**The Idea**

**Problem Statement**

Stack overflow provides one of the largest learning resources for programmers. Users post questions/doubts and his fellow peers try to provide solutions in the most helpful manner possible. The better an answer, the higher votes it gets, which also increase a user's reputation.

However, this huge amount of information makes it difficult to search for the solution you are looking for. It is not that big of an issue for Domain experts and other experienced professionals, because they are aware of the correct keywords required to get an appropriate answer. However, for a new programmer, this poses a great concern. For instance, if he needs to learn **‘how to make a server’** using Python, it is quite unlikely that he would use the terms **‘Django’** or **‘Flask’** in the search box. Thus, this might intimidate the user to use the platform.

**But why bother?**

A major source of income for Stack Overflow is through **Ad Revenue**. Therefore, their goal is to maximize readership in order to push more ads, and thus earn more money.

Due to poor performance of their search system, a user would have difficulty getting his doubts cleared through their website and would thus decide to use a more sophisticated search engine like Google for their purposes. The problem arises when Google suggests them a resource other than Stack overflow. **For every user that leaves their website in search of a better solution, they lose money they could potentially make.**

**Proposed Solution**

What we want is for the platform to actually understand the semantics of what the user is trying to search for, and then return the most helpful results for him. **Natural Language Processing (NLP)** has come a long way since its inception in the 20th century. We decided to use this subfield of Artificial Intelligence in order to solve our problem. NLP has proven to work very well in the past few years due to development of fast processors, GPUs and sophisticated model architectures.

***Note:*** *It wasn’t until a week before the submission date when I got know about the IBM Hack Challenge, thus I wasn’t able to experiment with more advanced NLP tasks such as Topic Modelling and Intent Detection for this task. I am confident that I could have built a better solution if provided a bit more time.*

In order to solve the semantic similarity task at hand, I needed to find a way to convert the text data in Stack overflow into a vector space so that I can use them as features for my predictive model. This is called **vectorizing.** Common approaches to vectorizing text data include Bag of Words approach and TF-IDF approach, however these are very sparse representations. In a vector of length 300, around 299 values would be 0. Thus, plugging it into our neural network would not work as the gradients would vanish. Therefore, for neural networks, we go for **Word Embeddings.** These are fixed length dense vector representations which work very well with neural network. I experimented with different options for pre-trained Word embeddings such as:

1. Google’s Universal Sentence Encoder
2. ELMO contextual word embeddings
3. Word2Vec embeddings
4. FastText embeddings

However, I found that the embeddings were not able to adapt to the data very well. This may be due to the fact that the data in Stackoverflow uses a very specific vocabulary.The pre-trained embeddings above are trained on plain English text such as Wikipedia or News corpus, and almost never encounter the words we are hoping to realize. Therefore, I decided to train the embeddings for my data from scratch. I tried both Word2Vec and FastText embeddings, and Word2Vec gave the best results so I decided to use that.

The word embeddings are then used to train a **Tag Classifier**, which predicts the most suitable tags for a user query.

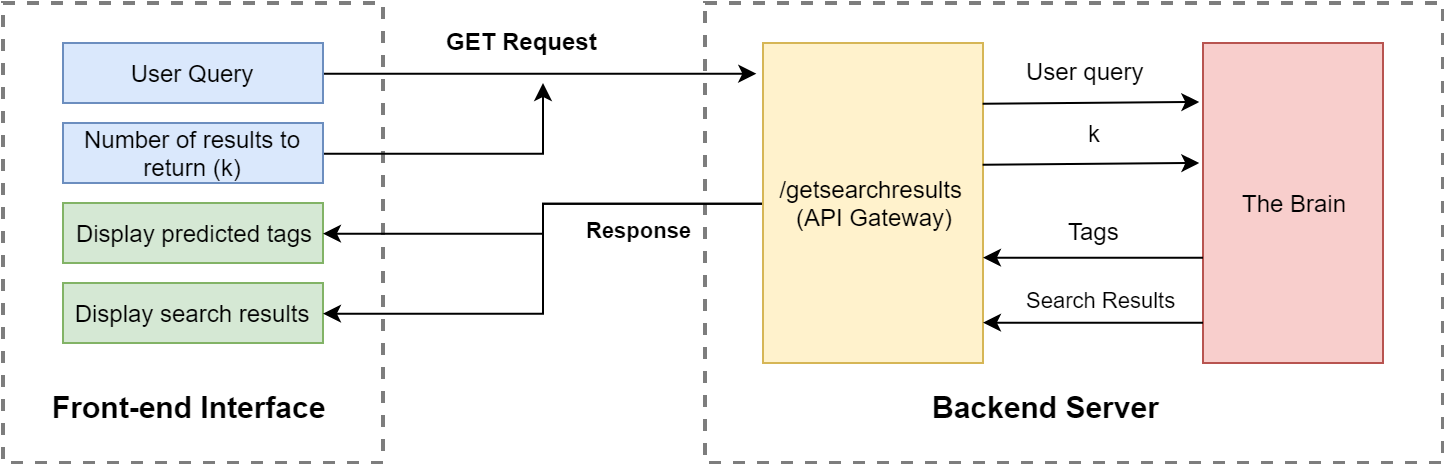
Finally, the trained word embeddings are used to calculate similarity measures between all the available titles and a user query and then retrieve the most suitable results.

***Note:*** *While typical semantic search implementations utilize ‘cosine similarity’ to rank results,* ***I have come up with a custom measure which is specifically created for our task****. It takes into factors such as the post popularity and sentiment while ranking the results. The weights assigned to each of them were tuned manually after several experiments based on sanity checks.*

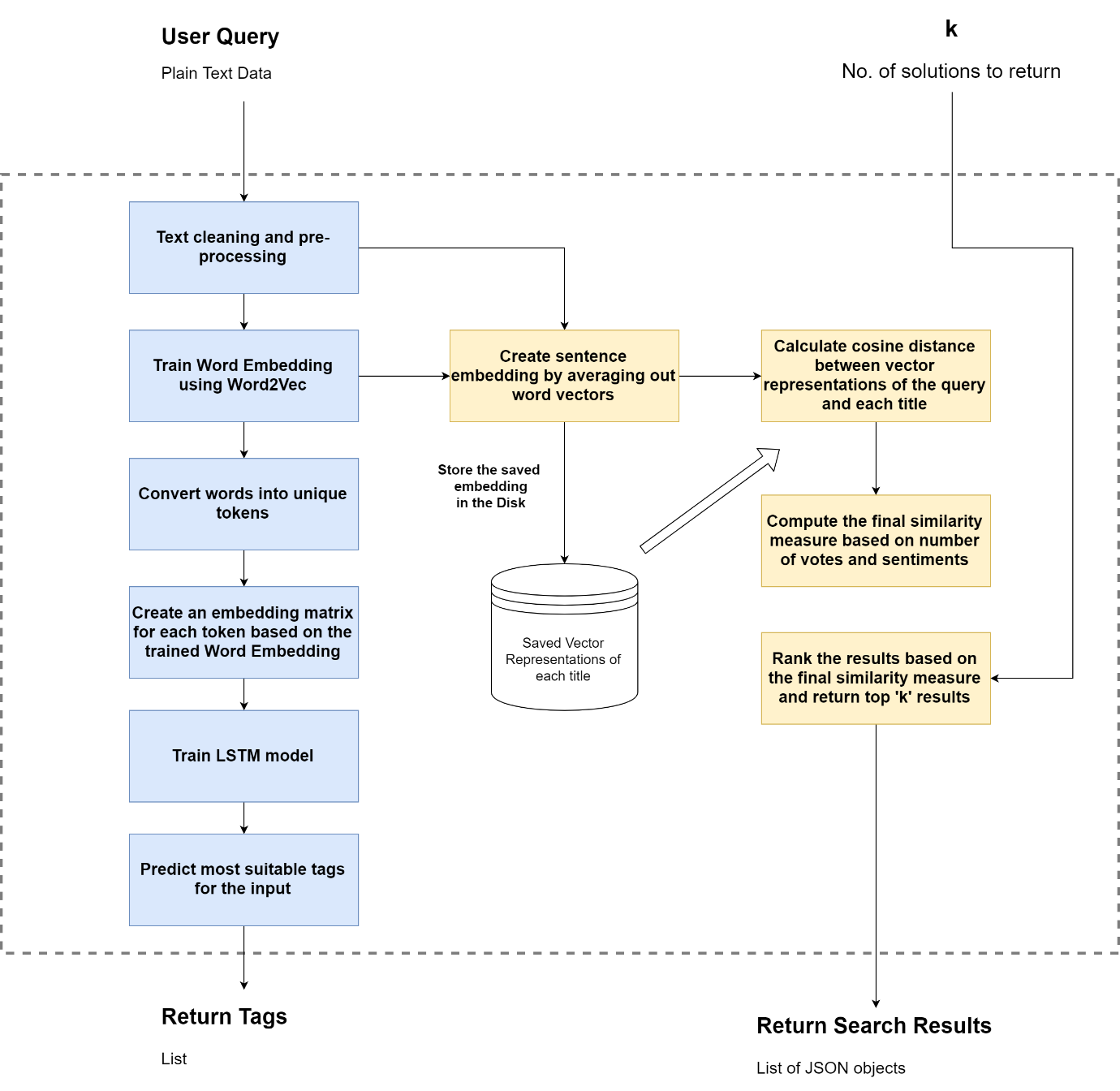
**Technologies Used**

The final solution is a Web application which is a combination of the following:

1. **A Deep learning model for classification of tags (written in Python):** 
   1. Pre-processing for the data: nltk, spacy and pandas
   2. Data preparation for the model: sklearn, keras
   3. The model definition and training: keras
   4. Semantic similarity calculation: sklearn
2. **A visually aesthetic front-end (written in ReactJS):**
   1. Structure and logic: ReactJS and Javascript
   2. Styling: Bootstrap and CSS
3. **A server backend for serving the deep learning algorithm (written in Python):**
   1. Server creation and API Gateways: Flask
   2. Model Serving: Tensorflow



**Figure 1: Application Architecture**

****

**Figure 2. The Brain**

**Scope of the Project**

Given the vast amount of data given on Stack overflow, I decided to exercise a few constraints for the proof of concept:

1. I have restricted the data to only Python Related Questions
2. I have restricted the possible tags to 500
3. I have used somewhat lower amounts of data points (~140,000) for faster processing
4. Since this project is mostly just a proof of concept, the web interface makes consecutive API calls to the server. **This is not optimal for a production environment**, and has only been added for visual aesthetic.

Further improvements may include:

* Experiment to solve the problem using Topic Modelling or other sophisticated NLP tasks
* Consider larger number of data points
* Experiment with different architectures for the final classification network