolved` column.

Hugging Face Spaces demo

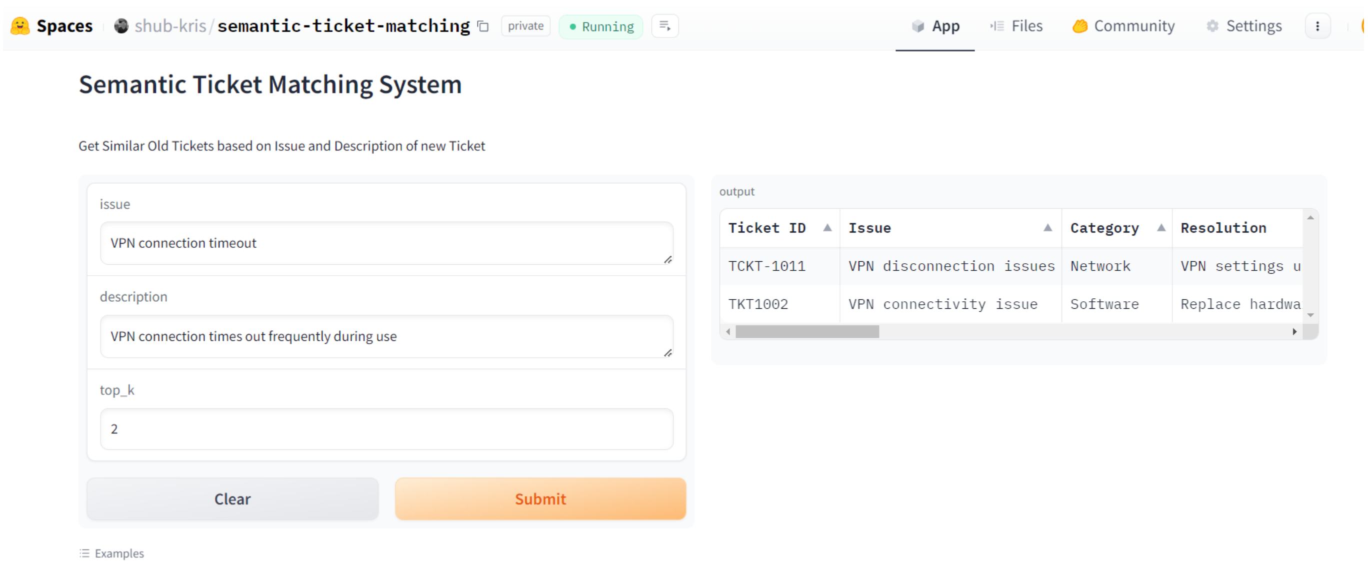
I have also deployed the solution on the Hugging Face Spaces platform. You can access the demo using the following link: Semantic Ticket Matching. All you need to provide is the following information:

- `issue`: Issue of the ticket
- `description`: Description of the ticket

- `top_k`: Number of top matches to retrieve

The system will then return the top `k` most similar tickets based on the provided information.

A screenshot of how the spaces looks is attached below:



Semantic Ticket Matching System

Get Similar Old Tickets based on Issue and Description of new Ticket

issue: VPN connection timeout

description: VPN connection times out frequently during use

top_k: 2

output

Ticket ID	Issue	Category	Resolution
TCKT-1011	VPN disconnection issues	Network	VPN settings u
TKT1002	VPN connectivity issue	Software	Replace hardwa

Note: As of now, the space is private and can't be accessed directly. Please let me know if you would like to access the space, and I can then make the space public or I can also demonstrate it during our discussion.

Observations and Improvements

- **Similarity Matching Suboptimal**: As we have only `30` samples in a `128` dimensional space, the similarity matching might not be optimal as the vector space is quite empty and hence the notion of similarity is distorted. The system can be improved by utilizing a larger dataset.
- **Smaller Language Model**: For calculating the embeddings I utilized the `luminous-base` LLM which contains `13B` parameters. It performs nicely but I expect smaller models like `distilbert-base-uncased` or `sentence-transformers` which contains some million parameters to also do a nice job. The advantage is the retrieval and embeddings can be faster compared to abigger LLM that contains billions parameters. So, I would have played with smaller models to see how they perform.
- **Embed Resolution**: The `Resolution` field can also be used to further refine the search as it also contains valuable information regarding a ticket. The embeddings of the `Resolution` field can be concatenated with the embeddings of the `Issue` and `Description` fields to provide a more comprehensive search mechanism.
- **Vector databases**: For storing the embeddings of the `old_tickets` data I utilised the `pandas DataFrame` but it won't scale as it is stored in memory. The system can be scaled by utilizing vector databases such as `Qdrant`, `Weaviate`, or others that can handle large-scale embeddings and provide efficient retrieval mechanisms. Using these vector databases, the similarity search will also be faster as their implementation is faster than the brute force search.
- **Reranker**: The system can be further improved by incorporating a reranker(Cross Encoder Models) that reorders the retrieved tickets based on the new ticket information. This can help in providing more relevant results to the agent.

- **Hybrid Search**: The solution can also be improved by leveraging additional search such as text search based on BM25 or ElasticSearch to provide a hybrid search mechanism. This can help in providing a more diverse set of results. A lot of vector databases also provide this kind of support out of the box.
- **Evaluation**: Qualitative Evaluation of the develop solution is a good first start. But, I would have also looked into doing a quantitative evaluation by calculating Information Retrieval metrics such as Mean Reciprocal Rank(MRR), Normalized Discounted Cumulative Gain (NDCG) and others to measure the performance of the system. I would have leveraged a LLM to build such a evaluation dataset.
- **Observability**: I would also have tracked the number of tokens being utilized in the LLM, the time taken for encoding the data, the time taken for retrieval, and other metrics to monitor the performance of the system. This can help in identifying bottlenecks and optimizing the system.
- **Retrieval Augmented Generation**: The system can be further improved by incorporating a retrieval augmented generation mechanism. The system can retrieve the top `k` most similar tickets and then generate a response based on the retrieved tickets. So, once we have top `k` tickets, an additional prompt together with the new ticket information can be fed to a LLM to generate a response. This directly generates a response that the agent can use to help the client. In this case, it is important to have a good generation model that can generate human-like responses.