

Research article

DOI: <https://doi.org/10.18721/JCSTCS.17309>

UDC 004.652:004.912



DEVELOPMENT OF THE SYSTEM OF AUTOMATIC GENERATION OF DATABASE MODEL ON THE BASIS OF THE TASK TEXT IN NATURAL LANGUAGE

I.A. Lapin ✉, *O.Yu. Sabinin*

Peter the Great St. Petersburg Polytechnic University,
St. Petersburg, Russian Federation

✉ lapin_ia@spbstu.ru

Abstract. This paper describes an approach to the implementation of a system that would allow automatic database model generation from a natural language description given by the user. Different machine learning technique, such as transformer, named entity recognition and relation extraction are considered and applied. The implementation of the neural network model uses the capabilities of the spaCy framework to organize a generic pipeline for training. Off-the-shelf implementations of some individual components from spaCy are also used, while the rest are custom. Moreover, we describe the process of gathering and preparing raw data for training a neural network model, and generating a proper corpus from them. For this purpose, a specialized annotating tool, Doccano, is used, which satisfies all requirements and is freely available. Finally, the paper presents the model parameters used in training and the performance metrics obtained. We've been able to achieve great results for the named entity recognition component, while the performance metrics of the relation extraction component can still be improved. The paper concludes with possible directions for further work on the implementation of the described system, including the relation extraction component improvements and new features implementation.

Keywords: natural language processing, named entity recognition, relation extraction, text analysis, classification, relational databases, model building

Citation: Lapin I.A., Sabinin O.Yu. Development of the system of automatic generation of database model on the basis of the task text in natural language. Computing, Telecommunications and Control, 2024, Vol. 17, No. 3, Pp. 93–102. DOI: 10.18721/JCSTCS.17309


Научная статья

DOI: <https://doi.org/10.18721/JCSTCS.17309>

УДК 004.652:004.912



РАЗРАБОТКА СИСТЕМЫ АВТОМАТИЧЕСКОЙ ГЕНЕРАЦИИ МОДЕЛИ БАЗЫ ДАННЫХ НА ОСНОВЕ ТЕКСТА ЗАДАНИЯ НА ЕСТЕСТВЕННОМ ЯЗЫКЕ

И.А. Лапин  , О.Ю. Сабинин

Санкт-Петербургский политехнический университет Петра Великого,
Санкт-Петербург, Российская Федерация

 lapin_ia@spbstu.ru

Аннотация. В данной статье описывается подход к реализации системы, которая позволила бы автоматически составлять модель базы данных по приведенному пользователем описанию на естественном языке. Рассматриваются и применяются различные методы машинного обучения, такие как трансформер, распознавание именованных сущностей и извлечение отношений. При реализации нейросетевой модели применяются возможности фреймворка spaCy для организации общего пайплайна для обучения. Также используются готовые реализации некоторых отдельных компонентов из spaCy, в то время как остальные являются пользовательскими. Кроме того, в статье описывается процесс сбора исходных данных для обучения нейросетевой модели, а также формирование из них надлежащего корпуса. Для этих целей используется специализированный инструмент для аннотирования – Dossano, который удовлетворяет всем функциональным требованиям, а также находится в свободном доступе. Наконец, в статье приводятся используемые при обучении параметры модели и полученные метрики производительности. В результате проведенного исследования авторам удалось достигнуть высоких показателей для компонента named entity recognition, в то время как показатели производительности для компонента relation extraction можно еще улучшить. В конце статьи приводятся возможные направления дальнейшей работы над реализацией описанной системы.

Ключевые слова: обработка естественного языка, распознавание именованных сущностей, извлечение отношений, анализ текста, классификация, реляционные базы данных, построение моделей

Для цитирования: Lapin I.A., Sabinin O.Yu. Development of the system of automatic generation of database model on the basis of the task text in natural language // Computing, Telecommunications and Control. 2024. Т. 17, № 3. С. 93–102. DOI: 10.18721/JCSTCS.17309

Introduction

In today's world, it is becoming somewhat of a *mauvais* to talk about the use of various information systems in a certain area – so global has become the digitalization of all areas of human activity. It is hard to imagine that today somewhere such systems are not used. This means that if information needs to be collected, processed and used, it also needs to be stored. Hence, there is no diminishing need for data storage tools and specialists, and approaches to this process are becoming more and more complex. Thus, in the field of databases, because it is through these tools that information storage is provided, specialists are increasingly in demand, with the growing number and complexity of information systems being developed, as well as with the need to support and expand existing ones.

It is human nature to look for simpler ways of solving the tasks we are facing, especially if these tasks become routine and take away time that should be spent on solving more complex requiring an individual approach. Therefore, we are trying to teach artificial intelligence (AI) to solve such routine tasks for humans, thereby freeing up resources for other tasks. Among other things, this is aimed at helping

people who do not have the necessary specialized knowledge in a certain field to get the opportunity to use various tools, at least their basic functionality.

This article will discuss the process of developing such a solution based on AI, which would allow one to create an initial representation of a relational database model based on the text description of the task of building a database in natural language. Such a tool should allow specialists to save time when implementing the developed database architecture, or when planning, being able to visualize different variants of possible architecture. In addition, such a solution will help to solve simple database development tasks for students, startup teams that are unwilling or unable to hire a specialist, as well as people who do not have professional knowledge in the database domain. The description of preliminary research as well as an assessment of the possibility of creating such a system, is given in [1].

Description of the chosen approach

When developing a software implementation of the system of automatic database model generation based on natural language text, we faced several major challenges:

- 1) to find logical entities in the text that represent the tables of the future model, as well as their attributes;
- 2) to relate the attributes to the logical entities, to which they refer;
- 3) to determine data types for the attributes;
- 4) to determine constraints for the tables (mainly foreign keys).

This article will discuss the solution to the first two challenges. Thus, at this stage, it is necessary to develop a software solution that would allow to input the text describing the modeling task in natural language and at the output get a certain set of logical entities with their attributes found in the text.

We decided to use the Python library spaCy¹ as a basis for building a software implementation, as this library offers a wide set of ready-made machine learning components for working with natural language text processing tasks, available for different languages, including Russian. In addition, this library offers a unified ecosystem of proprietary components, which can also include new, manually created components, which ultimately provide a unified pipeline for training and using the final model.

SpaCy uses a deep learning approach consisting of four main stages: embedding, encoding, attending, and prediction [2]. That is, first, tokenization takes place, where each representation is given a unique identifier and a table of word representations is formed, the size of which is determined by the size of the corpus dictionary. As a result, a set of vectors containing different tokens is formed.

The second stage involves the formation of a matrix containing context vectors for each token. This uses a combination of the Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) models instead of the usual recurrent neural network (RNN) implementation, since the hidden layer in RNN is constantly being rewritten and thus accumulation of information is problematic, while LSTM and GRU ensure that the results of the previous pass are saved for use in the next one. Finally, the resulting token vectors are composed of the vectors formed during the forward and backward passes.

The third stage uses the attention-mechanism to generate the final vectors. The last step, however, depends on the implementation of the particular component.

It is also worth noting that spaCy allows us to create joint models using a shared context for components, such as a vector store. We can use a transformer, or tok2vec component in the pipeline to form token vectors, and then use its results in each subsequent component in the pipeline without having to compute them again, as well as ensuring the integrity and equivalence of the store for all components. In addition, spaCy allows any transformer implementations, such as loading and inserting models with Huggingface into the pipeline.

When dealing with the first challenge – to find logical entities in the text that represent the tables of the future model, as well as their attributes – an approach to natural text processing using machine

¹ SpaCy, Industrial-Strength Natural Language Processing, Available: <https://spacy.io> (Accessed: 20.03.2024)

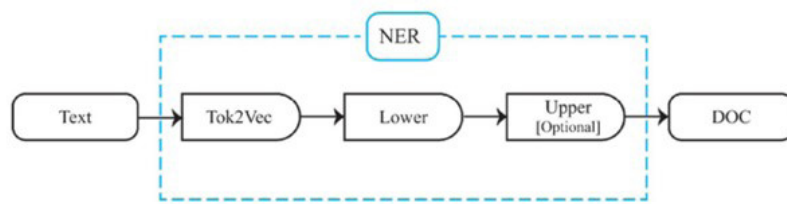


Fig. 1. NER model architecture [2]

learning technique was required, which could help to find specific designated classes of entities in the text depending on the context and the words themselves. Such a technique is called Named Entity Recognition (NER)², and has been used and improved for quite a long time. This technique allows solving tasks of determining named entities in text. Such tasks involve identifying certain entities that fall into different groups (classes) in unstructured text. NER technique is often used to locate people, organizations, geographic locations, etc. in text. Our task also fits well under this category of tasks, since we are solving a similar problem of locate entities belonging to a specific group in unstructured text. Thus, using NER component allows us to solve the problem of searching in text for logical entities and attributes.

To implement NER component, spaCy uses a transition-based system based on the fragmentation model proposed by Lample [3]. This is an approach based on computing different state transitions for prediction. The structure of NER component is shown in [2] (Fig. 1). The tok2vec component performs the entire process of translating tokens into vector representation. The “lower” component then creates special vectors for each property by token-property pairs, resulting in some general representation of the current state for each token. The “upper” component then uses the feed-forward network to predict weights based on the state representations.

To solve the second challenge we need to find a way to determine the relations in the text between specific attributes and the logical entities, to which they refer. The very problem of finding dependencies and relations between words in a text is not a new one and has already received different solutions many times. While earlier it was mainly pattern-matching, nowadays various machine learning technique are used, the most common and applicable of which are dependency parsing [4] and relation extraction³.

However, pattern-matching is still often used to solve various special problems, such as one described in [5], but in a more advanced form of rule-based analysis. Nevertheless, this technique cannot be used in our case, since the text of the task description can be arbitrary, therefore, it is impossible to identify specific rules that would be used to determine the relations.

The dependency parsing technique allows us to determine hierarchical dependencies between words, mostly within a single sentence, by constructing a directed graph with nodes (words) and edges (links). Despite the examples of existence of applications of this technique in tasks related to the search for relations between NER entities [6], dependency parsing does not allow us to obtain consistently desired result in our case, since it does not guarantee the construction of the same dependency graphs for different wording and word orders. In addition, the task of forming a dataset for training the dependency parsing component turns out to be quite extensive and redundant in this case.

Speaking of the relation extraction, it appears to be much more suitable, since it allows to search for relations between labeled entities, i.e., between entities found by NER or marked up manually in advance. In general, the algorithm of relation extraction component is shown in Fig. 2, and it can be described as follows⁴:

² What Is Named Entity Recognition? | IBM, Available: <https://www.ibm.com/topics/named-entity-recognition> (Accessed: 03.09.2024)

³ Relationship Extraction | NLP-progress, Available: https://nlpprogress.com/english/relationship_extraction.html (Accessed: 20.03.2024)

⁴ SPACY v3: Custom trainable relation extraction component, Available: https://youtu.be/8HL-Ap5_Axo?si=qRJOR-DbxhnZpcxj (Accessed: 13.03.2024)

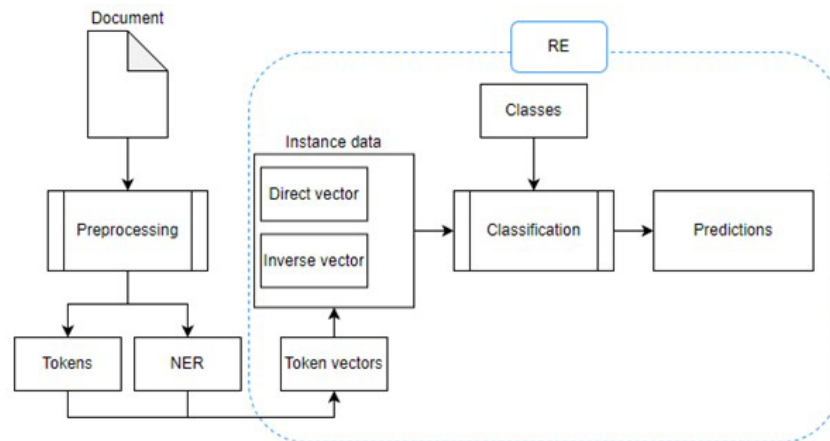


Fig. 2. Generalized RE component scheme

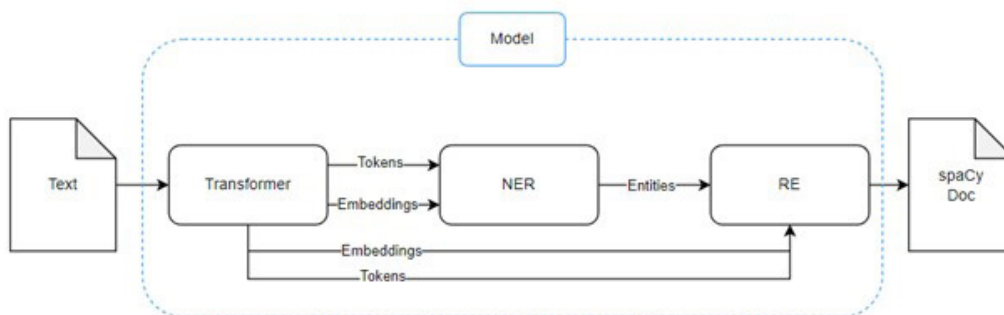


Fig. 3. Overall model scheme

1) Vector representations are analyzed and new vectors for relationships are constructed, including entity vectors and contextual information. In doing so, vectors will be created for all pairs of relations. This factor can be limited by setting the maximum window size as one of the hyperparameters. It is also worth noting that vectors will be created for both “direct” and “reverse” pairs.

2) The resulting vectors are assigned an assessment of belonging to a particular class (relation label). Any suitable classifier can be used, provided that the classification is performed with rejection, otherwise the relations will be assigned to any entities included in the window. Based on the obtained weights and taking into account the rejection threshold, the predicted labels are determined.

Since spaCy does not have a native implementation of the relation extraction component, we decided to use the standard generalized implementation provided by the spaCy developers⁵. It is worth noting that a big advantage of spaCy is the ability to easily add custom components to the standard pipeline due to the flexibility of their interfaces, as well as the capabilities of the thinc library⁶, used for simplified implementation of AI components.

We decided to use one of the standard spaCy transformers, the multi-language Bert model, as a common tok2vec component for our pipeline to improve the quality of vector representations and fully use

⁵ Custom spaCy Relation Extraction Component, Available: https://github.com/explosion/projects/tree/v3/tutorials/re_l_component (Accessed: 20.03.2024)

⁶ Thinc – A refreshing functional take on deep learning, compatible with your favorite libraries, Available: <https://thinc.ai> (Accessed: 20.03.2024)

Задача: Создать базу данных для курьерской службы, содержащую информацию о заказах, курьерах и доставке. База данных должна содержать следующие таблицы:

1. Таблица "Заказы" с полями: ID заказа, Дата заказа, Адрес отправителя, Адрес получателя, Описание заказа.
2. Таблица "Курьеры" с полями: ID курьера, ФИО курьера, Телефон курьера.
3. Таблица "Доставка" с полями: ID доставки, ID заказа, ID курьера, Дата доставки, Статус доставки.

Fig. 4. Easy level generated text

the attention-mechanism. NER and the relation extraction components rely on a shared store of vector representations obtained using the transformer. Fig. 3 shows the scheme of the resulting model.

Data mining and training set formation

To prepare a dataset for model training, we needed to collect a certain number of database modeling task texts. In the process of studying open access datasets on Kaggle and Huggingface, it became clear that, with a high degree of probability, the required dataset in ready or partially ready form does not exist, at least in the public domain. Therefore, a search on the Internet resulted in several open resources with the required tasks texts in Russian⁷ [7–8].

Unfortunately, the amount of data found was insufficient to form the minimum required dataset, since, according to the spaCy developers' recommendation, the dataset for training should contain at least a couple hundred records⁸. However, using the found sources, we managed to form a set of only 58 tasks. We decided that the size of the training set for the trial implementation could be about a hundred records, and yet, it was necessary to find the missing records somewhere.

Therefore, we decided to use Large Language Models (LLMs), since at this level of development of AI technologies we can use generative AI models to generate new task texts. We decided to use domestic solutions: Sber GigaChat⁹ and YandexGPT¹⁰. Web user interfaces in the public domain were used to work with the models. Using various queries to the models, 30 new tasks of varying levels of text complexity were generated (according to the empirical evaluation of the tasks already in the set), ranging from simple (Fig. 4) to medium (Fig. 5).

This resulted in a total dataset size of 88 records. This amount was still less than the recommended number, but was much closer to the minimum recommended one hundred records. There was also the possibility that generating more records could cause the model to “learn” the pattern of text construction offered by the generative AI and overtrained as a result. In addition, in general, the use of a large amount of surrogate data during training is considered to be an acceptable but undesirable practice, since the artificially generated data can often be very different from the real data that the model will encounter during its work. Thus, we decided to settle on a compromise – the number of records in the final dataset of about a hundred records and the ratio of generated texts to “natural” texts from open sources and their variability should allow us to obtain adequate results for real-life application.

After the initial dataset was formed, we wanted to improve the quality of the data collected. The data was examined for anomalies, problems with text formatting and overall low-quality texts that could lead to undesired interference with training were flagged.

⁷ Tekhnologii baz dannykh i znaniy – Individualnyye zadaniya dlya samostoyatelnoy raboty [Database and knowledge technologies – Individual tasks for self-study], Available: http://bseu.by/it/tohod/indv_zadaniya.htm (Accessed: 20.03.2024) (in Russ.); FKN+ANTITOTAL – Varianny zadach – proyektirovaniye baz dannykh [Task options – Database engineering], Available: <https://fkn.ktu10.com/?q=node/72> (Accessed: 20.03.2024) (in Russ.); Zadacha i teoriya po SQL, MySQL, PostgreSQL i bazam dannykh voobshche [Tasks and theory for SQL, MySQL, PostgreSQL and databases in general], Available: <https://gist.github.com/codedokode/10539213> (Accessed: 20.03.2024) (in Russ.)

⁸ Training Pipelines & Models spaCy Usage Documentation, Available: <https://spacy.io/usage/training> (Accessed: 20.03.2024)

⁹ Sber GigaChat, Available: <https://developers.sber.ru/gigachat> (Accessed: 20.03.2024) (in Russ.)

¹⁰ YandexGPT, Available: <https://ya.ru/ai/gpt-2> (Accessed: 20.03.2024) (in Russ.)

Задание: Создать логическую модель базы данных для системы управления задачами.

Система управления задачами предназначена для организации и контроля выполнения задач в команде. Она позволяет создавать задачи, назначать ответственных, устанавливать сроки выполнения, отслеживать прогресс и оставлять комментарии.

Логическая модель базы данных должна включать следующие элементы:

1. Таблица "Задачи" с полями:
 - Идентификатор задачи (ID)
 - Название задачи
 - Описание задачи
 - Ответственный за задачу
 - Дата создания задачи
 - Дата завершения задачи
 - Статус задачи (в процессе, выполнена, отменена)
2. Таблица "Комментарии" с полями:
 - Идентификатор комментария (ID)
 - Идентификатор задачи (ID)
 - Автор комментария
 - Текст комментария
 - Дата создания комментария
3. Таблица "Статусы" с полями:
 - Идентификатор статуса (ID)
 - Название статуса
4. Таблица "Ответственные" с полями:
 - Идентификатор ответственного (ID)
 - Имя ответственного
 - Фамилия ответственного
5. Таблица "Связи" с полями:
 - Идентификатор задачи (ID)
 - Идентификатор ответственного (ID)

При создании логической модели базы данных необходимо учесть следующие требования:

1. Каждая задача должна иметь уникальное название и описание.
2. Каждая задача должна быть связана с ответственным за ее выполнение.
3. Каждая задача должна иметь дату создания и дату завершения.
4. Каждая задача может иметь несколько комментариев.
5. Каждая задача может иметь несколько ответственных.
6. Каждая задача может иметь несколько статусов.

Fig. 5. Average level generated text

Formatting and punctuation problems were corrected first:

- 1) Unnecessary line breaks were eliminated.
- 2) Insignificant task headings without punctuation and with unnecessary hyphenation were eliminated.
- 3) Missing punctuation marks were added.

Then the task texts were analyzed for anomalies — errors, incorrect wording, etc. As a result, most of the anomalies were corrected; if too significant changes had to be made to correct anomalies, such texts were discarded, due to the excessive labor-intensive nature of such corrections and were not included in the updated dataset.

Finally, texts that differed too dramatically from others, while providing either insufficient or poor quality information about the database model, even from a human perception point of view, were also excluded from the final set. We decided to consider such tasks as inherently incorrect and not to include them in the training set. As a result of all the improvements made, the final dataset size was reduced to 80 records.

Data markup and corpus preparation

To use the collected data set as a corpus for training the neural network model, it was necessary to mark up the data, and to convert the obtained corpus to such a format that would be easily read by spaCy when generating binary files with a special extension, which would then be used to train the model.

Data markup, or annotation, involves selecting all necessary entities and relations in texts — in our case, these are logical entities and their attributes, as well as relations of attribute dependencies on specific entities. This required finding a suitable tool that would have sufficient functionality to perform the required actions, namely to annotate entities and relations between them throughout the text.

An ideal choice for such a tool would be Explosion Prodigy¹¹, an advanced text annotation software with extensive functionality that allows to quickly and conveniently solve a wide range of data annotation tasks, which, moreover, is developed by the spaCy developers. However, since this tool is a paid tool and this study was conducted without additional funding, we decided to consider free analogs.

Thus, it was necessary to find an alternative tool in the public domain that offered the required functionality. As a result of studying the market of solutions offered in this area, a free open source software focused on annotating text data, Doccano¹², was selected. This tool turned out to be the only one that fully met all the requirements and offered a convenient user interface to work with.

While annotating and re-evaluating records from the dataset, we decided to extend the originally planned set of labels for entities and relations. As a result, the following set of labels was used for annotation:

1) For entities: LENTITY (logical entity), LATATTRIBUTE (attribute of a logical entity), DESC (utility description for any other entities, primarily for attributes).

2) For relations: ATTRIBUTE_OF (is an attribute), DESCRIBED_BY (is described).

The LENTITY and LATATTRIBUTE labels were used to label logical entities and their attributes, while the DESC label was added to provide greater precision when labeling attributes in complex cases. This applies to cases when two or more attributes had the same description in the text, such as “date and time of order”. In this case, we defined LATATTRIBUTE labels for “date” and “time”, and DESC label for “order” to subsequently form two attributes “order date” and “order time”. To make such a solution work, DESCRIBED_BY relation label was introduced to show which entity with DESC label describes an entity with LATATTRIBUTE label. Finally, ATTRIBUTE_OF label was used to define the relation between the attribute and the logical entity to which it refers.

Thus, the original dataset was annotated using Doccano with labels described in the previous paragraph. The result was a corpus written in JSONL file format with the following structure:

1) The text field is the task text itself in its original form.

2) The entities field is a set of JSON objects with information on annotated entities in the text (id, start of entity, end of entity, label).

3) The relations field is a set of JSON objects with information on annotated relations in the text (id, id of parent entity (from), id of child entity (to), label).

Afterwards it was necessary to edit the obtained corpus so that it could be easily converted to the internal spaCy format. It is worth noting that in the implementation of the custom relation extraction component, spaCy developers also provide a script for parsing the annotated corpus obtained using their Prodigy tool to convert into the internal binary format. Since our corpus still lacked some of the information needed, according to spaCy internal notation, we decided to write a script to supply the corpus with missing details and convert it into the required format.

Experiments

The model described in this paper was trained using the generated corpus. The following hyperparameter values were used for training, as shown in Table 1. The settings for the transformer and NER component were left at their default values, while the relation extraction component window size was set to 1000 to ensure that it could cover large intervals between potentially related entities within all or most of the text. The rejection threshold was set to 0.3, as experiments with the current version of the model have shown this threshold to be better than the more obvious threshold of 0.5. Table 2 shows the performance metrics of the trained model.

¹¹ Prodigy – Radically efficient machine teaching. An annotation tool powered by active learning, Available: <https://prodi.gy> (Accessed: 20.03.2024)

¹² Doccano – open-source data labeling tool for machine learning practitioners, Available: <https://doccano.github.io/doccano> (Accessed: 20.03.2024)

Table 1

Hyperparameter values for training

Hyperparameter	Value
Overall batch size	1000
NER hidden layer width	64
Transformer maximum batch items	4096
Transformer window	128
Transformer stride	96
RE window	1000
RE threshold	0.3

It is worth noting the excellent performance result of NER component. Despite the relatively high total error, the indicators of accuracy, recall, and F-measure are very good, showing that the model correctly performs 98% of predictions.

Table 2

Performance metrics

		Metric			
		Loss	Precision	Recall	F-score
Component	NER	4872	0.96	1.0	0.98
Value	RE	8	0.31	0.44	0.36

At the same time, the indicators for the relation extraction component are not as good, with only 36% of the conditional accuracy of predictions. It can be assumed that the problem is largely due to the fact that we consider not individual sentences, but entire texts, including those that determine the presence of relations between entities in different sentences. After all, the originally used implementation of this component was supposed to work more with individual sentences.

Conclusion

This article outlined the details of a partial implementation of a system for automatic generation of database models based on a task text in natural language. As a result, excellent results were obtained on the trained model for searching logical entities and their attributes in texts, while the component for searching relations between attributes and logical entities needs to be improved.

It is worth noting that the system is still a work in progress, since in addition to the refinement of the existing components, more extensive functionality such as defining attribute data types and searching for restrictions should be implemented. In addition, it will be necessary to post-process the results provided by the model after processing the source texts, since it is necessary to form a visual representation of the database model described in the task for the user.

REFERENCES

1. **Lapin I.A., Sabinin O.Yu.** Research and planning the development of an automated system for building relational database models based on provided task text in natural language. *Theoretical & Applied Science*, 2023, Vol. 11, No. 127, Pp. 311–320. DOI: 10.15863/TAS.2023.11.127.39

2. **Singh N., Kumar M., Singh B. et al.** DeepSpacy-NER: an efficient deep learning model for named entity recognition for Punjabi language. *Evolving Systems*, 2023, Vol. 14, Pp. 673–683. DOI: 10.1007/s12530-022-09453-1
3. **Lample G., Ballesteros M., Subramanian S., Kawakami K., Dyer C.** Neural architectures for named entity recognition. *Proceedings of NAACL*, 2016. DOI: 10.48550/arXiv.1603.01360
4. **Jaiswal S.** Natural Language Processing – Dependency Parsing. Medium, 2021, Available: <https://towardsdatascience.com/natural-language-processing-dependency-parsing-cf094bbbe3f7> (Accessed: 20.03.2024)
5. **Kozhevnikov V.A., Sabinin O.Yu.** System of automatic verification of answers to open questions in Russian. *St. Petersburg State Polytechnical University Journal. Computer Science. Telecommunications and Control Systems*, 2018, Vol. 11, No. 3, Pp. 57–72. DOI: 10.18721/JCSTCS.11306
6. **Dy Dx.** Utilizing dependency trees and NER models for relation extraction task. Medium, 2021, Available: <https://medium.com/@dxdy/utilizing-dependency-trees-and-ner-models-for-relation-extraction-task-ffa5463cb8> (Accessed: 20.03.2024)
7. **Sidorova N.P.** Bazy dannyh: praktikum po proektirovaniyu relyacionnyh baz dannyh [Databases: A Hands-On Guide to Designing Relational Databases]. Moscow: Direkt-Media, 2020. 92 p.
8. **Potapov A.S., Astahova I.F., Chulyukov V.A., Starikov V.N.** Praktikum po informacionnym sistemam. Oracle [Oracle Information Systems Workshop]. Kiev: YUnior, 2004. 177 p.

INFORMATION ABOUT AUTHORS / СВЕДЕНИЯ ОБ АВТОРАХ

Lapin Igor A.

Лاپин Игорь Александрович

E-mail: lapin_ia@spbstu.ru

Sabinin Oleg Yu.

Сабинин Олег Юрьевич

E-mail: sabinin_oyu@spbstu.ru

Submitted: 22.04.2024; Approved: 29.07.2024; Accepted: 06.08.2024.

Поступила: 22.04.2024; Одобрена: 29.07.2024; Принята: 06.08.2024.