Implementation of transformer model for natural language to SQL query translation⋆

Kateryna Yalova[1,*,†], Mykhailo Babenko[1,†], Kseniia Yashyna[2,†], Artem Boyarchuk[3,†]

[1] *Dniprovsky State Technical University, Dniprobudivska str., 2, Kamianske, 51918, Ukraine*
[2] *SIRIS Academic, Francesc Cambo Av. 17, Barcelona, 08003, Spain*
[3] *Tallinna Tehhnikaülikool, Ehitajate tee 5, Tallinn, 12616, Estonia*

Abstract

The application of Natural Language Interfaces to Databases (NLIDB) serves as an important means of reducing the technical and IT competence requirements for database users. The objective of this study is the development and experimental evaluation of an NLIDB system that leverages a transformer-based architecture to generate SQL queries from user inputs formulated in natural language. The paper presents a generalized scheme of user interaction with the database via the NLIDB system and describes the main stages of input query preprocessing, including tokenization, embedding, positional encoding, and the architecture of the transformer-based neural network. The proposed model incorporates multi-head attention, which enables effective modeling of input query contexts, as well as the ADAM optimizer. The Spider corpus was utilized for training and evaluating the model. To assess the performance of the resulting model, in addition to accuracy, the BLEU metric was employed to quantitatively evaluate the degree of correspondence between the generated and expected queries, taking lexical similarity into account. The best experimental accuracy reached 69% using the BERT-base tokenizer and 63% with the basic Keras tokenizer. The highest BLEU score for the model with the BERT-base tokenizer was 46%, and 43.3% for the model with the basic Keras tokenizer. A comparative analysis demonstrated the competitiveness of the proposed model, indicating the effectiveness of the adopted solutions. The model performed best on simple and short queries, while the most challenging cases involved queries with literals that required inter-table relationships and domain-specific knowledge.

Keywords

Transformer-based neural network, natural language to database interface, Spider dataset

1. Introduction

The use of natural language as a means of interaction between humans and information or computer systems is one of the key research directions in the fields of data processing and artificial intelligence. One of the current challenges in Natural Language Processing (NLP) is the development of Natural Language Interfaces to Databases (NLIDB), which enable users to formulate queries in natural language without the need to know or use the Structured Query Language (SQL). A Natural Language Interface to Database is a system that provides users with a mechanism for transforming input text into SQL queries, thereby enabling natural language interaction with relational databases [1]. The integration of an NLIDB interface as an abstraction layer between users and relational database systems offers a key advantage – simplified interaction through natural language – and can be employed to lower the knowledge requirements for both developers and users of relational databases. The main purpose of embedding the NLIDB module into the architecture of an information system is to improve access to databases, facilitate interaction with information systems, and enhance overall user productivity. Practical applications of NLIDB include its use in business

ICyberPhyS'25: 2nd International Workshop on Intelligent & CyberPhysical Systems, July 04, 2025, Khmelnytskyi, Ukraine

[1*] Corresponding author.
[†] These authors contributed equally.

yalovakateryna@gmail.com (K. Yalova); mvbab130973@gmail.com (M. Babenko); yashinaksenia85@gmail.com (K. Yashyna); a.boyarchuk@taltech.ee (A. Boyarchuk)
0002-2687-5863 (K. Yalova); 0000-0003-1013-9383 (M. Babenko); 0000-0000-0002-8817-8609 (K. Yashyna); 0000-0001-7349-1371 (A. Boyarchuk)

contexts to simplify data access and improve analytics, as well as its integration with other intelligent technologies for automatic entity linking, sentiment analysis, and machine learning, with the goal of developing comprehensive solutions for business analytics and business process refactoring. The growing interest of the business sector in adopting this technology, particularly in combination with deep learning, underpins the rapid advancement and increasing popularity of NLIDB development tasks.

The automatic generation of SQL queries from natural language text sequences is a complex task that requires precise analysis, contextual processing, and the transformation of unstructured text into formalized queries [2]. The accuracy of such transformations remains insufficient due to the inherent complexity of natural language, including word polysemy, syntactic variability, and semantic ambiguity.

Significant progress in this field has been achieved by transitioning from rule-based and template-based approaches to the application of neural networks and deep learning techniques. Currently, various types of recurrent neural networks (RNNs) and convolutional neural networks (CNNs) are typically employed to solve natural language processing (NLP) tasks, including the Text-to-SQL problem. However, transformer-based neural networks are gradually replacing them due to their flexibility and high performance [3].

The objective of the present study is to design, adapt, and experimentally evaluate a transformer-based architecture for the task of automatic SQL query generation from natural language text sequences. The research is based on the hypothesis that the transformer neural network architecture, equipped with a self-attention mechanism, can be effectively applied to the Text-to-SQL problem.

2. Related works

Prior to 2017, most natural language processing models relied on RNNs incorporating encoder–decoder mechanisms to capture contextual relationships and dependencies between words in a sentence. However, this approach demonstrated lower effectiveness compared to the statistical methods dominant at the time, leading to the prevailing belief that neural networks were not well-suited for machine translation tasks. Moreover, RNN-based models suffered from a limitation in which the encoder tended to lose information from the beginning of the sequence if the input was too long [5]. For instance, the authors of [6] achieved only 36% BLEU score for their proposed Seq2Seq-based NLIDB model. Another model, NL2pSQL [7], implemented using a sequence-to-sequence architecture, improved the BLEU score from 27% to 31% by incorporating a denoising autoencoder mechanism. In [8], a model based on Long Short-Term Memory (LSTM) with two hidden layers and a dual-encoder mechanism, trained and evaluated on the SENLIDB and WikiSQL datasets, achieved an accuracy of 38.99% on the test set. Various strategies were employed to improve neural network performance, such as the use of bidirectional recurrent neural networks [9]. However, even in these cases, information from the middle of long sequences was often lost. Another significant drawback was the necessity for the encoder to compress the entire input sequence into a single hidden state vector, which adversely affected translation quality.

A breakthrough in addressing the limitations of recurrent neural networks (RNNs) emerged in 2015 with the introduction of the Attention mechanism, which was integrated into existing RNN architectures. This mechanism enabled models to assign weights to different parts of the input sequence regardless of their position, thereby allowing for a more flexible and context-aware processing of input data [10]. The implementation of attention significantly mitigated issues related to information loss in long sequences and the constraint of encoding the entire sentence into a single hidden state vector [11].

NLIDB models such as SQLova, described in [12], and IRNet [13], which are based on attention-enhanced bi-directional LSTM architectures, demonstrated considerable performance gains, achieving 80% accuracy on the WikiSQL dataset [12] and 53% accuracy on the Spider test set [13], respectively. In [14], the integration of a self-attention mechanism into an LSTM-based neural network led to translation accuracies ranging from 32% to 60% for natural language to SQL queries.

In 2017, a novel neural network architecture called the Transformer was introduced in [15]. This model proposed a new paradigm that eliminated the need for recurrent components entirely, relying solely on attention mechanisms and thereby avoiding sequential computation. This marked a significant advancement in the fields of natural language processing and deep learning [16]. The Transformer architecture substantially improved training speed and model efficiency, particularly when applied to large-scale datasets [17]. Transformer models opened a new frontier in data processing – especially textual data – and redefined the methodology for tackling NLP tasks.

The Transformer model has become one of the most prominent architectures for Natural Language Inference (NLI) tasks [18], which encompass a wide range of NLIDB-related challenges. Studies [8–15] report on the development, evaluation, and performance analysis of various neural network models, including TaPEx, which achieved 57–89% accuracy on the WikiSQL dataset [19], and SQL-PaLM [20], which demonstrated 77% execution accuracy on short queries from the Spider dataset. The SPSQL model, introduced in [21], reported an impressive 95% accuracy; however, this high recognition rate was attained on a significantly restricted dataset. Specifically, the experiments were conducted using a single database consisting of 37 tables, and the training and testing datasets included only 9,792 and 1,088 query pairs, respectively. In [22], the authors proposed the STAR framework, which leverages transformer-based pre-training and a self-attention mechanism to encode both text and SQL queries. This approach achieved 46.6% and 28.2% on the Interaction Match metric for the SParC and CoSQL test datasets, respectively. The RASAT model [23] extends a pretrained Seq2Seq transformer architecture and demonstrated execution accuracies of 37.4% and 52.6% on the CoSQL and SParC benchmarks, respectively.

An important milestone in the development of NLIDB is the modifications of the Transformer model, for example, the Bidirectional Encoder Representations from Transformers (BERT) model. It is based on the Transformer model, the improvements of which allow this model to more accurately understand the context through the analysis of sequences in two directions. Transformer and BERT architectures allows systems to analyze more complex phrases, better understand context, and form queries that require a deeper understanding of language. Model RAT-SQL [24] integrates BERT and demonstrates 65.6% accuracy on the Spider benchmark dataset. SDSQL [25] is a transformer-based model that incorporates BERT as a pre-trained encoder to generate rich contextual embeddings.

Its developers achieved up to 85% Logical Form Accuracy on the Spider test dataset. The results of recent studies demonstrate that, despite the original purpose of Transformer models being sequence-to-sequence translation, their application in the task of NLIDB proves to be a promising direction and remains a relevant scientific and practical challenge.

3. Proposed methodology

The implementation of the proposed task was carried out in several stages: data preprocessing, Transformer model development, evaluation of the obtained results, and optimization of the model architecture. To construct the model, both BERT tokenizers and the basic Keras tokenizer were employed to ensure high-quality vector representations of natural language input queries. The decoder was implemented based on the classical Transformer architecture, enabling efficient generation of SQL queries through step-by-step construction of output sequence tokens.

The Spider dataset was used for training and testing, allowing evaluation of the model's performance on multi-table schemas with complex joins, nested queries, and aggregations. The neural network was trained using the Adam optimizer, with categorical cross-entropy selected as the loss function. A series of experiments was conducted to determine the optimal model hyperparameters, including the number of Transformer layers, the dimensionality of hidden vectors, the number of attention heads, batch size, and the number of training epochs.

To quantitatively assess the proposed model, accuracy and the BLEU score (adapted for the Text-to-SQL task) were used as evaluation metrics.

4. Results

The primary type of data storage in modern information systems is the relational database, in which data are stored in the form of interrelated tables. While the use of SQL remains an effective tool for interacting with databases, it requires technical expertise and knowledge of the underlying database structure. The application of a Natural Language Interface to Database (NLIDB) helps reduce the entry barrier for using information systems and enhances data accessibility and usability, as it enables users to interact with databases using natural language. Text-to-SQL is one of the approaches to implementing NLIDB. It involves the automatic transformation of a natural language query into an executable SQL query over a relational database. A generalized schema of the Text-to-SQL problem implementation is presented in Figure 1.

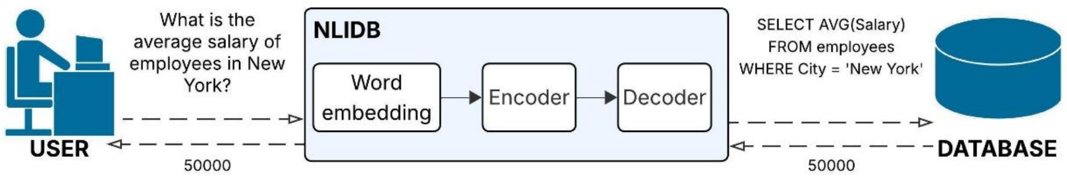

What is the average salary of employees in New York?
NLIDB
Word embedding
Encoder
Decoder
SELECT AVG(Salary) FROM employees WHERE City = 'New York'
USER
50000
50000
DATABASE

Figure 1: Generalized scheme of user-database interaction using NLIDB.

In the generalized case, the architecture of a neural network applied to solve the Text-to-SQL problem and enable the conversion of an input natural language text sequence into output SQL queries includes mandatory components such as:

- encoder – is a set of neural modules that provide taking of words from the input sequence by turns and formation of one or more hidden states characterizing the input sequence;
- decoder – is a collection of neural modules, that use the hidden state of the encoder to predict the result;
- components for translating natural language sentences to SQL queries.

The application of the Transformer model involves performing the following data preprocessing operations: tokenization, embedding, and positional encoding, the result of which is data prepared for input into the neural network.

4.1. Preprocessing stages

Tokenization plays an important role in sequence transformation problems. It is used to divide the input and output sequences into separate elements called tokens. A token can represent a word, word substring, or even a character, depending on the level of detail required. Tokenization methods vary depending on the details of the text breaking and the specific requirements of the task. During the development of the proposed NN model, both general and individual tokenizers were used to perform textual sequences to the form of tensors. In the development and testing process the BERT tokenizer was implemented. The developed and implemented into the NN tokenization algorithm can be realized in the following basic steps:

1. The initial textual sequence is encoded using a tokenizer, single for the input sequence, or separate for each of them.
2. The tokenizer denotes the beginning and the end of each sequence with < sos > and < eos > (start of sentence, end of sentence) markers.
3. The first token sent to the decoder is the sequence start label (< sos >).
4. The decoder creates a prediction by checking the output self-attention encoder's vectors.
5. The predicted token is then fed back to the decoder input. This operation is repeated until the model displays a label about the end of the sequence (< eos >).

The input sentence, presented in natural language, is transformed into numerical representations known as embeddings. Embeddings capture the semantic meaning of tokens within the input sequence. Through embedding, the input tokens are converted into their corresponding vector representations. The key parameters of this process include the vocabulary size *V* and the embedding vector dimension *d.* The embedding process can be described as follows:

$$E(x_i) = W_E \cdot o_i, \tag{1}$$

where x_i – is the i-th input token, W_E – is the embedding matrix, that can be described as $E = W_E \in R^{V \times d_{model}}$, o_i – is a one-hot vector for the token x_i.

The embedding mechanism is employed only in the lowest block of the encoder. Each encoder layer receives as input a vector whose size is a global parameter of the network and typically corresponds to the length of the longest sentence. The main parameters of the embedding are:

- the number of unique words or categories in the given dataset;
- the dimensionality of the vector space into which the words are embedded. This value is user-defined and serves as a hyperparameter of the model;
- the length of input sequences, i.e., the number of words in each input example. This parameter helps to fix the size of input sequences for more consistent processing by the model.

To handle the problem of different occurrences of the same word, positional encoding is used – vectors that represent the context of a word within the sequence. Positional encoding is added to the embedded tokens before their input to the model. It provides the model with information indicating the position or relative distance between tokens in the sequence. This is crucial because transformers lack recurrence or convolution mechanisms. For each position p and embedding dimension i:

$$\begin{gathered} PE(p, 2i) = \sin\left(\frac{p}{10000^{\frac{2i}{d_{model}}}}\right), \\ PE(p, 2i+1) = \cos\left(\frac{p}{10000^{\frac{2i}{d_{model}}}}\right), \end{gathered} \tag{2}$$

where p – is a position of the token in the sequence.

The result of positional encoding is a matrix of the same size as the token embeddings, containing encoded positional information.

4.2. Transformer-based architecture

Transformers consist of multiple layers that progressively refine data representations, enabling the capture of hierarchical and abstract features. When conceptualized as a black box, the internal structure of the model comprises two main components: the encoder and the decoder, within which transformer blocks are implemented, with the attention layer serving as the core mechanism. Figure 2 illustrates the architecture of the transformer-based neural network.

The encoder is the part of the neural network responsible for processing the input message, while the decoder's purpose is to generate the output sequence. In the Text-to-SQL task, the encoder is used to transform the input text into representations that preserve the semantics of the database query. The encoder receives vector representations of the input tokens and processes them through multiple layers. In turn, the decoder takes the processed input sequence from the encoder and translates it into SQL queries. The use of the encoder and decoder, along with data normalization through the application of a softmax layer, enables the transformation of natural language input text into syntactically correct SQL queries.

A key architectural difference in the decoder is the presence of an additional attention block, which helps the decoder attend to relevant parts of the input sentence. To construct a Transformer model, it is necessary to combine the encoder and decoder components and add, after the decoder output, a linear layer whose output size equals the target language vocabulary, followed by a softmax layer. Each transformer block consists of two main parts: self-attention and a feed-forward network. The input data first passes through the self-attention layer, which enables the model to consider other words in the sequence while encoding the current word. Subsequently, the output from this layer is fed into a fully connected feed-forward neural network, which remains the same across all layers.

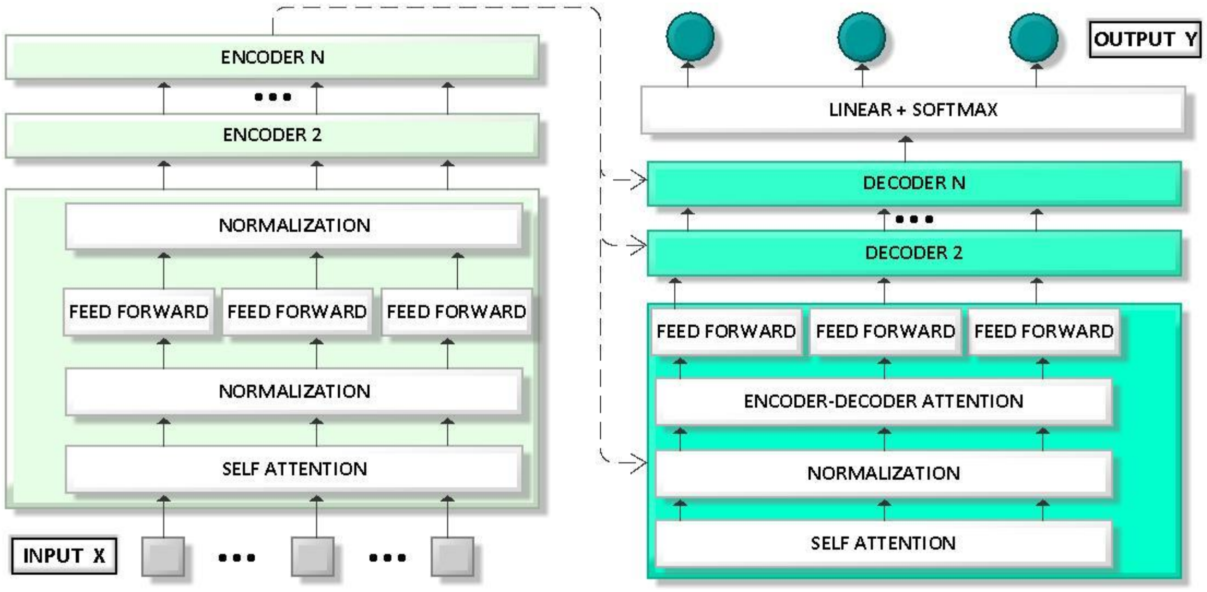


Figure 2: Transformer-based architecture.

The generalized algorithm for applying the transformer in the Text-to-SQL task consists of the following steps:

1. The input text, presented in natural language, is converted into a tokenized sequence.
2. The transformer processes this sequence by utilizing the following layers:
   a. Multi-Head Attention: determines relationships between tokens.
   b. Add layer: incorporates a residual connection where the output of the attention mechanism is added to the original tensor.
   c. Layer Normalization: stabilizes the data after the attention layer.
   d. Feed-Forward Network: performs a nonlinear transformation.
   e. Add layer: incorporates a residual connection where the output of the Feed-Forward Network layer is added to its input via an Add operation to preserve essential information from the previous layer.
   f. Layer Normalization: again stabilizes the data.
3. The resulting vector representation is decoded into an SQL query.

4.2.1. Self-attention mechanism implementation

The self-attention mechanism enhances the model's ability to focus on relevant parts of the input text during the generation of the output translation [22]. During training on a large dataset, the model internalizes this understanding. The self-attention mechanism allows each word to simultaneously attend to other words in the sequence, considering their importance relative to the current token. Thus, it can be argued that machine learning models can learn grammatical rules based on the statistical probabilities of word usage in language. To implement the self-attention mechanism, three vectors corresponding to each encoder vector must be created: $Q=\{q_1,...,q_n\}$ – is a set of queries, $K=\{k_1,...,k_m\}$ – is a set of keys, $V=\{v_1,...,v_l\}$ – is a set of values. These vectors are formed by multiplying the embedding by three weight matrices W_Q, W_K, W_V corresponding to queries, keys, and values, respectively:

$$Q = X \cdot W_Q, \qquad K = X \cdot W_K, \qquad V = X \cdot W_V, \tag{4}$$

where X – matrix of embedding vectors of input tokens.

Self-attention calculation is carried as:

$$S = softmax \frac{Q * K^T}{\sqrt{d_k}} V. \tag{5}$$

The generalized algorithm for applying the self-attention mechanism involves the following steps:

1. Calculation of the attention score for each current token, which is the result of the dot product between the query vector Q and the keys vector K of the current word.
2. Scaling of scores. To avoid excessively large values, the scores are divided by the square root of the vector dimension
3. The softmax function is applied to normalize the scores into the range between 0 and 1.
4. The resulting attention weights are applied to the value vectors V for each token.

This result represents a weighted representation of the input sequence, where the importance of each token is considered in the context of others. In this work, the basic attention mechanism was implemented, and all other attention mechanisms in the model inherited its implementation. Moreover, since it is necessary to determine not only the weight of each word in a sentence but also its relationship with other words, for each word it is necessary to compute multiple self-attention vectors and then calculate their arithmetic mean. This process is called Multi-Head Attention. The main parameters of Multi-Head Attention are:

- number of heads in the attention mechanism. Each attention head h_i has its own query vector, key and value $\forall h_i, \exists Q_i, K_i, V_i$;
- size of the keys set and values for each head, that usually obtained by dividing the total size of the layer by the number of self-attention heads;
- dimension of the values set for each block, calculated as a result of division of the total layer size by the number of self-attention heads;
- dropout probability for block responses before combining them into the final output.

After passing through n multi-head attention blocks, n matrices were obtained, one from each block. The computed results for each head h_i were concatenated and passed through an additional linear transformation:

$$MHA = Concat(h_1, h_2, .., h_n)W_o, \tag{6}$$

where Wo – is the weight matrix for combining the heads.

The attention block that connects the encoder and decoder parts is the most significant use of attention in the model and performs the task of identifying dependencies between the input and output sequences. In the developed model, this layer was termed Cross-Attention and implemented by passing the target sequence X as the query vector Q, and the context from the encoder as the key K and value V vectors.

4.2.2. Adaptive moment estimate algorithm

Transformer models are trained using supervised learning by minimizing a loss function. During training, the model's outputs are compared with the actual correct results. For this purpose, a vocabulary is first created, which consists of a vector of words as well as a vector indicating each word in the vocabulary. For model training, the decision was made to apply the Adaptive Moment

Estimation (ADAM) optimization algorithm, which combines the concepts of two other methods – Momentum and RMSProp – and was first proposed in 2014. This choice is justified by ADAM's ability to compute individual learning rates for each parameter. To stabilize the training process and reduce oscillations, exponentially weighted moving averages of the gradients and their squares are used:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \qquad (7)$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$$

where m_t – the first moment, v_t – the second moment, g_t – the gradient of the current mini-batch, β_1 i β_2 – are the coefficients for the first and second moments, respectively.

It implements correction of NN learning speed using the formula:

$$rate = d_{model}^{-0.5} \min\left({step_{num}}^{-0.5}, step_{num} \cdot {warmup_{steps}}^{-1.5}\right). \qquad (8)$$

ADAM has adaptive gradient descent properties and adapts the learning rate for each parameter, using an exponentially weighted average squared gradient, that allows to control the amplitude of updating weights effectively. The learning rate diagram of the developed model is shown in Fig. 3.

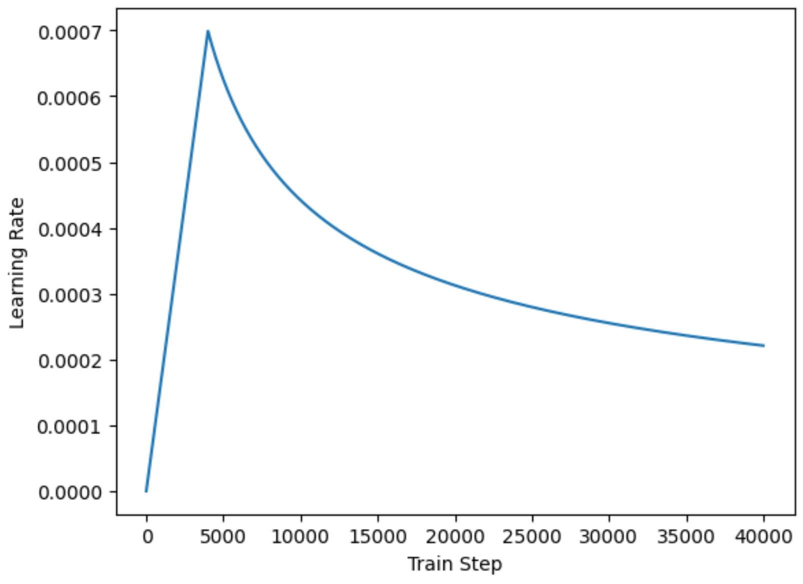


Figure 3: Learning rate for the proposed transformer-based neural network.

5. Experiments

For the development and testing of neural models, the Kaggle environment was chosen. This development platform provides cloud-based development and application deployment, offering fast graphical accelerators that significantly speed up the training of various models. Another advantage of this service is the ability to save input data within the project. Such data may include training datasets as well as saved and trained models, which allow using the model without retraining it from scratch.

For the experiments, it was decided to use 8 self-attention blocks, varying the number of layers (from 4 to 6), embedding vector sizes (from 128 to 512), and the dimensionality of the feed-forward network layer (from 512 to 2048). Compared to WikiSQL, the SQL syntax in the Spider dataset is more complex and diverse. Spider is a large-scale collection of interdomain semantic analysis and textual SQL datasets. The aim of using Spider is to develop natural language interfaces for interdomain databases. The Spider training set contains 10,181 questions and 5,693 unique complex SQL queries for 200 databases with multiple tables, covering 138 different data domains. The Spider dataset includes the use of JOIN operations, aggregate functions, nested subqueries, string and date operations, as well as operators such as LIKE, IN, and BETWEEN, which are especially useful for implementing real-world NLIDB systems. Spider has SQL queries of varying complexity, as well as sets with databases for NNs learning and testing. The Spider dataset consists of the following data:

- db_id – is a database identifier;
- question – is a query in natural language;
- question_toks – is a query in a natural language divided into tokens;
- query – is a SQL-query;
- query_toks_no_value – is a natural language query divided into tokens, where the parameters for conditional constructions are replaced with a "value" mask;
- query_toks – is a natural language query;
- sql – additional parameters related to query parameters (query type, availability of conditions or aggregation, parameters for conditional operators).

In a series of experiments using the Spider dataset, the field question was translated into the field query. Additionally, the query_toks_no_value field was used to transform natural language into an SQL query, where conditional parameters were replaced with the mask "value." Subsequently, these masks were substituted with specific values using auxiliary neural networks or syntactic analysis of the input sequence. For the insertion of conditional operator values, semantic analyzers and regular expressions were employed to identify numerical values or substrings within the input sequence. Table 1 presents the identified optimal hyperparameters of the model.

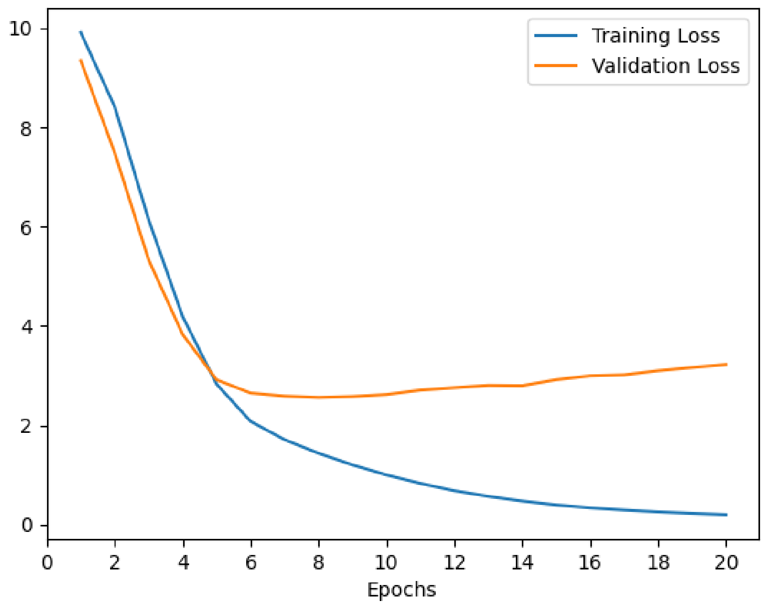
Table 1
Optimal model hyperparameters

Parameter	Value
Number of layers	6
Embedding vector size	256
Feed-forward network dimension	1024
Number of heads	8
Dropout	0,1
Tokenizer	BERT-base-uncased, Basic keras tokenizer

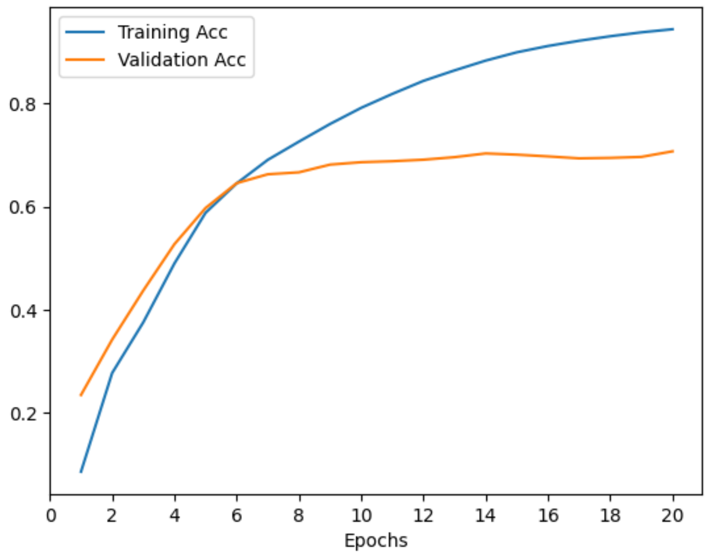
To analyze and compare the accuracy of each model, translation was performed on the first 100 pairs from the test set of the given dataset. This sample size was chosen due to the relatively long generation time of the neural model, while enabling an objective evaluation of each model's performance. The loss function and execution accuracy of the proposed model using the BERT-base tokenizer are presented in Figure 4. These graphs serve as important tools for monitoring and diagnosing the training process of the model and assist in making informed decisions regarding hyperparameter optimization and architectural improvements. During training, the neural model trained with Spider dataset demonstrated a gradual decrease in loss function and increased accuracy on the training sample.

Analyzing the loss function graph, it can be concluded that the proposed model trains well on the training data, with its error decreasing accordingly. However, the validation loss initially decreases but begins to increase after the 6th epoch, indicating model overfitting and a loss of generalization capability on new, unseen data. Since the minimum validation loss is reached approximately at the 6th epoch, this may indicate an optimal point to stop training in order to prevent further overfitting.

Training accuracy steadily increases, consistent with the decrease in training loss, confirming that the model becomes progressively more accurate. Validation accuracy also rises during the initial epochs but then plateaus, further confirming the presence of overfitting. Although the model continues to improve performance on the training data, its generalization ability on new data either stagnates or increases very slowly. The highest accuracy achieved by the model using the BERT-base tokenizer was 69%, compared to 63% using the basic tokenizer.



a)



b)

Figure 4: Diagrams for the model: a) loss, b) accuracy.

In addition to accuracy values, criteria that assess the naturalness of translation are used to evaluate the quality of neural translation. To assess the quality of results obtained using a transformer-based neural network, this work employed the Bilingual Evaluation Understudy (BLEU) metric in the context of the Text-to-SQL task. This metric measures the similarity between the translation and the original text based on a statistical analysis of word overlap. To apply the BLEU metric for evaluating the quality of NLIDB, it is necessary to implement the following steps:

1. The predicted and reference text are tokenized into separate words or phrases. SQL queries are tokenized by keywords, operators, and values (e.g., SELECT, FROM, WHERE, =, numbers, and identifiers). Some
2. Each token is given weight depending on its length.

3. BLEU calculates the number of n-grams (n token sequences) in the generated translation, which are also found in reference SQL-query.
4. BLEU calculates the accuracy of the generated translation by comparing the number of n-grams matches in the predicted SQL-query and reference one.
5. Accuracy is calculated for different levels of *n*-grams (usually from 1 to 4), and then combined into arithmetic mean to obtain the final BLEU score. In this work, bigrams were used for the BLEU calculation.

Table 2
Some examples of the used queries with Spider dataset

Initial textual sequence	Reference SQL-query	Predicted SQl-query
Training		
How many departments are led by heads who are not mentioned?	select count (*) from department where department_id not in (select department_id from management)	select count (*) from department where department _ id not in (select department _ id from department)
Count the number of farms	select count (*) from farm	select count (*) from farm
Testing		
How many concerts are there in year 2014 or 2015?	select count (*) from concert where YEAR = 2014 OR YEAR = 2015	select count (*) from campuses where year = 2014 or year = 2015
How many singers are from each country?	select country , count (*) from singer group by country	select country, count (*) from artist group by country

To prevent overestimation due to repetitions in the generated query, the frequency of n-grams in the hypothesis is limited by their frequency in the references. The BLEU metric value is defined as:

$$BLEU = \mathrm{BP} \cdot \exp\left(\sum_{n-1}^{N} w_n log P_n\right), \tag{9}$$

where w_n – are weights for each *n*-gram, *BP* stands for Brevity Penalty, which is used as a length penalty. The purpose of applying *BP* is to reduce the score of short queries that may have high precision. *BP* is defined by the following condition:

$$BP = \begin{cases} 1, if\ SQL - query\ lenth \geq reference\ length, \\ exp\left(1 - \dfrac{reference\ length}{sql - query\ length}\right), otherwise \end{cases}. \tag{10}$$

Analyzing the chosen combinations of NN parameters using the BLEU metric, the most accurate was a medium-sized with a BERT tokenizer NN model trained on the Spider dataset. Its BLEU score is 46% for the first 100 test sample conversions. A similar NN model with a basic tokenizer reached a BERT score of 43.3%.

Comparative analysis of the obtained results with neural network models tested on the different datasets is presents in the Table 3.

The presented model underperforms compared to SQL-PaLM [20] and SDSQL [25], which achieve accuracies of 77% and 85%, respectively; however, these models are significantly larger, employ large-scale fine-tuning, or incorporate built-in context. In other cases, the proposed model demonstrates competitive accuracy, which may indicate the effectiveness of the proposed design solutions. The accuracy of SQL query generation strongly depends on the neural network architecture, its hyperparameters, and the training algorithm. The use of neural network models enhances the power of NLIDB systems, making them more adaptable to various types of databases and improving their ability to interpret unstructured information. The diversity of proposed architectures and implementation approaches underscores the relevance of finding effective solutions for translating natural language into SQL queries.

Table 3
Comparative analysis results

Model	Architecture type	Accuracy,%	Dataset	BLEU, %
Presented model	Transformer-based	69	Spider	46
Neural model [6]	RNN	-	Spider	36
NL2pSQL [7]	RNN	-	Spider	31
IRNet [13]	Attention-based bi-LSTM	53	Spider	-
SQL-PaLM [19]	Transformer-based	77	Spider	-
STAR [22]	Pre-trained transformer-based	46.6	SParC	-
STAR [22]	Pre-trained transformer-based	28.2	CoSQL	-
RASAT [23]	Transformer-based	37.4	CoSQL	-
RASAT [23]	Transformer-based	52.6	SparC	-
RAT-SQL [24]	Transformer-based + BERT	65.6	Spider	-
SDSQL [25]	Transformer-based + BERT	85	Spider	-

The described model performed well in translating simple queries to tables used during training. The most problematic queries were:

1. Complex queries requiring joins between tables. When training and test sets contained queries related to different entities, the model sometimes failed to understand which entities to use, despite generating a structurally correct query.
   Queries containing constant values and keywords. Unmasked values for conditional operators did not distinguish values in the natural language query, resulting in incorrect outputs.

Possible ways to improve the accuracy of the developed model include:

1. Extracting specific words and entities from natural language queries in full. This approach removes from the natural language sequence those words that characterize important parts of the query, such as query type, target table, condition types, grouping, and table joins. After extracting these components, the query can be constructed according to a predefined template.
2. Training the model on database schemas, so that given existing entities, the model can recognize which words correspond to field and table names, identify their associated databases, understand relationships between tables, and recognize dynamic values used, for example, in conditional operators.

Regarding the strategy for improving model training, the authors consider it appropriate to implement Early Stopping to prevent overfitting and Data Augmentation by transforming existing data, which will be addressed in future research.

6. Conclusion

The main task of the NLIDB system developer is to implement functionality that allows to establish human-system interaction, so user queries are recognizable, can be turned into SQL command and provides the correct and expected results. Overall, the development of NLIDB systems can be justified by the need to improve data accessibility, facilitate interaction with information systems, and enhance user productivity. The task of creating an NLIDB system can be approached algorithmically in a manner analogous to automatic natural language translation, with the key distinction that the target language is not a natural language but the structured query language SQL. This implies that machine translation technologies can be adapted to transform textual queries into SQL.

In the present work, the application of a transformer-based neural network architecture for the task of automatic SQL query generation from natural language text sequences (Text-to-SQL), which constitutes a core component of NLIDB systems, was proposed. The objective of the study was to develop and adapt a transformer-based architecture for the efficient conversion of natural language queries into formalized SQL statements.

Although transformers are most commonly utilized for translation between natural languages, experimental results on the Spider dataset demonstrated the potential of the proposed architecture in solving the Text-to-SQL task. The highest achieved model accuracy using the BERT-base tokenizer was 69%, while using the basic Keras tokenizer it reached 63%. To evaluate the quality of the translation from natural language queries to SQL, the BLEU metric adapted for the Text-to-SQL task was additionally employed, enabling the assessment of similarity between generated SQL queries and reference queries. The use of BLEU allowed a quantitative estimation of the correspondence of generated queries to the expected results, taking lexical similarity into account. The highest BLEU score for the model with the BERT-base tokenizer was 46%, and 43.3% for the model with the basic Keras tokenizer.

Analysis of the obtained results revealed certain limitations of the developed model, particularly in handling complex SQL queries involving multiple table joins and non-standard conditional operator values. The model exhibited better performance on simpler queries to tables represented in the training dataset. To further improve accuracy and generalization capability, it is advisable to apply methods for extracting key entities from natural language queries, to train the model on database schemas for better understanding of context and relationships between tables, as well as to implement Early Stopping and Data Augmentation strategies to prevent overfitting.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] S. Abbas, M. Khan, S. Lee, A. Abbas, A. Bashir, A Review of NLIDB with deep learning: findings, challenges and open sssues. IEEE Access 10 (2022) 14927–14945. doi: 10.1109/ACCESS.2022.3147586.

[2] M. Liu, J. Xu, NLI4DB: A systematic review of natural language interfaces for databases, 2025. URL: https://arxiv.org/html/2503.02435v1#bib.bib53

[3] K. Majhadi, M. Mustapha, The history and recent advances of Natural Language Interfaces for databases querying. E3S Web of Conferences 229 (2021) 01039. doi: 10.1051/e3sconf/202122901039.

[4] A. Kumar, P. Nagarkar, P. Nalhe, S. Vijayakumar, Deep learning driven natural languages text to SQL query conversion: a survey. Journal of latex class files 14 (2022) 1–18.

[5] R. Iacob, F. Brad, E.-S. Apostol, C.-O. Truica, I. Hosu, T. Rebedea, Neural approaches for natural language interfaces to databases: a survey, Proceedings of the 28th International Conference on Computational Linguistics CL'20, Barcelona, Spain (Online), 2020, pp. 381–395.

[6] Y. Zhu, Y. Zhou, M. Xia, Generating semantically valid adversarial questions for TableQA AAAI. URL: https://arxiv.org/abs/2005.12696. doi:10.48550/arXiv.2005.12696.

[7] F. Chen, S. Hwang, J. Choo, J.-W. Ha, S. Kim. NL2pSQL: generating pseudo-SQL queries from under-specified natural language questions, Proceedings of the Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing EMNLP-IJCNLP'19, Hong Kong, China, 2019, pp. 2603–2613.

[8] I. Hosu, R. Iacob, F. Brad, S. Ruseti, T. Rebedea, Natural Language Interface for Databases using a dual-encoder model, Proceedings of the 27th International Conference on Computational Linguistics ICCL'18, Santa Fe, New Mexico, USA, 2018, pp. 514–524.

[9] R. Ghaeini, S. A. Hasan, V. Datla, J. Liu, K. Lee, A. Qadir, Y. Ling, A. Prakash, X. Z. Fern, O. Farri, DR-BiLSTM: Dependent Reading Bidirectional LSTM for Natural Language Inference, Proceedings of NAACL-HLT'18, New Orleans, Louisiana, 2018, pp. 1460–1469.

[10] A. Galassi, M. Lippi, P. Torroni. Attention in Natural Language Processing, IEEE Transactions on Neural Networks and Learning Systems 32 (2021) 4291–4308. doi: 10.1109/TNNLS.2020.3019893.

[11] D. Soydaner, Attention Mechanism in Neural Networks: Where it Comes and Where it Goes. URL: https://arxiv.org/pdf/2204.13154.pdf

[12] W. Hwang, J. Yim, S. Park, M. Seo, M. A comprehensive exploration on WikiSQL with table-aware word contextualization, Proceedings of the 33rd Conference on Neural Information Processing Systems NeurIPS'19, Vancouver, Canada, 2019.

[13] J. Guo, Z. Zhan, Y. Gao, Y. Xiao, J.-G. Lou, T. Liu, D. Zhang, Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation, Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics 2019. doi:10.18653/v1/P19-1444.

[14] X. Xu, C. Liu, D. Song, SQLNet: Generating structured queries from natural language without reinforcement learning, Proceedings of the Sixth International Conference on Learning Representations ICLR'18, Vancuver, Canada, 2019. doi:10.48550/arXiv.1711.04436.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need. URL: https://doi.org/10.48550/arXiv.1706.03762.

[16] K. Yemets, Time series forescasting model for solving cold start problem via temporal fusion transformer, Computer systems and information technologies 1 (2024) 57–64. doi: https://doi.org/10.31891/csit-2024-1-7

[17] N. Shahin, L. Ismail, From rule-based models to deep learning transformers architectures for natural language processing and sign language translation systems: survey, taxonomy and performance evaluation, Artifitial Intelligence Review 57 2024. doi: https://doi.org/10.1007/s10462-024-10895-z.

[18] T. Thoyyibah, W. Haryono, A.U. Zailani, Y.M. Djaksana, Transformers in machine learning: literature review, Jurnal Penelitian Pendidikan 9 (2023) 604-610. doi: 10.29303/jppipa.v9i9.5040.

[19] Q. Liu, B. Chen, J. Guo, M. Ziyadi, Z. Lin, W. Chen, J.-G. Lou, TAPEX: Table Pre-training via Learning a Neural SQL Executor, Proceedings of the Tenth International Conference on Learning Representations (Virtual) ICLR 2022 Vancuver, Canada, 2022. doi: https://arxiv.org/pdf/2107.07653.

[20] R. Sun, S. Ö. Arik, A. Muzio, L. Miculicich, S. Gundabathula, P. Yin, H. Dai, H. Nakhost, R. Sinha, Z. Wang, T. Pfister, SQL-PaLM: Improved Large Language Model Adaptation for Text-to-SQL. URL: https://arxiv.org/abs/2306.00739.

[21] R. Shen, G. Sun, H. Shen, Y. Li, L. Jin, H. Jiang, SPSQL: Step-by-step Parsing Based Framework for Text-to-SQL Generation. URL: https://arxiv.org/abs/2305.11061 doi: https://doi.org/10.48550/arXiv.2305.11061.

[22] Z. Cai, X. Li, B. Hui, M. Yang, B. Li, Bin. Li, Z. Cao, W. Li, F. Huang, L. Si, Y. Li, STAR: SQL Guided Pre-Training for Context-dependent Text-to-SQL Parsing, Proceedings of the Conference on Emperical Methods in Natural Language Processing EMNLP 2022, Abu-Dabi, United Arab Emirates, 2022, pp. 1235–1247.

[23] J. Qi, J.Tang, Z. He, X. Wan, Y. Cheng, C. Zhou, X.Wang, Q. Zhang, Z. Lin, RASAT: Integrating Relational Structures into Pretrained Seq2Seq Model for Text-to-SQL, Proceedings of the Conference on Empirical Methods in Natural Language Processing, Abu Dhabi, United Arab Emirates, 2022, pp. 3215–3229.

[24] B. Wang, R. Shin, X. Lin, S. Riedel, S. Singh, RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers, Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics ACL 2020, Online, 2020, pp. 7567–7578. doi: https://doi.org/10.18653/v1/2020.acl-main.677.

[25] B. Hui, X. Shi, R. Geng, B. Li, Y. Li, J. Sun. Improving Text-to-SQL with Schema Dependency Learning, URL: https://arxiv.org/abs/2103.04399, doi: https://doi.org/10.48550/arXiv.2103.04399.