

Kistmat AI: Advanced Mathematical Problem Solver

Kitsunp

July 6, 2024

Contents

1	Introduction	2
2	Model Architecture	2
2.1	Core Components	2
2.2	Architectural Choices and Reasoning	2
2.3	Model Definition	2
3	Mathematical Foundation	3
3.1	Tokenization and Embedding	3
3.2	LSTM and Attention	3
3.3	External Memory	3
3.4	Loss Function and Optimization	4
4	Training Process	4
4.1	Curriculum Stages	4
4.2	Smooth Curriculum Learning	4
4.3	Parallel Training	5
5	Real-time Modeling and Visualization	5
5.1	Real-time Plotter	5
5.2	Visualization Components	6
5.3	Implementation Details	6
6	Model Architecture	6
6.1	Bidirectional LSTM (BI-LSTM)	6
6.2	External Memory	7
7	System Architecture and Pipeline	8
8	Distributed Training	8
9	Problem Analysis and Evaluation	9
10	Innovative Features	9
11	Comprehensive System Overview	10
12	Results and Evaluation	10
12.1	Learning Curves	10
12.2	Performance Metrics	10
12.3	Sample Predictions	10
13	Conclusion and Future Work	11

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:

```
1 class Kistmat_AI(keras.Model):
2     def __init__(self, input_shape, output_shape, vocab_size=VOCAB_SIZE, name=None, **kwargs
3     ):
4         super(Kistmat_AI, self).__init__(name=name, **kwargs)
5
6         self.embedding = keras.layers.Embedding(input_dim=vocab_size, output_dim=64)
7         self.lstm1 = keras.layers.Bidirectional(keras.layers.LSTM(512, return_sequences=True)
8         ))
9         self.lstm2 = keras.layers.Bidirectional(keras.layers.LSTM(512))
10        self.attention = keras.layers.MultiHeadAttention(num_heads=8, key_dim=32)
11        self.memory = ExternalMemory(key_size=64, value_size=128)
12        self.reasoning_layer = keras.layers.Dense(512, activation='relu')
13
14        # Output layers for different learning stages...
```

```

13
14     def call(self, inputs, training=False):
15         # 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:

$$token = hash(word) \mod vocab_size \quad (1)$$

The embedding layer then maps these tokens to dense vectors:

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

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:

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

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:

$$\vec{h}_t = LSTM_{forward}(x_t, \vec{h}_{t-1}) \quad (4)$$

$$\overleftarrow{h}_t = LSTM_{backward}(x_t, \overleftarrow{h}_{t+1}) \quad (5)$$

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

$$h_t = [\vec{h}_t; \overleftarrow{h}_t] \quad (6)$$

The multi-head attention mechanism computes attention weights:

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

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:

$$similarity(q, k) = \sigma(q^T k) \quad (8)$$

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

The memory update mechanism is:

$$index =_i (usage_i) \quad (9)$$

$$keys[index] \leftarrow key \quad (10)$$

$$values[index] \leftarrow value \quad (11)$$

$$usage[index] \leftarrow 1.0 \quad (12)$$

$$usage \leftarrow usage * 0.99 \quad (13)$$

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:

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

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:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (15)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \quad (16)$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad (17)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (18)$$

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \quad (19)$$

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:

$$difficulty_{new} = difficulty_{current} + rate_{increase} \quad (20)$$

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

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

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

```

1: procedure PARALLELTRAIN(model, problems, epochs, n_folds)
2:    $kf \leftarrow \text{KFold}(n\_splits = n\_folds)$ 
3:    $fold\_data \leftarrow []$ 
4:   for  $train\_index, val\_index$  in  $kf.split(problems)$  do
5:      $train\_problems \leftarrow problems[train\_index]$ 
6:      $val\_problems \leftarrow problems[val\_index]$ 
7:      $fold\_data.append((model\_config, weights, train\_problems, val\_problems, epochs))$ 
8:   end for
9:    $fold\_histories \leftarrow \text{ParallelMap}(\text{TrainFold}, fold\_data)$ 
10:  return  $fold\_histories$ 
11: 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:

```

1 def real_time_plotter(plot_queue):
2     plt.switch_backend('agg')
3     fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))
4
5     epochs, losses, maes = [], [], []
6
7     while True:
8         try:
9             data = plot_queue.get(timeout=1)
10            if data is None:
11                break
12
13            # Update plot data...
14
15            # Save the plot
16            with open(f'training_progress_epoch_{data["epoch"]}.png', 'wb') as f:
17                f.write(buf.getvalue())
18
19            except queue.Empty:
20                pass
21
22     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

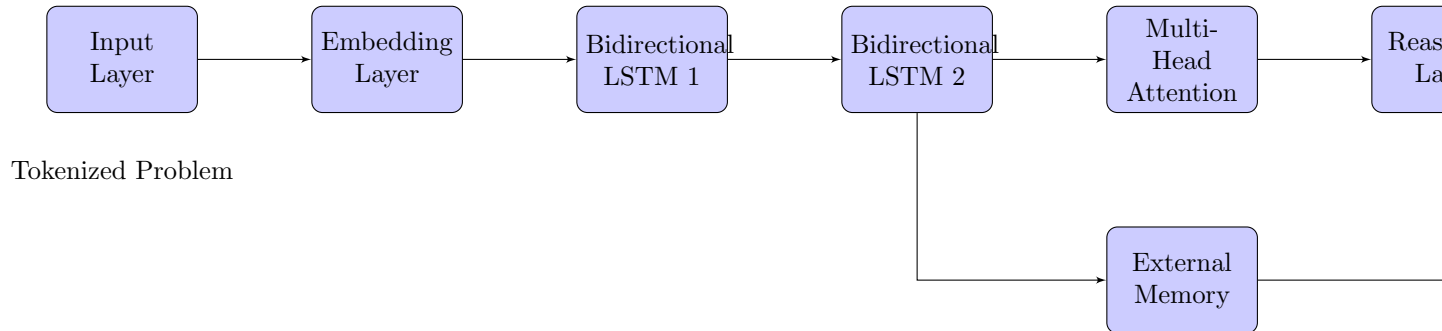


Figure 1: Kistmat AI Model Architecture

6.1 Bidirectional LSTM (BI-LSTM)

The BI-LSTM layers process input sequences in both forward and backward directions, allowing the model to capture context from both past and future states.

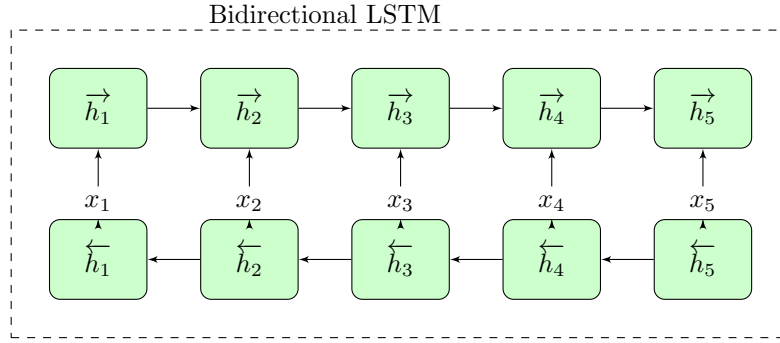


Figure 2: Bidirectional LSTM Structure

6.2 External Memory

The external memory component allows the model to store and retrieve information dynamically during problem-solving.

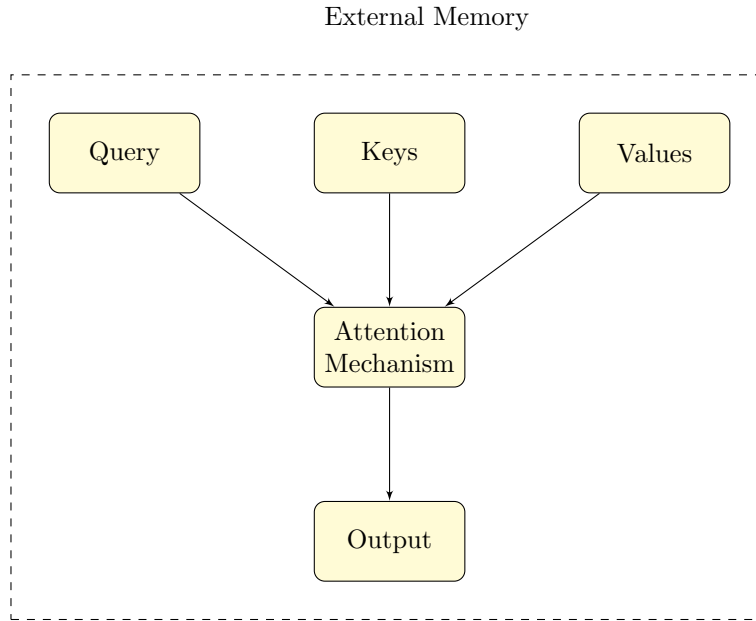


Figure 3: External Memory Structure

7 System Architecture and Pipeline

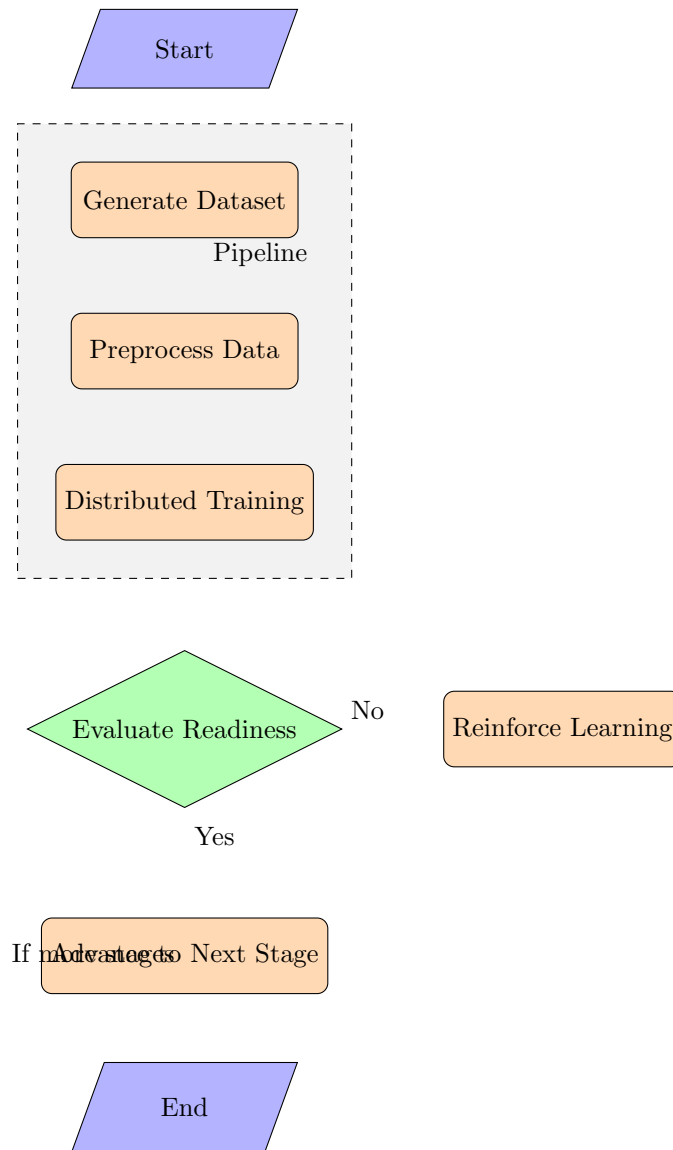


Figure 4: Kistmat AI System Architecture and Pipeline

8 Distributed Training

The Kistmat AI system employs distributed training to accelerate the learning process and handle large datasets efficiently.

Distributed Training

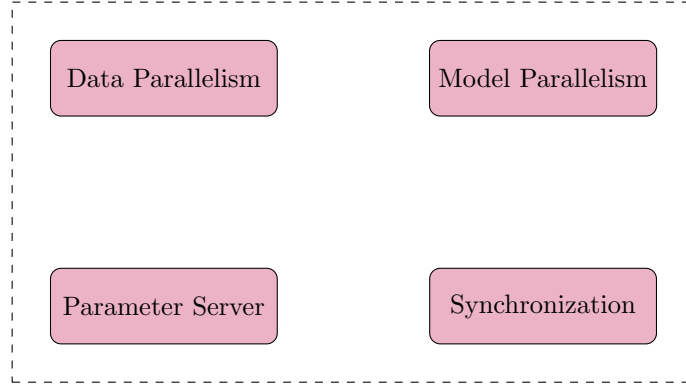


Figure 5: Distributed Training Approach

9 Problem Analysis and Evaluation

The Kistmat AI system analyzes problems through a multi-step process and evaluates results using various metrics.

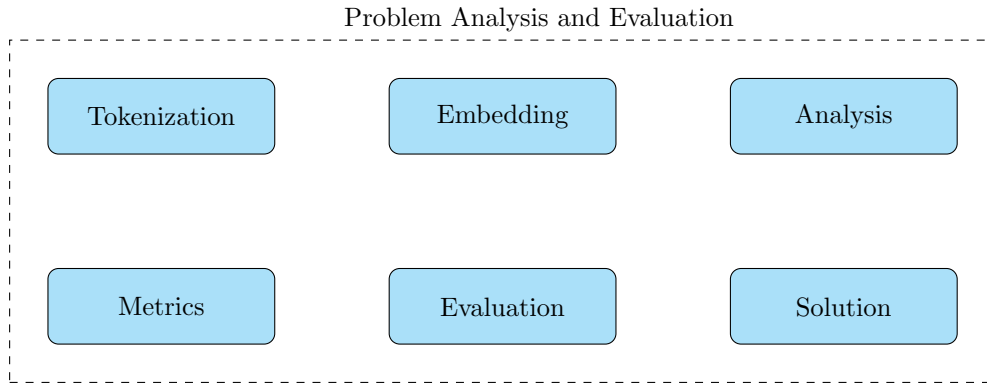


Figure 6: Problem Analysis and Evaluation Process

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

Kistmat AI Comprehensive System

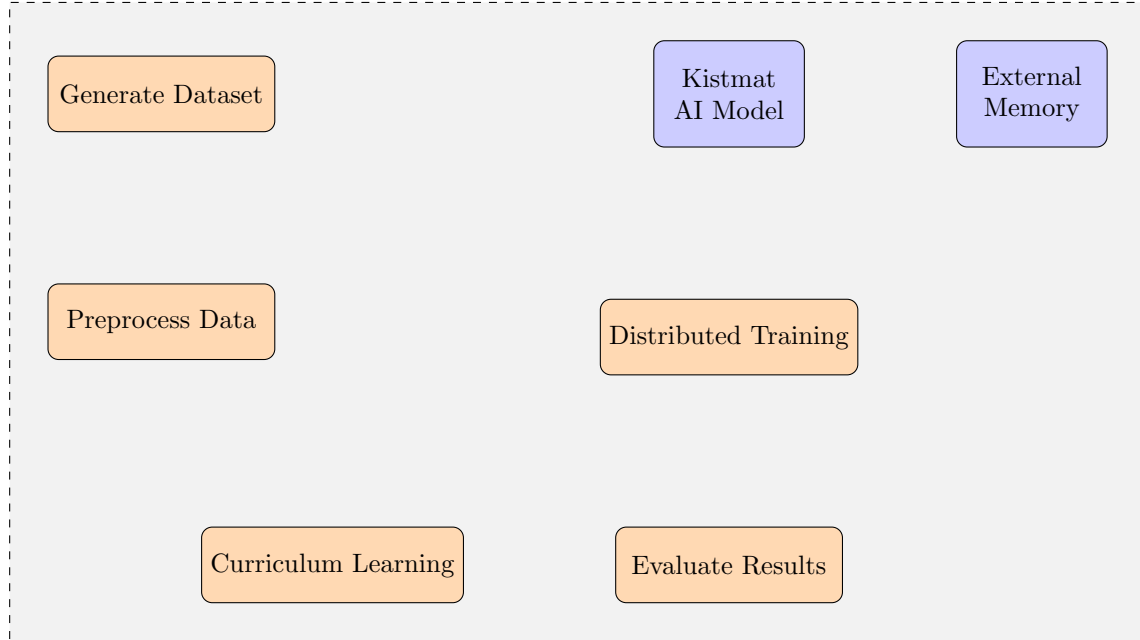


Figure 7: Comprehensive Kistmat AI System Overview

12 Results and Evaluation

12.1 Learning Curves

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:

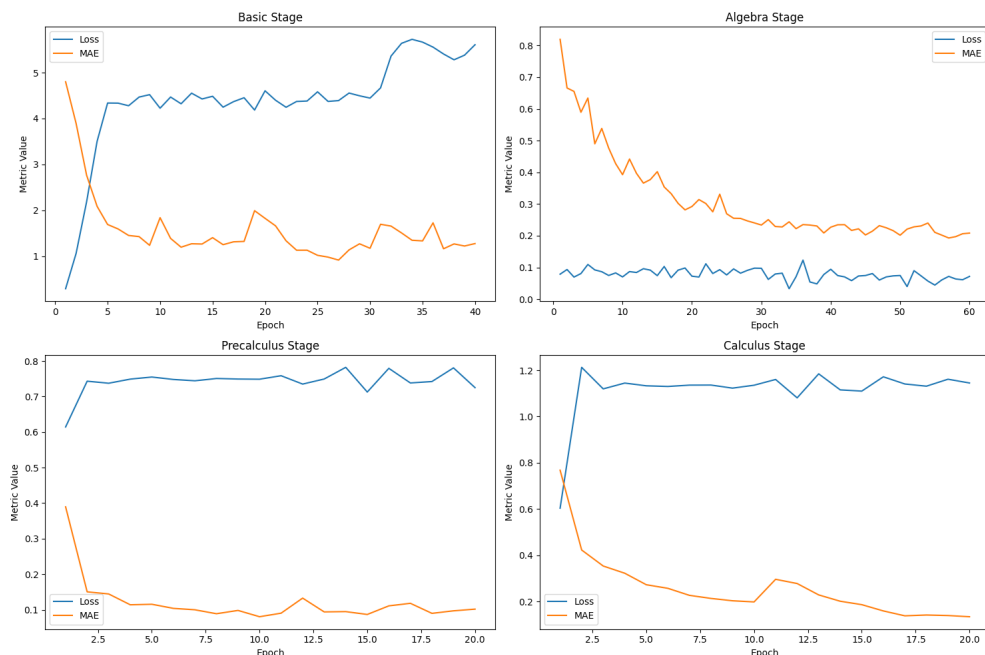


Figure 8: Learning Curves for Different Stages

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:

- 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

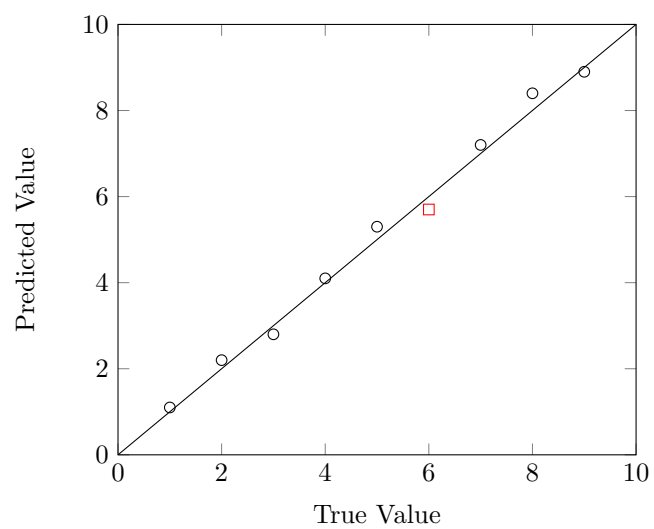


Figure 9: Sample Predictions vs True Values