Arabic Text Auto-Correction System

This project implements an Arabic text auto-correction system using a fine-tuned sequence-to-sequence transformer model. It detects and corrects common spelling mistakes and typographical errors in Arabic text. The model is based on the ARAT5 architecture and fine-tuned on a custom dataset with synthetically generated errors.

The system normalizes Arabic text by removing diacritics and standardizing characters, processes text at the word level for precise corrections, and includes a user-friendly GUI for interactive use.

by Reem Ashraf

Model Architecture and Parameters

Architecture

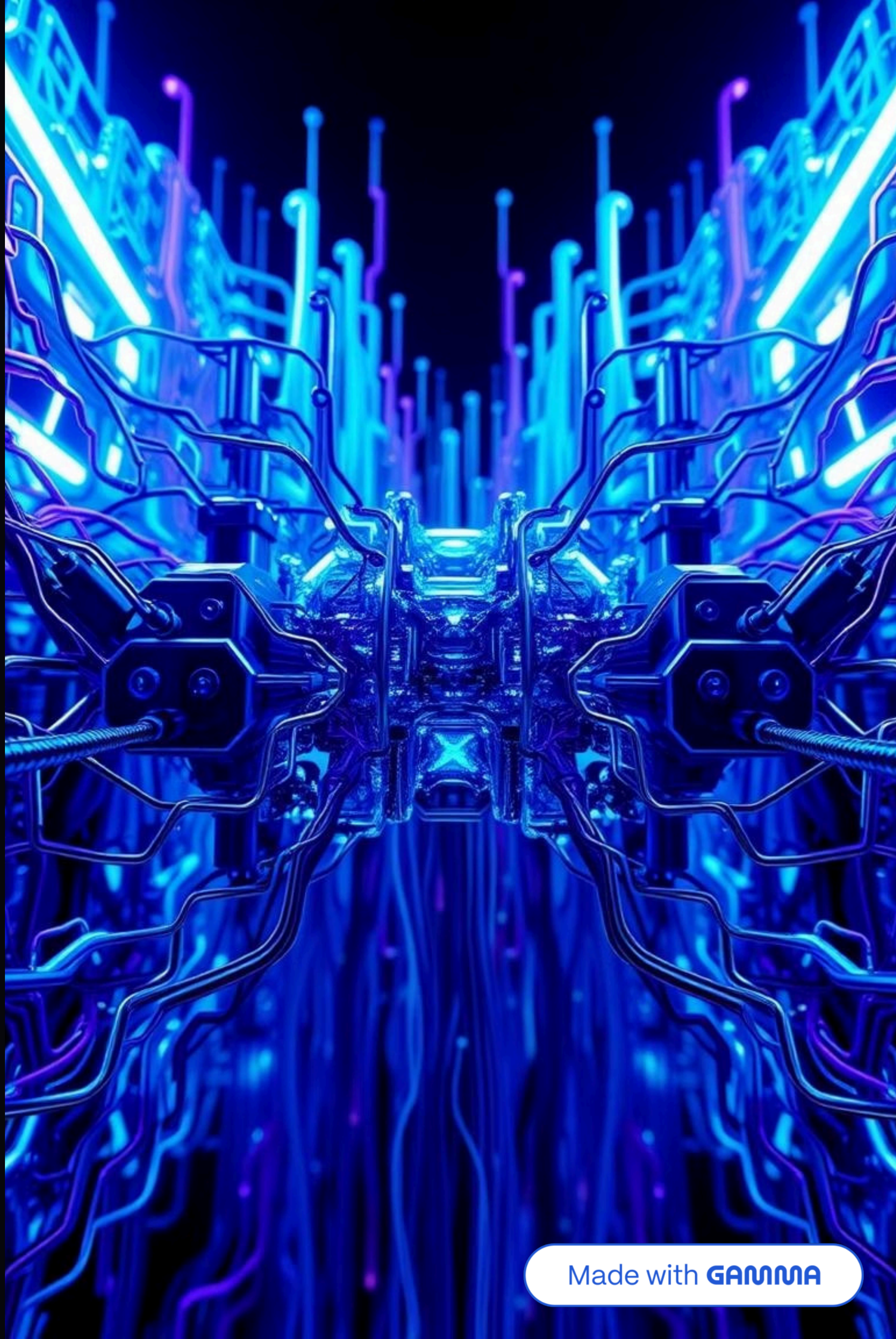
Based on ARAT5, a T5 variant for Arabic, with encoder-decoder structure and self-attention. Uses SentencePiece tokenizer and handles sequences up to 128 tokens.

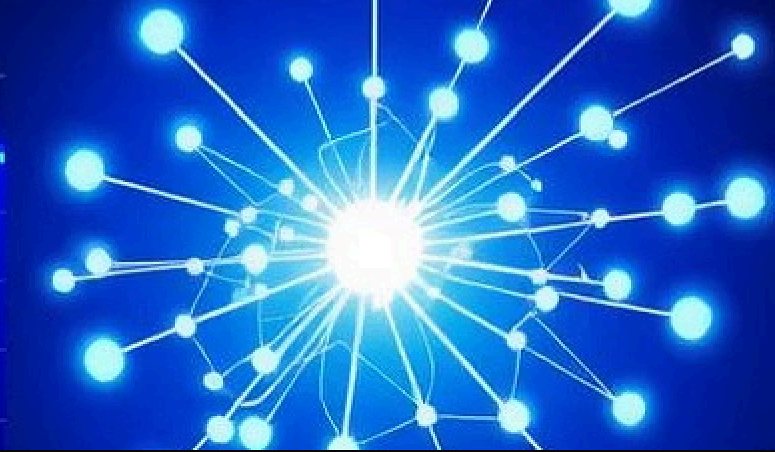
Key Parameters

- Hidden Size: 768
- Attention Heads: 12
- Layers: 12
- Vocabulary Size: 32,000

Training Setup

Learning rate 2e-5, batch size 16, optimizer Adafactor, cosine scheduler, FP16 training enabled.





Training Process and Monitoring

Fine-Tuning Configuration

Trained for 2 epochs with 200 warmup steps, weight decay 0.01, gradient accumulation of 1 step, and max gradient norm 1.0.

Monitoring Metrics

Training and evaluation loss tracked every epoch to ensure steady learning and avoid overfitting.

Loss Curve

Loss steadily decreased from initial 3.896 to final 1.556 in training, with evaluation loss at 1.328.

Dataset Overview

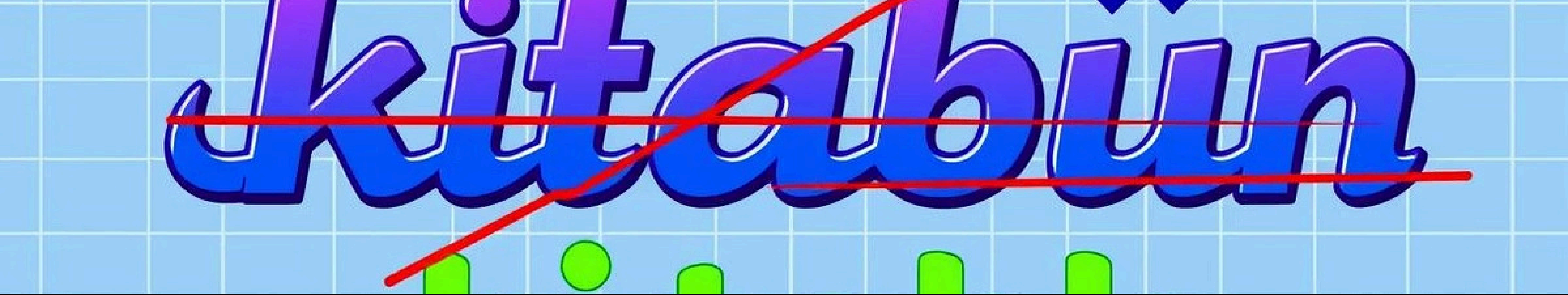
Dataset Source

Uses "zeydferhat/arabic_functional_text_dimensions" from Hugging Face with 2,380 training samples.

- Columns: index, Text, label (not used for correction)
- Vocabulary size after normalization: 72,804 unique words
- Text domains: Various functional Arabic text types

Data Cleaning

Steps include dropping NA values, removing non-string entries, filtering Arabic text only, and removing duplicate pairs.



Synthetic Data Generation

1. **Insertion**

   Randomly inserts Arabic letters into words to simulate errors.

2. **Deletion**

   Randomly removes letters from words to mimic typos.

3. **Replacement**

   Replaces letters with similar-looking Arabic letters to create confusion.

4. **Switching**

   Swaps adjacent letters to replicate common typing mistakes.

Common Error Patterns Targeted

1

Normalization Variations

Converts أ, إ, آ to ا for consistency.

2

Letter Confusions

Handles common confusions like ة to ه and ي to ى.

3

Typographical Errors

Focuses on frequent typing mistakes in Arabic script.

Preprocessing Steps

Character Normalization

Standardizes characters such as إ, أ, آ to ا; ى to ي; ؤ, ئ to ء; ة to ه; and گ to ك.

Diacritic and Punctuation Removal

Removes all Arabic diacritics (Tashkeel) and both Arabic and English punctuation marks.

Filtering

Excludes non-Arabic characters and numbers to ensure clean input for training.

Evaluation Metrics and Results

Training Metrics

- Initial Training Loss: 3.896
- Final Training Loss: 1.556
- Evaluation Loss: 1.328

Evaluation Metrics

BLEU score used for character-level correction quality. Accuracy on a 100-sample test set was 5%, indicating possible evaluation issues.

Model Limitations

Vocabulary Coverage

Limited to words in the original dataset vocabulary; struggles with out-of-vocabulary words.

Error Types

Primarily corrects spelling errors at the word level; does not handle grammar or semantics.

Context Understanding

Operates on individual words without broader sentence context; cannot handle context-dependent corrections.

Performance

Current evaluation metrics suggest room for improvement; may need more data or architecture changes.

Text Length

Limited to 128 tokens; longer texts require chunking.

Summary and Next Steps

Project Summary

Developed an Arabic text auto-correction system using a fine-tuned ARAT5 transformer model with synthetic error data and normalization preprocessing.

Key Challenges

Handling vocabulary limitations, context understanding, and improving evaluation accuracy remain challenges.

Future Directions

Enhance model with broader context, expand training data, and explore architecture improvements for better correction quality.