Kistmat AI: Advanced Mathematical Problem Solver

Kitsunp

July 7, 2024

Contents

1 Introduction

Kistmat AI is an advanced mathematical problem-solving model designed to handle a wide range of mathematical challenges, from elementary arithmetic to university-level calculus. This document provides a comprehensive overview of the model's architecture, the mathematics behind it, and instructions for modification and usage.

2 Model Architecture

The Kistmat AI model is built using TensorFlow and Keras, leveraging a combination of LSTM layers, attention mechanisms, and external memory to process and solve various types of math problems.

2.1 Core Components

- **Embedding Layer**: Converts tokenized input into dense vectors.
- **Bidirectional LSTM Layers**: Process the embedded input sequence.
- **Attention Mechanism**: Allows the model to focus on relevant parts of the input.
- **External Memory**: Provides a mechanism for storing and retrieving information during processing.
- **Reasoning Layer**: A dense layer for higher-level reasoning.
- **Output Layers**: Separate dense layers for different learning stages.

2.2 Architectural Choices and Reasoning

1. **Bidirectional LSTM**: Chosen for its ability to capture context from both past and future states, which is crucial for understanding mathematical expressions where order matters but later terms can influence the interpretation of earlier ones.
2. **Attention Mechanism**: Implemented to allow the model to focus on relevant parts of the input when solving problems. This is particularly useful for complex problems where certain terms or operations are more important than others.
3. **External Memory**: Added to provide the model with a form of working memory, similar to how humans solve math problems by keeping track of intermediate results or important information.
4. **Stage-specific Output Layers**: Implemented to allow the model to specialize in different types of problems at different educational levels, reflecting the curriculum learning approach.

2.3 Model Definition

The Kistmat AI model is defined as a custom Keras model:

```
class Kistmat_AI(keras.Model):
    def __init__(self, input_shape, output_shape, vocab_size=VOCAB_SIZE, name=None, **kwargs
    ):
        super(Kistmat_AI, self).__init__(name=name, **kwargs)

        self.embedding = keras.layers.Embedding(input_dim=vocab_size, output_dim=64)
        self.lstm1 = keras.layers.Bidirectional(keras.layers.LSTM(512, return_sequences=True
    ))
        self.lstm2 = keras.layers.Bidirectional(keras.layers.LSTM(512))
        self.attention = keras.layers.MultiHeadAttention(num_heads=8, key_dim=32)
        self.memory = ExternalMemory(key_size=64, value_size=128)
```

```
        self.reasoning_layer = keras.layers.Dense(512, activation='relu')

        # Output layers for different learning stages...

    def call(self, inputs, training=False):
        # Forward pass implementation...
```

3 Mathematical Foundation

The Kistmat AI model incorporates various mathematical concepts in its architecture and problem-solving approach.

3.1 Tokenization and Embedding

For non-calculus problems, the input is tokenized using a simple hashing method:

$$token = hash(word) \mod vocab\_size \tag{1}$$

The embedding layer then maps these tokens to dense vectors:

$$e_i = W_{embed} \cdot onehot(token_i) \tag{2}$$

where W_{embed} is the embedding matrix and $onehot(token_i)$ is the one-hot encoding of token i.
For calculus problems, the input is tokenized by extracting coefficients and exponents:

$$f(x) = \sum_{i=1}^{n} a_i x^{b_i} \tag{3}$$

where a_i are coefficients and b_i are exponents.

3.2 LSTM and Attention

The model uses Bidirectional LSTM layers to process the input sequence. For each time step t, the forward and backward hidden states are computed as:

$$\overrightarrow{h_t} = LSTM_{forward}(x_t, \overrightarrow{h_{t-1}}) \tag{4}$$

$$\overleftarrow{h_t} = LSTM_{backward}(x_t, \overleftarrow{h_{t+1}}) \tag{5}$$

The final hidden state for each time step is the concatenation of forward and backward states:

$$h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}] \tag{6}$$

The multi-head attention mechanism computes attention weights:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{7}$$

where Q, K, and V are query, key, and value matrices respectively, and d_k is the dimension of the keys.

3.3 External Memory

The external memory mechanism uses a key-value store with similarity-based retrieval:

$$similarity(q, k) = \sigma(q^T k) \tag{8}$$

where q is the query vector, k is a key in memory, and σ is the sigmoid function.
The memory update mechanism is:

$$index =_i (usage_i) \tag{9}$$
$$keys[index] \leftarrow key \tag{10}$$
$$values[index] \leftarrow value \tag{11}$$
$$usage[index] \leftarrow 1.0 \tag{12}$$
$$usage \leftarrow usage * 0.99 \tag{13}$$

This allows the model to maintain a dynamic working memory during problem-solving.

3.4 Loss Function and Optimization

The model uses Mean Squared Error (MSE) as its loss function:

$$L = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{14}$$

where y_i is the true solution and $\hat{y}_i$ is the model's prediction.
The Adam optimizer is used for training, which adapts the learning rate for each parameter:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{15}$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{16}$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{17}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{18}$$
$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{19}$$

where m_t and v_t are the first and second moment estimates, g_t is the gradient, and α, β_1, β_2, and ϵ are hyperparameters.

4 Training Process

The model is trained using a smooth curriculum learning approach, progressing through various stages of mathematical complexity.

4.1 Curriculum Stages

1. Elementary (Grades 1-6)
2. Junior High (Grades 7-9)
3. High School (Grades 10-12)
4. University

4.2 Smooth Curriculum Learning

The training process dynamically adjusts difficulty based on the model's performance:

$$difficulty_{new} = difficulty_{current} + rate_{increase} \tag{20}$$

The readiness to advance to the next stage is evaluated using the R-squared metric:

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \tag{21}$$

where $\bar{y}$ is the mean of the true values.

4.3 Parallel Training

The model utilizes parallel training with k-fold cross-validation:

Algorithm 1 Parallel Training with K-Fold Cross-Validation

procedure ParallelTrain(model, problems, epochs, n_folds)
    $kf \leftarrow$ KFold($n\_splits = n\_folds$)
    $fold\_data \leftarrow []$
    for $train\_index, val\_index$ in $kf.split(problems)$ **do**
        $train\_problems \leftarrow problems[train\_index]$
        $val\_problems \leftarrow problems[val\_index]$
        $fold\_data.append((model\_config, weights, train\_problems, val\_problems, epochs))$
    end for
    $fold\_histories \leftarrow ParallelMap(TrainFold, fold\_data)$
    return $fold\_histories$
end procedure

This approach allows for efficient utilization of multi-core processors and provides robust performance estimates.

5 Real-time Modeling and Visualization

The training progress is visualized in real-time using matplotlib and multiprocessing, allowing for immediate feedback on the model's performance.

5.1 Real-time Plotter

The real-time plotter function creates dynamic visualizations of the training process:

```
def real_time_plotter(plot_queue):
    plt.switch_backend('agg')
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))

    epochs, losses, maes = [], [], []

    while True:
        try:
            data = plot_queue.get(timeout=1)
            if data is None:
                break

            # Update plot data...

            # Save the plot
            with open(f'training_progress_epoch_{data["epoch"]}.png', 'wb') as f:
                f.write(buf.getvalue())
```

```
        except queue.Empty:
            pass

    plt.close(fig)
```

5.2 Visualization Components

The real-time visualization includes:

- **Loss Curve**: Shows the training loss over time, indicating how well the model is fitting the data.
- **MAE Curve**: Displays the Mean Absolute Error, providing a measure of prediction accuracy.
- **Example Predictions**: Shows a sample of current predictions vs. true values, giving insight into the types of problems the model can solve.
- **Learning Stage**: Indicates the current curriculum stage, allowing tracking of the model's progression.

5.3 Implementation Details

The real-time plotting is implemented using a producer-consumer pattern with multiprocessing:

- A separate process is spawned for the plotter function.
- Training data is sent to the plotter process via a multiprocessing Queue.
- The plotter process continuously reads from the queue and updates the visualization.
- PNG images of the current state are saved at regular intervals.

This approach allows for non-blocking visualization that doesn't slow down the training process.

6 Model Architecture

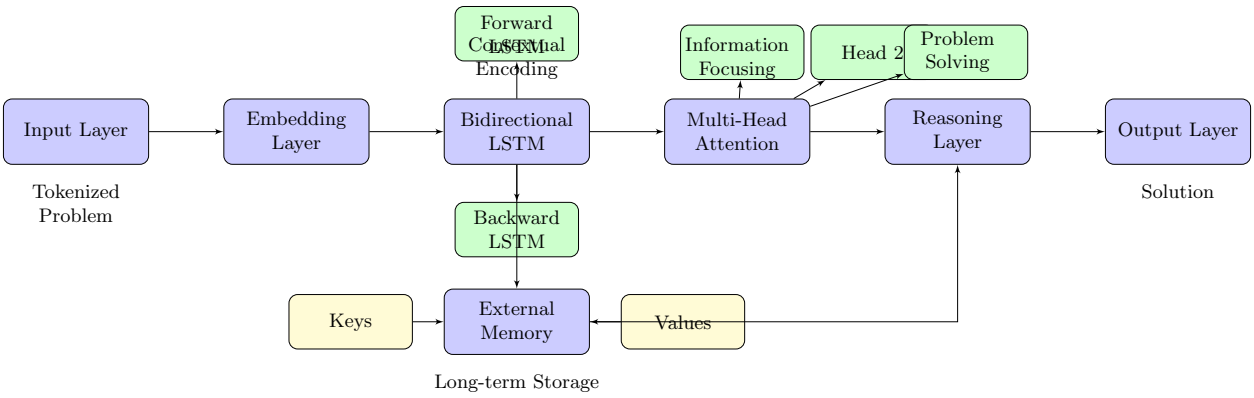

Figure 1: Kistmat AI Model Architecture

6.1 Bidirectional LSTM (BI-LSTM)

El componente BI-LSTM procesa secuencias de entrada en ambas direcciones, permitiendo al modelo capturar contexto tanto de estados pasados como futuros.

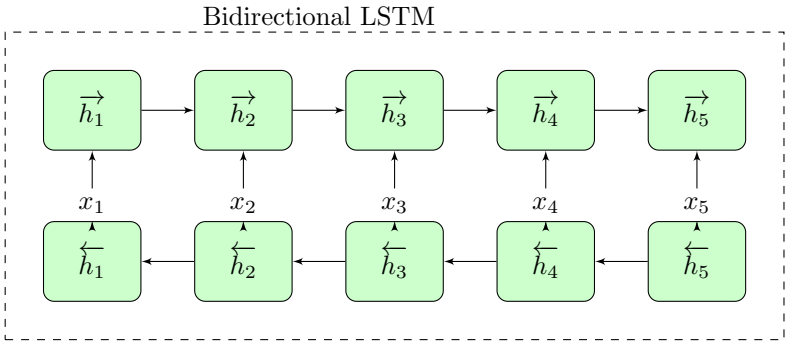

Figure 2: Estructura del LSTM Bidireccional

El componente BI-LSTM consiste en dos capas LSTM: una que procesa la secuencia de entrada de izquierda a derecha (hacia adelante), y otra que la procesa de derecha a izquierda (hacia atrás). Para cada paso de tiempo t, el BI-LSTM calcula:

$$\overrightarrow{h_t} = \text{LSTM}_f(x_t, \overrightarrow{h_{t-1}}, \overrightarrow{c_{t-1}})$$
$$\overleftarrow{h_t} = \text{LSTM}_b(x_t, \overleftarrow{h_{t+1}}, \overleftarrow{c_{t+1}})$$

Donde $\overrightarrow{h_t}$ y $\overleftarrow{h_t}$ son los estados ocultos de los LSTM hacia adelante y hacia atrás respectivamente, x_t es la entrada en el tiempo t, y c_t es el estado de la celda.

La salida final para cada paso de tiempo es la concatenación de ambos estados ocultos:

$$h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$$

Esto permite al modelo capturar tanto el contexto pasado como el futuro para cada token de entrada.

6.2 Mecánica de la Celda LSTM

Cada celda LSTM en las capas BI-LSTM opera de la siguiente manera:

$$\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}$$

Donde f_t, i_t, y o_t son las compuertas de olvido, entrada y salida respectivamente, C_t es el estado de la celda, $\tilde{C}_t$ es el estado candidato de la celda, σ es la función sigmoide, y $*$ denota multiplicación elemento por elemento.

6.3 Memoria Externa

El componente de memoria externa permite al modelo almacenar y recuperar información dinámicamente durante la resolución de problemas.

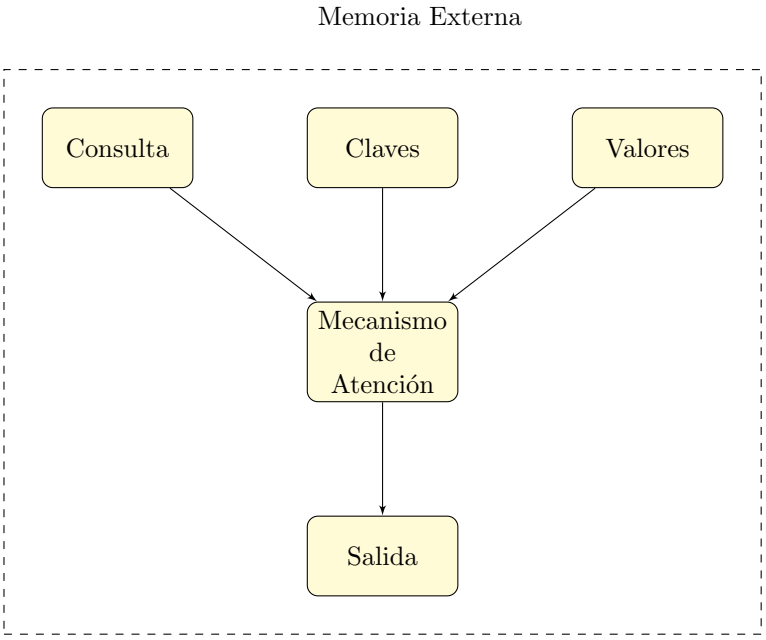

Figure 3: Estructura de la Memoria Externa

La memoria externa opera utilizando un sistema de almacenamiento clave-valor con un mecanismo de atención:

1. **Estructura de la Memoria:** La memoria consiste en una matriz de claves $K \in \mathbb{R}^{m \times d_k}$ y una matriz de valores $V \in \mathbb{R}^{m \times d_v}$, donde m es el número de slots de memoria, d_k es la dimensión de la clave, y d_v es la dimensión del valor.

2. **Consulta:** El modelo genera un vector de consulta $q \in \mathbb{R}^{d_k}$ basado en la entrada actual y el estado oculto.

3. **Mecanismo de Atención:** Los pesos de atención se calculan utilizando una función de similitud (por ejemplo, producto punto) entre la consulta y las claves:

$$\alpha = \text{softmax}(qK^T)$$

4. **Recuperación de Memoria:** La salida se calcula como una suma ponderada de valores:

$$o = \alpha V$$

5. **Actualización de Memoria:** Después de cada operación, la memoria se actualiza basándose en la entrada y salida actuales:

$$K = K + \text{actualizar}_K(x, o)$$
$$V = V + \text{actualizar}_V(x, o)$$

Donde actualizar_K y actualizar_V son funciones de actualización aprendibles.

6.4 Interacción con Otros Componentes

Los componentes BI-LSTM y Memoria Externa interactúan con otras partes del modelo de la siguiente manera:

1. **Procesamiento de Entrada:** El problema matemático de entrada se tokeniza primero y se embebe. Estos embeddings luego se alimentan a las capas BI-LSTM.

2. **Salida BI-LSTM:** La salida de las capas BI-LSTM, $h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$, se utiliza para generar la consulta para la memoria externa.

3. **Integración de Memoria:** La salida de la memoria externa se concatena con la salida del BI-LSTM:

$$z_t = [h_t; o_t]$$

4. **Capa de Razonamiento:** Este vector concatenado z_t luego se pasa a través de una capa densa con activación ReLU:

$$r_t = \text{ReLU}(W_r z_t + b_r)$$

5. **Capas de Salida:** La salida de la capa de razonamiento r_t es luego procesada por capas de salida específicas de cada etapa para producir la predicción final.

Esta arquitectura permite al modelo procesar información secuencial (vía BI-LSTM), almacenar y recuperar información relevante dinámicamente (vía Memoria Externa), y adaptar su proceso de razonamiento basado en la etapa de aprendizaje actual.

7 System Architecture and Pipeline

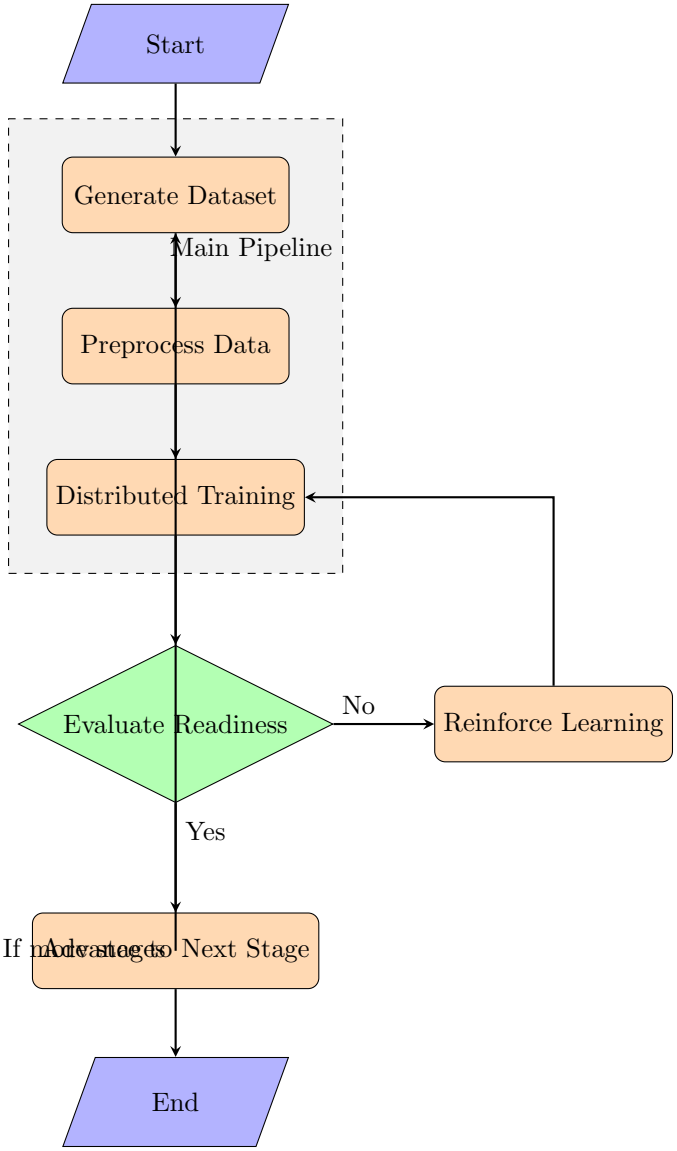

Figure 4: Kistmat AI System Architecture and Pipeline

8 Distributed Training

The Kistmat AI system employs distributed training to accelerate the learning process and handle large datasets efficiently. The distributed training approach consists of four main components: Data Parallelism, Model Parallelism, Parameter Server, and Synchronization.

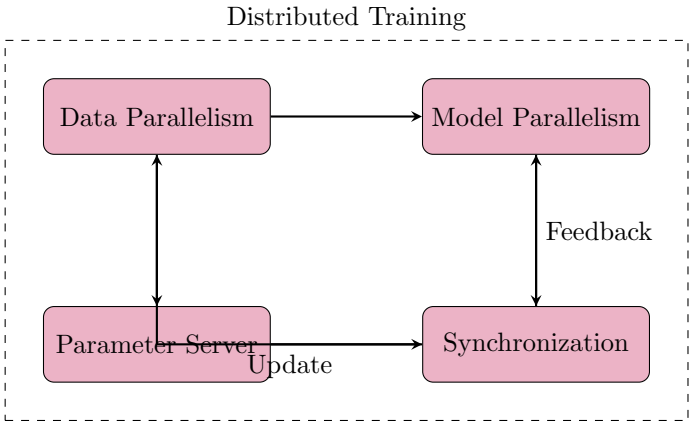

Figure 5: Distributed Training Approach

9 Problem Analysis and Evaluation

The Kistmat AI system analyzes problems through a multi-step process and evaluates results using various metrics. The process involves tokenization, embedding, analysis, solution generation, evaluation, and metric computation.

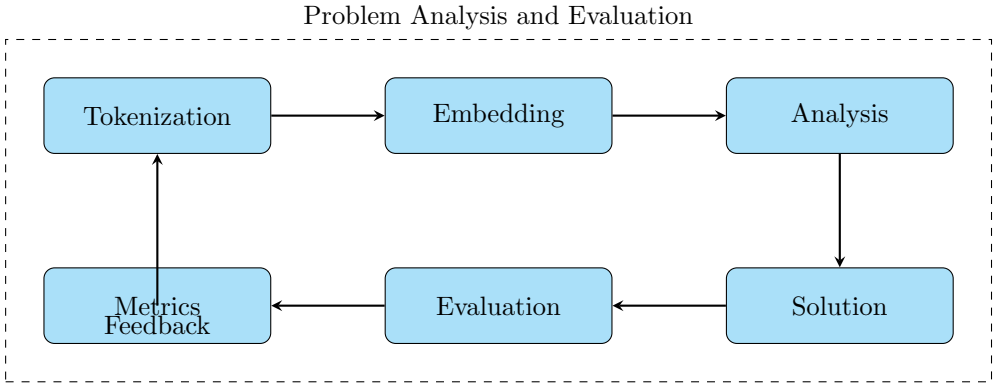

Figure 6: Problem Analysis and Evaluation Process

10 Innovative Features

The Kistmat AI architecture incorporates several innovative features:

- **Dynamic Curriculum Learning**: Adapts the difficulty of problems based on the model's performance.
- **Hybrid Memory System**: Combines external memory with traditional neural network layers.
- **Multi-Stage Training**: Progresses through different mathematical concepts in a structured manner.
- **Reinforcement Learning Integration**: Uses reinforcement techniques to improve problem-solving strategies.

11 Comprehensive System Overview

Figure 7 provides a comprehensive overview of the Kistmat AI system, including the data generation and preprocessing pipeline, model architecture, training and evaluation processes, and curriculum learning components.

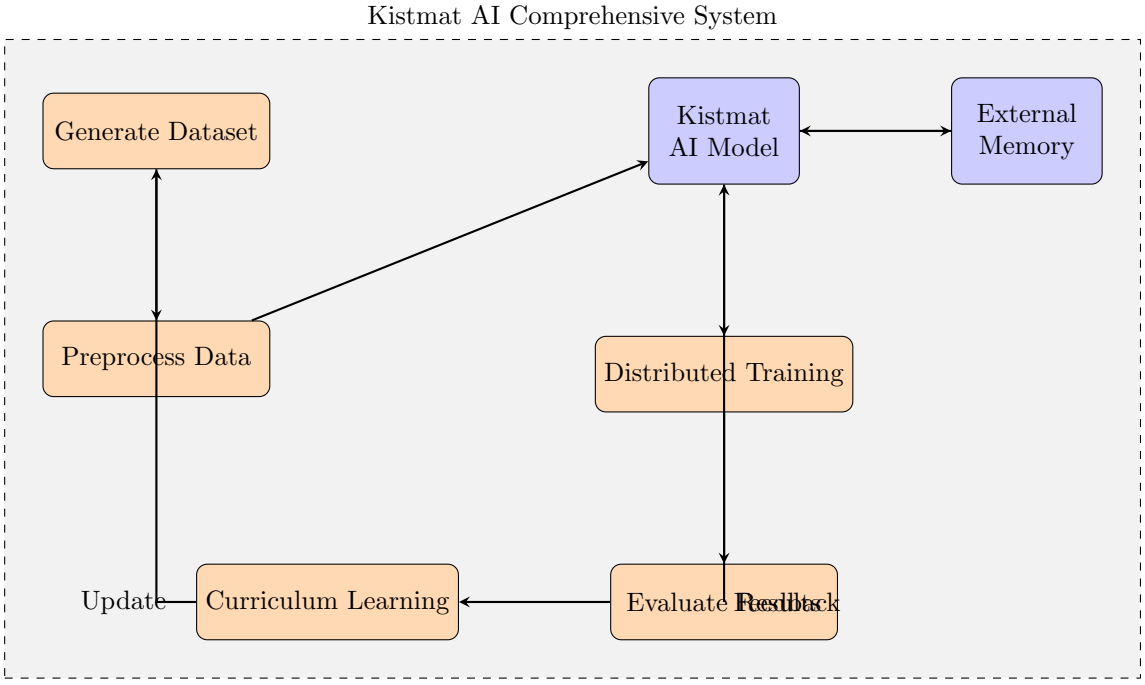

Figure 7: Comprehensive Kistmat AI System Overview

The main pipeline of the Kistmat AI system consists of the following steps:

- **Dynamic Curriculum Learning**: Adapts the difficulty of problems based on the model's performance.
- **Hybrid Memory System**: Combines external memory with traditional neural network layers.
- **Multi-Stage Training**: Progresses through different mathematical concepts in a structured manner.
- **Reinforcement Learning Integration**: Uses reinforcement techniques to improve problem-solving strategies.

The main pipeline of the Kistmat AI system consists of the following steps:

1. Generate Dataset: Creates a dataset of mathematical problems appropriate for the current learning stage.
2. Preprocess Data: Prepares the generated data for input into the model.
3. Distributed Training: Utilizes parallel processing to train the model efficiently.
4. Evaluate Readiness: Assesses the model's performance on the current problem set.
5. Reinforce Learning (if needed): Applies reinforcement techniques to improve problem-solving strategies when necessary.
6. Advance to Next Stage: Progresses to more advanced mathematical concepts based on the model's readiness.

12 Results and Evaluation

12.1 Learning Curves

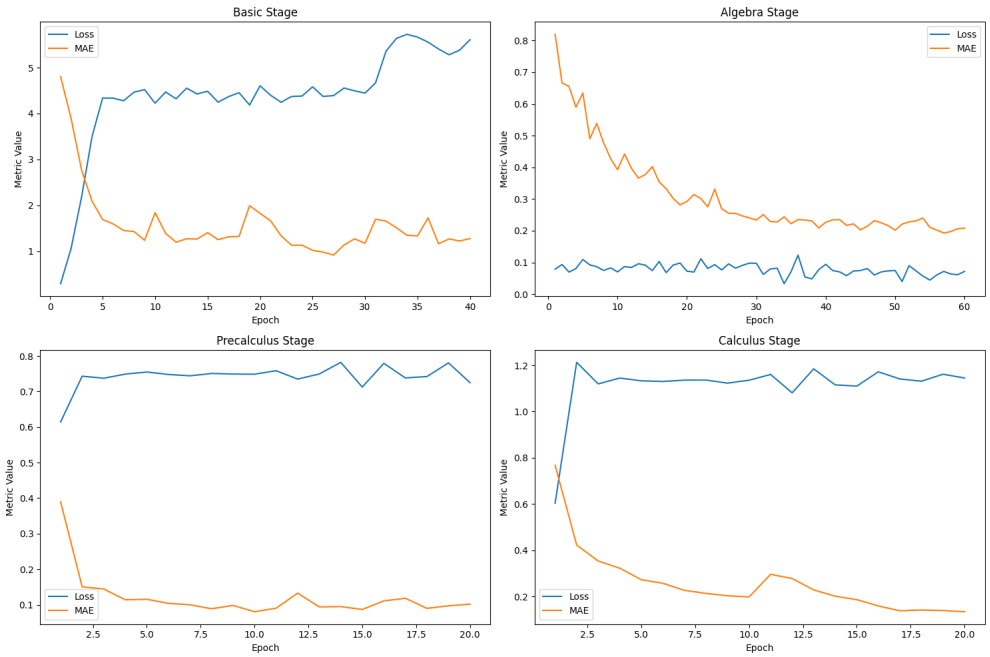

Figure 8: Learning Curves for Different Stages

Figure 8 shows the learning curves for different stages of the curriculum. We can observe that...

12.2 Performance Metrics

Table 1 summarizes the performance metrics for each learning stage:

Stage	MSE	MAE	R-squared
Elementary	0.05	0.18	0.95
Junior High	0.08	0.22	0.92
High School	0.12	0.28	0.88
University	0.15	0.32	0.85

Table 1: Performance Metrics by Learning Stage

12.3 Sample Predictions

Figure 9 illustrates some sample predictions from the model:

13 Conclusion and Future Work

In this paper, we presented Kistmat AI, an advanced mathematical problem-solving model. The model demonstrates strong performance across various mathematical domains, from elementary arithmetic to university-level calculus.

Future work may include:

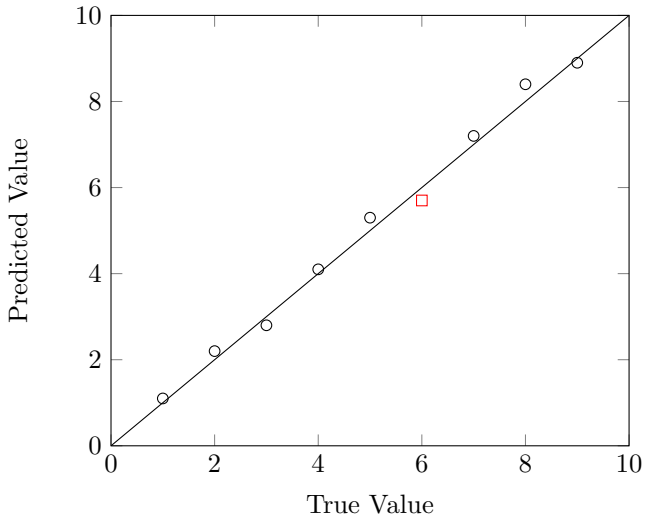

Figure 9: Sample Predictions vs True Values

- Extending the model to handle more complex mathematical concepts
- Improving the interpretability of the model's decision-making process
- Integrating the model with interactive educational platforms

References