**Phase 1: Foundation and Core Development**

The focus will be on setting up the necessary data preprocessing and initial model development.

**Plan**

1. Clean and Normalize Text
2. Tokenize and Annotate Text
3. Encode and Represent Data
4. Handle Contextual Data
5. Select Initial Model (LSTM)

**Phase 1 Tasks:**

Task Division:

1. **Data Preparation (Team Member 1)**: Cleaning and normalization of the Quranic text to prepare for tokenization and modeling.
2. **Dataset Creation (Team Member 2)**: Conversion of the text into a suitable format for fine-tuning the AraBERT model.
3. **Model Setup (Team Member 3)**: Setting up the AraBERT model for fine-tuning with the prepared dataset.
4. **Initial Model Training and Evaluation (Team Member 4)**: Fine-tuning the model on the prepared dataset and setting up a simple evaluation to assess performance.

**Division of Labor Among Team Members:**

* **Team Member 1** focuses on data cleaning and normalization, ensuring the text is ready for tokenization and modeling.
* **Team Member 2** is responsible for preparing the dataset, converting the cleaned text into a structured format suitable for model input.
* **Team Member 3** sets up the AraBERT model and prepares the tokenizer for processing Arabic text, making it ready for training.
* **Team Member 4** undertakes the initial training of the model on the dataset prepared by Team Member 2, followed by a basic evaluation to assess its performance.

This workflow ensures that the team can collaboratively prepare the data, set up the model, and commence training efficiently, adjusting as necessary based on initial performance outcomes.

**Team Member 1: Data Preparation**

import pandas as pd

import re

# Assuming we have the Quran text in a file: 'quran-text-uthmani.txt'

def clean\_normalize\_text(filepath):

    with open(filepath, 'r', encoding='utf-8') as file:

        text = file.readlines()

    cleaned\_text = []

    for line in text:

        # Normalize Arabic letters and remove diacritics

        line = re.sub(r'[أإآ]', 'ا', line)

        line = re.sub(r'[ًٌٍَُِّْ]', '', line)

        # Keep only Arabic letters and spaces

        line = re.sub(r'[^ء-ي ]', '', line)

        cleaned\_text.append(line)

    return cleaned\_text

cleaned\_text = clean\_normalize\_text('/path/to/quran-text-uthmani.txt')

**Team Member 2: Dataset Creation**

# Convert cleaned text to a DataFrame for easier manipulation

df = pd.DataFrame(cleaned\_text, columns=['text'])

# For simplicity, let's assume each line is a separate verse; hence, no need for further splitting.

# In practice, you might want to further process or split the text based on your requirements.

# Save the cleaned data to a new CSV file for easy access during model training

df.to\_csv('/path/to/cleaned\_quran\_text.csv', index=False)

**Team Member 3: Model Setup**

from transformers import AutoTokenizer, AutoModelForMaskedLM

# Initialize tokenizer and model for AraBERT

tokenizer = AutoTokenizer.from\_pretrained('aubmindlab/bert-base-arabertv02')

model = AutoModelForMaskedLM.from\_pretrained('aubmindlab/bert-base-arabertv02')

# Prepare the tokenizer for Arabic text handling

def prepare\_for\_tokenizer(text, tokenizer, max\_length=512):

    # Tokenize and encode sequences in the training set

    tokens = tokenizer(text, max\_length=max\_length, truncation=True, padding="max\_length", return\_tensors="pt")

    return tokens

# Example usage with one line of text

example\_tokens = prepare\_for\_tokenizer(df['text'].iloc[0], tokenizer)

**Team Member 4: Initial Model Training and Evaluation**

from torch.utils.data import DataLoader, Dataset

import torch

# Define a custom dataset for the Quran text

class QuranTextDataset(Dataset):

    def \_\_init\_\_(self, encodings):

        self.encodings = encodings

    def \_\_getitem\_\_(self, idx):

        return {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}

    def \_\_len\_\_(self):

        return len(self.encodings.input\_ids)

# Assuming 'example\_tokens' contains tokenized input for multiple verses

dataset = QuranTextDataset(example\_tokens)

loader = DataLoader(dataset, batch\_size=16, shuffle=True)

# Simplified training loop - for demonstration purposes

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model.to(device).train()

optimizer = torch.optim.AdamW(model.parameters(), lr=5e-5)

for epoch in range(3):  # For demonstration, let's just do 3 epochs

    for batch in loader:

        optimizer.zero\_grad()

        input\_ids = batch['input\_ids'].to(device)

        labels = batch['input\_ids'].to(device)

        outputs = model(input\_ids, labels=labels)

        loss = outputs.loss

        loss.backward()

        optimizer.step()

    print(f"Epoch {epoch} Loss: {loss.item()}")

**Phase 2: Enhancement and Expansion**

**Includes**: Python > Natural Language Processing, Deep Learning **Libraries**: pandas, numpy, nltk, regex, keras, matplotlib **Requirements**: V=3 verbose, with comments explaining each step for team collaboration

**Plan**

1. Enhance Model Training and Data Handling
2. Develop Immediate Correction with Self-Correction Window
3. Implement Session Review and Progress Tracking
4. Introduce Progressive Difficulty Levels and Gamification
5. Build a Web Application Skeleton

**Phase 2 Tasks:**

Task Division:

1. **Fine-Tuning AraBERT (Team Member 1)**: Adjust AraBERT for the task of next-word prediction, including data preparation for fine-tuning.
2. **Developing Custom Seq2Seq Model (Team Member 2)**: Build and train a Seq2Seq model with attention tailored to Quranic text.
3. **Model Evaluation and Selection (Team Member 3)**: Evaluate both models' performance and devise a strategy for using them in tandem.
4. **Integration and Testing (Team Member 4)**: Integrate the models into the application with a mechanism to select or blend their outputs.

**Division of Labor Among Team Members:**

* **Team Member 1** focuses on fine-tuning AraBERT for next-word prediction, handling data preparation and model training.
* **Team Member 2** builds a custom Seq2Seq model with attention, developing and training the model specifically for Quranic Arabic.
* **Team Member 3** evaluates both models to determine their performance and develops a strategy for using them together effectively.
* **Team Member 4** integrates the models into the application, creating a system to select or blend their outputs based on performance.

This setup ensures a comprehensive approach, leveraging the strengths of both Transformer and Seq2Seq models for understanding and generating Quranic Arabic text.

**Team Member 1: Fine-Tuning AraBERT**

from transformers import AutoModelForMaskedLM, AutoTokenizer, Trainer, TrainingArguments

from datasets import load\_dataset

# Load the tokenizer and model

tokenizer = AutoTokenizer.from\_pretrained('aubmindlab/bert-base-arabertv02')

model = AutoModelForMaskedLM.from\_pretrained('aubmindlab/bert-base-arabertv02')

# Prepare the dataset for fine-tuning

def encode(examples):

    return tokenizer(examples['text'], padding='max\_length', truncation=True, max\_length=128)

# Assuming you have your dataset in a CSV file after preprocessing in Phase 1

dataset = load\_dataset('csv', data\_files={'train': 'path/to/cleaned\_quran\_text.csv'})

dataset = dataset.map(encode, batched=True)

dataset.set\_format(type='torch', columns=['input\_ids', 'attention\_mask', 'labels'])

# Define training arguments

training\_args = TrainingArguments(

    output\_dir='./results',          # output directory for model and checkpoints

    num\_train\_epochs=3,              # total number of training epochs

    per\_device\_train\_batch\_size=8,   # batch size per device during training

    logging\_dir='./logs',            # directory for storing logs

)

# Initialize Trainer

trainer = Trainer(

    model=model,

    args=training\_args,

    train\_dataset=dataset['train'],

)

# Start fine-tuning

trainer.train()

**Team Member 2: Developing Custom Seq2Seq Model**

import torch

from torch import nn

from torch.nn import functional as F

# Define a simple Seq2Seq model for demonstration purposes

class Seq2SeqModel(nn.Module):

    def \_\_init\_\_(self, vocab\_size, embed\_dim, hidden\_dim):

        super(Seq2SeqModel, self).\_\_init\_\_()

        self.embedding = nn.Embedding(vocab\_size, embed\_dim)

        self.encoder = nn.LSTM(embed\_dim, hidden\_dim, batch\_first=True)

        self.decoder = nn.LSTM(embed\_dim, hidden\_dim, batch\_first=True)

        self.fc = nn.Linear(hidden\_dim, vocab\_size)

    def forward(self, src, trg):

        embedded\_src = self.embedding(src)

        \_, (hidden, \_) = self.encoder(embedded\_src)

        embedded\_trg = self.embedding(trg)

        output, \_ = self.decoder(embedded\_trg, (hidden, \_))

        prediction = self.fc(output)

        return prediction

# Assuming `vocab\_size` is defined

model = Seq2SeqModel(vocab\_size, 256, 512)

**Team Member 3: Model Evaluation and Selection**

def evaluate\_model(model, dataloader, criterion):

    model.eval()

    total\_loss = 0

    with torch.no\_grad():

        for batch in dataloader:

            inputs, targets = batch

            outputs = model(inputs, targets)

            loss = criterion(outputs, targets)

            total\_loss += loss.item()

    avg\_loss = total\_loss / len(dataloader)

    print(f'Average Loss: {avg\_loss}')

# Placeholder for evaluation code. In practice, you would set up dataloaders for each model,

# choose a suitable loss function, and call `evaluate\_model` for both the AraBERT and Seq2Seq models.

**Team Member 4: Integration and Testing**

def integrate\_and\_test\_models(arabert\_model, seq2seq\_model, text):

    # Placeholder for integration code

    # Assume `arabert\_predict` and `seq2seq\_predict` are functions that use each model to predict the next word

    arabert\_prediction = arabert\_predict(arabert\_model, text)

    seq2seq\_prediction = seq2seq\_predict(seq2seq\_model, text)

    # Decision logic for choosing between models or combining outputs

    # This could be as simple as a confidence threshold or more complex ensemble methods

    final\_prediction = choose\_best\_prediction(arabert\_prediction, seq2seq\_prediction)

    return final\_prediction

# This function illustrates how you might set up a simple mechanism to utilize both models.

# Actual implementation would depend on your application's architecture and requirements.

**Phase 3: Deployment, Community, and Continuous Improvement**

**Plan for Phase 3:**

1. **Deployment of the Web Application**:
   * **Team Member 1** will handle the deployment setup, including server configuration, domain setup, and ensuring secure HTTPS connections.
   * **Team Member 2** will oversee finalizing the database schema and deployment, including the setup of a robust backup and recovery process.
2. **User Feedback Loop**:
   * **Team Member 3** will implement a feedback collection system in the web application to gather user experiences and suggestions.
   * **Team Member 4** will create a dashboard for admins to review collected feedback and prioritize feature requests and bug reports.
3. **Community Building**:
   * **Team Member 1** will set up community features like forums or social integrations, allowing users to connect and share their experiences.
   * **Team Member 2** will work on implementing a system for community moderation and support to ensure a positive environment.
4. **Continuous Model Improvement**:
   * **Team Member 3** will develop a process for retraining the model with new data as it is collected, ensuring the application continues to improve over time.
   * **Team Member 4** will focus on setting up A/B testing environments to test new features and models before they are fully deployed.
5. **Tutorial and Onboarding**:
   * **Team Member 1** will develop interactive tutorials to help new users familiarize themselves with the application.
   * **Team Member 2** will design an onboarding process for new users, incorporating tips and guidance for using the application effectively.
6. **Marketing and Outreach**:
   * **Team Member 3** will create marketing materials and strategies for promoting the application to the target audience.
   * **Team Member 4** will reach out to educational institutions and Islamic centers to introduce the application and encourage its use.

**Phase 3 Tasks:**

**Task Division:**

1. **Deployment and User Interface Integration (Team Member 1):** Deploy the application and integrate the models with the user interface.
2. **User Feedback Collection System (Team Member 2):** Develop a system to collect user feedback on predictions.
3. **Model Continuous Improvement (Team Member 3):** Set up a process for continuous model evaluation and retraining based on new data and feedback.
4. **Community Engagement and Support (Team Member 4):** Establish community features and support mechanisms.

**Division of Labor Among Team Members:**

* **Team Member 1:** Handles the technical aspects of deployment, ensuring the models are integrated with the user interface for real-time predictions.
* **Team Member 2:** Develops the feedback collection system, enabling users to provide valuable insights on the model's predictions.
* **Team Member 3:** Focuses on setting up a framework for continuous model improvement, incorporating user feedback and new data into the training process.
* **Team Member 4:** Builds community engagement strategies and support mechanisms to foster a supportive user environment and gather community insights.

This collaborative approach ensures that the application not only benefits from advanced NLP models but also evolves based on user interactions and feedback, leading to a more effective and user-centered Quran memorization aid.

**Team Member 1: Deployment and User Interface Integration**

Deployment involves making the application available to users, typically through a web server or cloud service, and ensuring the model is correctly integrated and functioning within the application's user interface.

# No direct Python code for deployment; this involves configuring web servers, cloud services, etc.

# Example Flask route integrating model predictions:

@app.route('/predict', methods=['POST'])

def predict():

    user\_input = request.form['text']

    # Assuming a function `make\_prediction` that uses the selected model to predict the next word

    prediction = make\_prediction(user\_input)

    return jsonify({'prediction': prediction})

# The `make\_prediction` function would call either AraBERT, the Seq2Seq model, or an ensemble method based on the setup from Phase 2.

**Team Member 2: User Feedback Collection System**

Collecting user feedback is crucial for understanding the model's real-world performance and identifying areas for improvement.

@app.route('/feedback', methods=['POST'])

def collect\_feedback():

    feedback = request.form['feedback']

    user\_input = request.form['user\_input']

    # Save feedback along with the user input for analysis

    save\_feedback(user\_input, feedback)

    return 'Feedback received, thank you!'

def save\_feedback(user\_input, feedback):

    # Placeholder for saving feedback to a database or file system for later analysis

    pass

**Team Member 3: Model Continuous Improvement**

Continuous improvement involves regularly updating the model based on new data, user feedback, and ongoing evaluation to enhance performance and accuracy.

def retrain\_model(model, new\_data):

    # Placeholder for the retraining process

    # This could involve loading new training data (including user corrections) and fine-tuning the model

def evaluate\_model(model, evaluation\_data):

    # Placeholder for model evaluation

    # Evaluate the model's performance on a separate validation set or through user studies

# Continuous improvement might also include automated scripts or scheduled tasks to retrain and evaluate the model periodically.

**Team Member 4: Community Engagement and Support**

Engaging with the user community and providing support are essential for fostering a positive user experience and encouraging active use of the application.

# Example of setting up a community forum - typically done using third-party services or platforms

# Establish a support system for users to ask questions, report issues, or suggest improvements

# This might involve setting up email support, a ticketing system, or integrating a chatbot within the application for immediate assistance.

# Engaging with the community can also include hosting webinars, Q&A sessions, or creating tutorial content to help users get the most out of the application.