"Doc2DB-Gen": Multimodal LLM System that Converts PDFs, Tables & Images into Normalized Databases

Oleksii Rubezhanskyi
Wydział Informatyki, Grafiki i Architektury
Akademia Finansów i Biznesu Vistula
Warszawa, Polska
Student ID:77905
ORCID:0009-0000-9132-7987
77905@office.mans.org.pl

Andriy Nikolayuk
Wydział Informatyki, Grafiki i Architektury
Akademia Finansów i Biznesu Vistula
Warszawa, Polska
Student ID:77600
ORCID:0009-0009-0770-643X
77600@office.mans.org.pl

Andrii Borovyi
Wydział Informatyki, Grafiki i Architektury
Akademia Finansów i Biznesu Vistula
Warszawa, Polska
Student ID:77849
ORCID:0009-0009-6149-5039
77849@office.mans.org.pl

Mohcine Baloul
Wydział Informatyki, Grafiki i Architektury
Akademia Finansów i Biznesu Vistula
Warszawa, Polska
Student ID:77532
ORCID:0009-0002-2443-8210
77532@office.mans.org.pl

Kumar Nalinaksh
Wydział Informatyki, Grafiki i Architektury
Akademia Finansów i Biznesu Vistula
Warszawa, Polska
kumar.nalinaksh@office.mans.org.pl

***Abstract*—Despite significant progress in multimodal AI for document understanding, there is still no fully integrated system capable of transforming unstructured documents into normalized relational databases. Doc2DB-Gen is a small document-to-database pipeline that helps converting uploaded files (PDFs, images, and spreadsheets) into a normalized relational database. The user uploads a file, then the backend extracts a schema, generates an ER diagram (Mermaid text) and SQL DDL, and after that it applies the schema and shows a database preview in the UI.**

***Index Terms*—document understanding, multimodal LLM, schema extraction, ER diagram, SQL DDL, FastAPI**

I. Introduction

Motivation: Many companies and researchers store important information inside documents such as PDFs, scanned images, and spreadsheets. Even when these files contains tables and structured facts, it is still hard to use them for analytics, because the data is not inside a relational database and it is not easy to be a query. In real work, people often needs to manually read a document, understand the entities and columns, and then design tables and insert the data. This process is slow and it can produces mistakes. Because of this, there is motivation to create a tool that can help converting documents into a database automatically, with less manual effort.

Contributions: In this paper it's presented *Doc2DB-Gen*, a document-to-database pipeline implemented as a small full-stack web application. The system:

- uses a `FastAPI` backend and a lightweight web interface.
- integrates an LLM through the OpenAI API to extract entities, relationships, and possible table rows from uploaded files.
- generates an ER diagram (Mermaid text) and SQL DDL based on extraction.
- applies the schema automatically to a project database.
- provides a database preview in the UI.

Paper organization: The rest of this paper are organized as follows. Section II present the *Literature Review* and summarize the main related works about document understanding, table extraction, and schema reasoning. Section III describe the key *Challenges* that appears when to transform heterogeneous documents into normalized relational databases. After that, Section IV explains the *Proposed Solution* and introduces the Doc2DB-Gen pipeline, including the main processing steps and used components. Section V describes the *Implementation* of the project. Next, Section VI reports the *Results*. Section VII discusses the *Future Work* and gives directions for improvement. Finally, Section VIII provide the *Conclusion* of this paper.

II. Literature Review

Recent progress in multimodal large language models (LLMs) has significantly improved document interpretation by jointly modeling textual, visual, and spatial information. LayoutLM introduced multimodal pretraining that integrates text tokens with two-dimensional layout embeddings, enabling improved performance in tasks such as form understanding and invoice processing [1]. LayoutLMv2 extends this approach by incorporating image features and spatially aware attention mechanisms, further improving extraction accuracy and document reasoning [2]. LayoutLMv3 unified text and image masking during pretraining, demonstrated improved generalization across various document understanding tasks [3]. In contrast to OCR-dependent approaches, Donut proposed an end-to-end transformer architecture capable of generating structured outputs directly from document images [4]. Other multimodal models such as DocFormer and StrucTexT further enhanced extraction of key-value pairs and relationships by integrating textual, visual, and structural representations [5], [6]. While these studies demonstrate strong capabilities in information extraction, they typically do not address the transformation of extracted data into normalized relational database schemas, which is still missing in practice.

Several widely known benchmark datasets have supported the development of models that understand documents. For example, the FUNSD dataset provides annotated forms for entity and relation extraction [7]. The CORD dataset offers structured annotations for receipt parsing and line-item extraction [8]. The DocVQA benchmark evaluates documents through visual question answering, measuring the ability of models to query document content based on layout and textual information [9]. Although these datasets provide valuable evaluation resources, they often focus on specific document types and may not fully represent diverse real-world business documents, especially when the documents are noisy or messy.

Another key direction involves table detection and structural reconstruction, which is critical for converting document information into structured formats. Deep learning approaches such as TableNet and DeepDeSRT demonstrate effective table detection and structure recognition from document images [10], [11]. Subsequent models, including CascadeTabNet and Table Transformer, improved detection accuracy and structural reconstruction using advanced architectures [12], [14]. Large-scale datasets such as SciTSR and PubTables-1M enabled more comprehensive training and evaluation of table extraction systems [13], [15]. Despite this progress, extracting complex tables with unusual layouts, merged cells, and domain-specific structures remains challenging and does not always produce correct outputs. In addition, classic document image understanding tasks such as classification and retrieval also remain important as a supporting layer for many pipelines; Harley et al. evaluated deep convolutional networks for document image classification and retrieval and showed that CNN-based representations can be effective for document images [16]. Related to this direction, the ICDAR 2019 Scene Text Visual Question Answering competition (ST-VQA) also evaluates question answering over images that contain text, and it is often used as an OCR/VQA benchmark for reasoning about scene text [17].

In addition to extraction, transforming document-derived data into usable relational databases requires schema reasoning and evaluation of queryability. The Spider dataset introduced a large-scale benchmark for cross-domain text-to-SQL generation [18], while RAT-SQL improved schema-aware encoding to enhance query generation accuracy [19]. PICARD further improved SQL validity by applying constrained decoding during query generation [20]. In database research, automated discovery of functional dependencies and normalization has been explored through frameworks such as Metanome and algorithms like TANE, which support schema refinement and relational consistency [21], [22]. However, these methods generally assume structured and relatively clean input data, which is not always the case for document-derived information and can lead to failures.

Recent studies have also investigated the integration of large language models with external tools and multimodal reasoning. Toolformer demonstrated that language models can learn to invoke external tools during inference [23], while ReAct introduced a reasoning-and-action framework for controlled task execution [24]. Vision-language models such as LLaVA and BLIP-2 further expanded multimodal reasoning by combining visual encoders with large language models [25], [26]. Although these approaches provide flexible multimodal capabilities, they are not specifically designed for end-to-end document-to-database transformation and still require many manual steps.

III. Challenges

Based on research conducted in Section II the authors identify crucial research gaps:

- Absence of end-to-end integration: Existing approaches typically focus on text, layout or table extraction, but do not provide a complete solution that automatically generates normalized relational schemes.
- Limited schema and normalization reasoning: Current models often fail to correctly infer primary keys, foreign keys and dependencies required for proper normalization and data consistency.
- Weak robustness on real-world documents: Many methods are evaluated on clean and domain-specific datasets (forms, receipts, scientific tables etc) and may perform poorly on messy, different business documents with complex layouts.
- Lack of controlled and safe execution: Some LLM-based systems generate database schemas and SQL queries without strong validation, which can lead to incorrect or unsafe database operations.
- Insufficient database-level evaluation: Prior work mainly measures extraction accuracy, while rarely evaluating normalization quality, relational integrity and practical queryability of the generated database.

IV. Proposed Solution

To address research gaps identified Section III the proposal is *Doc2DB-Gen* that is document-to-database pipeline that converts uploaded files (PDF, images, spreadsheets) into a normalized relational database. The system works as a sequence of steps: the user uploads a file, the backend extracts a schema, it generates an ER diagram and SQL DDL, then it apply the schema and show a database preview in the UI. A key element of the solution is using an OpenAI API key stored in the backend `.env` file. The backend reads this key from environment settings and uses it to call the LLM for schema extraction. If the key is missing or invalid, the system returns an authentication error, and if the API quota/rate limit is exceeded, the system returns a `429` error.

In terms of data processing, the system supports two extraction modes:

1) Image/PDF mode: The PDF is converted to an image (first page) and then processed by a vision-capable model thruogh the OpenAI API.
2) Text/spreadsheet mode: CSV/TXT is parsed into rows and XLSX is read via `openpyxl`, then the content is sent as text to the LLM, and the system also tries to create `table_data` from file rows when possible.

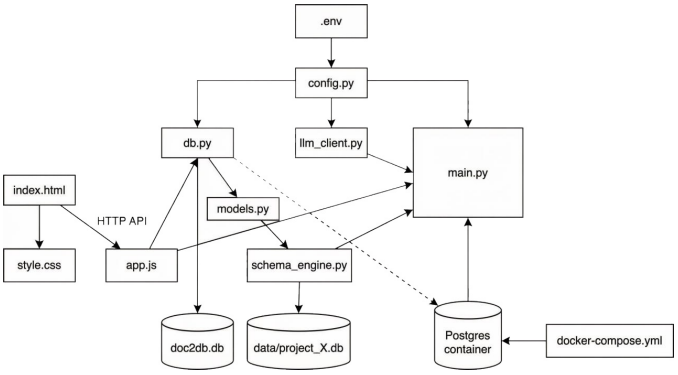

Fig. 1. Doc2DB-Gen system architecture and module dependencies

The output of extraction is a JSON structure with entities, relationships, and `table_data` (rows) that later can populate the created tables.

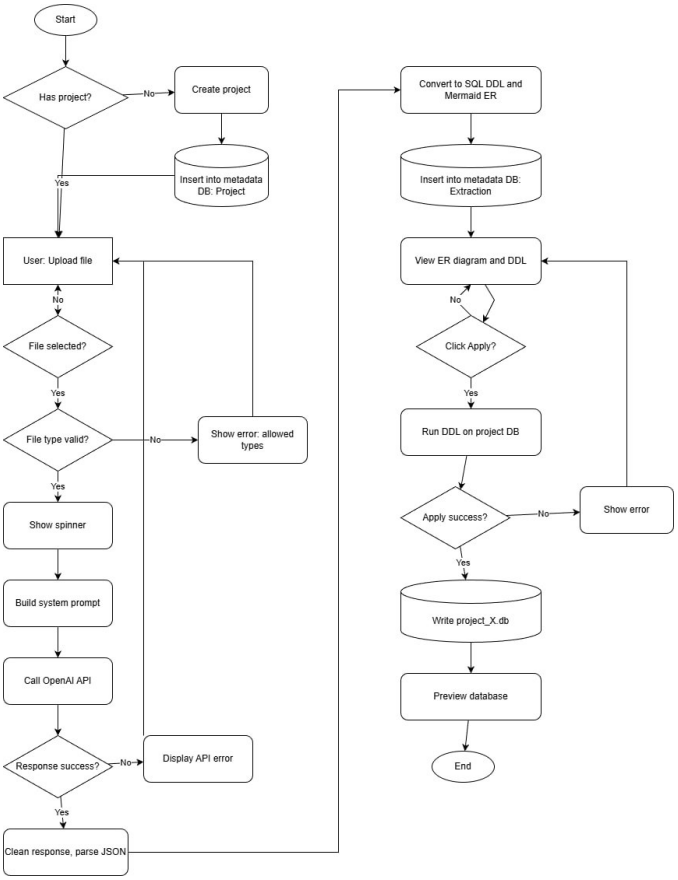

Fig. 2. Doc2DB-Gen workflow

V. Implementation

Doc2DB-Gen is implemented as a small full-stack web application. The backend is written in `FastAPI` and exposes REST endpoints for creating projects, uploading files, extracting schema, applying schema, previewing the database, and health monitoring.

Backend layer: The backend uses:

- FastAPI + Uvicorn for running the server and serving API endpoints.
- OpenAI SDK with `AsyncOpenAI` to call a chat completion model (`gpt-4o-mini`) for vision-based schema extraction.
- Pydantic settings + dotenv to load `.env` from the backend folder (not depending on the current working directory).
- SQLAlchemy (async) for the metadata database (projects and extractions).

The API flow is implemented as:

- `POST /api/projects` creates a new project in the metadata DB and returns `project_id`.
- `POST /api/upload` stores the file in `./uploads` with a `project_id` prefix and checks the maximum size (`MAX_UPLOAD_MB`).
- `POST /api/extract` reads the file, converts PDF to an image if needed, calls LLM extraction, generates ER Mermaid and SQL DDL, and stores results in the `Extraction` table (including `extraction_data` as JSON).
- `POST /api/apply-schema` runs DDL and inserts extracted rows into a project-specific SQLite database stored in `./data/project_<id>.db`.
- `GET /api/preview/<project_id>` returns the table list and sample rows for the UI preview.
- `GET /api/health` checks if the OpenAI key exists and if the DB session can execute `SELECT 1`.

An important detail is that the system has two database concepts:

- a metadata DB configured by `DATABASE_URL` (SQLite by default, PostgreSQL optional), and
- a target DB used for extracted data preview, which is a SQLite file per project stored in `./data/`.

Frontend layer: The frontend is a simple HTML/CSS/JS page served from the same origin (`http://localhost:8000`) to avoid CORS issues. It creates a project, uploads a file, calls extraction, shows the ER/DDL output, applies the schema, and previews the database.

There is also a helper script `start_server.ps1` which tries to free port `8000` and then runs `uvicorn` using the `.venv` Python interpreter.

As shown in Fig. 3 and Fig. 4 *Doc2DB-Gen* web interface running the full pipeline from an uploaded document to a database preview:

1) Create project & upload (Fig. 3)
   - The user clicks “New project” (it creates a new project, *Project #23*).
   - The user selects a file (`book_list.pdf`) and clicks “Upload file”.
   - The UI shows the status “Uploaded: book_list.pdf”.
2) Extract schema (LLM) (Fig. 3)

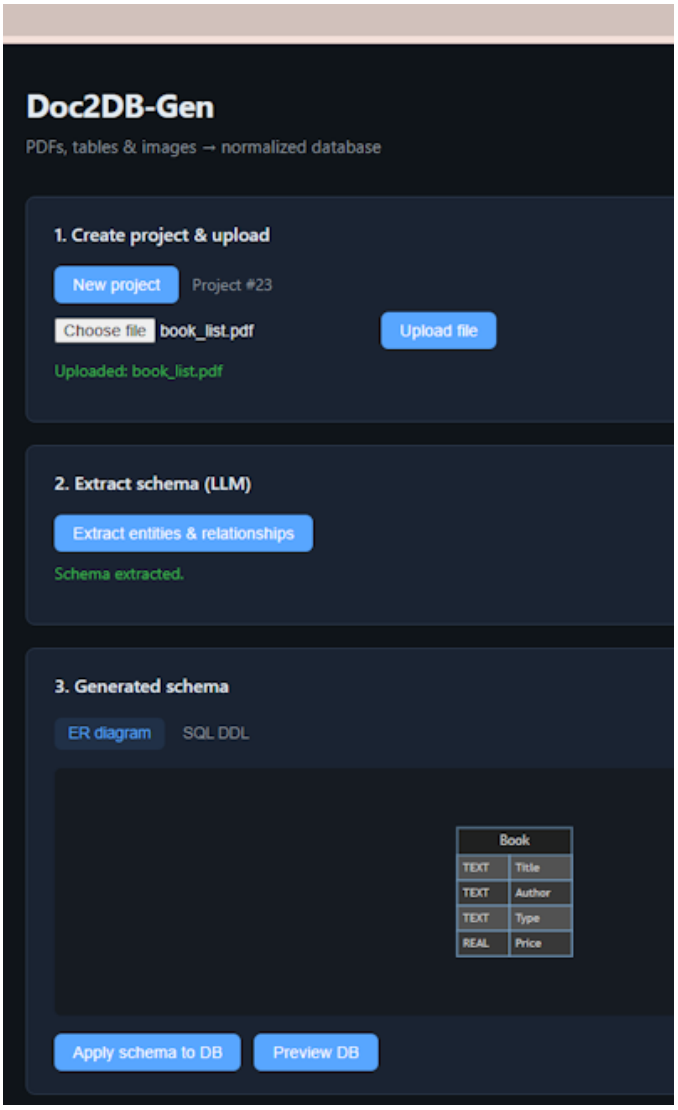

Fig. 3. Doc2DB-Gen ER diagram

- The user clicks "Extract entities & relationships".
- The backend sends the document content to the LLM and extracts a structured result (entities, fields, and possible rows).
- The UI confirms with "Schema extracted."

3) Generated schema
   - The UI can show two outputs:
     - ER diagram (schema visualization / Mermaid text),
     - SQL DDL (generated `CREATE TABLE` statements).
   - As shown in Fig. 3, the system generated a table `Book` with columns like *Id, Title, Author, Type, Price*.
4) Apply schema to DB
   - When the user clicks "Apply schema to DB" (Fig. 3), the backend executes the generated SQL DDL and creates the tables in the project database.
   - It also inserts extracted rows (if table data was extracted).
5) Preview DB
   - When the user clicks "Preview DB" (Fig. 4), the UI displays a Database preview: a table view with sample rows.
   - In Fig. 4 list of books is shown (*1984*, *To Kill a Mockingbird*, *The Hobbit*, etc.) with their extracted fields and values.

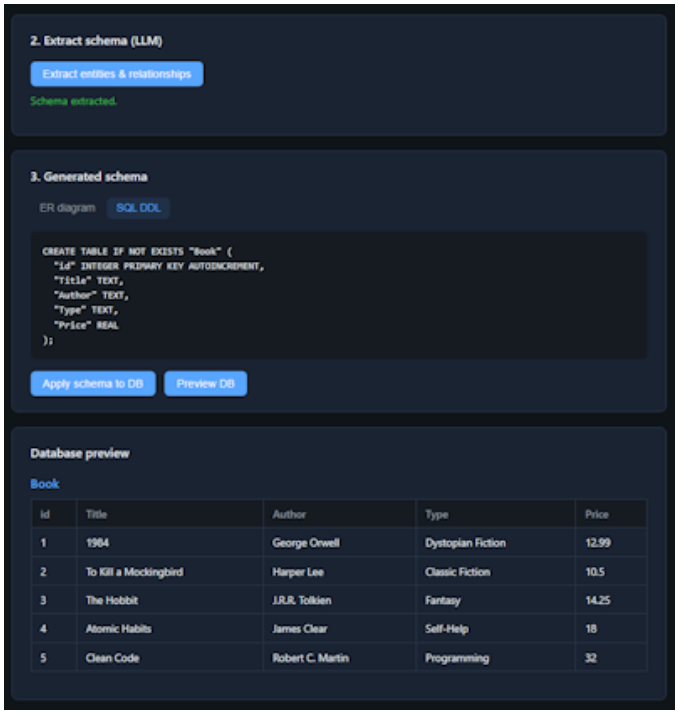

Fig. 4. Doc2DB-Gen SQL DDL

VI. Results

The implemented system provides a working end-to-end pipeline from document upload to database preview. In a normal run, a user can create a project, upload a file, call extraction, and obtain:

- an ER diagram (Mermaid text), and
- SQL DDL (`CREATE TABLE` statements) generated from extracted entities and relationships, and optionally extracted rows as `table_data`.

After pressing "Apply schema to DB", the system executes the generated DDL and inserts extracted rows into the project SQLite database. Then the user can preview tables and a few sample rows (the default limit is 5).

From the robustness side, the backend includes user-friendly error mapping:

- missing or invalid OpenAI key triggers `401` "Invalid or missing `OPENAI_API_KEY`",
- API quota/rate limit triggers `429` "OpenAI rate limit or quota exceeded",

which is important for real usage and debugging.

A separate test script (`test_flow.py`) demonstrates the full flow automatically: create project → upload PDF → extract → apply schema → preview, so the pipeline can be tested without manual UI steps.

VII. Future Work

Future iterations of this paper will focus on consolidating the data architecture by moving from isolated database files to a unified storage system with `project_id` tagging, facilitating more effective cross-project analysis. To improve the reliability and cost-efficiency of LLM-based extraction, it is planned to implement exponential backoff retries, result caching, and optimized document chunking. Furthermore, the evaluation module will be expanded from its current stub state into a robust benchmarking suite to systematically measure extraction accuracy against ground truth schemas. Finally, the user experience will be enhanced through an interactive UI for schema editing, allowing users to manually refine entity names and relationships before the final DDL application.

VIII. Conclusion

Paper presented *Doc2DB-Gen*, a practical pipeline that converts documents (PDFs, images, spreadsheets) into a normalized relational database by using a `FastAPI` backend and a lightweight web interface. The system uses the OpenAI API to extract entities, relationships, and table rows, and then it automatically generates ER diagram text and SQL DDL. The implementation shows that LLM-assisted schema generation can reduce the manual work for database design and makes document data quickly queryable thruogh an automatically created project database.

Acknowledgment

All authors participated in writing and approved the final manuscript.

1) Funding: Not applicable.
2) Conflicts of interest: None.
3) Ethics approval and consent: Not applicable.
4) Consent for publication: All authors have provided consent.
5) Data availability: Publicly available as indicated in the relevant section.
6) Code availability: https://github.com/MohcineBaloul/doc2db-final

References

[1] Y. Xu, M. Li, L. Cui, S. Huang, F. Wei, and M. Zhou, "LayoutLM: Pre-training of Text and Layout for Document Image Understanding," in Proc. ACM SIGKDD, 2020.

[2] Y. Xu, Y. Li, T. Zhang, et al., "LayoutLMv2: Multi-modal Pre-training for Visually-rich Document Understanding," in Proc. ACL, 2021.

[3] Y. Huang, T. Lv, L. Cui, Y. Lu, and F. Wei, "LayoutLMv3: Pre-training for Document AI with Unified Text and Image Masking," arXiv preprint arXiv:2204.08387, 2022.

[4] G. Kim, T. Hong, M. Yim, et al., "Donut: Document Understanding Transformer without OCR," in Proc. ECCV, 2022.

[5] S. Appalaraju, S. S. Datta, Y. N. Murthy, and R. T. Iyer, "DocFormer: End-to-End Transformer for Document Understanding," arXiv preprint arXiv:2106.11539, 2021.

[6] Z. Li, W. Chen, X. Xu, et al., "StrucTexT: Structured Text Understanding with Multi-Modal Transformers," arXiv preprint arXiv:2108.02923, 2021.

[7] G. Jaume, H. K. Ekenel, and J. Thiran, "FUNSD: A Dataset for Form Understanding in Noisy Scanned Documents," in Proc. ICDAR Workshops, 2019.

[8] S. Park, S. Shin, B. Lee, et al., "CORD: A Consolidated Receipt Dataset for Post-OCR Parsing," in Proc. ICDAR, 2019.

[9] M. Mathew, D. Karatzas, and C. V. Jawahar, "DocVQA: A Dataset for VQA on Document Images," in Proc. WACV, 2021.

[10] A. Paliwal, D. Vishwanath, R. Rahul, M. Sharma, and L. Vig, "TableNet: Deep Learning Model for End-to-End Table Detection and Tabular Data Extraction from Scanned Document Images," in Proc. ICDAR, 2019.

[11] S. Schreiber, S. Agne, I. Wolf, A. Dengel, and S. Ahmed, "DeepDeSRT: Deep Learning for Detection and Structure Recognition of Tables in Document Images," in Proc. ICDAR, 2017.

[12] P. Prasad, A. Sarkar, M. P. K. Reddy, and M. N. S. Swamy, "CascadeTabNet: An Approach for End to End Table Detection and Structure Recognition from Image-Based Documents," in Proc. CVPR Workshops, 2020.

[13] Z. Chi, H. Huang, H. Yu, et al., "SciTSR: Extracting Table Structure from Scientific Tables," in Proc. ICDAR, 2019.

[14] B. Smock and R. Abraham, "Table Transformer: A Transformer-based Approach to Table Detection and Structure Recognition," arXiv preprint arXiv:2110.00061, 2021.

[15] B. Smock, R. Pesala, and R. Abraham, "PubTables-1M: Towards Comprehensive Table Extraction from Unstructured Documents," in Proc. CVPR, 2022.

[16] A. W. Harley, A. Ufkes, and K. G. Derpanis, "Evaluation of Deep Convolutional Nets for Document Image Classification and Retrieval," in Proc. ICDAR, 2015.

[17] Y. Huang, Q. Liu, and D. Karatzas, "ICDAR 2019 Competition on Scene Text Visual Question Answering (ST-VQA)," in Proc. ICDAR, 2019.

[18] T. Yu, R. Zhang, K. Yasunaga, et al., "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task," in Proc. EMNLP, 2018.

[19] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, "RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers," in Proc. ACL, 2020.

[20] T. Scholak, R. Schucher, and P. Bahdanau, "PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models," in Proc. EMNLP, 2021.

[21] F. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, T. Kruse, J. Schmidl, and F. Naumann, "Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms," PVLDB, vol. 8, no. 10, 2015.

[22] J. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, "TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies," The Computer Journal, vol. 42, no. 2, 1999.

[23] T. Schick, J. Dwivedi-Yu, R. Dessì, et al., "Toolformer: Language Models Can Teach Themselves to Use Tools," arXiv preprint arXiv:2302.04761, 2023.

[24] S. Yao, J. Zhao, D. Yu, et al., "ReAct: Synergizing Reasoning and Acting in Language Models," arXiv preprint arXiv:2210.03629, 2022.

[25] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual Instruction Tuning," arXiv preprint arXiv:2304.08485, 2023.

[26] J. Li, D. Li, S. Savarese, and S. Ermon, "BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models," arXiv preprint arXiv:2301.12597, 2023.