Design and Implementation of a Chinese Semantic Matching Model Based on Siamese LSTM

Lin Liu[1], Wanyue Liu[2], Xiaoying Liu[1]
[1]School of Artificial Intelligence and Big Data, Henan University of Technology, Zhengzhou, China
[2]iFLYTEK Co., Ltd., Hefei, China

Abstract: To address the limited semantic discrimination of term-frequency-based methods and the high deployment cost of large pretrained models, this paper develops a Chinese semantic matching model that combines a Siamese architecture, bidirectional long short-term memory networks, and attention pooling. Two weight-sharing encoders project a sentence pair into the same semantic space. Contextual representations are extracted by a BiLSTM, salient tokens are aggregated by an attention mechanism, and the two sentence vectors are compared through concatenation, absolute difference, and element-wise product before binary classification. Experiments on the LCQMC corpus show that the proposed model achieves 82.48% accuracy, 76.84% precision, 92.98% recall, 84.14% F1-score, and 92.79% AUC. Compared with the TF-IDF baseline, accuracy and F1-score improve by 26.38 and 15.78 percentage points, respectively. With about 4.8 million parameters, the model uses only approximately one twenty-third of the parameters of BERT-base. The model is further integrated into the knowledge retrieval pipeline of the Zhixue learning platform to rank candidate passages and return Top-5 results, demonstrating a favorable trade-off among accuracy, inference efficiency, and deployability.

Keywords: Chinese Text Matching; Siamese Model; BiLSTM; Attention; Knowledge Retrieval

1. Introduction

Semantic matching checks whether two pieces of text have the same or a close meaning. It is used in search, question answering, dialogue systems, duplicate-question checks, and knowledge-base search. Chinese questions are often short. They may leave out words, use everyday language, restate the same idea, or change word order. For this reason, word overlap alone is not enough. For example, “How can I improve my grades?” and “How can I improve learning efficiency?” use different words but have related meanings. But “The laptop starts slowly” and “The laptop shuts down slowly” share many words, while the actions and intents are different.

TF-IDF builds sparse text vectors from term frequency and inverse document frequency [1]. The method is fast and easy to use, but it does not model context. BERT and other pretrained language models give better semantic vectors [2]. But they have many parameters and need much more computing, which makes them costly to run on common GPUs and in real-time search systems. Finding a balance between accuracy and resource consumption therefore retains substantial engineering value.

A Siamese network processes a sentence pair with two encoders that share parameters, allowing both inputs to be compared in a unified semantic space [3]. Using the large-scale LCQMC Chinese question-matching corpus [4], this study constructs a Siamese BiLSTM-attention model. A bidirectional LSTM first learns contextual dependencies, attention pooling then produces sentence-level representations, and concatenation, absolute difference, and element-wise product features are finally combined for classification. The main work comprises: (1) a complete pipeline from Chinese tokenization, vocabulary construction, and fixed-length encoding to mini-batch training; (2) a weight-sharing BiLSTM-attention encoder and a multi-feature fusion classifier; (3) evaluation using accuracy, F1, AUC, a confusion matrix, and sentence-length analyses; and (4) integration into the Zhixue knowledge-base retrieval workflow to assess the practicality of lightweight semantic matching.

Chinese semantic matching is a binary sentence-pair classification task, but its difficulty differs

from that of ordinary text classification. Ordinary classification only assigns a category to one sentence, whereas semantic matching must model both the internal semantics of two sentences and the interaction between them. Concatenating the sentences before classification can make the model sensitive to their order and length. Encoding them separately without shared parameters can instead produce inconsistent representation spaces. A Siamese structure uses the same encoder for both inputs and is therefore naturally suited to metric learning and pairwise matching.

BiLSTM is used because words in a Chinese question may depend on what comes before and after them. In "How do I turn off automatic updates on an iPhone?", the model needs both directions to link "turn off" with "automatic updates." Attention pooling uses information from all hidden states instead of only the last one. It can also give more weight to words that matter for the task. The feature fusion layer gives the classifier the vector difference and element-by-element match directly, so the classifier does not have to learn the whole comparison on its own.

Section 2 reviews statistical methods, neural matching models, and pretrained models. Section 3 describes the LCQMC data and preprocessing. Section 4 explains the model and training loss. Section 5 reports the experiments and results. Section 6 shows how the model is used in Zhixue. Section 7 discusses its uses, limits, and possible changes. Section 8 gives the conclusion.

2. Related work

2.1 Basic Text Similarity Methods

Early text-matching systems used bag-of-words, TF-IDF, edit distance, or hand-written rules. These methods count shared terms, so they do not handle multiple meanings, negation, word order, or different wording well. Deep learning then made it possible to model text as a sequence. The LSTM from Hochreiter and Schmidhuber uses gates to reduce the vanishing-gradient problem in long sequences [5]. The attention method from Bahdanau et al. lets a network give a different weight to each position [6].

2.2 Neural-Network-Based Matching Methods

Mueller and Thyagarajan applied a Siamese recurrent architecture to sentence similarity learning and obtained comparable sentence vectors through shared encoding parameters [3]. Subsequent Chinese studies introduced bidirectional encoders, attention, combined word-character representations, and feature interaction to improve semantic matching [7,8]. BERT derives contextual representations with a bidirectional Transformer [2], and Sentence-BERT uses a Siamese structure to improve sentence-vector retrieval efficiency. These approaches offer strong accuracy, but their parameter counts and inference latency may also increase. The present study uses Siamese BiLSTM as the main architecture and adds attention pooling and multidimensional interaction features to balance performance and efficiency in a resource-constrained educational retrieval scenario.

2.3 Pretrained-Model-Based Matching Methods

Pretrained language models learn general linguistic knowledge from large-scale unlabeled corpora. BERT uses multiple bidirectional Transformer layers and performs strongly on question matching and natural-language inference [2]. A common implementation feeds sentence pairs in the form [CLS] sentence A [SEP] sentence B [SEP] and classifies them using the representation at [CLS]. Such cross-encoding enables full interaction at every layer, but knowledge-base retrieval must execute the whole Transformer for each query–candidate pair, producing substantial latency when the candidate set is large.

Sentence-BERT converts BERT into a dual-tower architecture so that candidate passages can be encoded offline. Nevertheless, BERT-base still contains approximately 110 million parameters and places significant demands on GPU memory, system memory, and inference hardware. Recent embedding studies have expanded multilingual and Chinese benchmarks, training resources, and large-language-model-based representation learning [9-12]. They demonstrate rapid progress in general text embeddings, while also reinforcing the need to consider model scale and deployment cost. In medium or small course knowledge bases, the model must support semantic discrimination while respecting training conditions, service concurrency, and maintenance cost. This study

does not try to match the best results of pretrained models. It asks a narrower question: can Siamese LSTM beat TF-IDF with a much smaller model that runs on a regular laptop GPU?

2.4 Where the Proposed Method Fits

Term overlap misses many Chinese rewrites, but large pretrained models are hard to run on limited hardware. Our method sits between these two options. Word embeddings and BiLSTM model the context, attention gives more weight to useful words, and the two towers share one encoder so candidate vectors can be reused. The model still learns semantic features from data, but it does not run a large cross-encoder for every pair. It can replace a basic search module or work as a small recall model in a larger system.

3. Dataset and Preprocessing

3.1 LCQMC Dataset

Researchers at the Shenzhen Graduate School of Harbin Institute of Technology and Alibaba's DAMO Academy built LCQMC [4]. It is a common dataset for Chinese semantic matching. Each record has two Chinese questions and a label. Label 1 means that their meanings are the same or very close, while label 0 means that they are different. After cleaning, removing duplicates, and manual labeling, the official data has 238,766 training pairs, 8,802 validation pairs, and 12,500 test pairs. The total is 260,068. The split is summarized in Table 1.

Table 1. LCQMC Dataset Split

Subset	Number of examples	Purpose
Training	238,766	Parameter learning
Validation	8,802	Tuning and early stopping
Test	12,500	Final evaluation
Total	260,068	—

At the raw character level, the average and median question lengths are approximately 15 and 12 characters, respectively, and more than 90% of the questions contain no more than 30 characters. After jieba tokenization, Figure 1 shows mean and median lengths of 10.90 and 10.00 tokens. The positive and negative classes account for 58.00% and 42.00% of the training set. Accuracy is therefore used as the primary metric, while precision, recall, F1, and AUC are also reported to avoid conclusions based on a single measure.

3.2 Text Preprocessing

Preprocessing consists of Chinese tokenization, vocabulary construction, index mapping, and fixed-length padding. Sentences are first segmented with jieba in accurate mode. Word frequencies are then calculated only on the training set, and the 30,000 most frequent items are retained with two special tokens, <PAD> and <UNK>. Each sentence is mapped to an integer sequence with a maximum length of 40; longer sequences are truncated and shorter ones are padded at the end. This length covers most questions while controlling LSTM computation. Because interrogatives, negative words, and modal particles may be important for matching, stop words are retained. During training, DataLoader supplies mini-batches and shuffles the training set.

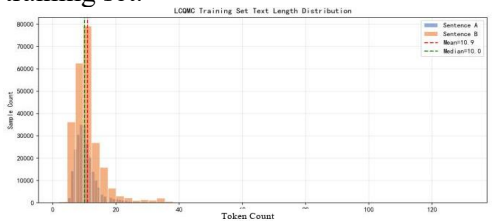

Figure 1. Token-Length Distribution of the LCQMC Training Set

3.3 Class Distribution and Data Quality

As shown in Figure 2, the LCQMC training set contains 138,574 similar pairs (58.00%) and 100,192 dissimilar pairs (42.00%). This moderate imbalance does not require class weighting or oversampling in the present experiments, although it is not an exact 1:1 ratio. Multiple annotators labeled the data, and disagreements were adjudicated to reduce individual subjectivity. Nevertheless, semantic boundaries in Chinese remain partly ambiguous. For example, “How can I learn English well?” and “How can I improve English listening?” are topically related but do not express exactly the same intent, so such examples may still lead to disagreement between the model and human judgment. Table 2 presents representative linguistic phenomena in Chinese semantic matching.

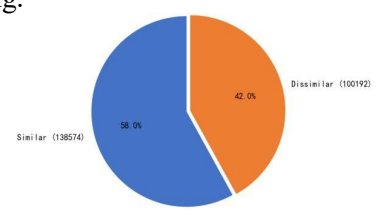

Figure 2. Class Distribution of the LCQMC Training Set

Table 2. Typical Linguistic Phenomena in Chinese Semantic Matching

Phenomenon	Sentence A	Sentence B	Analysis
Synonymous rewrite	How can grades be improved?	How can learning outcomes be improved?	Large literal difference but close meaning
Local word overlap	The computer starts slowly.	The computer shuts down slowly.	One changed action yields a different intent
Negation	Can the password be changed?	Why can the password not be changed?	Negation changes polarity
Entity replacement	What is the weather in Beijing?	What is the weather in Shanghai?	Same structure but different queried entity
Word-order change	How do students select courses?	How should courses be selected?	Order changes while the core intent remains close

Synonymous rewrites require the model to move beyond surface frequency; local overlap and entity replacement require attention to the small number of words that determine meaning; negation requires the sequence encoder to model polarity correctly; and word-order changes require sentence vectors to remain partly invariant to expression form. Together, these phenomena show that semantic matching must exploit both contextual information and cross-sentence interaction features.

3.4 Preprocessing Flow and Parameter Selection

The complete preprocessing flow is shown in Figure 3, and its key settings are summarized in Table 3. After the original TSV files are loaded, field completeness is checked and records with missing sentences or invalid labels are removed. The two sentences are then tokenized separately. The vocabulary is constructed only from the training set, and unseen words in the validation and test sets are mapped to <UNK> to prevent test-information leakage. An attention mask is generated after indexing to distinguish real tokens from <PAD>. The final Dataset returns the indices of sentences A and B, their valid lengths or masks, and the label; DataLoader performs batching and training-set shuffling.

Vocabulary size and maximum length strongly affect both model size and performance. If the vocabulary is too small, many low-frequency entities become <UNK>, weakening fine-grained discrimination. If it is too large, the embedding matrix and overfitting risk both increase. The maximum length of 40 follows the corpus-length statistics, covers most questions, and keeps each batch within the 4 GB GPU-memory budget. Tail truncation is used for sentences longer than 40 tokens, which may remove important information near the end and contributes to the lower performance observed on longer inputs.

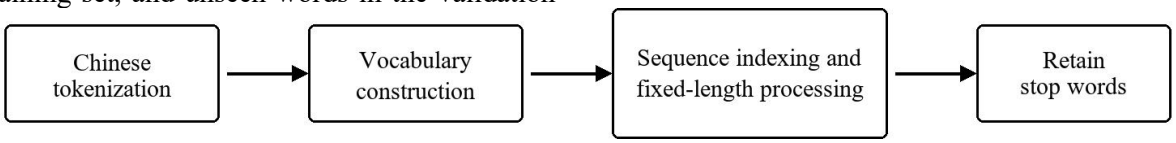

Figure 3. Chinese Text Preprocessing Flow

Table 3. Key Data-Preprocessing Settings

Stage	Setting	Purpose
File reading	UTF-8, tab separated	Preserve Chinese characters and field boundaries
Chinese tokenization	jieba accurate mode	Obtain word-level semantic units
Vocabulary size	30,000	Balance coverage, parameters, and OOV rate
Special tokens	<PAD>, <UNK>	Support batching and unseen words
Maximum length	40	Cover most questions and control computation
Stop words	Retained	Preserve negation, interrogatives, and particles
Data shuffling	Enabled for training	Reduce ordering effects on optimization

3.5 Data loading and Mini-Batch Organization

A custom PyTorch Dataset encapsulates preprocessed sentence pairs. Each example contains three logical fields, sentence_a, sentence_b, and label. Vocabulary lookup converts the two sentences into fixed-length LongTensor objects, while the label becomes a floating-point Tensor for BCEWithLogitsLoss. Preprocessing and caching during Dataset initialization avoid repeated tokenization and

lookup in every epoch, at the cost of additional main memory. For LCQMC's roughly 260,000 examples, this design substantially reduces CPU waiting time in the training loop.

During training, DataLoader shuffles the data at the start of each epoch. This reduces the effect of the original order and nearby examples with the same label. The validation and test sets are not shuffled, so their evaluation stays consistent. At batch size 128, each batch has two sequences, and each sequence has 40 tokens. After embedding, each branch produces a tensor of shape 128 × 40 × 256 before entering the shared BiLSTM. The identical configuration for both sentences allows efficient batched matrix computation on the GPU.

To prevent data leakage, vocabulary construction, threshold selection, and early-stopping decisions use only the training or validation data. The test set is evaluated once after the model and threshold have been fixed. If term frequencies were calculated on the complete corpus, words unique to the test set would enter the vocabulary in advance and make results optimistic. Similarly, adjusting the classification threshold using test-set F1 would violate the test set's role as unseen data. The workflow therefore assigns distinct responsibilities to the three subsets.

For reproducible deployment, the fixed word-to-index mapping must be saved with the model. Even if a rebuilt vocabulary contains the same words, changes in ordering or special-token indices make the trained embedding matrix incompatible. A model checkpoint should therefore be released together with vocab.json, maximum length, tokenization settings, and label definitions. The platform inference service verifies configuration versions at load time to avoid silent accuracy degradation caused by inconsistent preprocessing.

4. Siamese BiLSTM-Attention Model

4.1 Overall Architecture

The model consists of a weight-sharing dual-tower encoder, an interaction-feature fusion layer, and a multilayer perceptron classifier. Sentences A and B pass through the same embedding, BiLSTM, and attention-pooling modules to produce fixed-length vectors u and v. Shared parameters ensure consistent feature extraction and reduce parameter count. The original representations, difference representation, and co-occurrence representation are then combined and passed to the classifier, which outputs the semantic-similarity probability. The architecture is shown in Figure 4.

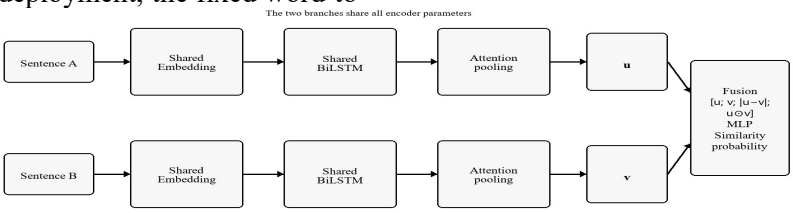

Figure 4. Siamese BiLSTM-Attention Model Architecture

4.2 Bidirectional LSTM Encoding

Input word indices are first mapped to 256-dimensional dense vectors. The encoder uses a two-layer bidirectional LSTM with a hidden dimension of 128 in each direction. Concatenating the forward and backward states yields a 256-dimensional contextual representation at every time step. The LSTM controls information flow through forget, input, and output gates [5]. Its principal computations include:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \quad (1)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad (2)$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (3)$$

Here, x_t is the word vector at time step t, while h_t and C_t denote the hidden state and cell state. The symbol σ denotes the sigmoid function and ⊙ denotes element-wise multiplication. Bidirectionality uses both left and right context and reduces information loss caused by one-directional encoding. Inter-layer dropout is set to 0.30 to limit overfitting.

4.3 Attention Pooling

Not every word contributes equally to the matching decision. Additive attention is used to aggregate the BiLSTM outputs H = {h₁, h₂, …, h_T} [6]. The attention score and normalized weight are:

$$e_t = v_a^T \cdot \tanh(W_a h_t + b_a) \quad (4)$$

$$\alpha_t = \frac{\exp(e_t)}{\sum_j \exp(e_j)} \quad (5)$$

$$s = \sum_t \alpha_t h_t \quad (6)$$

W_a, b_a, and v_a are learnable parameters. The mask prevents <PAD> positions from participating in weight normalization, and the resulting sentence vector s has 256 dimensions. This process increases the influence of interrogatives, entity words, negation, and other salient information on the final decision.

4.4 Feature Fusion and Classification

For the two sentence vectors u and v, the model retains the original semantics, difference pattern, and co-occurrence pattern and constructs a 1,024-dimensional fused feature:

$$\text{feature} = [u; v; |u-v|; u \odot v] \quad (7)$$

|u − v| describes dimension-wise differences, and u ⊙ v describes dimension-wise matching strength. The fused vector then goes through Linear, ReLU, Dropout, and Linear layers, and the last layer returns one logit. BCEWithLogitsLoss combines the sigmoid function with binary cross-entropy for improved numerical stability.

4.5 Word Embeddings and Masking

The embedding matrix E ∈ R^(V×d) maps discrete indices to a continuous space, where V = 30,000 and d = 256. It is updated jointly with the remaining parameters so that words with similar functions in the matching task acquire nearby representations. Compared with fixed pretrained word vectors, end-to-end training from random initialization adapts directly to the LCQMC question distribution, but it is less effective for rare and out-of-domain terms. Future work can load Chinese pretrained vectors or combine character- and word-level encoding to reduce the unknown-word problem.

Mini-batches require sentences of different lengths to be padded to the same size, so attention must explicitly mask <PAD>. A Boolean mask is constructed from the original lengths, and padding scores are set to a very small value before Softmax, making their normalized weights approach zero. Without this step, large numbers of padding positions may contaminate the sentence vector, particularly for short questions. Within the LSTM, pack_padded_sequence can also skip invalid time steps and reduce unnecessary computation.

4.6 Loss Function and Optimization

Let x_i be the logit for sentence pair i, let p_i = σ(x_i), and let y_i ∈ {0,1} be its true label. The binary cross-entropy is:

$$L = -\frac{1}{N}\sum_i [y_i \log p_i + (1-y_i)\log(1-p_i)] \quad (8)$$

BCEWithLogitsLoss implements a stable form that combines sigmoid and logarithmic computation and avoids numerical overflow when a probability approaches 0 or 1. Because the class distribution is moderately balanced, no positive-class weight is used. We use Adam [13] and set weight decay to 1 × 10^−4 to keep the weights from growing too large.

Training starts with a learning rate of 0.001. ReduceLROnPlateau checks the validation result. If it does not improve for several epochs, the scheduler cuts the learning rate in half. This gives larger updates early in training and smaller updates later. We save the model with the best validation result instead of the model from the last epoch. This limits the effect of overfitting or small changes near the end of training.

4.7 Model Size and Computing Cost

The embedding matrix, BiLSTM, and MLP contain most of the parameters. The embedding matrix grows with the vocabulary and holds the largest share. Both towers use the same encoder, so processing two sentences does not require two sets of encoder weights. BiLSTM running time grows roughly with sequence length. Setting max_len = 40 keeps the cost low for short questions. Knowledge passages can be encoded and saved before a query arrives. The online step then needs only query encoding and vector comparison. Table 4 lists the main design and deployment differences among the four methods.

Table 4. Structural and Deployment Characteristics of Four Schemes

Scheme	Semantic capability	Parameter scale	Candidate reuse	Suitable scenario
TF-IDF	Weak	No trainable parameters	Supported	Fast literal retrieval and fallback
Siamese LSTM	Strong	≈4.8M	Supported	Resource-constrained semantic recall
BERT cross-encoder	Very strong	≈110M	Inconvenient	High-accuracy reranking of few candidates
Dual-tower BERT	Very strong	≈110M	Supported	Large-scale retrieval with sufficient compute

4.8 Forward Pass and Prediction

One forward pass has six steps. First, the shared embedding matrix maps the two index sequences to vectors. Second, BiLSTM builds a context vector for each position in both directions. Third, attention uses the padding mask and creates one vector for each sentence. Fourth, the model forms [u; v; |u − v|; u ⊙ v]. Fifth, the MLP returns one logit. Sixth, sigmoid changes the logit to a similarity score between 0 and 1 for evaluation or prediction. During training, the loss function takes the logit directly, so a separate sigmoid step is not used.

The shared encoder uses the same weights for both branches, but the branches do not share hidden states. Sentences A and B go through separate forward calculations. The errors from both sides update the same embedding, BiLSTM, and attention weights during backpropagation. The fusion vector keeps u and v in order, so swapping the sentences can give a small change in the output. A strictly symmetric version can use only |u − v|, u ⊙ v, and u + v. Another option is to swap sentence order at random during training.

The default threshold is 0.50. A pair is marked as similar when $p = \sigma(x) \geq 0.50$. This works for this fairly balanced dataset because the two types of error have similar costs. In a real search system, the threshold should match the task. Lowering it finds more relevant content but may add weak results. Raising it gives fewer and more precise results. The model score can also be used with the TF-IDF score, course-section rules, or click data.

The model is saved as state_dict rather than as a complete Python object to reduce compatibility problems caused by code changes. After loading, model.eval() disables dropout, and inference runs under torch.no_grad() or inference_mode. One query can be paired with multiple candidates in a batch to reduce repeated GPU-kernel startup. For CPU deployment, dynamic quantization of linear and LSTM layers can trade a small accuracy loss for lower memory use and faster inference.

5. Experiments and Result Analysis

5.1 Experimental Environment and Parameters

Experiments were conducted under Windows 11. The main software and hardware configuration is given in Table 5, and the hyperparameters are listed in Table 6. Because the RTX 3050 Ti Laptop GPU has 4 GB of memory, batch size was set to 128; a batch size of 256 caused an out-of-memory error. Adam and ReduceLROnPlateau were used, and the learning rate was multiplied by 0.50 when the validation metric stagnated.

Table 5. Experimental Environment

Item	Configuration	Description
Operating system	Windows 11 64-bit	Development platform
GPU	RTX 3050 Ti Laptop	4 GB memory
CUDA	12.1	GPU computing
Python	3.10	Programming language
PyTorch	2.5.1+cu121	Deep-learning framework
jieba	0.42+	Chinese tokenization
scikit-learn	1.6	Baseline and evaluation

Table 6. Main Hyperparameters

Parameter	Value	Parameter	Value
vocab_size	30,000	embed_dim	256
hidden_dim	128	num_layers	2
bidirectional	True	attention_dim	64
dropout	0.30	batch_size	128
learning_rate	0.001	weight_decay	1 × 10^−4
epochs	20	optimizer	Adam
max_len	40	loss	BCEWithLogitsLoss

5.2 Comparison Methods and Evaluation Metrics

A TF-IDF baseline is used to verify the effectiveness of deep semantic encoding. It extracts local Chinese patterns with character-level n-grams, computes cosine similarity between sentence vectors, and selects the classification threshold on the validation set. A BERT-base scheme is also included to discuss the performance ceiling and parameter cost. Metrics include accuracy, precision, recall, F1, and AUC. F1 jointly reflects precision and recall, while AUC measures overall discrimination across thresholds.

5.2.1 Definitions of evaluation metrics

Let TP, TN, FP, and FN denote true positives, true negatives, false positives, and false negatives. Accuracy is the proportion of all correctly classified examples and is suitable for the moderately balanced classes in this study:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad \text{.....(9)}$$

Precision measures how many pairs predicted as similar are truly similar. It shows how reliable the returned results are:

$$\text{Precision} = \frac{TP}{TP+FP} \quad (10)$$

Recall measures how many truly similar pairs the model finds. It shows how often relevant content is missed:

$$\text{Recall} = \frac{TP}{TP+FN} \quad (11)$$

F1 combines precision P and recall R. It falls when either value is low, so it gives a clearer view than accuracy alone for this matching task:

$$F_1 = \frac{2PR}{P+R} \quad (12)$$

The ROC curve shows the true-positive rate and false-positive rate at different thresholds. AUC is the area under this curve. It can also be read as the chance that a random positive example gets a higher score than a random negative example. It does not depend on one threshold and is appropriate for ranking and discrimination. The precision–recall curve focuses on the positive class and directly shows the trade-off between result reliability and relevant-content coverage.

5.3 Training Process

The model was trained for at most 20 epochs. Figure 5 shows a continuing decrease in training loss and an overall increase in validation accuracy. Near epoch 14, the scheduler reduced the learning rate from 0.001 to 0.0005, after which validation performance continued to improve. This indicates that dynamic learning-rate adjustment supports finer parameter updates late in convergence. The curves contain no severe oscillation, suggesting that the current regularization strength and batch size maintain stable optimization.

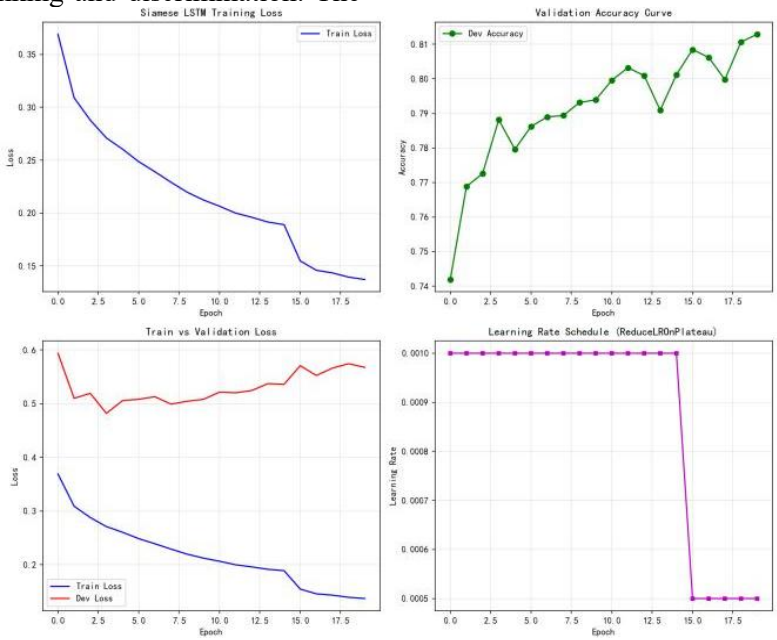

Figure 5. Training Curves of the Siamese LSTM

5.3.1 Training stability and model selection

A declining training loss does not necessarily imply continuously improving generalization, so validation accuracy and F1 are computed after every epoch. Early in training, the model rapidly learns frequent words and common sentence patterns. Later epochs mainly optimize boundary examples, so validation improvement slows. In Figure 5, training loss continues downward while validation loss fluctuates slightly. This is common in deep learning and shows that checkpoint selection should use validation accuracy rather than training loss alone.

After the learning rate is halved near epoch 14, the training curve enters another slow-decline phase and validation accuracy improves further, implying that 0.001 had become too large for fine updates. A fixed learning rate might oscillate near a local optimum, whereas reducing it too early would lengthen convergence. Triggering ReduceLROnPlateau from actual validation performance is more adaptable than decay at a fixed epoch.

Early stopping can end training when validation performance fails to improve for several epochs, avoiding wasted computation and reducing overfitting risk. The present experiment uses 20 epochs as the upper limit and saves the best validation checkpoint. Final test metrics come from that checkpoint rather than the final epoch. A stricter experiment would repeat training with multiple random seeds and report mean and standard deviation. Because of the computational limits of the course project, the

reported results reflect one complete run.

5.4 Overall Performance Comparison

The test results are reported in Table 7. All classification metrics are expressed as percentages with two decimal places, and AUC is also given as a percentage for unit consistency.

Table 7. Performance on the LCQMC Test Set

Model	Accuracy/%	Precision/%	Recall/%	F1/%	AUC/%	Parameters
TF-IDF	56.10	53.80	93.74	68.36	61.60	0
Siamese LSTM	82.48	76.84	92.98	84.14	92.79	≈4.8M
BERT-base	87.00	86.00	88.00	87.00	93.00	≈110M

Table 7 and Figure 6 show that TF-IDF reaches 93.74% recall but only 53.80% precision, indicating a tendency to classify pairs with shared characters as similar and thereby create many false positives. Siamese LSTM achieves 82.48% accuracy, 26.38 percentage points higher than TF-IDF. F1 increases from 68.36% to 84.14%, a gain of 15.78 percentage points. Recall decreases by only 0.76 percentage points, while precision increases by 23.04 percentage points, showing stronger discrimination of literally similar but semantically different pairs.

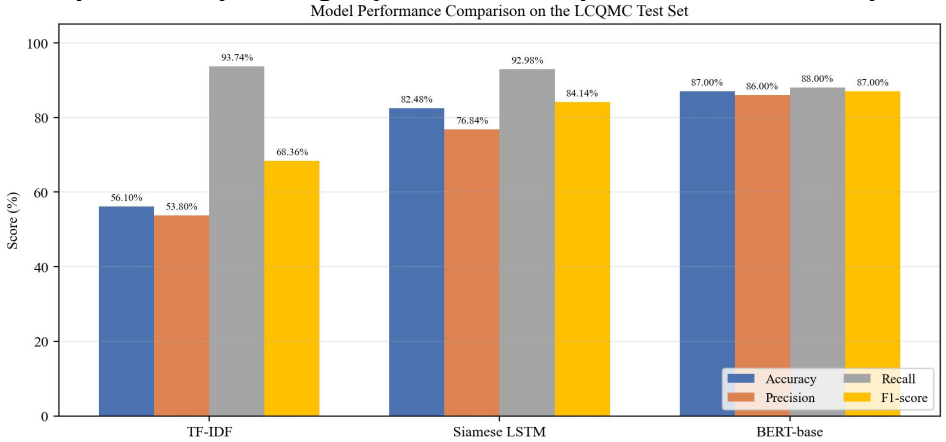

Figure 6. Comparison of Major Performance Metrics

BERT-base is 4.52 percentage points more accurate than the proposed model, but its approximately 110 million parameters are about 23 times the size of Siamese LSTM. Pretrained models remain advantageous when accuracy is the overriding requirement and computation is abundant. Siamese LSTM has lower storage and inference costs on ordinary servers, laptop GPUs, or educational platforms with high retrieval concurrency.

5.5 Error and Robustness Analysis

Figure 7 gives the Siamese LSTM confusion matrix for all 12,500 test examples. The model correctly predicts 10,310 and incorrectly predicts 2,190. Specifically, it contains 1,751 false positives and 439 false negatives. These counts reproduce 76.84% precision, 92.98% recall, and 82.48% accuracy and show that the model favors recall over precision rather than producing two equally sized error categories. Errors mainly arise from information-poor short questions, entity replacement, ambiguous negation scope, and colloquial expressions that are rare in training.

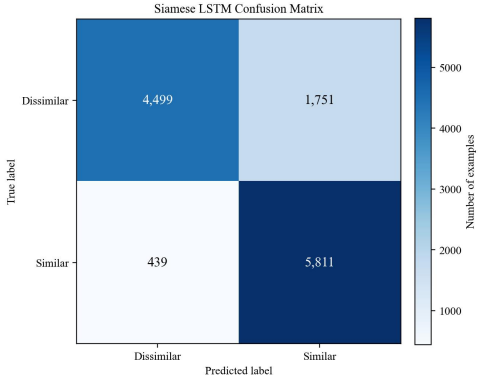

Figure 7. Siamese LSTM Confusion Matrix on the Test Set

Figure 8 uses the conventional normalized 0–1 scale for ROC axes and AUC labels. TF-IDF has an AUC of 0.6160 (61.60%), close to a random classifier. Siamese LSTM reaches 0.9279 (92.79%) and maintains a high true-positive rate with a relatively low false-positive rate over most thresholds. Precision–recall analysis further shows that the proposed model maintains approximately 0.80 precision when recall is about 0.90. Confidence is polarized, with most probabilities near 0–0.1 or 0.9–1.0. Sentence-length analysis is reported separately in Section

5.9.

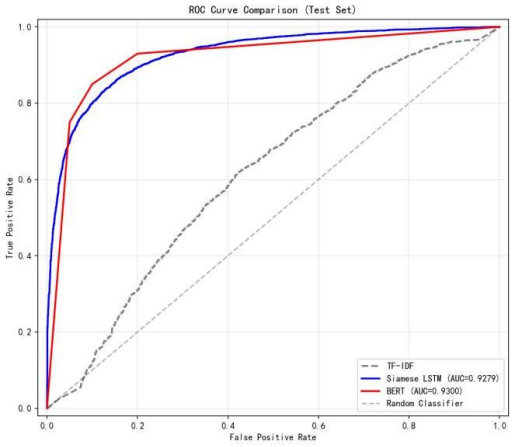

Figure 8. ROC Curves of the Compared Models

5.6 Precision–Recall Curve Analysis

ROC curves characterize overall ranking when the two classes are moderately balanced, whereas precision–recall curves more directly describe the positive class. In knowledge retrieval, missing relevant content reduces the chance that a user obtains an answer, but returning many irrelevant passages also harms the experience. Both recall and precision must therefore be considered. Figure 9 shows that the TF-IDF curve remains low and cannot achieve high precision and recall simultaneously by changing the threshold. Siamese LSTM maintains about 0.80 precision at approximately 0.90 recall, making it better suited to high-recall candidate filtering.

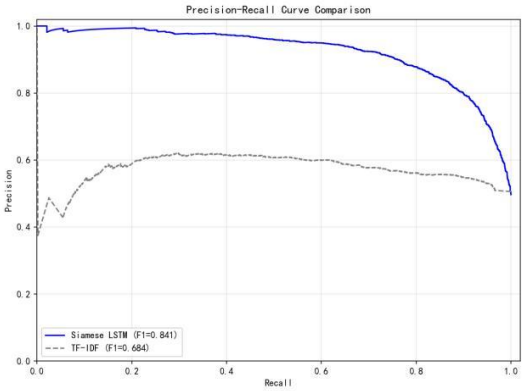

Figure 9. Precision–Recall Curves of Siamese LSTM and TF-IDF

5.7 Prediction-Confidence Distribution

Confidence distribution reveals whether model outputs cluster near the decision boundary. Ideally, positive probabilities are close to 1 and negative probabilities close to 0. Figure 10 visualizes a 5,000-example diagnostic subset, containing 4,092 correct and 908 incorrect predictions. Approximately 45% of the probabilities exceed 0.90, approximately 40% are below 0.10, and roughly 15% lie in the middle range. Thus, most examples receive decisive predictions. Examples between 0.40 and 0.60 often have ambiguous semantic boundaries, insufficient information, or complex entity relationships and are suitable for manual review and hard-example mining.

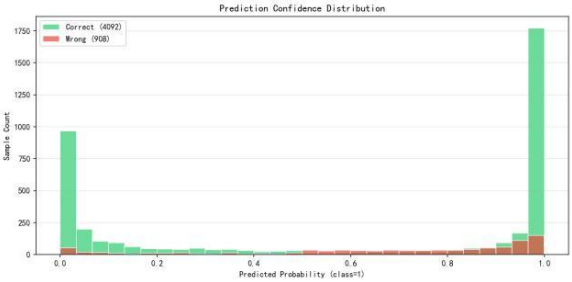

Figure 10. Prediction-Confidence Distribution of Siamese LSTM

5.8 TF-IDF Threshold Tuning

TF-IDF produces continuous cosine similarities and therefore requires a threshold for binary classification. Figure 11 shows the validation-set relationship between threshold and performance. The best validation threshold is 0.17, at which validation F1 is 75.43%. Applying the selected TF-IDF configuration to the held-out test set yields the 68.36% F1 reported in Table 7. A lower threshold creates more false positives by classifying weakly overlapping pairs as similar; a higher threshold increases false negatives among synonymous rewrites with limited literal overlap. The relatively narrow peak indicates that TF-IDF is sensitive to threshold selection.

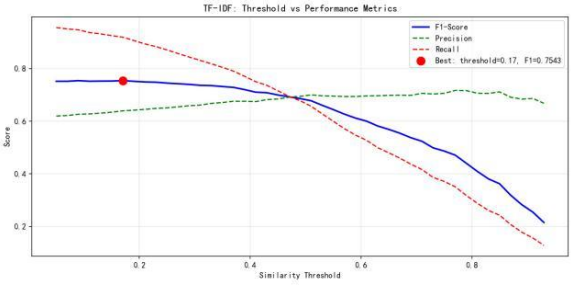

Figure 11. TF-IDF Similarity Threshold and Validation Performance

5.9 Performance Across Sentence Lengths

Test examples are grouped by the maximum tokenized length of the two sentences. Figure 12 reports percentage accuracy with two decimal places for the three populated diagnostic buckets. Siamese LSTM achieves 85.00%, 80.00%, and

79.00% accuracy in the 0–10, 10–20, and 20–30 buckets, respectively. TF-IDF obtains 56.00%, 54.00%, and 33.00%. The proposed model therefore remains clearly stronger across the reported ranges, although its accuracy declines as length increases. Longer questions contain more semantic material but can also introduce complex dependencies; sequences beyond max_len = 40 may additionally lose tail information through truncation.

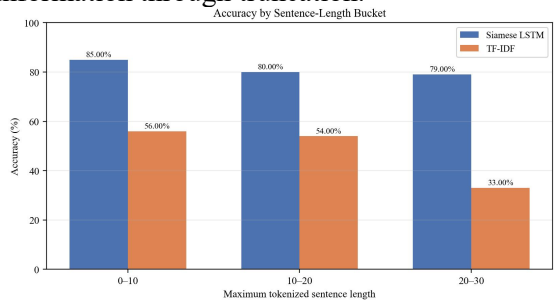

Figure 12. Accuracy across Tokenized Sentence-Length Buckets

5.10 Discussion of Results and Error Sources

The major improvement comes from the large precision gain: the model reduces misclassification of literally similar but semantically different pairs while preserving high recall. Attention helps emphasize entity words, action words, and negation, while multi-feature fusion makes sentence-vector differences easier for the classifier to detect. However, a dual-tower model interacts only after sentence-level encoding, so its fine-grained token alignment remains weaker than that of a BERT cross-encoder. Errors can still occur when only one key entity changes, when negation scope is complex, or when commonsense reasoning is required. Table 8 summarizes the principal error types and possible improvements.

Table 8. Main Error Types and Potential Improvements

Error type	Cause	Potential improvement
Unknown domain terms	General vocabulary has limited course terminology	Add domain data and pretrained word vectors
Key-entity replacement	A global sentence vector masks local entity differences	Introduce entity features or cross-attention
Complex negation scope	LSTM is weak at long-range polarity relations	Add hard examples and a lightweight Transformer
Long-text truncation	max_len = 40 removes tail semantics	Use dynamic length, segmented encoding, or hierarchical models
Ambiguous annotation boundary	Similarity and relatedness are partly subjective	Use multiple reviewers and soft-label training

5.11 Practical retrieval thresholds and Top-K selection

Offline binary classification usually converts a fixed threshold into 0 or 1, whereas knowledge-base retrieval ranks continuous scores and returns the top K candidates. The threshold removes results with very low scores, and K sets the number of results shown. A low threshold can let in unrelated passages. A high one can leave the user with no result when the knowledge base is small. Top-5 gives several useful choices without making the list too long.

The online threshold should be set with labels from the real platform, not copied from LCQMC. Student questions and passages checked by teachers can be used to measure precision, recall, and the no-result rate at several thresholds. The final value should match the cost of mistakes. Learning support needs high recall and can accept a few weak results. Exam knowledge-point search needs high precision because a wrong result may mislead the student.

The interface can also change its response based on the scores. If Top-1 is much higher than Top-2 and its score is high, the first result can be shown more clearly. If several results have close scores, the system can ask the student for a more specific question. If every score is too low, it should return “No reliable answer found” instead of showing a weak answer. These rules turn model scores into clearer choices for users.

6. Integration into the Zhixue platform

To test practical usability, Siamese LSTM is integrated into the knowledge-base retrieval module of the Zhixue intelligent learning platform. The platform uses a separated front end and back end. FastAPI and MySQL support the back end, while Vue 3 and Vite support the front end. After a teacher uploads course materials, a background task parses the documents, extracts text, divides it into overlapping chunks, and stores the result in the KnowledgeChunk table. When a student submits a natural-language question, the system sends the query and candidate chunks to the shared encoder, calculates matching scores, sorts them, and returns the Top-5 results.

Compared with a high-dimensional sparse TF-IDF index, the model recognizes synonymous

rewrites and word-order changes without maintaining a corpus-specific sparse matrix. Compared with BERT-class models, its approximately 4.8 million parameters are more suitable for real-time service on an educational platform. Candidate vectors can be cached so that only the user query is encoded online. If the model service becomes unavailable, TF-IDF remains as a fallback to preserve system continuity. Figure 13 presents the overall platform architecture.

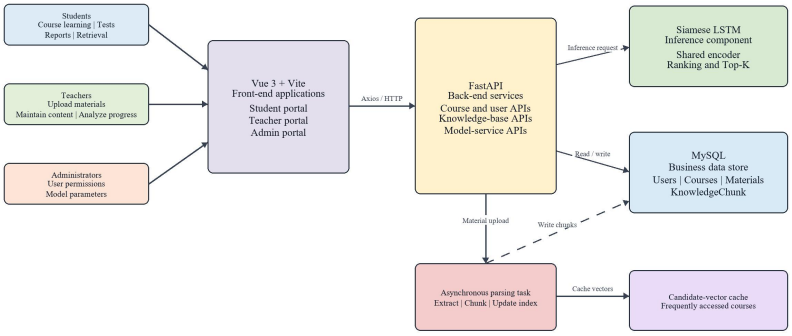

Figure 13. Overall Architecture of the Zhixue Platform

6.1 Overall platform architecture

Zhixue serves students, teachers, and administrators. The student side provides course learning, staged tests, learning reports, and knowledge retrieval. Teachers upload course materials, maintain content, and analyze learning progress. Administrators configure user permissions and model parameters. FastAPI exposes back-end interfaces, MySQL stores users, courses, materials, and knowledge chunks, and Axios connects the front end to the back end. The semantic matching model is embedded as an independent inference component and does not alter the existing business data structure.

6.2 Knowledge-Base indexing service

After a teacher uploads a PDF, Word, or text file, the system starts an asynchronous parsing task, extracts plain text, and divides it using a fixed window with overlap. Overlap reduces information breaks when a knowledge point lies at a chunk boundary. Each KnowledgeChunk records its course, source document, page or paragraph location, and text. With the Siamese scheme, passage vectors can be computed and saved during ingestion. When material changes, only affected chunks are re-encoded, avoiding full text encoding for every student query. The offline knowledge-base indexing workflow is shown in Figure 14.

6.3 Online Query and Result Ranking

After a student enters a question, the service applies the same tokenization, indexing, and length processing used during training and generates a query vector. Candidate chunks are retrieved from the current course, scored with the Siamese matching head or vector similarity, sorted in descending order, and returned as Top-5. The front end shows passage content, source material, and similarity so that students can locate course content quickly. Figure 15 shows the online semantic-retrieval and fallback workflow.

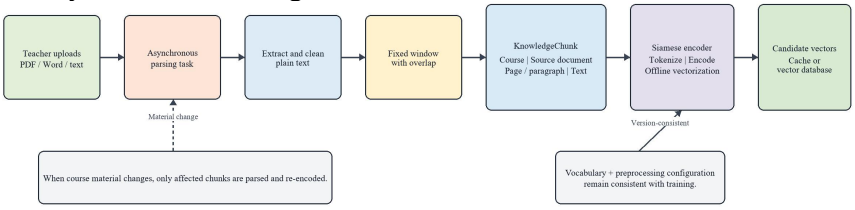

Figure 14. Offline Knowledge-Base Indexing Workflow

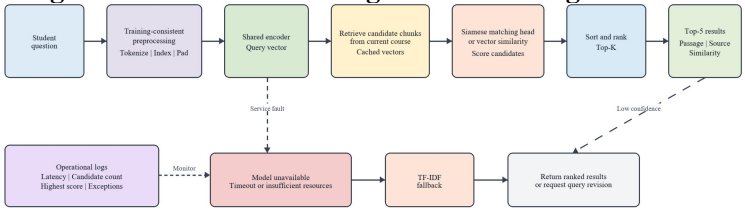

Figure 15. Online Semantic Retrieval and Fallback Workflow

6.4 Model Service and Fallback Mechanism

In deployment, the model file, vocabulary, and preprocessing configuration must remain version-consistent. The service loads the parameters once at startup, switches to evaluation mode, and performs inference under torch.no_grad to reduce gradient computation and GPU-memory use. Vectors for frequently accessed courses can remain in memory or a vector database; those for low-frequency courses can be loaded on demand. The model interface logs request latency, candidate count, highest score, and exception information for performance diagnosis.

Table 9. Responsibilities of the Semantic-Retrieval Modules

Module	Input	Main processing	Output
Material parsing	Teacher-uploaded files	Extract, clean, and chunk text	Knowledge chunks
Offline indexing	Knowledge chunks	Tokenize, encode, and cache	Candidate vectors
Query encoding	Student question	Apply training-consistent preprocessing and encoding	Query vector
Semantic ranking	Query and candidates	Score and sort in descending order	Top-5 results
Service monitoring	Logs and latency	Detect faults and control fallback	Status and alerts

A layered fallback strategy preserves service continuity. Semantic matching is preferred when Siamese LSTM is healthy. If loading fails, a timeout occurs, or resources are insufficient, the system switches automatically to TF-IDF. If neither algorithm returns a reliable result, the user is prompted to revise the question and the query is logged. A two-stage design using TF-IDF or vector recall followed by lightweight reranking can further improve Top-5 quality while controlling computation. The responsibilities of the retrieval module are listed in Table 9.

6.5 Interface Design and Caching

The model service can expose /encode and /match. /encode accepts one sentence or a sentence list and returns vectors, mainly for offline indexing. /match accepts a query and candidate list and returns similarity probabilities and ranking results, which is suitable for small-scale online comparison. Maximum text length and batch count must be limited to prevent abnormal requests from occupying excessive GPU memory. Responses include model version and latency in addition to scores, helping the front and back ends diagnose problems.

Table 10. Main Interfaces of the Semantic-Matching Service

Interface	Main input	Main output	Use stage
/encode	Text list, model version	Sentence vectors, latency	Offline index/cache update
/match	Query, candidate list	Probability, ranking, Top-K	Online retrieval
/health	None or probe parameters	Model status, device information	Monitoring and fault detection
/reload	Model and vocabulary versions	Load result	Model update

Caching includes model cache, candidate-vector cache, and query cache. The model remains resident after process startup. Candidate vectors are recomputed after materials change and organized by course. Top-5 results for popular questions repeated over short intervals can also be cached. Each cache key must include the course, material version, and model version. Without all three, an update may leave old results in the cache.

For a small candidate set, the service can compute batched dot products or run the matching head directly. For tens of thousands of passages or more, a vector index can first find a broad set of close items. Siamese LSTM then reranks this smaller set. This two-step process keeps the response time under control while still using the stronger matcher. A later BERT cross-encoder could rerank only the final Top-20 instead of reading the whole knowledge base. Table 10 lists the main interfaces.

6.6 Service Monitoring and User Feedback

The service needs checks for both system health and search quality. System measures include average and P95 latency, GPU or CPU use, memory use, error rate, and fallback count. Search measures include the no-result rate, Top-1 click rate, time spent on a result, question rewrites, and teacher corrections. The first group finds system problems. The second shows whether search is helping students.

Questions with low scores, questions that users rewrite many times, and results corrected by teachers can be saved as hard examples. After

personal data is removed and the examples are checked by people, they can be used for more training on course data. Clicks alone should not be treated as correct labels because bad feedback may be repeated by the model. Time on a page, later actions, and teacher review should be checked as well.

Table 11. Online Monitoring Indicators and Their Significance

Category	Indicator	Significance
Performance	Average/P95 latency	Determine whether real-time response meets requirements
Resources	GPU memory, RAM, utilization	Detect insufficient capacity and abnormal use
Stability	Error rate, fallback count	Evaluate service reliability
Retrieval effect	Top-1 click-through, no-result rate	Measure relevance and coverage
Data loop	Rewrite rate, teacher corrections	Identify hard examples and domain shift

New model versions should be released to a small group first. The old version stays available while the two versions are compared on latency, click rate, and no-result rate for a few courses or users. More traffic is sent to the new version only when these measures do not get worse. The service can quickly return to the old version if a problem appears. Each release stores the model, vocabulary, settings, and test report together, so any online result can be traced. Table 11 lists the monitoring measures.

7. Discussion

7.1 Accuracy and Deployment Cost

The results do not show a direct link between model size and task performance. BERT-base is about 4.52 percentage points more accurate than Siamese LSTM, but it has about 23 times more parameters. That cost may be acceptable for offline tests or a few important requests. It matters more in real-time course search, where loading time, memory, concurrent users, and maintenance all affect the service. A smaller model is easier to run in this setting. The model should be chosen for the actual task, not for accuracy alone.

7.2 Domain Transfer

LCQMC mainly contains questions from general question-answering sites. Zhixue also has course terms, definitions, and long passages. The writing style and the meaning of the labels are different. In LCQMC, similarity asks whether two questions have the same intent. In search, relevance may link a question to an answer or a title to a passage, so the relation is often one-way. A model trained only on LCQMC may therefore learn the wrong target for the platform. A better method is to collect real queries, build question-passage pairs, fine-tune the model on them, and often review questions with no result or a low score.

7.3 Reproducibility and Test Limits

All tests used one RTX 3050 Ti Laptop GPU with 4 GB of memory. This limited the batch size and ruled out larger hidden layers. Training became stable after the batch size was changed from 256 to 128 and unused cache was cleared when needed. To repeat the results, the Python, PyTorch, and CUDA versions must be fixed, along with random seeds, data versions, vocabulary files, the best epoch, and evaluation code. A different tokenizer dictionary or library version can change the token sequence and the final scores.

7.4 Ways to Improve the Model

There are four main next steps. First, more course data and hard negative examples can be used for training. Second, pretrained word vectors, a character branch, or a small Transformer may help with rare words and long links in a sentence. Third, separate tests can show how much attention, absolute difference, and element-wise product each add. Fourth, quantization, distillation, ONNX, and batch processing may shorten online response time. For a larger knowledge base, FAISS or another vector index can find nearby passages before the Siamese model ranks them.

7.5 Choosing a Model

Different stages need different models. TF-IDF is useful at the start of a course project because it quickly checks data loading, evaluation, and interfaces. Siamese LSTM is a better choice when semantic matching must improve but computing power is limited. When the platform has more labeled data and stronger servers, it can test dual-tower BERT for recall and a cross-encoder for reranking. Adding these parts in steps makes it easier to tell where a gain comes from and avoids a heavy model before the data

and tests are ready.

If the main goal is to find knowledge passages, the encoder should favor fast vector search. For duplicate short questions, the current binary matching head can stay. If the task changes to one-way question-answer relevance, the system should use separate query and document encoders or add cross-attention. The model design should match the input pair and the required output.

These experiments do not show that one model is best for every case. They show a practical process: begin with a statistical baseline, add a shared sequence encoder for better meaning, and then judge both model scores and system cost on the platform. This makes the cost and benefit of each model choice clear.

8. Conclusion

This study built a Siamese BiLSTM-attention model for Chinese sentence matching and knowledge search in education. The two towers use the same encoder, so both sentences are placed in one vector space. BiLSTM reads the context in both directions, and attention gives more weight to useful words. The classifier uses the two sentence vectors, their absolute difference, and their element-wise product. On LCQMC, the model reached 82.48% accuracy, 84.14% F1, and 92.79% AUC. These results are much better than TF-IDF. The model has about 4.8 million parameters, so it offers a useful balance between accuracy and cost. In Zhixue, it ranks knowledge passages and returns the Top-5 results, which shows that it can be used in the platform.

The work still has three limits. First, LCQMC contains general questions, not educational knowledge text. Second, the hyperparameters were mostly chosen from experience, and a full grid search and ablation study were not done. Third, quantization, distillation, and vector-index tuning are not complete. Later work will gather sentence pairs from education, test pretrained word vectors or a small Transformer, and use quantization, distillation, and vector caching to make online search faster.

References

[1] Manning C D, Raghavan P, Schütze H. Introduction to Information Retrieval. Cambridge: Cambridge University Press, 2008.

[2] Devlin J, Chang M W, Lee K, et al. BERT: Pre-training of deep bidirectional transformers for language understanding. Proceedings of NAACL-HLT. Minneapolis: ACL, 2019: 4171-4186.

[3] Mueller J, Thyagarajan A. Siamese recurrent architectures for learning sentence similarity. Proceedings of the 30th AAAI Conference on Artificial Intelligence. Phoenix: AAAI Press, 2016: 2786-2792.

[4] Liu X, Chen Q, Deng C, et al. LCQMC: A large-scale Chinese question matching corpus. Proceedings of the 27th International Conference on Computational Linguistics. Santa Fe: ACL, 2018: 1952-1962.

[5] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation, 1997, 9(8): 1735-1780.

[6] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. International Conference on Learning Representations. San Diego, 2015.

[7] Meng J X, Shan H T, Wan J J, et al. BSLA: An improved Siamese-LSTM text similarity model. Computer Engineering and Applications, 2022, 58(23): 178-185. (in Chinese).

[8] Li Y L, Zhou Y P. Text similarity matching based on a Siamese network and combined character-word vectors. Computer Systems & Applications, 2022, 31(10): 295-302. (in Chinese).

[9] Muennighoff N, Tazi N, Magne L, et al. MTEB: Massive text embedding benchmark. Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, 2023: 2014-2037. doi:10.18653/v1/2023.eacl-main.148.

[10] Xiao S, Liu Z, Zhang P, et al. C-Pack: Packed resources for general Chinese embeddings. Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2024: 641-649. doi:10.1145/3626772.3657878.

[11] Chen J, Xiao S, Zhang P, et al. M3-Embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. Findings of the Association for Computational Linguistics: ACL 2024, 2024: 2318-2335.

doi:10.18653/v1/2024.findings-acl.137.
[12] Wang L, Yang N, Huang X, et al. Improving text embeddings with large language models. Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics, 2024: 11897-11916. doi:10.18653/v1/2024.acl-long.642.
[13] Kingma D P, Ba J. Adam: A method for stochastic optimization. International Conference on Learning Representations. San Diego, 2015

