**Explanation of My Approach**

The objective of this assignment was to design a deep learning model that could learn to compute the derivative of a function with respect to the requested variable. Since the task requires machine learning prowess and avoids heuristic-based solutions, I took a purely data-driven approach. I aimed to leverage the power of sequence models like LSTMs to predict mathematical operations symbolically, which mirrors the structure of language models. Here's a summary of my thought process:

**Model Architecture**:

* + Embedding Layers: I utilized embedding layers to transform symbolic inputs into dense vector representations, facilitating meaningful learning from the input functions.
  + LSTM Layers: I integrated two LSTM layers to capture temporal dependencies in the function sequences. This design ensures the model learns patterns and relationships over variable-length inputs, as derivatives involve step-by-step transformations.
  + Attention Mechanism: I added an attention mechanism to focus on relevant portions of the input when calculating derivatives, improving performance by enabling the model to 'attend' selectively to key components.
  + Concatenation and Dense Output Layer: I concatenated the LSTM and attention outputs to provide rich, contextualized representations, followed by a dense layer to predict the derivative sequence.

1. **Training Strategy**:
   * I trained the model using symbolic function-derivative pairs, optimizing for sequence accuracy. Given the sequential nature of the task, I expected LSTM layers to gradually learn how to compute derivatives through pattern recognition over multiple epochs.

**Challenges and Current Results**

The primary issue I encountered was 0.17% accuracy, which indicates the model isn't learning effectively. One reason could be that derivatives are mathematically complex operations, and the model might require more layers and data to converge. Another bottleneck is computational time: on my laptop, one epoch takes 1.9 hours, making extensive experimentation difficult.

**Suggestions for Improvement**

1. Add Dropouts: Introduce Dropout layers with probabilities around 0.1-0.2 after the LSTM and dense layers. This will help regularize the network and prevent overfitting.
2. Increase Model Depth: Add 5-7 layers to enhance the model's capacity. More layers can capture deeper patterns in the data, especially for complex mathematical tasks like derivatives.
3. Train for More Epochs with Early Stopping: To ensure convergence, train the model for 100 epochs with early stopping (patience = 5 epochs). Early stopping will prevent unnecessary computations if the model ceases to improve.
4. Parallelize Data Loading: Implement multiprocessing for data preprocessing to accelerate training, especially when working with large datasets.
5. Use a Cloud GPU or Colab: Given the computational challenges, consider using Google Colab or another cloud-based platform to leverage GPUs. This will significantly reduce training time and allow for more extensive experimentation.
6. Hyperparameter Tuning: Experiment with different learning rates, optimizers, and batch sizes. Sometimes, a small change in these parameters can lead to significant improvements.
7. Symbolic Data Augmentation: Generate synthetic function-derivative pairs or augment existing data to ensure the model sees diverse patterns during training.

This project was a challenging yet rewarding attempt to demonstrate my ability to solve mathematical problems with deep learning. Although I couldn’t achieve 0.70 accuracy within the given constraints, I believe the foundational architecture and approach are sound. Implementing the above suggestions—such as dropout layers, deeper networks, and early stopping—would improve the model's performance. My next step would be to move training to a GPU environment to reduce time and explore more sophisticated architectures, like Transformer-based models, for symbolic learning.