COMPUTER RESEARCH AND MODELING
2025 VOL. 16 NO. 7 P. 1703–1713
DOI: 10.20537/2076-7633-2025-16-7-1703-1713

Ки&М

SPECIAL ISSUE

UDC: 004.42

Generating database schema from requirement specification based on natural language processing and large language model

N. Salem[1,a], K. Al-Tarawneh[1,b], A. Hudaib[1,c], H. Salem[2,d], A. Tareef[3,e], H. Salloum[2,f], M. Mazzara[2,g]

[1]King Abdullah II School for Information Technology, University of Jordan, Amman, Jordan
[2]Innopolis University, 1 Universitetskaya st., Innopolis, 420500, Russia
[3]Faculty of Information Technology, Mutah University, Karak, Jordan

E-mail: [a] NAD9220478@ju.edu.jo, [b] Khawla_t@mutah.edu.jo, [c] ahudaib@ju.edu.jo, [d] H.salem@innopolis.ru, [e] a.tareef@mutah.edu.jo, [f] h.salloum@innopolis.university, [g] m.mazzara@innopolis.ru

Received 26.10.2024, after completion — 21.11.2024
Accepted for publication 25.11.2024

A Large Language Model (LLM) is an advanced artificial intelligence algorithm that utilizes deep learning methodologies and extensive datasets to process, understand, and generate human-like text. These models are capable of performing various tasks, such as summarization, content creation, translation, and predictive text generation, making them highly versatile in applications involving natural language understanding. Generative AI, often associated with LLMs, specifically focuses on creating new content, particularly text, by leveraging the capabilities of these models. Developers can harness LLMs to automate complex processes, such as extracting relevant information from system requirement documents and translating them into a structured database schema. This capability has the potential to streamline the database design phase, saving significant time and effort while ensuring that the resulting schema aligns closely with the given requirements. By integrating LLM technology with Natural Language Processing (NLP) techniques, the efficiency and accuracy of generating database schemas based on textual requirement specifications can be significantly enhanced. The proposed tool will utilize these capabilities to read system requirement specifications, which may be provided as text descriptions or as Entity-Relationship Diagrams (ERDs). It will then analyze the input and automatically generate a relational database schema in the form of SQL commands. This innovation eliminates much of the manual effort involved in database design, reduces human errors, and accelerates development timelines. The aim of this work is to provide a tool can be invaluable for software developers, database architects, and organizations aiming to optimize their workflow and align technical deliverables with business requirements seamlessly.

Keywords: large language model, natural language processing entity-relationship diagrams, SQL

Citation: *Computer Research and Modeling*, 2025, vol. 16, no. 7, pp. 1703–1713.

Introduction

In the ever-evolving domain of database management, the development of an efficient and accurate database schema is essential yet often complicated and time-consuming. Traditionally, converting requirements into a cohesive database schema necessitated considerable expertise and manual effort. However, the advent of advanced technologies, such as NLP and LLMs, signals a significant transformation in this process. With the capability of NLP, computers can now comprehend and analyze requirement specifications articulated in natural language with remarkable accuracy. This technology enables machines to grasp the subtleties and context of human language, facilitating the extraction of pertinent information necessary for database design. When coupled with the sophisticated capabilities of LLMs, these tools can generate robust database schemas, streamlining the process and mitigating errors.

The integration of NLP and LLMs in the creation of database schemas yields numerous advantages. It enhances the precision and consistency of database design by minimizing the likelihood of human error. Automated schema creation ensures adherence to specified criteria, which is particularly beneficial in complex projects characterized by intricate requirements. Furthermore, it accelerates the design process by automating the transformation of requirements into functional schemas. This efficiency allows developers to concentrate on other critical project tasks, such as optimization and testing.

The proposed tools that can generate relational schemas from natural language requirements and entity-relationship diagrams (ERDs) utilizing LLMs represent a significant advancement in the field of database design and management. These tools leverage NLP to accurately interpret user specifications and employ the advanced capabilities of LLMs to create detailed and efficient database schemas. This automation not only enhances the accuracy and consistency of database structures but also democratizes the design process, enabling individuals with varying levels of technical expertise to contribute. By bridging the gap between human language and complex database architecture, these tools streamline the design process, reduce errors, and foster the development of more effective and user-friendly database systems. This paradigm shift not only enhances productivity but also paves the way for future innovations in intelligent, automated database management solutions.

Background

At the planning phase, stakeholders collaborate to ascertain the purpose of the database [Teniente, Urpi, 2003]. Subsequently, stakeholders will establish objectives for the database, which help formalize its purpose into measurable goals. These objectives must encompass the database systems, applications, and technologies involved. The second stage, termed system definition, refines the identification of users and the types of tasks they will undertake. The requirements collection and analysis phase, often the most intricate, encompasses requirements determination, structured analysis, and data collection. Subsequently, the database designer begins to ascertain the data to be stored and the interrelations among data elements.

The creation of the database schema is contingent upon the database conceptual model, which underpins all system development. The conceptual model is designed to represent the overall structure of the target system and the intended usage by prospective users. Database design, a comprehensive planning process, is essential for acquiring, storing, managing, sharing data among multiple users, recovering data, and ensuring its availability on demand. The database design lifecycle encompasses six steps: planning, system definition, requirements collection and analysis, database design, implementation and conversion, and operation and maintenance.

A database schema is a description of the structure of the database and the data it contains, viewed from specific perspectives referred to as schema levels [Burgin, 2005]. The view at different

schema levels is pertinent to database users, administrators, and designers for varying purposes. The design of the database schema is tailored to meet the specific needs of the target user. Understanding the intended application of the database aids the schema designer in making critical design decisions regarding the organization and representation of data. Three schema types exist: physical schema, logical schema, and external schema. The physical schema represents the internal organization of the database; the logical schema delineates the organization of information, while the external schema defines the user's view of the data, including data structures, query language, and data manipulation.

Grasping the specifications (requirements) of a text involves addressing numerous issues related to enterprise databases, particularly schema integration challenges and potential solutions [Jin et al., 2018]. Recent advancements have led to the creation of programs (schemas) capable of identifying the optimal alignment of source schemas, primarily relying on the Structure Query Language (SQL) definitions of both input schemas. Unfortunately, in many instances, data must be exchanged (integrated) between two different organizations utilizing distinct database management systems, and often only the original organization's schema SQL definition is accessible.

Consequently, a multiresolution schema mapping system utilizing XML file structures for dataset description has been demonstrated.

To gain a deeper understanding of the database structure, designers meticulously analyze the entity sets using a systematic methodology. The process of constructing a database according to the entity relationship diagram (ERD) follows a series of steps [Pieris, Wijegunesekera, Dias, 2020]. This process commences with an analysis of the provided ERD, during which the time consumed and any redundancies or extraneous data types are assessed. Designers are responsible for identifying the parent entity set and segmenting its related attribute sets to mitigate the complexity of the tables [Kim, Kanezaki, Tanaka, 2020]. By reducing table complexity, query performance can be enhanced. Designers will also identify all attribute sets participating in the relationship sets linked to their respective entity sets.

The schema information of a database system reflects the underlying data structure in the form of tables/relations, keys, integrity constraints, indexes, and access structures [Candel, García-Molina, Ruiz, 2023]. The schema is often referred to as metadata, as it describes the content of the data stored in the database [Trummer, 2021]. This schema metadata is utilized in various database applications, including query processing, query optimization, physical data storage, search, data integration, and indexing [Proper, Halpin, 2021].

Humans possess a remarkable capacity for using language to express themselves and establish connections. This capability begins developing in early childhood and continues to grow throughout life. Conversely, machines lack this innate ability to understand and communicate like humans, unless equipped with sophisticated AI algorithms. The quest to enable machines to read, write, and communicate like humans has long been a challenge and aspiration [Chowdhary, Chowdhary, 2020; Hadi et al., 2023].

Artificial Intelligence (AI) aims to create systems that can emulate human intelligence and skills [Zhao et al., 2023]. In the 18th century, philosopher Denis Diderot proposed that if a parrot could answer every question, it might as well be considered intelligent [Liu et al., 2023a]. While Diderot's contemplation centered on living creatures, it sparked the notion that a highly intelligent entity could mimic human behavior.

In the 1950s, Alan Turing furthered this concept by introducing the Turing Test, a pivotal criterion in AI designed to assess whether machines can exhibit intelligent behavior indistinguishable from that of humans [Sumers et al., 2023].

In AI discourse, these intelligent entities are often referred to as "agents", which are fundamental components of AI systems. An agent in AI functions like an artificial being capable of sensing its environment, making decisions, and acting upon those decisions using actuators [Weng, 2023].

In [Zhao et al., 2023], the investigator reviews recent advancements in LLMs and their profound impact on the AI community. The examination covers various aspects of LLMs, including pre-configuration, adaptation, utilization, and performance assessment, alongside potential future directions. Key insights highlight the robust capabilities of pre-trained models, particularly in addressing diverse NLP challenges. Furthermore, the article underscores the significance of model scaling, which enhances LLM performance and unveils specialized capabilities that may not be evident in smaller models. The technological evolution of LLMs has revolutionized AI algorithm development, resulting in substantial progress and garnering considerable attention within academic and industrial spheres. As advancements continue, remaining challenges and future research directions will be explored to determine the necessity for ongoing investigation and innovation in LLM studies.

Figure 1 below an evolution process of the four generations of language models (LM) from the perspective of task-solving capacity. For: A neural probabilistic language model (NPLM) [Manhaeve et al., 2021], and Natural language processing (NLPS) [Chowdhary, Chowdhary, 2020].

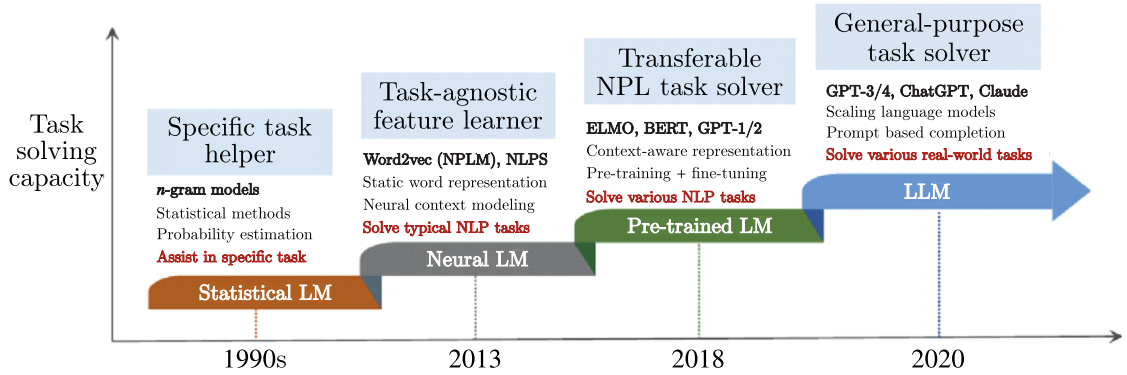


Figure 1. The evolution of language models across four generations, highlighting task-solving capabilities [Zhao et al., 2023]

By exploring the impact of LLMs on NLP tasks, it is clear that these findings provide transformative benefits in many areas. With their extensive prior training in diverse groups, LLM holders demonstrate strong abilities in understanding and generating human expressions, and thus language understanding and generation tasks. Specifically, the greatly expanded LLM, which has achieved success through its large model size, is highly optimized for linguistic programming tasks. It has a large volume of more parameters, allowing it to accurately capture and reproduce subtle linguistic features in tasks such as language translation, sentiment analysis, question answering, and text. Additionally, MBA workers have been highlighted for revolutionizing applications such as login software, virtual assistants and translation systems, promising more natural and engaging user experiences. LLM's impact scale has ushered in a new era in AI algorithm development, identifying important and most likely non-NLP tasks for innovation and progress. Important benefits such as language translation, sentiment analysis, and question answering include particularly notable texts from the larger MBA, demonstrating the ability to make impacts across diverse applications and domains. Notwithstanding this, significant research and exploration remain to address the remaining challenges in the field of LLM towards new frontiers of sole discovery. Global language volumes (LLMs) have become prominent in a variety of applications. In the field of language translation, they go a long way in dramatically improving the performance of translation software by capturing subtle linguistic features and producing more distinctive translations. Likewise, in sentiment analysis, larger language models show greater understanding and analysis of the poet in the text, leading to better results on sentiment analysis tasks. Furthermore, in the question-answering task, MBA majors show an improved understanding of the micro-advertising questions, thus increasing the overall accuracy of answering

questions. In addition, larger LLM programs excel at creating text assignments, producing performative and contextually connected text suitable for use in applications such as chatbots, virtual assistants, and content. Overall, it is a significant expansion of cutting-edge MBAs (LLMs) for performance in various related fields, making it the most important in natural and industrial technological applications. Figure 2 presents the applications of LLMs in research directions, options, and downstream fields [Zhao et al., 2023].

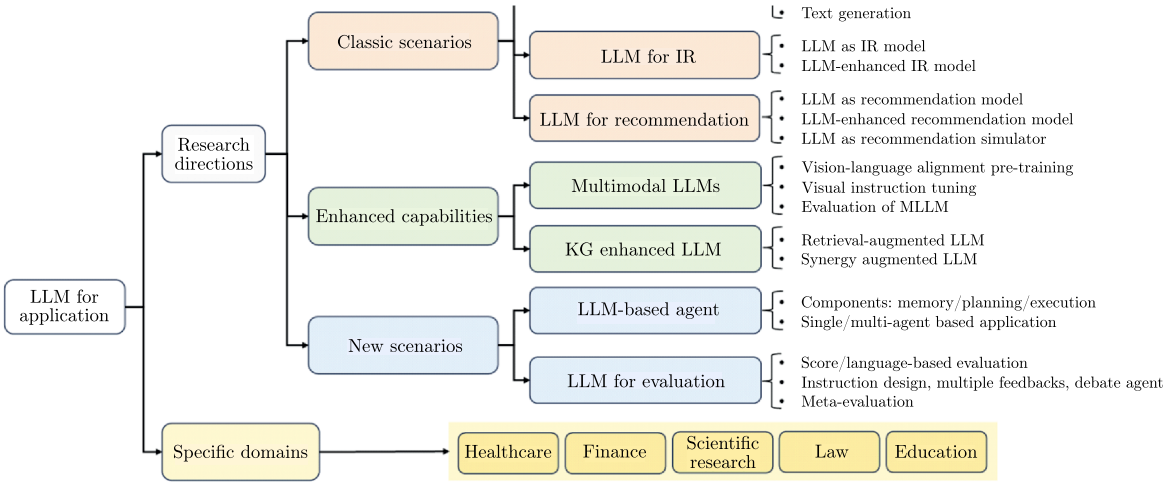


Figure 2. Applications of LLMs in various research directions and industrial fields [Zhao et al., 2023]

LLMs' ability to handle large volumes of data and generate contextually appropriate text makes them invaluable in a range of applications. Their performance in tasks such as language translation and sentiment analysis demonstrates their capacity to improve accuracy and produce more refined outputs. Furthermore, their proficiency in generating diverse and creative text has made LLMs indispensable in fields like content creation, virtual assistants, and beyond.

Overall, LLMs have ushered in a new era of AI development, presenting remarkable opportunities across both natural language and non-NLP tasks. The research community continues to explore new frontiers, seeking to address the challenges and unlock the full potential of these advanced models.

Literature review

LLMs have been successfully utilized in many research areas within the software engineering field. In this section, an overview of existing studies employing LLMs for Software Engineering (SE) is provided. These studies are categorized into four main phases of the Software Development Life Cycle (SDLC): software requirements and design, software development, software testing, and software maintenance. Each phase involves various code-related tasks, such as fault localization and program repair in the software maintenance phase.

The proposed ERD bot

The proposed model, **ERD bot**, can read any system requirement specifications in either text or Entity-Relationship Diagrams (ERD) and analyze them to generate a relational database schema script in SQL. Figure 3 illustrates the working framework of the proposed ERD bot.

The ERD bot uses the Telegram messenger to input system requirement specifications in two formats: human-written text (Fig. 4, *a*) and an ERD image (Fig. 4, *b*). The bot analyzes these inputs

Table 1. Using LLMs in software requirements

Specific field	Reference	Description
Software Requirements Generation	[Xie et al., 2023]	The study provides suggestions for enhancing the performance of LLMs in specification generation. It includes exploring hybrid approaches and improving the effectiveness of prompts used in LLM-based processes.
Software Specifications Repair	[Hasan et al., 2023]	A study evaluates ChatGPT's use in repairing software specifications in Alloy language. ChatGPT's performance is compared against existing automated repair methods, revealing both strengths and areas for improvement.
Categorizing Software Requirements	[Hey et al., 2020]	NoRBERT efficiently classifies types of requirements using transfer learning. The functional section of the NFR dataset is labeled into classes like Function, Data, and Behavior.
GUI Layouts	[Brie et al., 2023]	Investigates whether LLM-based systems can enhance the design process of GUI layouts. The study shows that Instigator, an LLM-based system, parses the code of over 100k websites to create relevant GUI layouts.

Table 2. Using LLMs in software development

Specific field	Author(s)	Description
Code Generation	[Li et al., 2023; Zhang et al., 2023]	The study [Li et al., 2023] uses AceCoder utilizes requirement-guided generation combined with example retrieval to improve code understanding and implementation, and the work [Zhang et al., 2023] introduce a method called Self-Edit, which enhances code quality for competitive programming tasks by utilizing execution results of generated code from LLMs. The core of this approach is a fault-aware code editor that can edit and optimize the generated code.
Code Summarization	[Sun et al., 2023]	ChatGPT's performance in code summarization is evaluated, with comparisons against NCS, CodeBERT, and CodeT5 models. While ChatGPT shows deeper semantic understanding, it struggles with complex code logic.
Code Search	[Li et al., 2022a]	Code Retriever learns semantic representations of code functions using a large dataset of code-text pairs, excelling in various code search tasks across multiple languages.
Code Translation	[Baltaji et al., 2023]	A transformer-based LLM is used for cross-lingual code translation, revealing practical insights on language transfer, with Kotlin and JavaScript showing high transferability.

using an LLM to generate the corresponding relational database schema script in SQL. The results are then returned to the user through Telegram.

The second step involves analyzing the input data to generate SQL commands in the ERD bot using an LLM. When a user sends a natural language query, the bot forwards it to the LLM, which processes the query to generate the appropriate SQL commands that create the relational schema. The database schema is then returned to the user via Telegram.

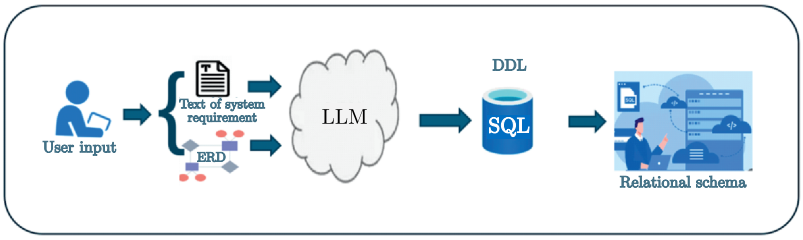


Figure 3. The working framework of the proposed ERD bot

Table 3. Using LLMs in software testing

Specific field	Reference	Description
Fault Localization	[Mohsen et al., 2023]	Introduces a phase-based bug localization method using BERT for package classification and source code recommendation.
Vulnerability Detection	[Steenhoek et al., 2023]	Empirical study on LLM-based models for identifying software vulnerabilities, analyzing nine approaches including two LLM-based methods.
Unit Test Generation	[Tang et al., 2023]	Compares ChatGPT's ability to generate unit tests against the SBST tool EvoSuite, demonstrating that ChatGPT generates executable unit tests with significant success.
GUI Testing	[Liu et al., 2023b]	GPTDroid reframes the GUI testing challenge as a question-answer task using LLMs, improving activity coverage by 32 % and identifying numerous bugs.

Table 4. Using LLMs in software maintenance

Specific field	Reference	Description
Automated Program Repair	[Jiang et al., 2023]	Evaluates LLMs on different Program Repair (PR) benchmarks, showing that fine-tuned LLMs can fix more bugs than traditional deep learning methods.
Patch Correctness Assessment	[Tian et al., 2023]	Explores the use of representation learning models to predict the correctness of patches generated by PR tools, achieving high recall and precision.
Code Review	[Li et al., 2022b]	CodeReviewer, a Transformer-based model inspired by CodeT5, automates code reviews by understanding code differences and generating relevant review comments.
Test Update	[Hu et al., 2023]	CEPROT identifies and updates outdated test cases, achieving impressive precision, recall, and F1 score in identifying obsolete tests.

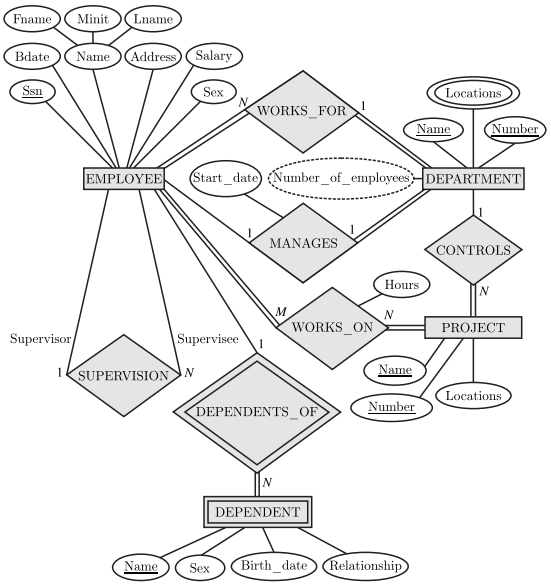
Results and discussion

To evaluate the performance and accuracy of the proposed tool, we tested it on five different database schemas. The first database schema is from the University of Jordan's ACAD system, which assists academic staff in managing semester tasks. These tasks include exam-related operations such as converting percentages to points, transferring, and updating marks, which require approvals from instructors, department heads, and deans. Furthermore, the system generates inquiries and reports

Consider a CONFERENCE_REVIEW database in which reserchers submit their research papers for consideration. Reviews by reviewers are recorded for use in the paper selection process. The database system caters primarily to reviewers who record answers to evaluation questions for each paper they review and make recommendations regarding whether to accept or reject the paper. The data requirements are summarized as follows:

- Authors of papers are uniquely identified by e-mail id. First and last names are also recorded.
- Each paper is assigned a unique identifier by the system and is described by a title, abstract, and the name of the electronic file containing the paper.
- A paper may have multiple authors, but one of the authors is designated as the contact author.
- Reviewers of papers are uniquely identified by e-mail address. Each reviewer's first name, last name, phone number, affilation, and topics of interest are also recorded.
- Each paper is assigned between two and four reviewers. A reviewer rates each paper assigned to him or her on a scale of 1 to 10 in four categories: technical merit, readability, originality, and relevance to the conference. Finally, each reviewer provides an overall recommendation regarding each paper.
- Each review contains two types of written comments: one to be seen by the review committee only and the other as feedback to the author(s).

(a)



(b)

Figure 4. (a) Text-based system requirement specifications input, (b) ERD image input

covering registered students, instructors, departments, and student numbers per course. It also manages academic advising, reassignments, and other advising-related activities.

The second database schema pertains to a conference attendance system at Mutah University. In this system, teachers submit conference attendance applications with details such as sponsorship and fees. The application undergoes review at multiple levels, from the department head to the deanship of scientific research. Notifications are sent via SMS at various stages of the review process, making the system dynamic and interactive.

We compared key metrics such as the number of tables, fields per table, and relationships between tables in the real databases with those generated by the proposed tool. The tool generally generated a greater number of tables compared to the actual databases. This is attributed to its ability to perform more accurate normalization of tables. The results are shown in Table 5.

Table 5. Comparison between real database system specification and proposed tool database system specification

Reference	Input	Metrics/Parameters	Real DB system	Proposed tool
Company schema [Elmasri, 2021]	ERD	# Entity # Relationship # Attribute	6 8 28	9 8 34
University schema [Elmasri, 2021]	Text for system requirements	# Entity # Relationship # Attribute	7 9 34	7 6 37
Conference system from Mutah University	Text for system requirements	# Entity # Relationship # Attribute	8 11 52	11 12 48
Bank system [Scaler, 2023]	Text for system requirements	# Entity # Relationship # Attribute	9 12 36	8 8 41
ACAD system from University of Jordan	Text for system requirements	# Entity # Relationship # Attribute		14 18 44

The proposed tool offers significant advantages for systems analysts and software engineers by aiding in the accurate construction of databases. This tool automates the processes of building tables, defining attributes, identifying relationships between tables, and applying constraints, thereby ensuring efficient and accurate database design. This automation results in reduced time, effort, and costs associated with database construction, leading to the simultaneous generation of reliable and optimized database schemas.

Conclusion and future works

In summary, the proposed tool is a valuable asset for systems analysts and software engineers, enhancing their ability to understand system requirements and facilitating the accurate development of databases. By automating tasks such as table creation, attribute specification, relationship definition, and constraint enforcement, the tool greatly simplifies the database construction process. This results in significant time, effort, and cost savings when compared to manual methods. Moreover, it ensures a high level of accuracy, leading to more reliable and efficient database management systems. Looking ahead, our goal is to enhance this tool by incorporating the ability to generate UML diagrams. UML diagrams, such as class diagrams, sequence diagrams, and use case diagrams, provide visual representations of system architectures, helping developers and stakeholders better understand the relationships between entities and the data flow within the system. By utilizing UML diagrams, the interpretation of complex database structures becomes more unified and accurate, ultimately improving the overall development and maintenance process.

References

Baltaji R., Pujar S., Mandel L., Hirzel M., Buratti L., Varshney L. Learning transfers over several programming languages // arXiv preprint. — 2023. — arXiv:2310.16937

Brie P., Burny N., Sluÿters A., Vanderdonckt J. Evaluating a large language model on searching for gui layouts // Proceedings of the ACM on Human-Computer Interaction. — 2023. — Vol. 7, No. EICS. — P. 1–37.

Burgin M. Mathematical models in schema theory // arXiv preprint. — 2005. — arXiv:cs/0512099

Candel C. J. F., García-Molina J. J., Ruiz D. S. Skiql: A unified schema query language // Data & Knowledge Engineering. — 2023. — Vol. 148. — P. 102234.

Chowdhary K., Chowdhary K. R. Natural language processing // Fundamentals of artificial intelligence. — 2020. — P. 603–649.

Elmasri R. Fundamentals of database systems. — 7th edition. — 2021.

Hadi M. U., Qureshi R., Shah A., Irfan M., Zafar A., Shaikh M. B., Mirjalili S. A survey on large language models: Applications, challenges, limitations, and practical usage // Authorea Preprints. — 2023.

Hasan Md. R., Li J., Ahmed I., Bagheri H. Automated repair of declarative software specifications in the era of large language models // arXiv preprint. — 2023. — arXiv:2310.12425

Hey T., Keim J., Koziolek A., Tichy W. F. Norbert: Transfer learning for requirements classification // 2020 IEEE 28th international requirements engineering conference (RE). — 2020. — P. 169–179.

Hu X., Liu Z., Xia X., Liu Z., Xu T., Yang X. Identify and update test cases when production code changes: A transformer-based approach // 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). — 2023. — P. 1111–1122.

Jiang N., Liu K., Lutellier T., Tan L. Impact of code language models on automated program repair // 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). — 2023. — P. 1430–1442.

Jin Z., Baik C., Cafarella M., Jagadish H. V. Lou Y. Demonstration of a multiresolution schema mapping system // arXiv preprint. — 2018. — arXiv:1812.07658

Kim W., Kanezaki A., Tanaka M. Unsupervised learning of image segmentation based on differentiable feature clustering // IEEE Transactions on Image Processing. — 2020. — Vol. 29. — p. 8055–8068.

Li J., Zhao Y., Li Y., Li G., Jin Z. Acecoder: Utilizing existing code to enhance code generation // arXiv preprint. — 2023. — arXiv:2303.17780

Li X., Gong Y., Shen Y., Qiu X., Zhang H., Yao B., Qi W., Jiang D., Chen W., Duan N. Coderetriever: A large scale contrastive pre-training method for code search // Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing. — 2022a. — P. 2898–2910.

Li Z., Lu S., Guo D., Duan N., Jannu S., Jenks G., Sundaresan N. Automating code review activities by large-scale pre-training // Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. — 2022b. — P. 1035–1047.

Liu R., Yang R., Jia C., Zhang G., Zhou D., Dai A. M., Yang D., Vosoughi S. Training socially aligned language models in simulated human society // arXiv. — 2023a. — https://arxiv.org/abs/2305.16960

Liu Z., Chen C., Wang J., Chen M., Wu B., Che X., Wang Q. Make LLM a testing expert: Bringing human-like interaction to mobile GUI testing via functionality-aware decisions // arXiv preprint. — 2023b. — arXiv:2310.15780

Manhaeve R., Dumančić S., Kimmig A., Demeester T., De Raedt L. Neural probabilistic logic programming in deepproblog // Artificial Intelligence. — 2021. — Vol. 298. — p. 103504.

Mohsen A. M., Hassan H., Wassif K., Moawad R., Makady S. Enhancing bug localization using phase-based approach // IEEE Access. — 2023.

Pieris D., Wijegunesekera M. C., Dias N. G. ER model partitioning: Towards trustworthy automated systems development // arXiv preprint. — 2020. — arXiv:2007.00999

Proper H. A., Halpin T. A. Conceptual schema optimisation – database optimisation before sliding down the waterfall // arXiv preprint. — 2021. — arXiv:2105.12647

Scaler. ER diagram for bank database. — [Electronic resource]. — 2023. — https://www.scaler.com/topics/er-diagram-for-bank-database/

Steenhoek B., Rahman M. M., Jiles R., Le W. An empirical study of deep learning models for vulnerability detection // 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). — 2023. — P. 2237–2248.

Sumers T. R., Yao S., Narasimhan K., Griffiths T. L. Cognitive architectures for language agents // arXiv. — 2023. — https://arxiv.org/abs/2309.02427

Sun W., Fang C., You Y., Miao Y., Liu Y., Li Y., Chen Z. Automatic code summarization via chatgpt: How far are we? // arXiv preprint. — 2023. — arXiv:2305.12865

Tang Y., Liu Z., Zhou Z., Luo X. Chatgpt vs sbst: A comparative assessment of unit test suite generation // arXiv preprint. — 2023. — arXiv:2307.00588

Teniente E., Urpi T. On the abductive or deductive nature of database schema validation and update processing problems // Theory and Practice of Logic Programming. — 2003. — Vol. 3, No. 3. — P. 287–327.

Tian H., Liu K., Li Y., Kaboré A. K., Koyuncu A., Habib A., Bissyandé T. F. The best of both worlds: Combining learned embeddings with engineered features for accurate prediction of correct patches // ACM Transactions on Software Engineering and Methodology. — 2023. — Vol. 32, No. 4. — P. 1–34.

Trummer I. Can deep neural networks predict data correlations from column names? // arXiv preprint. — 2021. — arXiv:2107.04553

Weng L. LLM-powered autonomous agents // 2023. — lilianweng.github.io

Xie D., Yoo B., Jiang N., Kim M., Tan L., Zhang X., Lee J. S. Impact of large language models on generating software specifications // arXiv preprint. — 2023. — arXiv:2306.03324

Zhang K., Li Z., Li J., Li G., Jin Z. Self-edit: Fault-aware code editor for code generation // arXiv preprint. — 2023. — arXiv:2305.04087

Zhao W. X., Zhou K., Li J., Tang T., Wang X., Hou Y., Min Y., Zhang B., Zhang J., Dong Z., Du Y., Yang C., Chen Y., Chen Z., Jiang J., Ren R., Li Y., Tang X., Liu Z., Liu P., Nie J.-Y., Wen J.-R. A survey of large language models // arXiv preprint. — 2023. — arXiv:2303.18223