**Models To generate Solutions from Queries in Jira Ticket**

**Approach 1: Creating our own Model Using RNN:**

* M-1: LSTM model :

LSTM helps in generating sequence models for our data, and has long-term memory context which offers more accurate and flexible model in generating text solutions to our query.

We will first, preprocess the text by tokenizing, removing stopwords, and converting it into numerical representations using embeddings. Next, design an LSTM architecture that includes an embedding layer to handle the input sequences, followed by one or more LSTM layers to capture sequential dependencies. Train the model using labeled data where Jira issues are paired with their solutions, optimizing it to minimize loss during training. Evaluate the model to ensure it generalizes well, and utilize it for predicting the best-matching solution for new Jira issues based on their descriptions.

Example code:

model = Sequential()

model.add(LSTM(units=100, input\_shape=(embeddings\_proxy.shape[1], embeddings\_proxy.shape[2])))

model.add(Dense(units=len(label\_dict), activation='softmax'))

# Compile model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Print model summary

print(model.summary())

# Train model

history = model.fit(embeddings\_proxy, encoded\_answers, epochs=10, batch\_size=32, validation\_split=0.2)

* **STRENGTHS:**
* **Sequence Modeling**: Particularly advantageous for tasks where understanding the context and flow of information (such as Jira issue descriptions) is crucial for accurate matching to solutions.
* **Long-term Dependencies**: This capability allows them to accurately handle Jira issue descriptions of varying lengths and complexities, ensuring comprehensive understanding and accurate matching.
* **High Contextual Understanding and Flexibilty**: Helps in detailed queries and adapts to different datasets and query sets.
* **Scalable and have high generalization**
* **Weakness:**
* **Complex in Training**: while powerful, these can be complex to train and fine-tune effectively This complexity can sometimes lead to longer development cycles and higher computational costs compared to simpler models.
* **Limited Contextual Understanding**: While LSTMs are adept at capturing long-term dependencies within sequences, they may struggle with capturing broader contextual information or domain-specific knowledge that could impact the matching of Jira issues to solutions.
* **Fixed Length Input Limitations**
* **Might give errors in unstructured data**
* **Resource Intensive**
* **Opportunity for our Task:**
* **Very good at handling sequence data for** our jira ticket issue description. This helps with dependencies and patterns within the text, which is crucial for accurately mapping issues to corresponding solutions.
* Can be customized according to domain specific question sets which will help accuracy and relevance.
* Has long term memory so helps so has the ability to capture long context in jira issues.
* **Limitations for our Task:**
* Complex in Training and utilizes huge amount of resouces as well. LSTM models require careful parameter tuning and sufficient training data to perform effectively.
* cases where Jira issue descriptions are brief or lack sufficient contextual information, LSTM models may struggle to extract meaningful patterns or dependencies, potentially affecting the accuracy of matching solutions.
* Needs handling of preprocessing data as it has Fixed Input limit
* M-2: GRU model :

Using a GRU (Gated Recurrent Unit) model for our project involves preprocessing Jira issue descriptions, designing a GRU-based neural network with embedding and GRU layers to capture dependencies, training it on labeled data, evaluating its performance, and deploying it to predict solutions for new issues based on descriptions. GRU's efficiency in handling sequential data makes it ideal for accurately matching Jira issues with appropriate solutions, optimizing support processes and user satisfaction.

It optimizes our approach from LSTM and then helps us save resources. It utilizes update and reset box which enables to have short term as well as long term memory and saving resources.

Example code:

# Define the GRU model

model\_gru = Sequential()

# Add GRU layer

model\_gru.add(GRU(units=100, input\_shape=(embeddings\_proxy.shape[1], embeddings\_proxy.shape[2])))

# Add Dense output layer

model\_gru.add(Dense(units=len(label\_dict), activation='softmax'))

# Compile the model

model\_gru.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Print model summary

print(model\_gru.summary())

# Train the model

history\_gru = model\_gru.fit(embeddings\_proxy, encoded\_answers, epochs=10, batch\_size=32, validation\_split=0.2)

* **STRENGTHS:**
* **Sequence Modeling**: Particularly advantageous for tasks where understanding the context and flow of information (such as Jira issue descriptions) is crucial for accurate matching to solutions.
* **Simpler Architecture**: GRUs have a simpler architecture compared to LSTMs, which makes them easier to understand, implement, and train. This simplicity often results in faster training times and reduced computational overhead.
* **Efficient Training and Less Time to Train**: Due to their fewer parameters and operations (e.g., fewer gates), GRUs can be quicker to train compared to LSTMs
* **Less Susceptible to Overfitting**: With fewer parameters, GRUs are less prone to overfitting compared to LSTMs, especially when the training dataset is not extensive. This attribute can lead to better generalization on validation and test data.
* **Weakness:**
* **Limited Long-Term Dependency**: GRUs may struggle with capturing long-term dependencies in sequential data as effectively as LSTMs. This limitation can impact their ability to model relationships that span across longer sequences or require retaining information over extended periods.
* **Less Control over Memory**: Unlike LSTMs, which have separate mechanisms to control the input, forget, and output gates, GRUs combine these functionalities into a single update gate and reset gate
* **Less Efficient on Complex Task**
* **Learning Contextual Information**: In some applications, especially those requiring comprehensive context understanding and precise sequence modeling (e.g., language translation, text generation), LSTMs' ability to explicitly manage long-term dependencies can be advantageous over GRUs.
* **Opportunity for our Task:**
* **Enhanced Efficiency in Training**: reducing training time and computational resources needed for processing Jira issue descriptions.
* **Effective Handling of Short-Term Dependencies**: beneficial for tasks involving quick contextual analysis of Jira issues to map them to solutions efficiently.
* easier handling of variable-length Jira issue descriptions without the need for extensive preprocessing, making them more flexible in real-world applications.
* **Limitations for our Task:**
* P**otential Limitations in Capturing Long-Term Dependencies**: Compared to LSTMs, GRUs may not perform as well in capturing long-term dependencies in Jira issue descriptions, which could affect their ability to understand complex relationships over extended sequences.
* **Complexity in Contextual Understanding**: In cases where Jira issue descriptions require deep contextual analysis spanning multiple interactions or stages, the simplified gating mechanism of GRUs may not be as effective as LSTMs in retaining and utilizing long-term context.
* T**rade-off in Resource Efficiency**: While GRUs are generally more efficient in terms of training speed and computational resources compared to LSTMs, they may sacrifice some long-term memory capabilities, which could impact accuracy in certain complex Jira issue resolution tasks.