**How to Build a Mini GPT in Java**

The Generative Pre-trained Transformer (GPT) model has become extremely popular in the field of natural language processing (NLP). GPT models are used for a wide range of applications, such as text generation, translation, summarization, and many more. Although many versions of GPT are built in Python with frameworks such as TensorFlow or PyTorch, you may be curious about how to develop a simplified version of GPT in Java. In this blog, we'll explore the steps to create a mini GPT model in Java.

**What is GPT ?**

GPT relies on the Transformer design, which consists of self-attention mechanisms enabling the model to determine the significance of various words within a sequence. It utilises a series of Transformer blocks, with each one consisting of a self-attention layer and feedforward neural networks. The model is pre-trained on large text datasets and fine-tuned for specific tasks.

**Getting Started**

## **1. Establishing the Environment**

In order to begin, you will require the tools and libraries listed below:

* Make sure you have **Java Development Kit (JDK) 11** or a newer version installed.
* **Apache Maven** or **Gradle** are used for managing dependencies and building the project.
* **DJL**, which stands for **Deep Java Library**, is a freely available library that enables Java to utilise deep learning functionalities.
* **MXNet:** A deep learning platform that works with DJL.

You have the option to create a Maven or Gradle project and include the specified dependencies in your **pom.xml** or **build.gradle** file.

**Maven (pom.xml):**

| <dependencies>  <dependency>  <groupId>ai.djl</groupId>  <artifactId>djl-api</artifactId>  <version>0.16.0</version>  </dependency>  <dependency>  <groupId>ai.djl.mxnet</groupId>  <artifactId>mxnet-engine</artifactId>  <version>0.16.0</version>  </dependency> </dependencies> |
| --- |

**Gradle (build.gradle):**

| dependencies {  implementation 'ai.djl:djl-api:0.16.0'  implementation 'ai.djl.mxnet:mxnet-engine:0.16.0' } |
| --- |

**2. Implementing the Mini GPT Model**

Next, we will create a simple Transformer model using DJL and MXNet in Java. We will go over the key elements.

**Developing the Transformer Model**

The Transformer model is composed of blocks for both encoding and decoding. To keep things simple, we will concentrate on developing a basic version of the decoder as GPT is specifically a model for decoding.

| import ai.djl.Model; import ai.djl.ModelException; import ai.djl.nn.Block; import ai.djl.nn.SequentialBlock; import ai.djl.training.util.ProgressBar;  public class MiniGPT {   public static Model buildModel() {  Model model = Model.newInstance("mini-gpt");   Block transformerBlock = new SequentialBlock()  .add(/\* Add Transformer Decoder Layers Here \*/);   model.setBlock(transformerBlock);  return model;  }   public static void main(String[] args) throws ModelException {  Model model = buildModel();  model.load(new ProgressBar());  // Further training and inference logic  } } |
| --- |

**Implementing the Self-Attention Mechanism**

The self-attention mechanism plays a critical role in GPT. It evaluates the significance of each word compared to the other words in the series. Here is the implementation of a self-attention layer:

| import ai.djl.ndarray.NDArray; import ai.djl.ndarray.NDManager; import ai.djl.nn.core.Linear; import ai.djl.nn.Blocks; import ai.djl.nn.SequentialBlock;  public class SelfAttention extends SequentialBlock {   public SelfAttention(int hiddenSize) {  this.add(Linear.builder().setUnits(hiddenSize).build())  .add(Blocks.identity())  .add(Linear.builder().setUnits(hiddenSize).build());  }   @Override  protected NDArray forwardInternal(NDManager manager, NDArray[] inputs) {  NDArray query = inputs[0];  NDArray key = inputs[1];  NDArray value = inputs[2];   NDArray attentionWeights = query.dot(key.transpose());  attentionWeights = attentionWeights.softmax(-1);  return attentionWeights.dot(value);  } } |
| --- |

**Model training**

Training consists of adjusting the model's parameters in order to reduce the loss function. To train effectively, you must have a dataset and specify both a loss function and an optimizer.

| import ai.djl.training.Trainer; import ai.djl.training.loss.Loss; import ai.djl.training.dataset.Dataset; import ai.djl.training.dataset.RandomAccessDataset; import ai.djl.training.optimizer.Optimizer;  public class TrainerSetup {   public void trainModel(Model model, RandomAccessDataset trainingData) {  Trainer trainer = model.newTrainer(new Trainer.Config()  .optLoss(Loss.softmaxCrossEntropyLoss())  .optOptimizer(Optimizer.adam())  .optDevices(Device.getDevices(1)));   for (int epoch = 0; epoch < 10; epoch++) {  for (Batch batch : trainer.iterateDataset(trainingData)) {  trainer.trainBatch(batch);  trainer.step();  batch.close();  }  trainer.notifyListeners(listener -> listener.onEpoch(trainer));  }  } } |
| --- |

**Making deductions**

Once you have completed your training, you will be able to generate text by utilising the model that was trained. The process of inference includes feeding a prompt into the model and allowing it to predict the subsequent tokens.

| import ai.djl.Model; import ai.djl.inference.Predictor; import ai.djl.translate.TranslateException;  public class Inference {   public String generateText(Model model, String prompt) throws TranslateException {  try (Predictor<String, String> predictor = model.newPredictor(/\* Translator Implementation \*/)) {  return predictor.predict(prompt);  }  } } |
| --- |

**Conclusion**

Creating a Mini GPT in Java presents a difficult but fulfilling challenge. Even though the model we've talked about is basic, it presents important ideas such as self-attention and Transformer blocks. When implementing a GPT for production, think about using TensorFlow or PyTorch with Java bindings or utilising pre-trained models in the DJL model zoo.

After going through these steps, you should now have a fundamental grasp of how to develop and train a Transformer-based model in Java.

**References:**

1. Feel free to use the reference links for more detailed information and guides:

* [**DJL Documentation**](https://docs.djl.ai/master/docs/index.html)
* [**MXNET Documentation**](https://mxnet.apache.org/versions/1.9.1/api)

1. Check out the GitHub repository for detailed source code and functional examples. Explore practical implementations to enhance your projects.

* [**Mini GPT Demo on GitHub**](https://github.com/vtg04/Mini_GPT)