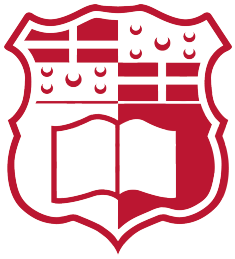


Table Selection using Information Retrieval Techniques for Table-Agnostic Text-to-SQL

Timothy J. Bartolo

Supervised by Prof Claudia Borg

Department of Artificial Intelligence
Faculty of Information and Communication Technology
University of Malta

June, 2025

A dissertation submitted in partial fulfilment of the requirements for the degree of M.Sc. in Artificial Intelligence.

WWW.UM.EDU.MT

First edition, October 14, 2025

L-Università ta' Malta

Declaration by Postgraduate Students

(a) Authenticity of Dissertation

I hereby declare that I am the legitimate author of this Dissertation and that it is my original work.
No portion of this work has been submitted in support of an application for another degree or qualification of this or any other university or institution of higher education.
I hold the University of Malta harmless against any third party claims with regard to copyright violation, breach of confidentiality, defamation and any other third party right infringement.

(b) Research Code of Practice and Ethics Review Procedures

I declare that I have abided by the University's Research Ethics Review Procedures.
As a Master's student, as per Regulation 58 of the General Regulations for University Postgraduate Awards, I accept that should my dissertation be awarded a Grade A, it will be made publicly available on the University of Malta Institutional Repository.

Faculty/Institute/Centre/School	Faculty of Information and Communication Technology
Degree	M.Sc. in Artificial Intelligence
Title	Table Selection using Information Retrieval Techniques for Table-Agnostic Text-to-SQL
Candidate (Id.)	Timothy J. Bartolo (0138194M)

Signature of Student	*Timothy J. Bartolo*
Date	October 14, 2025

To My Loving and Patient Wife

For her unwavering support

Abstract

Text-to-SQL has been effectively addressed using various NLP approaches, enabling the translation of natural language queries into SQL queries. A common prerequisite for these implementations, however, is the availability of the database table during inference. This requirement can pose challenges in scenarios where the table is not readily accessible to users. This work is motivated by the ongoing development of a chatbot tool within a private company, aimed at streamlining database interactions for the users. To address the table accessibility limitation, this study leverages Information Retrieval techniques to implement table selection based solely on the natural language query. We finetune pretrained models like BERT and GIST-NoInstruct using the ColBERT method. We train our models using data we curate in-house by employing established LLM-prompting techniques. We prepare individual training datasets using two negative sampling techniques: uniform distribution and weighted probability distribution. We also experiment with various data fusion techniques such as RRF, CombMNZ, and Linear Combination to combine results from multiple search strategies. Our approach outperforms baseline methods in table retrieval, while also providing a comparative analysis of various retrieval strategies.

Contents

1 Introduction **1**

1.1 Motivation . . . 1
1.2 Objectives . . . 3
1.3 Proposed Solution . . . 4
1.4 Document Structure . . . 5

2 Background & Literature Overview **7**

2.1 Information Retrieval Techniques . . . 7
2.1.1 BM25 . . . 7
2.1.2 HNSW . . . 8
2.1.3 Context Embeddings . . . 9
2.1.4 OpenSearch . . . 13
2.1.5 Data Fusion . . . 15
2.1.6 Anthropic Claude . . . 17
2.1.7 Negative Sampling . . . 18
2.1.8 Natural Language Toolkit . . . 19
2.2 Literature Review . . . 20
2.2.1 Table Selection . . . 21
2.2.2 Information Retrieval for Structured Data . . . 25
2.3 Summary . . . 29

3 Materials & Methods **31**

3.1 Dataset . . . 31
3.1.1 Data Synthesis . . . 32
3.1.2 Negative Sampling . . . 34

3.2 Data Augmentation . . . 35
3.2.1 Sanitization . . . 35
3.2.2 Table Flattening . . . 35
3.2.3 Query Entity Extraction . . . 36
3.3 OpenSearch . . . 36
3.3.1 Index Setup . . . 37
3.4 BM25 . . . 37
3.5 Context Embeddings . . . 38
3.6 Data Fusion . . . 42
3.7 Evaluation Metrics . . . 43
3.8 Summary . . . 43

4 Results & Evaluation **45**
4.1 Base Retrievers . . . 45
4.2 Finetuned Models . . . 48
4.3 Data Fusion . . . 55
4.4 Final Remarks . . . 57
4.5 Summary . . . 58

5 Conclusions **60**
5.1 Achieved Objectives . . . 60
5.1.1 Objective 1 . . . 60
5.1.2 Objective 2 . . . 61
5.1.3 Objective 3 . . . 61
5.1.4 Objective 4 . . . 61
5.1.5 Objective 5 . . . 62
5.1.6 Objective 6 . . . 62
5.2 Critique & Limitations . . . 63
5.3 Future Work . . . 64
5.4 Conclusion . . . 64

Appendix A Sample Training Examples **67**
A.0.1 Uniform Distribution . . . 67
A.0.2 Weighted Probability Distribution . . . 69

Appendix B Source Code **70**

References **72**

List of Figures

1.1 Text-to-SQL high-level flow for the business use-case. 3

2.1 Depiction of ColBERT's late interaction maximum similarity architecture. . . 28
2.2 Illustration of fine-grained semantic matching over tables 29

3.1 An overview of the dense retrieval system implemented by this work. 39

4.1 HitRate@1 performance across all finetuned models. 53
4.2 HitRate@3 performance across all finetuned models. 53
4.3 HitRate@5 performance across all finetuned models 54
4.4 HitRate@10 performance across all finetuned models. 54
4.5 A visual representation of the HitRate@k performance metrics for the evaluated models. 58

List of Tables

3.1 Linear Combination Weights 42

4.1 Base Retriever Results 46
4.2 Finetuned BERT Results (UD) 49
4.3 Finetuned NoInstruct Results (UD) 49
4.4 Finetuned NoInstruct Results (WPD) 50
4.5 Data Fusion Results 55
4.6 Data Fusion Improvements 57
4.7 Result Comparison 58

B.1 Directory Information 71

List of Abbreviations

AI Artificial Intelligence
ANN Approximate Nearest Neighbor
ANNS Approximate Nearest Neighbor Search
AP Average Precision 24
API Abstract Programming Interface
AWS Amazon Web Services 5
BAAI Beijing Academy of Artificial Intelligence 11
BERT Bidirectional Encoder Representations from Transformers
BGE BAII General Embedding 11
BM25 Best Matching 25 7
CombMNZ Combined Mass Normalized Z-score 15
CPU Central Processing Unit
DPR Dense Passage Retrieval
DSL Domain Specific Language 1
EM Exact-Match
FAISS Facebook AI Similairty Search 14
FP False Positive
GIST GIST-small-Embedding-v0
GPT Generative Pre-trained Transformer
GPU Graphics Processing Unit
HF HuggingFace 13
HNSW Hierarchical Navigable Small World 8
HR Hit Rate

IDF Inverse Document Frequency
IR Information Retrieval 2
IVFPQ Inverted File with Product Quantization
JSON JavaScript Object Notation 35
NI NoInstruct-small-v0
NLP Natural Language Processing 1
KNN K-Nearest Neighbor
LC Linear Combination 5
LLM Large Language Model 3
LSE Least Squares Error 16
LSTM Long Short-Term Memory
MAP Mean Average Precision 23
MaxSim Late Interaction Maximum Similarity
MEP Masked Entity Prediction 25
MLM Masked Language Modeling 10
MLR Multiple Linear Regression 16
MRR Mean Reciprocal Rank 5
MS MARCO Microsoft Machine Reading Comprehension
MTEB Massive Text Embedding Benchmark 13
NDCG Normalized Discounted Cumulative Gain 23
NLTK Natural Language Toolkit 19
NN Nearest Neighbor
NNS Nearest Neighbor Search 8
NSP Next Sentence Prediction 10
ODQA Open Domain Question Answering 7
ODTQA Open Domain Table Question Answering 21
PII Personally Identifiable Information
PLM Pretrained Language Model
QA Question-Answering
RAM Random Access Memory
REST Representational State Transfer
RNN Recurrent Neural Network
RRF Reciprocal Rank Fusion 5
SDA Structured Data Alignment 25

SDK Software Development Kit
SQL Structured Query Langauge 1
TREC Text Retrieval Conference
TREC-CAR Text Retrieval Conference Complex Answer Retrieval
TN True Negative
TP True Positive
UI User Interface 1
WSL Windows Subsystem for Linux 40

1

Introduction

This section introduces the text-to-SQL limitation we aim to solve, its relevance to our real-world business use case, and the specific research objectives and experiments we conduct. Our goal is to advance studies in this area and develop solutions to overcome the identified challenge.

1.1 | Motivation

Structured Query Langauge (SQL) is the most popular and utilized data querying language today (Hong et al., 2025). Although various alternative data management systems exist, SQL has remained a staple tool due to its early introduction in 1970 and widespread use since. Despite its popularity, its usability requires the knowledge of a complex and technical syntax, which for many users hinders thorough adoption (Hong et al., 2025). To mitigate such gaps, many use-cases resort to a middleman approach where the Domain Specific Language (DSL) is hidden from the user. As an example, an application which allows users insight into a pool of data may establish a User Interface (UI) comprised of graphical components to dictate the nature of the query operation. While satisfactory, this poses two issues; firstly, considerable development effort is required to build and maintain such a layer, and secondly, users are bounded to the limits of the UI.

Using Natural Language Processing (NLP) techniques, such issues can be drastically reduced, and in certain use-cases completely mitigated. Through direct conversion of a natural query to its counterpart SQL statement, users can operate on the database without the knowledge of the querying syntax. Furthermore, by allowing a trained model to handle the conversion, much of the development and maintenance effort can be drastically reduced. Despite this major success, the leading text-to-SQL models today still

require the table to be provided along with the natural query (Gao et al. (2023), Pourreza et al. (2025), Xu et al. (2022)). Although this may not be an issue for certain applications, others may not have the advantage of readily accessible tables. For instance, whether for privacy or security concerns, an end-user may not be granted access to the full database schema. Another scenario may be when the database entities are simply too numerous in number, causing unknowns and ambiguity during the process of manual table selection. While there exist many works (Yin et al. (2020), Herzig et al. (2020), Dong and Wang (2024), Wang et al. (2024)) which focus on schema-agnostic runtime, almost all include the table and metadata during the training phase, allowing the model to learn the explicit tables. Despite being successful in a static setting, such an implementation may not be viable for real-world business use-cases where the underlying tables are constantly changing. It is also not practical to simply provide all possible tables during inference, since the input length increases proportionally with the number of tables. For LLMs, longer inputs can exacerbate issues such as losing track of earlier context, in addition to hitting token limits, which can significantly degrade their performance. While individual inference of query–table pairs can be a viable solution, as demonstrated by Chopra and Azam (2024), latency issues become increasingly apparent

This research effort proposes a retrieval system that leverages context and metadata extracted from user queries to identify relevant SQL tables. Techniques and architectures established in prior research will be adopted and evaluated for their applicability. Given the business motivation underpinning this work, cost-effectiveness and time-efficiency will significantly influence the choice of models, tools, and techniques used to develop the system. This research focuses exclusively on the Information Retrieval (IR) aspect for selecting relevant tables. It does not extend beyond this specific stage. Instead, it is intended to serve as a first-stage retriever for downstream tasks, such as the text-to-SQL mechanism required by the business use-case. Figure 1.1 demonstrates a high level overview of the business product being built. As a final note, the company associated with this business use-case will remain anonymous, and any identifiable data is replaced with similar mock data. The changes to the data do not impact the reproducibility of our experiments or results.

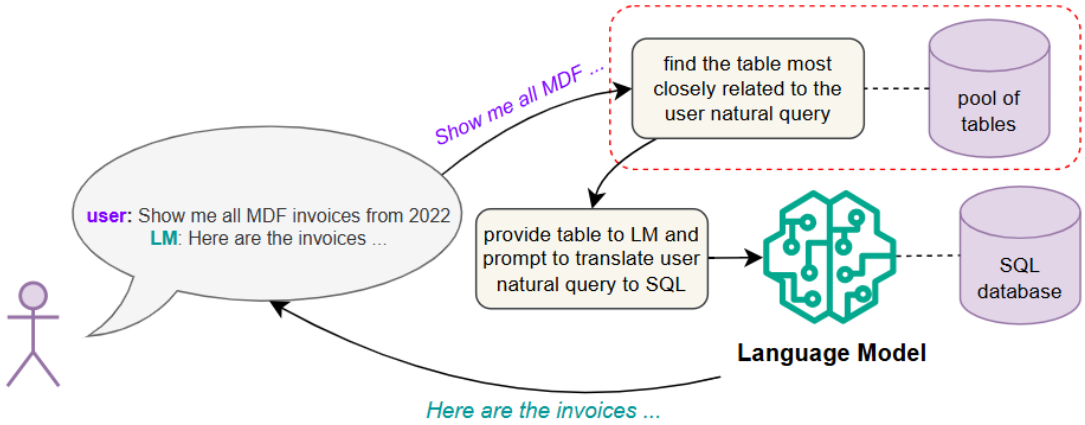


Figure 1.1: **A high-level view of the pipeline that is being implemented for the business. The user's query is used to find the target table, which is then passed to the LLM to build the final SQL query which is executed against the SQL database. The retrieved data is passed back to the LLM, which then presents it to the user. This work aims to implement the component encapsulated within the red dashed-line.**

1.2 | Objectives

This work aims to build a tool, which given a SQL-translatable natural-language-based query, selects all tables necessary to build the intended SQL statement. The tables are indexed into a storage system and retrieved based on the metadata and context of the natural query alone. To implement such a tool, this research focuses on IR techniques for both structured and unstructured data. Given the structured nature of SQL tables, we naturally place more emphasis on the latter. This work researches different approaches to IR; namely sparse, dense, and a mixture of both, which is termed data fusion. For dense retrieval, related components are investigated, such as base-model selection and finetuning techniques. We also require a dataset for training the base models, which is curated from scratch. Given the resource and time limitations of this work, hand-curated data is not possible, nor feasible, and we instead rely on the curation of synthetic data through Large Language Model (LLM). For data fusion, various multi-retrieval combinations and fusing techniques are researched and experimented with. Lastly, given the real-world business requirement of this research, it is crucial to implement a solution with limited resource consumption in mind, while also maintaining high accuracy. The objectives of this research are as follows:

- **O1**: Select the LLM and identify the prompting techniques for synthetically curat-

ing the training and test datasets which will be used to finetune the base embedding models for dense retrieval.

- **O2**: Identify the negative sampling techniques to introduce negative examples into the base synthetic datasets curated in **O1**. Compare the identified sampling techniques with one another.
- **O3**: Identify the base embedding models to use for embedding the queries and tables.
- **O4**: Identify the finetuning technique to train the base models selected in **O3** using the datasets curated in **O1** and **O2**.
- **O5**: Finetune and compare the base models selected in **O3** using the training technique from **O4** and the curated datasets from **O1** and **O2**, and identify the optimal resulting model for the table selection task.
- **O6**: Evaluate and compare different data fusion techniques to potentially improve on the optimal model identified in **O5**.

1.3 | Proposed Solution

The initial part of the research focused on determining the training technique we would use to finetune a pretrained language model. We considered various approaches, but ultimately settled on ColBERT (Khattab and Zaharia, 2020) given that our tables needed to be represented with individual columns in mind. ColBERT's use of late interaction maximum similarity not only captures context across the entire table, but also excels at capturing context within the individual columns themselves. We believed our retrieval requirement would benefit from such a characteristic. Further to this idea, Jin et al. (2023), from whom our work takes much inspiration, demonstrate successful table selection using the late interaction retrieval technique.

We also tasked ourselves with choosing a strong pretrained baseline model which we would eventually finetune using the ColBERT method. We consulted a diversity of models such as BERT (Devlin et al., 2019), SimCSE (Gao et al., 2021), BGE-small-en-v1.5 (Xiao et al., 2024), and NoInstruct[1], ultimately selecting the final model due to it's strong Massive Text Embedding Benchmark[2] IR score. Further to its strength, we found the model to also retain a smaller parameter and dimension size, which was

[1] `https://huggingface.co/avsolatorio/NoInstruct-small-Embedding-v0`
[2] `https://huggingface.co/spaces/mteb/leaderboard`

an important prerequisite for our business use-case. We also executed preliminary experiments using some of the *Titan v2*[3] series embedding models provided by Amazon Web Services (AWS), but upon achieving slightly less optimal results when compared to NoInstruct, we decided to not employ the models. This decision was taken due to the closed-source nature of the models, which would prevent us from finetuning them any further.

Given the complete lack of training data, we decided to curate synthetic data using LLM prompting. We used the Anthropic Claude 3 LLM to generate both our training and test datasets, as at the time of this work this was the only LLM included within our company's subscription plan. Our prompt engineering setup was entirely inspired by the work published by Gao et al. (2023), which prepends instructions with examples to help encourage the model to generate more accurately. From this base dataset, we then branched off into two training datasets, each using different sampling techniques to select the negative examples per query-table pair. Given the success of NoInstruct, we decided to adopt their uniform distribution and weighted probability distribution sampling techniques for our work.

To achieve a broad spectrum of results, we decided to evaluate and compare three retrieval techniques. We compared sparse retrieval through BM25, dense retrieval using the NoInstruct and $\text{BERT}_{\text{BASE}}$ embedding models, and data fusion techniques through Reciprocal Rank Fusion (RRF) (Cormack et al., 2020), CombMNZ (Fox and Shaw, 1993), and Linear Combination (LC) (Wu et al., 2011). For dense retrieval we evaluated both unfinetuned models, as well as finetuned using the ColBERT method. For data fusion, we combined and evaluated the results of both sparse and dense retrieval methods. Finally, we evaluated all approaches using the Mean Reciprocal Rank (MRR) and HitRate@k (HR) metrics.

1.4 | Document Structure

In the second section (2), we provide a detailed overview of the tools and techniques used in this work, ensuring the reader gains a foundational understanding. Additionally, we review relevant literature, including works addressing similar problems, as well as those tackling different challenges but employing techniques applicable to our study. In the next section, (3) we cover the details of our experiments by discussing tools, hardware, techniques, configuration, and overall reasoning for our decisions. In this section, the user will find all the necessary details to replicate our experiments. In the fourth

[3] `https://aws.amazon.com/bedrock/amazon-models/titan/`

section (4), we demonstrate the raw results of our experiments while also evaluating and discussing them. Finally, in the last section (5), we summarize our work, while also discussing limitations and potential future research.

2

Background & Literature Overview

The task of selecting the best table to aid in downstream tasks, such as text-to-SQL, can usually be simplified into a classification problem, where the question context is mapped to a known class, which is ultimately mapped to a table. For the possibility of such a system, all possible tables must be known from beforehand, while also remaining static in their respective makeups. The challenge in this research lies in the table-agnostic nature of the business use case, where tables are both entirely unknown as well as ever-changing. It is for this reason that this research focuses on Information Retrieval (IR) techniques within the field of Open Domain Question Answering (ODQA), where the aim is to find the best matching entities within an unbounded corpus of data. This section consists of two subsections. The first discusses the IR techniques (2.1) central to this work and the tools used for their implementation. The second subsection (2.2) reviews existing research with similar objectives, as well as other works that employ techniques relevant to our work but ultimately pursue different goals.

2.1 | Information Retrieval Techniques

In this section, we cover the tools and techniques this work uses to implement IR for the task of table selection.

2.1.1 | BM25

Best Matching 25 (BM25) is a search relevancy algorithm used in IR and developed over the course of several decades starting in 1960s until the 1990s (Robertson et al., 2009). The algorithm functions on the premise of probability weighting, where shared

terms between a query and document are given individual scores. The term scores are summed and an overall score is applied to the query-document pair, allowing it to be ranked for relevancy amongst other pairs. BM25 is formally described with the following equation:

$$\text{score}(D, Q) = \sum_{t \in Q} \frac{\text{IDF}(t) \cdot f(t, D) \cdot (k_1 + 1)}{f(t, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgDL}}\right)} \tag{2.1}$$

where Q is the query containing the terms, D is the document being scored, $f(t, D)$ is the frequency of term t in document D, $|D|$ is number of words in D, avgDL is the average document length in the collection, k_1 is the term frequency saturation parameter, b is the length normalization parameter, and $\text{IDF}(t)$ is the Inverse Document Frequency of term t, defined as:

$$\text{IDF}(t) = \ln\left(\frac{N - n_t + 0.5}{n_t + 0.5} + 1\right) \tag{2.2}$$

where N is the total number of documents in the collection and n_t is the number of documents containing the term t (Robertson et al., 1995).

2.1.2 | HNSW

Nearest Neighbor Search (NNS) serves as a crucial component across multiple fields, including information retrieval, data mining, pattern recognition, machine learning, and recommendation systems. It plays a fundamental role in enabling efficient data analysis and decision-making in these domains (Wang et al., 2021). As put by Indyk and Motwani (1998), the nearest neighbor (NN) problem involves preprocessing a set of n points $P = \{p_1, \ldots, p_n\}$ in a given metric space X to efficiently answer queries. Each query requires finding the point in P that is closest to a given query point $q \in X$. A major drawback of traditional NNS algorithms is their high time complexity of $O(n)$, as they rely on a brute-force approach that compares the query point to every document point (Indyk and Motwani, 1998). This approach naturally suffers from the curse of dimensionality, hindering the algorithm's use in settings retaining higher data volumes. Over the years, researchers have developed various algorithms that address this challenge by approximating neighboring values, often at the cost of some accuracy. Indyk and Motwani (1998) describe approximate nearest neighbor (ANN) as finding a point $p \in P$ that is an ϵ-approximate nearest neighbor of the query q such that for all:

$$p' \in P, d(p, q) \leq (1 + \epsilon) d(p', q) \tag{2.3}$$

Hierarchical Navigable Small World (HNSW), introduced by Malkov and Yashunin (2020) is one such NN approximation algorithm employed by this work. HNSW is an efficient approximate nearest neighbor search algorithm that organizes data in a multi-layer graph structure, where each layer represents a different level of abstraction motivated by the skip-list data structure. It leverages a navigable graph to quickly traverse and find nearest neighbors. The algorithm significantly improves time complexity compared to brute-force search, with query time complexity improving from $O(n)$ to $O(\log n)$. The HNSW algorithm can be further finetuned by using the adjustable parameters originally introduced by Malkov and Yashunin (2020). For instance, *ef_search* controls the search effort and *ef_construction* affects the quality of the graph construction, while m is the maximum number of neighbors permitted for each node, impacting both the graph's connectivity and the search efficiency. By adjusting these parameters, one can balance accuracy and speed. A higher *ef_construction* improves the graph quality and search accuracy, but comes with a higher construction cost; similarly, a higher *ef_search* improves search precision but at the expense of slower queries.

2.1.3 | Context Embeddings

Context embeddings are floating-point vector representations which capture the semantics meaning of text. Different texts represented in this form and mapped to the same vector space can be used to perform powerful operations concerning their correlation with one another. In this work, context embeddings are used to perform ANN Search (ANNS) for selecting the table with greatest relation to the natural query. This subsection covers some of the different algorithms, architectures, and techniques used to generate such embeddings.

2.1.3.1 | Transformers

The Transformer model, introduced by Vaswani et al. in *Attention Is All You Need* (2017), revolutionized natural language processing by eliminating recurrence and relying entirely on self-attention mechanisms. This novel approach solved the memory and computation-time problems of the recurrent neural network (RNN) and long short-term memory (LSTM) architectures. At the time, these architectures were considered the state of the art (Vaswani et al., 2017). The Transformer model efficiently processes input sequences in parallel, making it significantly faster than traditional recurrent models while achieving superior performance in various tasks. Central to the Transformer architecture is self-attention, a mechanism that dynamically assigns weights to words based on

their contextual relevance within the sequence. This is computed using the scaled dot-product attention function:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.4}$$

where Q is the input-query matrix, K is the input-key matrix, V is the representation matrix, d_k is the dimentionality of the input-key matrix, QK^T is the dot-product of the input-query and input-key matrices, and $\frac{QK^T}{\sqrt{d_k}}$ is used as a scaling factor to prevent large values during training. As explained by Vaswani et al., the Transformer model utilizes multi-head self-attention to capture diverse word relationships in parallel, enriching feature representation through concatenation and linear transformation. Positional encodings address the absence of recurrence, enabling word order understanding. Each layer is comprised of multi-head self-attention and a position-wise feedforward network, both followed by layer normalization and residual connections. The encoder stacks multiple layers, while the decoder adds masked self-attention to prevent future token influence during training.

2.1.3.2 | BERT

BERT (Devlin et al., 2019), which stands for Bidirectional Encoder Representations from Transformers, is an encoder Transformer model which processes both left and right token contexts concurrently. Previous to BERT, tokens were processed in a unidirectional fashion such as the model presented by Kiros et al. (2015). Other models like ELMo (Peters et al., 2018) also processed both left-to-right and right-to-left contexts, but independent of one another. Instead, the authors of ELMo individually processed the two contexts and later concatenated them to form a single representation. BERT's novel approach found major success, and consequently set new state of the art results for 11 NLP tasks (Devlin et al., 2019).

BERT uses a technique called Masked Language Modeling (MLM). In MLM, 15% of the tokens in a sentence are selected for possible masking. Of these, 80% are replaced with the reserved *[MASK]* token, while the remaining 20% are either left unchanged or replaced with a random word. The model learns to reconstruct the original words by considering context from both the left and right. This two-way contextual understanding allows BERT to grasp word meanings and their relationships with greater accuracy. Additionally, BERT uses a technique called Next Sentence Prediction (NSP). In NSP, the model is trained to determine whether one sentence logically comes after another in a sequence. This dual pre-training strategy allows BERT to excel at various natural

language processing tasks like question answering, sentiment analysis, and language inference.

Besides the *[MASK]* token, BERT is trained utilizing additional tokens such as the *[CLS]*, *[SEP]*, and *[PAD]* tokens. The *[CLS]* is added once at the beginning of the input sequence indicating the start of a sequence, while *[SEP]* is used to show separation between sentences within a single sequence. The *[SEP]* token is also expected at the end of the entire input. Lastly, the *[PAD]* token is trained to be ignored by the BERT architecture as is simply a means of maintaining the same sequence lengths across different sequences sharing the same processing batch. Batching is a common technique used to improve efficiency when processing multiple sequences.

Along with the release of their paper, Devlin et al. established two BERT models - $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$. The former having 12 layers, 768 dimension vectors, 12 self-attention heads, and 110M parameters, while the latter having 24, 1024, 16, and 320M, respectively. While $\text{BERT}_{\text{LARGE}}$ outperformed it's smaller sibling across all NLP tasks, $\text{BERT}_{\text{BASE}}$ still showed considerable improvement over the other state of the art models at the time (Devlin et al., 2019). For our experiments shown later (2.2), we use $\text{BERT}_{\text{BASE}}$ as it is naturally less computationally intensive during both training and inference time.

2.1.3.3 | BAAI BGE

BAII General Embedding (BGE) is a group of embedding models released by the Beijing Academy of Artificial Intelligence (BAAI) by authors Xiao et al. (2024). The family of models consists of three models, offering the flexibility to trade off efficiency and effectiveness as required: small, base, and large, with 24M, 102M, and 326M parameters, respectively. All models are based on BERT-like (Devlin et al., 2019) architecture. Their smallest model, which we again reference later in this work (2.1.3.4), is formally called *BGE-small-en-v1.5*, and it generates vectors of 384 dimension. To train the models, the authors implement a three-stage recipe consisting of pre-training, contrastive learning, and multi-task learning.

The models are first pretrained using a large collection of plain text data, optimizing it specifically for embedding tasks. They utilize a self-curated corpora gathered from Wikipedia, social media websites, and news websites. Their approach is based on the RetroMAE (Xiao et al., 2022). In this process, they introduce noise into the text, encode it into embeddings, and then use a lightweight decoder to reconstruct the original clean text. The objective is to minimize the reconstruction loss, ensuring the model learns meaningful representations through this encoding-decoding framework:

$$\min \sum_{x \in X} -\log \mathrm{Dec}(x \mid e\tilde{X}), \quad e_{\tilde{X}} \leftarrow \mathrm{Enc}(\tilde{X}). \tag{2.5}$$

where *Enc* indicates the encoding operation, *Dec* indicates the decoding operation, and X and $\tilde{X}$ indicate the clean and polluted text, respectively.

For the second step, the models are finetuned on unlabeled data via contrastive learning, where true-positives (TP) are learned to be discriminated from their true-negatives (TN). Negatives are selected using in-batch sampling (Karpukhin et al., 2020) with batch sizes of up to 19,200. The authors use the following contrastive loss formula:

$$\min \sum_{(p,q)} -\log \left(\frac{e^{\langle e_p, e_q \rangle / \tau}}{e^{\langle e_p, e_q \rangle / \tau} + I \sum_{Q'} e^{\langle e_p, e_{q'} \rangle / \tau}} \right) \tag{2.6}$$

where p and q are the paired passage and query, $q' \in Q'$ is a sampled negative, and τ is the temperature.

For the final step in the recipe, the pre-trained models are further finetuned using smaller, but higher quality data sets. During training, the authors leverage instruction-based tuning (Asai et al., 2023) by attaching a verbal instruction prompt to the query side of the query-passage pair. This is done to mitigate conflicts between ambiguous pairs naturally occurring in the different datasets. Additionally, one hard negative is mined per query-passage pair following the ANN-style (Xiong et al., 2021) sampling strategy.

2.1.3.4 | GIST

Built on top of the *BGE-small-en-v1.5* model (Xiao et al., 2024) is the *GIST-small-Embedding-v0* (GIST) (Solatorio, 2024) embedding model. During the training of this model, Solatorio takes into consideration only those negatives which are truly unrelated to the positive query-passage pair. For filtering out the negatives, the author utilizes a secondary scoring model originally trained by Li and Li named *WhereIsAI/UAE-Large-V1* (Li and Li, 2024). Solatorio uses this model to first vectorize both positive and in-batch-negative queries and passages so that the negative vectors can then be compared with the positive using cosine similarity. Those negative texts producing a similarity score greater than the positive-query-passage pair are removed from the negative training batch (Solatorio, 2024). This is done in an effort to remove any negative passages or queries which are highly likely to be relevant to the target query. InfoNCE loss (van den Oord et al., 2018) is used to train the model to distinguish between related and unrelated pairs:

$$L \sim \frac{e^{\text{sim}(q_i, p_i^+)/\tau}}{e^{\text{sim}(q_i, p_i^+)/\tau} + \sum_{j \in B} e^{\text{sim}(q_i, p_j^-)/\tau}} \tag{2.7}$$

where (q_i, p_i^+) are the related pairs, p_j^- are the unrelated samples, *sim* represents the cosine similarity in the case of GIST, and B is the universe of batch negatives from which p_j^- are sampled. Temperature τ, typically within the range $[0.01, 0.1]$, regulates the concentration of the similarity scores.

Extending further on the GIST work, Solatorio has since released a followup model which demonstrates an improvement in IR. This model outperforms both the original GIST model, as well as the base *BGE-small-en-v1.5* model across most IR tasks. The same encoder is used for both queries and passages, but with different embedding strategies: for passages, the embedding of the *[CLS]* token from the last layer is used, while for queries, a single representation is used for all token embeddings by using mean-pooling:

$$\mathbf{e} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \tag{2.8}$$

where e is the mean-pooled embedding, n is the number of embeddings, and $\mathbf{x}_i$ represents each embedding of equal dimension. The new model, called *NoInstruct-small-Embedding-v0* (NoInstruct), is available for download on HuggingFace (HF)[1]. However, there are currently no scientific references available for the author's work. As of this writing, the author notes on the model's HF page that technical details will be published soon. Despite this, we choose to use the model in later experiments (3) due to its smaller size and strong performance in IR, as demonstrated by its Massive Text Embedding Benchmark (MTEB) score (Muennighoff et al., 2023) - a large-scale benchmarking suite for evaluating text embedding models across diverse tasks. The MTEB leader board can be viewed on HF[2].

2.1.4 | OpenSearch

OpenSearch[3] is an open-source full-text search platform built on top of the Apache Lucene[4] engine. More specifically, it is a community-driven fork of the popular Elasticsearch[5], managed and led by Amazon Web Services (AWS). OpenSearch supports both sparse and dense search, providing options for a multitude of algorithms, two

[1] `https://huggingface.co/avsolatorio/NoInstruct-small-Embedding-v0`
[2] `https://huggingface.co/spaces/mteb/leaderboard`
[3] `https://opensearch.org/`
[4] `https://lucene.apache.org/`
[5] `https://www.elastic.co/elasticsearch`

of which being BM25 and HNSW. For the former, various tokenization techniques are available, with the default one being the *standard* tokenizer. This tokenizer uses grammar-based tokenization, following the Unicode Text Segmentation algorithm outlined in Unicode Standard Annex #29[6]. For example, using the *standard* tokenizer on the text "*The 2 QUICK Brown-Foxes jumped over the lazy dog's bone.*" would result in the following tokens:

[*the*, *2*, *quick*, *brown*, *foxes*, *jumped*, *over*, *the*, *lazy*, *dog's*, *bone*]

where a search term would only match if any of the terms in the tokenized list are a case-insensitive exact-match of the term. For instance, both "*quick*", as well as "*QUICK*" would match the document, while "*quickly*" or "*quickest*" would not. During a search, the query is also tokenized in the same fashion. In this work, we only make use of this default tokenizer when performing BM25 search.

For HNSW, OpenSearch uses the Facebook AI Similairty Search (FAISS)[7] library as the underlying similarity search engine. Using this library, OpenSearch indexes provided vectors into the HNSW graph space and performs similarity search using either cosine similarity or Euclidean Distance, depending on which function is configured. Cosine similarity can be formally described as:

$$cos\_sim = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \tag{2.9}$$

where $\|\mathbf{A}\|$ is the length of vector $\mathbf{A}$, $\|\mathbf{B}\|$ is the length of vector $\mathbf{B}$, and $\mathbf{A} \cdot \mathbf{B}$ is the dot product between vectors $\mathbf{A}$ and $\mathbf{B}$. The dot product is defined as:

$$\mathbf{A} \cdot \mathbf{B} = |\mathbf{A}||\mathbf{B}| \cos\theta \tag{2.10}$$

where $|\mathbf{A}|$ and $|\mathbf{B}|$ denote the lengths of vectors $\mathbf{A}$ and $\mathbf{B}$, while θ is the angle separating them. Two dimensional Euclidean Distance, on the other hand, can be formally described as:

$$d = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2} \tag{2.11}$$

where p and q represent points with coordinates (p_1, p_2) and (q_1, q_2), respectively. Furthermore, OpenSearch exposes the configurable HNSW parameters discussed in previous sections (2.1.2), such as *ef_construction*, *m*, and *ef_search*. All configurations used for our experiments are described in later sections (3).

[6] `https://unicode.org/reports/tr29/`

[7] `https://github.com/facebookresearch/faiss/`

2.1.5 | Data Fusion

Frank Hsu and Taksa (2005) define data fusion as a collection of methods that integrate multiple relevant pieces of evidence to enhance the performance of IR systems. According to Beitzel et al. (2004), under the right circumstances, data fusion can improve overall results, but on the contrary can cause them to degrade if certain rules are not followed. For instance, they state that fusing the results of retrieval strategies which are already strong in isolation is one scenario where performance is more likely to degrade. In our work, we experiment with this observation in later sections (4.3). We employ both rank-based and score-based fusion techniques: the former only considers the document's ranking position, while the latter relies solely on its relevance score (Wu et al., 2011). The following subsections discuss some of the specific data fusion techniques used by this work.

2.1.5.1 | Reciprocal Rank Fusion

Reciprocal Rank Fusion (RRF) (Cormack et al., 2009) is a simple rank-based data fusion algorithm which requires no training. The algorithm can be formally defined as:

$$\text{RRF}(d \in D) = \sum_{r \in R} \frac{1}{k + r(d)} \tag{2.12}$$

where D represents the set of documents to be ranked, R represents the set of rankings per retrieval strategy, and k is a constant which mitigates the impact of outlier rankings. The authors settle on $k = 60$ as it performed best in their initial investigations.

2.1.5.2 | Combined Mass Normalized Z-score

Combined Mass Normalized Z-score (CombMNZ) (Fox and Shaw, 1993) is a score-based data fusion technique which takes into consideration both document scores and document occurrences across all ranked lists. The scores for all ranked lists are first individually normalized using a normalization technique. CombMNZ can be defined as:

$$\text{CombMNZ}(d) = \left(\sum_{i=1}^{N} \text{Score}_i(d) \right) \times F(d) \tag{2.13}$$

where $\text{CombMNZ}(d)$ is the normalized score of document d in the i^{th} retrieval system, $F(d)$ is the number of retrieval systems where document d appears in, and N is the number of retrieval systems or number of rank lists. While *MinMax* and *Z-score* are both popular normalization techniques, in this work we only make use of *MinMax*:

$$X' = a + \frac{(X - X_{\min})(b - a)}{X_{\max} - X_{\min}} \tag{2.14}$$

where X is the document's relevancy score assigned by the retrieval algorithm, X' is the normalized score of X, $X_{\min}$ and $X_{\max}$ are the minimum and maximum relevancy scores in the set of scores, respectively, and a and b are the lower and upper bounds of the desired range to which the data is normalized. Common values for a an b are 0 and 1.

2.1.5.3 | Linear Combination

Linear Combination (LC) (Wu et al., 2011) is another data fusion technique which uses a score-based approach. The difference in this algorithm is that it uses a pretraining step to learn the weights which are used during final ranking. The algorithm uses Multiple Linear Regression (MLR) to find the combination of weights that produce the least amount of error, or Least Squares Error (LSE). According to the authors, in the LSE approach, the coefficients derived from MLR enable the optimal fusion outcomes through the Linear Combination technique, as they allow for the most precise estimation of the relevance scores for all documents in relation to all queries. Wu et al. define Linear Combination as:

$$M(d, q) = \sum_{i=1}^{n} \beta_i \cdot s_i(d, q) \tag{2.15}$$

where n is the number or IR systems $ir_1, ir_2, ..., ir_n$, q is the target query, and d is the document. Each IR provides a result r_i comprised of relevancy scores. $s_i(d, q)$ is the normalized score of the document d in result r_i. β_i is the weight assigned to IR system ir_i and $M(d, q)$ is the Linear Combination score for d and q. For score normalization, the authors use a binary logistic regression model to estimate the scores, but they also state that other techniques may be used. In our work, we use *MinMax* normalization as shown later (3). As mentioned, for learning the weights β_i to β_n, MLR is employed, which is defined as:

$$q = \sum_{i=1}^{m} \sum_{k=1}^{r} \left[y_k^i - \left(\hat{\beta}_0 + \hat{\beta}_1 s_{1k}^i + \hat{\beta}_2 s_{2k}^i + \cdots + \hat{\beta}_n s_{nk}^i \right) \right]^2 \tag{2.16}$$

where m is the number of queries, n is the number of IR systems, and r is the total number of documents in document collection D. The authors conduct two experiments, first with $r = 1000$ and then with $r = 20$, observing only a 0.3% difference in performance. However, the latter reduces training time by a factor of 20. For each query q^i,

each IR system provides a score for each document in D. Every IR system ir_j assigns score s^i_{jk} for document d_k and query q^i. y^i_k is the judged relevance score of d_k for query q^i. In this work, we use 1 for relevant and 0 for irreverent since our business requirement does not allow a table to be partially correct. Finally, LSE is used to learn the combination of coefficients $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, \ldots, \hat{\beta}_n$ for which the error is the least. Here, $\hat{\beta}_1$ represents the first IR system, $\hat{\beta}_2$ represents the second IR system, and so on. The authors do not include $\hat{\beta}_0$ since it has no effect on the outcome.

2.1.6 | Anthropic Claude

The introduction of context-aware models such as the LSTM-based ELMo (Peters et al., 2018) and the Transformer-based BERT (Devlin et al., 2019) marked the emergence of a new class of models known as pretrained language models (PLM). These models established a strong baseline, providing a foundation for further fine-tuning to optimize performance on specific downstream tasks. Researchers discovered that significantly scaling up base PLMs led to different behaviors compared to their smaller predecessors, giving rise to a new category known as Large Language Models (LLM). GPT-3 (Brown et al., 2020), having 175B parameters, is one such model within this category demonstrating remarkable chat dialog capabilities and in-context learning (Zhao et al., 2023). Other popular LLMs include a successor to GPT-3 called GPT-4 (Achiam et al., 2023), Google's T5 (Raffel et al., 2020), Meta's LLaMa (Touvron et al., 2023), and various others, each retaining their individual strengths.

In our work, we utilize the Claude 3 Sonnet LLM developed by Anthropic, with its predecessor models being introduced in the work titled *A General Language Assistant as a Laboratory for Alignment* (Askell et al., 2021). In this work, Askell et al. aim to develop an AI assistant that aligns with human values and expectations. The authors encapsulate this goal through three key principles: *helpfulness*, *honesty*, and *harmlessness*. Unfortunately, Claude 3 Sonnet is only recently released and is not open-sourced, so details into the inner-workings and training procedures would be speculative. It should also be noted that our work is limited to the use of this LLM due to the limitations of the current subscription plan subscribed to by our company. For the synthesis of our datasets, we use the Amazon Bedrock[8] iteration of Claude 3 Sonnet called *anthropic.claude-3-sonnet-20240229-v1:0*, which has a parameter size of roughly 175B, a 200k context window, and 8,192 output token size. This is done using prompt-engineering techniques described later (3).

[8] `https://aws.amazon.com/bedrock/`

2.1.7 | Negative Sampling

Given that our work takes inspiration from existing training methods which utilize contrastive loss functions, it is vital that our datasets include high-quality examples. Contrastive learning relies on two main pieces of information: positive pairs (q, p^+) and negative pairs (q, p^-), where q is the query and p^+ and p^- are the positive and negative passages for q, respectively. The training goal directs the learned representation to position positive pairs close to each other and negative pairs farther apart (Robinson et al., 2020). For our specific task of table selection, the choice of positive example is straightforward, as the answer to a query is of a binary nature; for a given query, only a single table can be the TP, while all others are fully irrelevant. Contrary to this, the task of selecting a negative example is not as straightforward, as different negative sampling techniques lead to a difference in the behavior of the model ultimately trained. While there are a multitude of negative sampling techniques, our work implements and compares just two of them - these techniques are uniform distribution and weighted probability distribution.

2.1.7.1 | Uniform Distribution

While training GIST (Solatorio, 2024), the author uses uniform distribution to select the negative examples for their training dataset. In our work, we take inspiration from this technique to also select our own negatives. Solatorio defines their approach in two steps. First, the top-K most similar passages are retrieved using cosine similarity and an embedding model of choice. For K, 100 is chosen. Secondly, a random selection is made on the top-K, where all members have equal chance of being selected. In our work, we decide to incorporate the entire corpus instead of the top-K since only a single table can be relevant for any given query; although we believe that closely following the top-K approach is a strong candidate for future research.

2.1.7.2 | Weighted Probability Distribution

Taking further inspiration from Solatorio (2024), we use weighted probability distribution to build an additional training dataset. It should we noted that Solatorio uses this technique to select their query positives, but we use it to select our negatives. The author defines the technique as:

$$\pi = \text{Softmax}\left(\frac{\text{TopK}(\theta)}{\tau}\right) \tag{2.17}$$

where TopK(θ) is the top-K-most similar passages for a query using the cosine similarity method and an embedding model of choice, and τ is a temperature parameter to control the distribution density of the $Softmax$ function. The authors then assign a positive example to the query by probabilistically selecting from the top-K using the weighted probability. They use 100 for K and 0.005 for temperature τ. Unlike uniform distribution, the more similar a passage is to the target query, the higher the chance it has of being selected. For our work, we implement a slight variation of this method by negating the $Softmax$ step. Instead, we first sort on similarity score descending and then build the probability weights based on positional ranking, resulting in:

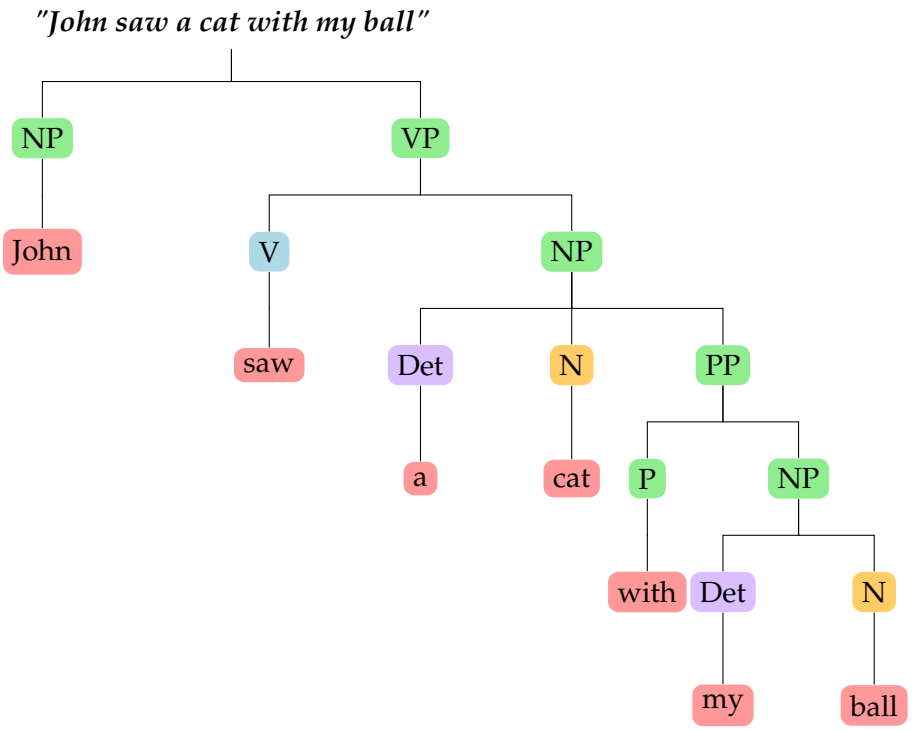
$$w_i = \frac{i+1}{1}, \quad \text{for position } i = 1, 2, \ldots, n \tag{2.18}$$

As with Solatorio's $Softmax$ technique, passages with higher similarity have higher chances of selection. Actual implementation values are discussed in further detail later in this work (3).

2.1.8 | Natural Language Toolkit

Natural Language Toolkit (NLTK) (Bird, 2006) is a comprehensive library for natural language processing (NLP) in Python, designed to assist researchers in working with natural language data. It offers access to a wide range of linguistic datasets and lexical tools, including WordNet, and comes equipped with modules for various natural language processing tasks like text analysis, classification, tokenization, stemming, part-of-speech tagging, and syntactic parsing. NLTK is highly modular, offering high-level functionality making it a versatile toolkit for handling various NLP tasks.

This work primarily makes use of the NLTK's parsing module, which is designed to analyze and process sentence structure, enabling the extraction of grammatical relationships within text. It includes several powerful parsers which can handle context-free grammars. NLTK's parsing module allows users to build syntax trees that represent the grammatical structure of sentences, providing insights into sentence composition. This functionality is essential for tasks like syntax analysis, syntactic parsing, and understanding the hierarchical structure of language, all of which are foundational for more complex NLP tasks. As an example taken directly from Bird's work (2006), the sentence *"John saw a cat with my ball"* can be parsed and tagged by the NLTK parser module to output the following grammar tree:



As demonstrated in the example grammar tree, natural language text can be decomposed into individual words and phrases, each associated with its respective grammatical tag. In the example, the abbreviations have the following meanings: **NP**=*noun phrase*, **VP**=*verb phrase*, **V**=*verb*, **Det**=*determiner*, **N**=*noun*, **PP**=*prepositional phrase*, and **P**=*phrase*. This functionality can then be leveraged for subsequent tasks, as illustrated in later experiments (3).

2.2 | Literature Review

This section explores existing works that align with our objectives (2.2.1), as well as those which employ techniques relevant to our research (2.2.2). We first start out by discussing the former, which to the best of our knowledge are quite limited in number. We then discuss some of those works which do not share the same objectives, but make use of IR techniques which relate to our task of table-selection.

2.2.1 | Table Selection

Open Domain Table Question Answering (ODTQA) by Jin et al. (2023) is the work that most closely aligns with our objectives, particularly due to its ability to handle a large and diverse number of tables. Given a natural query, the authors train a model to differentiate between relevant and irrelevant tables. Jin et al. utilize NLTK to extract noun phrases from the input question. These noun phrases are subsequently passed through the $\text{BERT}_{\text{BASE}}$ model, where mean pooling is applied to the embeddings of each phrase, resulting in a single pooled embedding per noun phrase. The table structure is then linearized by concatenating column names with their respective cell values in a sequential format. The linearization process follows a structured order, where column names are followed by a subset of their corresponding cell values, ensuring a consistent format for downstream processing. As an example, a table with three columns and two entries may be linearized like so:

NAME *john amy* **AGE** *31 38* **ADDRESS** *4th street apt 15 17 pioneer way*

where **NAME**, **AGE**, and **ADDRESS** are the column names, and *john 31 4th street apt 15* is the first entry, and *amy 38 17 pioneer way* is the second entry.

For each table column, the authors compute two separate embeddings: one for the column name and another for the first value in the column. The column name embedding is generated through mean pooling over all token embeddings corresponding to the column name, accounting for multi-token column names. A similar pooling process is applied to the first cell value within each column. This results in each column being represented by two distinct embeddings: one for the column name and another for its first value.

During the retrieval process, each pooled query noun-phrase embedding is compared against the column-name and column-value embeddings using NNS. The highest similarity score for each noun phrase is selected, and these scores are aggregated at the table level. The table with the highest cumulative score is chosen as the most relevant. The score calculation can be formulated by:

$$w_{ij} = q_i \cdot c_j \tag{2.19}$$

$$\text{Score} = \sum_{i}^{n} \max_{j \in [1,2m]} w_{ij} \tag{2.20}$$

where q_i is a single query phrase representation, c_j is a single table representation, and w_{ij} is their similarity calculated using the Dot Product. The number of phrase rep-

resentations for the target query is represented by n, while m represents the number of columns for the target table. The second equation is a scoring approach called *late interaction maximum similarity* (MaxSim) adopted from the ColBERT (Khattab and Zaharia, 2020) paper, which is discussed in the upcoming section (2.2.2).

Jin et al. finetune their model using two datasets called WikiSQL (Zhong et al., 2017) and NQ-TABLES (Herzig et al., 2021), both consisting of natural language query and table pairs. To measure the accuracy of their implementation, the authors use the recall@k (R@k) and exact-match (EM) metrics. The former can be described as:

$$\text{Recall@}k = \frac{|\mathcal{R} \cap \mathcal{A}_k|}{|\mathcal{R}|} \tag{2.21}$$

where $|\mathcal{R} \cap \mathcal{A}_k|$ is the set of relevant items for a given query, and $|\mathcal{R}|$ is the set of top-k items retrieved by the retrieval system, while EM can be described as a strict measurement where the true answer must be contained within the set of returned results. If the answer is present, a score of 1 is given, otherwise 0. Using their novel table selection approach, Jin et al. achieve strong results for both datasets, with a R@5 score of 77.63 and an EM score of 45.42 for WikiSQL, and a R@10 of 90.41 and an EM of 39.72 for NQ-TABLES.

Unfortunately, we find the full implementation details of this effort to be somewhat ambiguous, and therefore a replication of this work is not attempted. For instance, it is not certain what contrastive learning functions were used to finetune the BERT encoder. It is also not evident whether a single encoder approach was implemented, or rather a dual encoder. Furthermore, it is not clear to us what negative sampling techniques were utilized. While the authors do indeed reference other works throughout their explanations, we feel that a certain degree of speculation would be required. At present, the code for ODTQA is not publicly available, but after reaching out to the authors, we were provided with the NLTK logic which was used to extract query noun phrases. Despite our uncertainty, we would like to highlight that our work is heavily inspired by Jin et al.'s work, as will be seen later in our experiments (3).

A second work which also aims to select the best table given a natural query is the research conducted by Chopra and Azam (2024). Despite similar objectives, the approach implemented by the authors suffers from sufficient enough throughput to handle higher numbers of tables. The authors state that databases containing a few hundred tables or more may pose processing time issues. As already mentioned, the business use-case of our work requires fast response times and the ability to handle such table numbers. Nevertheless, the work published by Chopra and Azam provides useful knowledge and techniques for the task of table selection.

In their work, Chopra and Azam first curate an in-house dataset based on the popular Spider (Yu et al., 2018) text-to-SQL dataset. Taking the natural queries and structured tables from Spider, the authors construct a three-attribute training dataset consisting of the natural query, SQL query, and an array of table names representing the label. Unlike our work, Chopra and Azam train their models to support a multi-tble setting, by incorporating natural queries which require data from multiple tables. The authors employ a binary classification approach: given a natural query and a table, infer the classes (table names) which are required to answer the query. They formalize their approach as:

$$f(q, d) \rightarrow \{T_i \dots T_j\} \in \{T_i \dots T_j\} \subseteq d, \quad d = \{T_1, T_2, \dots, T_k\} \tag{2.22}$$

where q represents the query, d represents all tables $\{T_1, T_2, \dots, T_k\}$ in the corpus, and $f(q, d)$ represents the task to select the tables $\{T_i \dots T_j\} \in \{T_i \dots T_j\}$ in d which are required for the formulation of the SQL query based on q. As base models, the authors compare the performance of a mixture of encoder and decoder models, with the best performing model, DeBERTa (He et al., 2020), originating from the former architecture. During inference, the input is provided in the BERT-style form of:

$$[CLS][W_1] \dots [W_n][SEP][S_1] \dots [S_n][SEP][PAD] \dots [PAD] \tag{2.23}$$

where $[W_1] \dots [W_n]$ represents the query tokens and $[S_1] \dots [S_n]$ represents the tokens of a single table definition. The embedding outputs from DeBERTa are then pooled and fed forward to a fully connected layer, whose output is subsequently passed to a classifier. Finally, the raw logits produced by the classifier are used to assess the scores of each possible table, retaining only those surpassing a defined threshold of 0.5. The model is trained for a total of 10 epochs with a learning rate of $5e-5$ using the Adam optimizer (Kingma and Ba, 2014) to enhance performance and accelerate convergence during training.

For measuring the performance of their approach, Chopra and Azam utilize two separate metrics called Normalized Discounted Cumulative Gain (NDCG) and Mean Average Precision (MAP). The former metric can be described as:

$$\begin{aligned} \text{NDCG}_p &= \frac{\text{DCG}_p}{\text{IDCG}_p} \\ \text{DCG}_p &= \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)} \\ \text{IDCG}_p &= \sum_{i=1}^{p} \frac{2^{rel_i^*} - 1}{\log_2(i+1)} \end{aligned} \tag{2.24}$$

where DCG_p denotes the Discounted Cumulative Gain at position p, and $IDCG_p$ represents the optimal DCG at the same position. The variable rel_i refers to the relevance score of the item ranked at position i, while rel_i^* indicates the relevance scores of the best possible ranking. MAP can be formally described as:

$$\text{MAP} = \frac{1}{|Q|} \sum_{q \in Q} \text{AP}(q) \tag{2.25}$$

where Q represents all queries, $|Q|$ is the total number of queries, and $AP(q)$ is the Average Precision (AP) for a single query q. AP can be formulated using:

$$\text{AP}(q) = \frac{1}{R} \sum_{k=1}^{n} P(k) \cdot \text{rel}(k) \tag{2.26}$$

where R is the total number of relevant documents for query q, $P(k)$ is the precision at rank k, and $rel(k)$ is a binary function which outputs 1 when rank k is relevant, and 0 when irrelevant. Lastly, n is the total number of retrieved documents. Using these two metrics, Chopra and Azam report strong scores of 0.947 and 0.938, respectively.

As a final work, we also consider the table selection implementation published by Zhang and Balog (2018). Using tables from the WikiTables (Bhagavatula et al. dataset, 2015) and queries sourced from Amazon's Mechanical Turk platform, Google's Squared, as well as crowdsourcing, Zhang and Balog extract their respective terms using two methods. The first method extracts only the unique terms from the individual queries and tables. For the former, all words in the query are considered, while for latter, all words in the table's title, caption, and headings are taken into account. The second method follows an entity-based approach, considering only the entities present in queries and tables by utilizing a lookup knowledge base known as DBpedia (Auer et al., 2007). For queries, the process is straightforward by selecting the produced entities, but for tables the process is further finetuned. Here, only entities which are from the core-column are chosen, which the authors define as the column housing the greatest number of entities. Further entities are retrieved from the table's page title and caption by limiting the number of entities to a k-value. All three table components are aggregated and represented using the following:

$$\{t_1, \ldots, t_m\} = E_{cc} \cup R_k(T_{pt}) \cup R_k(T_{tc}) \tag{2.27}$$

where $\{t_1, \ldots, t_m\}$ represents the table terms, E_{cc} represents the entities belonging to the core-column, and $R_k(T_{pt})$ and $R_k(T_{tc})$ represent the k-selected entities from the table's page title and caption, respectively.

The query and table terms are then individually vectorized using two methods. The first method uses a sparse vector approach where each term is represented as a 0 or 1 depending on which entities in the DBPedia knowledge base match the current term. The second method also vectorizes the individual terms, but instead using dense embeddings generated by both a word embedder called Word2Vec (Mikolov et al., 2013), as well as a Graph embedder called RDF2vec (Ristoski and Paulheim, 2016). During inference, a combination of early-fusion and late fusion are used to measure the similarity between a target query and the corpus of tables. In early-fusion, the centroid of the query term embeddings is compared with the centroid of the table term embeddings. In late-fusion, pairwise similarity is calculated amongst all query and table term embeddings, which are then aggregated. For calculating the similarity in either fusion type, the authors use cosine similarity. Finally, the similarity scores for both fusion types are combined. Zhang and Balog achieve a NDCG@5 score of 0.5951, and a NDCG@10 score of 0.6293.

2.2.2 | Information Retrieval for Structured Data

Table selection involves mapping a natural language query to a structured representation. Given the nature of our research, we naturally consulted studies that focus on data with similar characteristics, even if their objectives differed from ours. One such example is SANTA (Li et al., 2023), where a single T5 (Raffel et al., 2020) model is finetuned to represent the vectors of product descriptions and structured product bullet points in proximity. Given the text-to-text nature of T5, Li et al. intercept the encodings before they are forwarded to the decoder and instead use the embeddings to train the model via contrastive learning. The authors integrate two pretraining strategies, Structured Data Alignment (SDA) and Masked Entity Prediction (MEP), jointly optimized using:

$$L = L_{\text{SDA}} + L_{\text{MEP}} \tag{2.28}$$

SDA treats structured data as documents and descriptions as queries, learning to align them through contrastive loss and in-batch negatives. Meanwhile, MEP masks structured data entities by assigning a mask-token to recurring entities, training the model to predict masked entities without relying on negative sampling. This approach emphasizes structure-aware learning, specifically selecting noun phrases and code identifiers as entities.

Other works, such as TaPas for QA (Herzig et al., 2021) and UTP (Chen et al., 2023) also implement IR systems based on structured data. More specifically, these works focus on generating representations for natural language text and structured tabular

data, each with a different approach. For instance, Herzig et al. leverage the base TaPas (Herzig et al., 2020) model which supports queries and flattened tables in a single sequence. This is possible due to the TaPaS model itself originating from the $\text{BERT}_{\text{BASE}}$ model, which allows for separation of texts by using the $[SEP]$ token. Queries are provided using the form:

$$[CLS][Q_1]\dots[Q_n][SEP][PAD]\dots[PAD]$$

where Q_i is the token of the i^{th} word in the query. Tables are provided as a concatenation of table title and flattened body, where body, in turn, is a concatenation of all columns followed by all cell values.

$$[CLS][TT_1]\dots[TT_n][SEP][T_1]\dots[T_n][SEP][PAD]\dots[PAD]$$

where TT_i is the token of the i^{th} word in the table title, and T_j is the token of the j^{th} word in the table's flattened body. The authors employ a dual-encoder setup with two TaPaS models: one for queries and another for flattened tables. They utilize the $[CLS]$ token output embeddings from each model for subsequent table retrieval via NNS. The retrieved tables are then used for further downstream QA tasks.

Similar to TaPaS for QA, Chen et al. also utilize the TaPaS base model for further finetuning, but instead only use a single model to embed representations. Furthermore, instead of natural queries to accompany the tables, the authors use natural text descriptions surrounding the respective tables. Chen et al. implement their training in two phases. First, they employ the BERT-style MLM technique by masking 15% of tokens. They then utilize a cross-modal contrastive loss function to enhance representation learning. To encourage learning and input diversification, the authors train the model to learn on three types of input: table description only, flattened table only, and both table description and flattened table:

$$X_w = [CLS][w_1]\dots[w_n][SEP][PAD]\dots[PAD]$$

$$X_t = [CLS][t_1]\dots[t_n][SEP][PAD]\dots[PAD]$$

$$X_{wt} = [CLS][w_1]\dots[w_n][SEP][t_1]\dots[t_n][SEP][PAD]\dots[PAD]$$

where w_i is the token of the i^{th} word in the table description text and t_j is the token of the j^{th} word in the flattened table. Finally, the authors use last-layer mean pooling to generate representations of the input tokens.

As our final discussion, we focus on ColBERT, introduced by Khattab and Zaharia (2020). Our interest in this approach was initially prompted by our reading of ODTQA

by Jin et al., which inspired us to explore the potential of ColBERT. Unlike the previous works discussed in this section, ColBERT does not primarily focus on structured data, but instead serves as a training technique for general IR. The benefit of ColBERT for our table selection task is it's approach to representation and scoring, which uses contextualized late interaction to estimate the similarity between a query and document. Given query q, document d, and their relevance $S_{q,d}$, the late interaction technique employed by ColBERT can be formularized as:

$$S_{q,d} := \sum_{i \in |E_q|} \max_{j \in |E_d|} E_{q_i} \cdot E_{d_j}^T \tag{2.29}$$

where $|E_q|$ is the number of embeddings for q, $|E_d|$ is the number of embeddings for d, and $(\max_{j \in |E_d|} E_{q_i} \cdot E_{d_j}^T)$ is the single document embedding d_j within E_d which is most similar to the target query embedding q_i. This operation is performed for all documents, ultimately producing a score for the query in question and all corpus documents. More specifically, the authors title this approach as maximum similarity contextualized late interaction, or *MaxSim* for brevity. Figure 2.1 illustrates ColBERT's late interaction MaxSim architecture. We believe this technique works well for the task of table selection based on the granular comparison of individual components making up the query and table. As put by Jin et al. (2023), the MaxSim technique is analogous to how a human would choose a table based on a query, which is by choosing the one with the most fine-grained keyword matches between itself and the query. Figure 2.2, taken from Jin et al., illustrates granular semantic matching over tables.

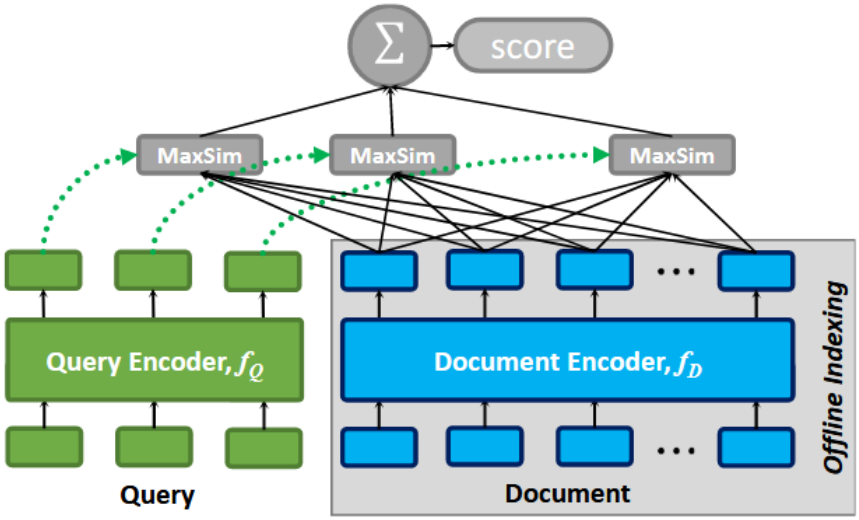


Figure 2.1: **A depiction of ColBERT's late interaction maximum similarity architecture, taken from Khattab and Zaharia (2020).**

In the ColBERT paper, the authors employ a single $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2019) encoder model, which is shared for both the query and document. Instead of forming a single sequence separated by $[SEP]$ tokens, the queries and documents are passed into the model individually. Custom tokens $[Q]$ and $[D]$ are respectively added just after the $[CLS]$ token. The query tokens are padded with BERT's special $[MASK]$ token up to a max length of N_q, which the authors set to 32. In the case that the query's base token length exceeds N_q, then the query is truncated to N_q. The authors label the addition of the $[MASK]$ tokens as *query augmentation*, which encourages the model to learn new terms and re-weight existing ones. For documents, no masking tokens are appended, but instead tokens correlating to punctuation are entirely removed due to their ineffectiveness. By default, $\text{BERT}_{\text{BASE}}$ outputs embeddings with a dimension of 768, but Khattab and Zaharia alter this output to a reduced value of 128.

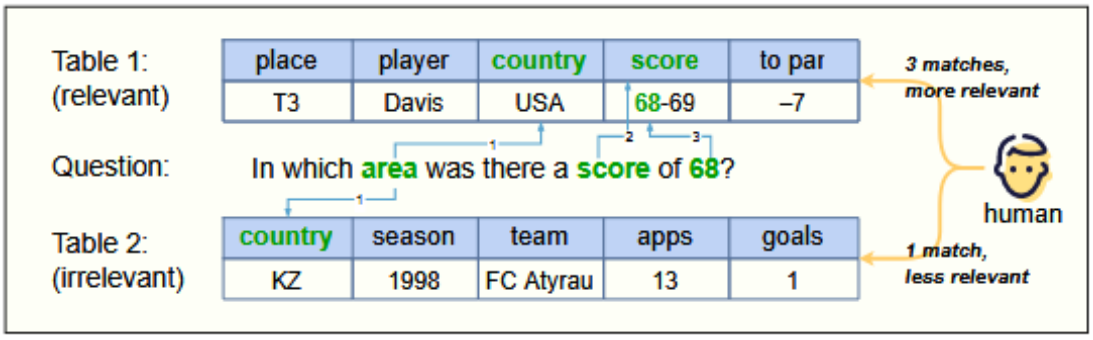


Figure 2.2: **Illustration of fine-grained semantic matching over tables, taken from Jin et al. (2023).**

The authors train the model's weights using cosine similarity to assess query-document relevance, and apply the Adam optimizer with cross-entropy loss over the positive and negative document scores d^+ and d^-. They use MS MARCO (Bajaj et al., 2018), a dataset consisting of 8.8M passages and 1M queries built on Microsoft Bing search results, as well as TREC Complex Answer Retrieval (TREC-CAR) (Dietz, 2019), another dataset consisting of 27M Wikipedia passages and 3M queries. For evaluation, the authors make use of 7k and 2.2k test queries, respectively. Given the exhaustive search nature of the MaxSim algorithm, the authors also mention the potential use of libraries like FAISS for an accuracy-latency tradeoff. They demonstrate the use of an inverted file with product quantization (IVFPQ) index, which drastically improves search time. The IVFPQ algorithm is able to do this through K-means clustering (MacQueen, 1967), a machine learning algorithm that groups data into a specified number of clusters based on their similarities, minimizing the variance within each cluster. In our work, we instead use FAISS' HNSW implementation.

2.3 | Summary

In this section, we covered the tools and techniques this work uses in its experiments to retrieve the best table given a natural query. For instance, we covered search techniques such as BM25, HNSW, and others, while also covering tools like OpenSearch and NLTK. Their usages are demonstrated throughout our experiments (3). We also discussed literature which we separated into two categories. The first category are those works with similar objectives to ours, such as ODTQA, the work published by Chopra and Azam (2024), as well as the work by Zhang and Balog (2018). The second group we categorized

as those works, which while ultimately having different goals, make use of techniques which we deem useful towards our own efforts. An example of this is ColBERT (Khattab and Zaharia, 2020).

3

Materials & Methods

Our methodology consists of: (a) synthesizing a dataset of tables and queries using an LLM, (b) preparing training triples with different negative sampling strategies, (c) finetuning two base language models ($\text{BERT}_{\text{BASE}}$ and NoInstruct) using a ColBERT-inspired late interaction method, and (d) experimenting with data fusion to combine different retrievers. We also implement a BM25 baseline for comparison. In this section, we describe in detail all components making up our implementation and experiments.

3.1 | Dataset

To the best of our knowledge, there is no existing public dataset which satisfies the requirements of this work's business use-case. While datasets like WikiSQL and NQ-Tables can be manipulated to our training structure, they ultimately contain data which is not suitable for the business use-case as they mostly consist of English words which are understood globally. In our case, we required examples of closed-domain acronyms which do not belong to any particular language, and are not necessarily understood by individuals outside of the business. For instance, the queries *"What XMA purchases have been submitted by Amy?"* and *"Which DTMs are pending for Application M5?"* contain terms *XMA*, *DTM*, and *M5*, respectively, which are both language-agnostic as well as closed-domain. It is for this reason that we decided to synthesize our training and test datasets using an LLM.

When synthesizing the datasets, we also took into consideration additional requirements. For instance, a user of our system will have access to an average of 50 tables and a maximum of 100, so for this reason we created a test data set of roughly 200 tables, coupled with 4,940 queries. We also ensured that a query is linked to exactly one table, as

the business use-case does not allow for table joining in the downstream SQL translation task. For the training dataset, we synthesized a total of 1,140 tables and 31,183 queries. For both datasets, each table contains between 25 and 50 queries, and the number of columns per table ranges from 6 to 20, reflecting the minimum and maximum number of columns a business user may access, respectively. Lastly, we ensured to synthesize both tables and queries to include a mixture of underscores, dashes, and camel-casing when referring to columns names; for example, instead of *"M5 Purchases"*, we encouraged the LLM to generate *"M5_Purchases"*, *"M5-Purchases"*, or *"m5Purchases"*. This was done in an effort to mimic the real-world data of our business.

3.1.1 | Data Synthesis

For synthesizing our datasets, we made use of the Anthropic Claude 3 Sonnet LLM hosted in AWS Bedrock under the *anthropic.claude-3-sonnet-20240229-v1:0* alias and having version *bedrock-2023-05-31*. We communicated with the LLM through the abstract programming interface (API) exposed by Bedrock using AWS' software development kit (SDK) for Python 3.12 called *Boto3*[1]. Our work is limited to the use of this LLM due to the constraints of our company's current subscription plan. While we also experimented with data synthesis using GPT-4, this was highly restricted by the API rate limits imposed on the free tier. Despite these limitations, we found Anthropic Claude 3 to produce more versatile and diverse table definitions. In contrast, GPT-4 often reused a narrow set of column names across multiple tables, leading to highly similar tables.

We synthesized the tables by building a LLM prompt with inspiration from Gao et al. (2024), where examples are prepended to the instructions. For the examples, we used three tables which were taken from our real-world business data stores, but manually replaced personally identifiable information (PII) with similar mock data. In an effort to encourage the LLM to focus on the task at hand and avoid incorrectness, we only included the instructions which were absolutely necessary, by avoiding tasks which could be done later programmatically such as shuffling and flattening of the tables. The following listing outlines the structure of the prompts used to generate the tables:

[1] `https://boto3.amazonaws.com/v1/documentation/api/latest/index.html`

Listing 3.1: Table Generation LLM Prompt

```
{
    "name": "mdi_genericdoc",
    "body": {
        "MDI_ID": "varchar",
        "AccountingEntity": "varchar"
        ...
    }
}
...
Here are 3 examples of tables. Taking inspiration from these
3 tables, create another table with the title '{table_name}'
using field names of your decision. Create a total of
{field_count} fields. Incorporate field names using acronyms,
underscores, dashes, and camel-case. Only respond with the
JSON result.
```

where *field_count* is a random number in the range $[6, 20]$ and *table_name* is generated from a separate LLM prompt:

Listing 3.2: Table Name Generation LLM Prompt

```
examples: [aarf_incidents,document_submission_report,
m5_retrieval,bens_document_file,login_credentials,
user_interaction_report,task_management_plc,
audit_log_archive,transaction_abc_monitoring,
csr_feedback_log,mxf_user_roles]

Come up with a fake table name with inspiration taken from
the provided examples. Be sure to incorporate unrealistic
acronyms for some of the names, but NOT all. Only respond
with a JsonArray.
```

Once the tables were synthesized, we ensured to remove any which shared the same name or those sharing at least half of its column names with another table. Given the new tables, we then synthesized up to 50 natural queries by prepending the instructions with the body of the table:

Listing 3.3: Query Generation LLM Prompt

```
{tables}

Come up with up to 50 questions that a user might ask when
querying this SQL table. The questions must be in natural
language English. Incorporate the table's name into some
of the questions, but not all. For the values, use
hypothetical values. Only respond with a JsonObject
of this form {{"name": "name_of_table", "queries" : []}}
```

3.1.2 | Negative Sampling

Since our models would eventually be finetuned using contrastive learning, we required our training pairs (q, d^+) to also retain an example of a negative by transforming them into triplets (q, d^+, d^-). To accomplish this, we employed the two sampling techniques inspired by Solatorio (2024), which we discuss in detail in the earlier section (2.1.7). The uniform distribution technique is straightforward, and only requires that the TP is not mistakenly selected as the negative. The weighted probability distribution technique on the other hand requires further explanation of our implementation.

We first flattened all training tables based on the technique which is described in the upcoming section (3.2.2), and indexed into an OpenSearch HNSW index with *ef_search*=100, *ef_construction*=128, and m=16. For each query, we then performed an ANNS using Euclidean distance late interaction MaxSim to select the top-100 candidates. Similar to the uniform distribution sampler, we excluded the TP table from the retrieved results if it was present. Queries were first sanitized using the sanitization operation explained shortly (3.2.1). The model used to generate the embeddings for the ANNS were generated using a finetuned model which we name *NoInstruct_6* and introduce in detail in section 3.5.0.3. By default, OpenSearch sorts the retrieved tables by most relevant to least. We then assigned weights to each retrieved result based on their positional ranking as explained in 2.1.7.2. Using the Python *Random* module, we randomly selected a negative example while actively applying the calculated weights.

Once both datasets were populated with their respective negative examples, we ensured to randomly shuffle their entries so that the training batches ultimately provided during training were not made up of queries having the same positive table. We provide samples for both datasets in Appendix A.

3.2 | Data Augmentation

The datasets we synthesized were designed with future research in mind, so queries and tables were left in their raw state. As a result, special characters like underscores and dashes, as well as formatting styles like camel-casing, were preserved. Just as Khattab and Zaharia (2020) perform punctuation cleaning during their experimentation, we too applied necessary data augmentation techniques. In this section, we discuss some of pre-index and pre-search operations.

3.2.1 | Sanitization

Before running our experiments, we first sanitized all queries and tables by replacing underscores and dashes with a single whitespace. Similarly, we also separated all camel-cased text with a single whitespace. As a final measure, we lower cased all text. We performed these prerequisite tasks for two reasons. The first was to reduce any ambiguity between words making up an entity, table name, or column name. The second reason was purely based on inspiration taken from works like ColBERT (Khattab and Zaharia, 2020), which drop all tokens relating to punctuation. As an example, *"Which mdTasks have status ACTIVE for file-ag12?"* would be sanitized to *"which md tasks have status active for file ag12?"*.

3.2.2 | Table Flattening

Since the tables in our datasets are stored in JavaScript Object Notation (JSON) form, we had to first extract the respective table names and bodies into usable text before indexing into OpenSearch. To the best of our knowledge, no existing research addresses the indexing of table data consisting solely of table names and columns; we therefore followed techniques provided by ODTQA (Jin et al., 2023) and TaPaS (Herzig et al., 2020). Both groups of authors embed their table data along with the raw cell values. For instance, Jin et al. embed their tables using the following input form:

$$\{column_1\}\{column_1\_cell\_values\}\{column_2\}\{column_2\_cell\_values\}\{column_3\}\ldots$$

We, on the other hand, prefixed the column names with the table's name, and do not make use of any cell data since it does not apply to our work. We included the table's name because we believe this to be an information-rich part of the table definition, and an integral component if present in the query. The following demonstrates the flattened structure we used in our work:

$$\{table\_name\}\{column_1\}\{column_2\}\{column_3\}\ldots\{column_n\}$$

where every entity is separated by a single whitespace and n is the total number of columns for a given table, which in our case is within range $[6, 20]$.

3.2.3 | Query Entity Extraction

When performing BM25, we did not simply provide the entire query as our search term, but instead we extracted only those words and phrases which were based on the NLTK Python function used in ODTQA (Jin et al., 2023). The function extracts words and phrases from text according to any of the following rules:

1. If the word is a singular noun (**NN**), plural noun (**NNS**), proper singular noun (**NNP**), proper plural noun (**NNPS**), or cardinal number (**CD**)
2. If the word is a descriptive noun, meaning the noun must be preceded by an adjective (**JJ**), a comparative adjective (**JJR**), or a superlative adjective (**JJS**)

We chose this approach for two reasons. Firstly, Jin et al. demonstrated successful results using this method for table selection. Secondly, when comparing full unaugmented query search to extracted phrase search, we observed improvements for the latter, albeit subtle. The results are detailed in later sections (4). It should be noted that we only extracted entities from queries when performing BM25 search. We did not apply this logic to other search techniques such as dense embedding retrieval. It should also be noted that we did perform an experiment where we used the unaugmented natural query during a single BM25 experiment, but this was only done in an effort to benchmark against entity extraction.

3.3 | OpenSearch

All search operations performed for our experiments were done using a uni-node cluster running OpenSearch 2.17.1 with the KNN plugin enabled. This plugin was required to allow for HNSW indices and searches. We communicated with OpenSearch via a tool called Postman[2], an API testing and development tool that allows users to test REST APIs efficiently. We also called OpenSearch directly from within our Python code using the official OpenSearch Python 3.12 client SDK[3].

[2] `https://www.postman.com/`
[3] `https://github.com/opensearch-project/opensearch-py`

3.3.1 | Index Setup

We created two types of OpenSearch indices, one for BM25-related querying, and another for ANNS dense retrieval. For the former, we created a simple index mapping with two fields: *PHRASE* and *TABLE_ID*, where *PHRASE* stores the textual data of the entire flattened table using the *standard* tokenizer, and *TABLE_ID* represents the primary key storing of the name the table. For the latter type, we defined the mapping with two fields: *TABLE_ID* and *EMBEDDING*, where *TABLE_ID* is the name of the table and *EMBEDDING* is the respective token vector embedding of the flattened table. Unlike the index created for the BM25, the *TABLE_ID* value for the dense retrieval index is shared amongst all tokens making up the flattened table representation. Following the late interaction methodology of MaxSim, this field value was later used to consolidate scores based on the table name. The dense index was also configured to use the following settings: *dimension=128*, *engine=faiss*, *similarity_scorer=l2* (Euclidean distance), *algorithm=hnsw*, *ef_construction=128*, and *m=16*. We did not explicitly configure *ef_search*, as by default OpenSearch sets this to the value of *k* (neighbors).

3.4 | BM25

Since Jin et al. (2023) and Khattab and Zaharia (2020) use BM25 as a basis of comparison, we too performed similar experiments. We performed two BM25-only experiments using the base dataset consisting of the 4,940 query-table pairs (q, d^+). As there is no training involved in these experiments, we did not make use of the other datasets consisting of query-positive-negative triplets (q, d^+, d^-). The first experiment we performed consisted of the full raw natural query, solely subjecting the text to our sanitization method. For the second experiment, we performed both sanitization as well as entity extraction. The entities extracted using the ODTQA (Jin et al., 2023) method were then joined by a single whitespace and indexed into the OpenSearch index as a single sequence of text. For our second experiment, a query like *"Show me all MDS-files which are ACTIVE status"* would be translated to *"mds files active status"*. For our initial experiment, a simpler translation would take place given the absence of entity extraction: *"show me all mds files which are active status"*. For both experiments, *match*, a convenient full-text operation provided by OpenSearch, was used to perform the BM25 search against the *PHRASE* field. As we refrained from specifying a search-time tokenizer, by default, OpenSearch used the tokenizer configured for the target match field to also tokenize the query. In our case, the *PHRASE* field's *standard* analyzer was also applied to the query. This would result in the first query to be tokenized into terms [*mds*, *files*, *active*, *status*], and the sec-

ond query into [*show*, *me*, *all*, *mds*, *files*, *which*, *are*, *active*, *status*]. Finally, OpenSearch responded with the most relevant tables, sorted in descending order by their respective scores.

3.5 | Context Embeddings

In this section, we introduce the approaches we use to perform table-selection using only dense retrieval. We discuss in detail the base embedding models we chose to use, the training techniques used to finetune them, and the available training resources available at the time of this work. Finally, we explain the experiments we ran using different combinations of models and search parameters.

3.5.0.1 | Base Models

We made use of two base embedding models for our work, $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2019) and *NoInstruct-small-Embedding-v0*. We chose the former model because we employed the ColBERT (Khattab and Zaharia, 2020) training method, and wanted to remain as close to the original work as possible, which also uses this same BERT model. We also decided that using the same model would encourage more accurate replication, and serve as a strong comparison model when attempting to improve the results with subsequent models. Secondly, we decided to adopt the latter model by Solatorio, replacing the original BERT model in hopes of achieving stronger table-selection results. When selecting a replacement model, we considered three key factors: IR quality, dimension size, and parameter size. Since our task involves IR, prioritizing higher IR quality was a natural choice. For this, we relied on the model's MTEB results[4], which reported a competitive IR score of 51.99, an impressive 41.4 points higher than $\text{BERT}_{\text{BASE}}$. Additionally, to minimize runtime, training latency, and financial costs, we preferred models with fewer dimensions and parameters. The *NoInstruct-small-Embedding-v0* model offered a well-balanced trade-off among all three factors, with a dimension size of 384 and 33M parameters.

3.5.0.2 | Finetuning

We decided to use the ColBERT (Khattab and Zaharia, 2020) training method to finetune our base models for a few reasons. Firstly, as already mentioned, this work took inspiration from the work published by Jin et al. (2023), which demonstrates success

[4] `https://huggingface.co/spaces/mteb/leaderboard`

in their table-selection approach using late interaction MaxSim. Secondly, as opposed to standard dense retrieval techniques such as DPR (Karpukhin et al., 2020) and GIST (Solatorio, 2024), we believed the term-based granular matching of ColBERT to be a superior approach for our table-selection task. Lastly, we found the ColBERT resources and code to be publicly available[5] and well-documented, ensuring that we would be more accurate in our implementation and replication. We provide a detailed overview in Figure 3.1 of the dense retrieval system described thus far.

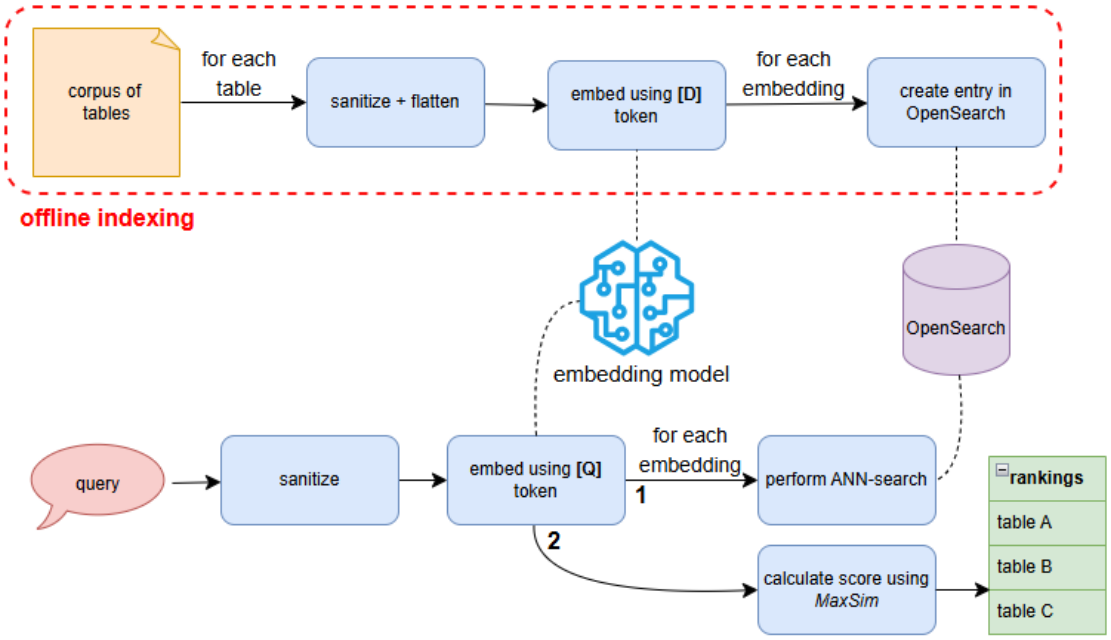


Figure 3.1: **An overview of the dense retrieval system implemented by this work.**

We trained our base models using ColBERT's publicly available code, which follows the algorithm outlined in previous sections (2.2.2). The authors expose various parameters which can be configured by the user and passed during training startup. We used the following parameters when finetuning our models:

[5] `https://github.com/stanford-futuredata/ColBERT/tree/colbertv1`

Listing 3.4: ColBERT v1 Parameters

```
CUDA_VISIBLE_DEVICES="0" OMP_NUM_THREADS=8 python -m
torch.distributed.launch --nproc_per_node=1 -m colbert.train
--amp --doc_maxlen 180 --mask-punctuation --bsize 32 --accum 1
--triples {train_path.tsv} --root {save_path} --experiment
MSMARCO-psg --similarity l2 --run-path msmarco.psg.l2
```

where *train_path.tsv* is the path to the training dataset and *save_path* is the path where the learned model weights should be saved. All parameters, except for the similarity function, which we set to l2 instead of cosine, are identical to the parameters used in the original ColBERT paper. It should also be noted that Khattab and Zaharia train their models using two Intel Xeon Gold 6132 14-core CPUs, each with 469 GiBs of RAM. They also use four Titan V GPUs, each having 12 GiBs of memory. Unfortunately, we were not able to acquire similar hardware, but instead were limited to a single Intel Core Ultra 7 165H CPU (3.8 GHz) and one Nvidia RTX 500 Ada Generation GPU with 32 GiBs of RAM. For this reason, we also set *CUDA_VISIBLE_DEVICES* to a single GPU, while also defining the number of available threads to 8, which is the maximum capability of our CPU.

We set up our training on a Windows 10 machine using an Ubuntu 24.04.1 LTS sub-system via Windows Subsystem for Linux (WSL). We then ran the ColBERT code via Miniconda[6] using their Conda environment definition found within the root of the project. Unfortunately, due to various compatibility issues caused by outdated dependencies, we were not able to run the Python program using the original *conda.yml* file. As a workaround, we updated all dependencies to their latest versions at the time, which mitigated the issue. By default, the ColBERT code uses the $\text{BERT}_{\text{BASE}}$ model as a base, so for our experiments concerning the NoInstruct we had to manually update the code to reflect these requirements. Due to the abundant and technical nature of the code changes, we do not detail them here; however, we encourage interested readers to explore the code themselves for a deeper understanding. The code can be found at the links provided in Appendix B.

3.5.0.3 | Experiments

Before performing any training, we first established two foundational results as a baseline for later comparison with our finetuned models. For this, we used the late interaction MaxSim retrieval scores for our synthetic dataset of (q, d^+) pairs using the $\text{BERT}_{\text{BASE}}$

[6] https://www.anaconda.com/docs/getting-started/miniconda/main

(Devlin et al., 2019) and NoInstruct embedding models. We set a maximum of *k=1000* (neighbors), as we wanted to avoid increased inference times in an effort to satisfy the business requirements. Once the foundational results were established, we proceeded to finetuning the models.

We first aimed to create base results to later compare against by replicating the ColBERT paper (Khattab and Zaharia, 2020) using the author's default $\text{BERT}_{\text{BASE}}$ setup. Unfortunately, true replication of the work was not possible due to the size of the datasets used in the original work. For instance, the authors finetune $\text{BERT}_{\text{BASE}}$ on 200k iterations, which given our limited hardware would have taken us over two weeks of continuous training. This would not have been feasible given our original aims to train several epochs. Furthermore, the two datasets used by the authors were not relevant to our task. We instead used our own dataset which required just over four hours of training per epoch, each consisting of roughly 970 iterations. Using the synthetic dataset built with negative examples sampled using uniform distribution, we finetuned the $\text{BERT}_{\text{BASE}}$ model for seven epochs, producing a total of seven sets of model weights. The individual weights were then used to perform seven separate retrieval tasks using the late interaction MaxSim technique. The same k value of 1000 was used for all tasks. We then performed a similar training, but using the NoInstruct model instead, ultimately producing an additional seven sets of retrieval results. It should also be noted that due to the much smaller parameter and dimension size, finetuning NoInstruct only required approximately fifteen minutes per epoch.

We repeated the experiment yet again using the NoInstruct base model, but this time using the second synthetic dataset we curated using weighted probability distribution negative sampling. Setting the same value of *k=1000*, we produced seven more sets of search results using MaxSim. We did not train $\text{BERT}_{\text{BASE}}$ against this second dataset, as by this stage in our experimentation, the NoInstruct model was proving to be superior for table selection; we leave this experiment for future potential research. Finally, we set up one more experiment where we initialized the NoInstruct base model with the weights produced by the 6^{th} epoch uniform distribution training session. We then further trained this model for a single epoch using the weighted probability distribution dataset. We termed this finetuned model as *NoInstruct_6.1* (NI_6.1), where the first ordinal refers to the 6^{th} epoch of the original training, and the second refers to the single epoch trained during the followup finetuning. Similar to the previous experiments, we set k to 1000 and performed retrieval using MaxSim.

3.6 | Data Fusion

In an effort to increase search accuracy for our task of table selection, we explored IR when combining multiple search techniques. We therefore experimented with merging different sets of results using RRF (Cormack et al., 2009), CombMNZ (Fox and Shaw, 1993), and Linear Combination (Wu et al., 2011). Our initial experiment retrieved results using two search techniques we demonstrated in previous experiments. The first was BM25 retrieval using the sanitized entities extracted from the query with the ODTQA (Jin et al., 2023) NLTK function. The second retrieval method used the MaxSim approach for embeddings generated using our NoInstruct_6 model. The results from both retrieval methods were then combined in three separate experiments using RRF@60, CombMNZ, and Linear Combination, respectively. For Linear Combination, we learned weights (Table 3.1) using a search size of 10,000, which encompasses all tokens in our table corpus. To explore the effects of different search size values, we repeated the same experiment for Linear Combination using search size 20 and 10.

Table 3.1: Linear Combination Weights

Retriever 1 & 2	Search Size	Ret. 1 Weight	Ret. 2 Weight
BM25_on & NI_6.1	10,000	1.186782	-0.01132026
BM25_on & NI_6.1	20	1.47596013	0.14829506
BM25_on & NI_6.1	10	1.61055509	0.24312852
NI & NI_6.1	10,000	0.51345614	-0.00508008
NI & NI_6.1	20	0.32620289	0.20911481
NI & NI_6.1	10	0.50718321	0.34719289

The learned weights for our experiments with Linear Combination for data fusion, which were performed using two combinations of two retrievers. *on* represents BM25 with entities extracted using the odtqa nltk function

We again repeated the experiments for all three data fusion techniques, but replacing the BM25 retrieval with an additional embedding-based retriever. We instead employed the base NoInstruct (NI) model originally introduced by Solatorio. Instead of using late interaction MaxSim, we used the model to embed text into a single representation as originally intended by the author. As described in previous sections (2.1.3.4), NoInstruct uses token mean-pooling for queries, and the $[CLS]$ token embedding for passages, which in our case were the flattened tables. Our inspiration behind this approach was to explore data fusion performance when incorporating different embedding techniques, specifically. All results are discussed in detail in the upcoming section (4).

3.7 | Evaluation Metrics

The choice of evaluation metrics for our table selection results is dependent upon two core features of the business requirements. Firstly, a user's query will always be in the context of a single table, and therefore the retrieved results can be treated as binary, where at most one table is relevant. Secondly, the accuracy of our downstream text-to-SQL task, which is beyond the scope of this work, is inversely correlated to the number of retrieved table. It is critical that our table selection mechanism not only selects the correct table, but also encourages higher ranking. We decided to use Mean Reciprocal Rank (MRR), which provides a fine-grained measurement based on the ranking position across all test queries, which is defined as:

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i} \tag{3.1}$$

where $|Q|$ denotes the total number of queries and rank_i represents the rank of the first relevant table in the retrieval results. We also introduced a secondary measure called HitRate@k (HR@k) which provides accuracy rates at different threshold intervals. The different accuracy rates per threshold aid in comparing the accuracy-latency tradeoffs for the downstream tasks. Given only one table can be relevant per query, we decided to make use of a simplified version of HR@k:

$$\text{HR@k} = \frac{|\text{rank}(d_q) \leq k\}|}{|Q|} \tag{3.2}$$

where $|Q|$ denotes the total number of queries, q is a query within Q, d represents a table document within the set of all table documents, $\text{rank}(d_q)$ is the single relevant table document d for q, and k is the rank threshold. We used rank thresholds of 1, 3, 5, and 10, with preference for the former two thresholds given the business latency requirements. We include thresholds at 5 and 10 to ensure thoroughness in our observations.

3.8 | Summary

In this section, we described the process of synthesizing our dataset through LLM prompting and the various negative sampling techniques used to construct our datasets. We also detailed our use of OpenSearch, including its configuration for storing tables and conducting searches. Additionally, we discussed data preparation and augmentation techniques, the base models employed, the training methods applied, and the hardware resources available at the time. Furthermore, we provided an in-depth analysis

of our search experiments, covering BM25, similarity search using context embeddings, and the fusion of different retrieval techniques.

4

Results & Evaluation

In this section, we present the results through visualization and discussion, grouping the experiments into three categories: base models, finetuned models, and data fusion. We analyze and compare the results within each category, while also attempting to provide an explanation as to why certain approaches performed better or worst compared to others.

4.1 | Base Retrievers

Using the late interaction MaxSim technique, along with the base NoInstruct model yielded the highest score when measured using MRR, HR@1, and HR@3. When using HR@5 and HR@10 for evaluation, single-representation NoInstruct performed best. Despite this, NoInstruct-MaxSim presented itself as the optimal base model for our business use-case given it's dominance in the lower HR thresholds. When swapping out the MaxSim embedding model with $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2019), all metric scores declined, making it the worst performing retrieval strategy. Surprisingly, both BM25 retrievers outperformed this model by more than double across all metrics. However, it should be noted that later fine-tuning results showed improvement (4.2). We enlist all results in Table 4.1, and for completeness we also include model output dimension size where applicable.

Table 4.1: Base Retriever Results

Retriever	Dim.	MRR	HR@1	HR@3	HR@5	HR@10
BM25_fq	n/a	0.656	55.58	72.43	78.70	84.86
BM25_on	n/a	0.657	55.70	73.60	75.58	83.30
NI	384	0.737	61.59	82.51	**90.26**	**95.85**
BERT-MaxSim	128	0.283	20.02	30.76	35.06	42.18
NI-MaxSim	128	**0.748**	**64.19**	**83.25**	88.52	91.78

The results of the base retrieval techniques which we compare our other experimentation models against. No finetuning was performed for any of these retrievers. *fq*=full query and *on*=odtqa nltk.

As we initially anticipated, the experiments concerning the NoInstruct model proved to be stronger retrievers when compared to BM25. We attribute this to the natural language form of the queries, where the stricter nature of BM25 leads to a higher number of unmatched table terms. Using the ODTQA method for entity extraction did show slight improvements for the BM25 results, which we attribute to the reduction in noise words. Taking the query "*What is the latest UPDATED_ON date for the project 'PQR789'?*", the ODTQA method extracted entities "*latest UPDATED ON date*" and "*project*" and resulted in the TP at the 3rd position, while the full-query approach resulted in 8th position. In this example, noise words "*what*" "*is*", "*the*", "*for*", and "*PQR789*" resulted higher relevance being given to FPs by the full-query technique. We noticed such behavior throughout most of the BM25 full-query results.

NoInstruct's less rigid context-based matching, on the other hand, allows for broader matches. For instance, the TP table (Listing 4.1) for the test query "*Retrieve the email addresses of users whose passwords are set to a default value*" was ranked 2nd by NI-MaxSim, and 15th by BM25. The poor BM25 performance can be attributed to the lack of exact terms shared between the query and table attributes. Unlike BM25, the context-based approach was able to correctly match terms like *email* and *password* with *email-addr* and *pw_hash*, respectively, while also using other fields like *user_name* and *lastLoginDt* as context. While also context-based, we believe that the $\text{BERT}_{\text{BASE}}$ retriever performed as poorly as it did due to its intended role as a base model, originally trained by Devlin et al. to be further finetuned for more specific downstream tasks. As we will demonstrate shortly (4.2), additional finetuning of the model provides evidence supporting the viability of our theory.

Listing 4.1: JKS Identity Management Table

```
{
    "name": "jks_identity_management",
    "body": {
        "user_name": "varchar",
        "pwd_hash": "varchar",
        "email-addr": "varchar",
        "lastLoginDt": "date",
        "acct_status": "varchar",
        "secQ_ans": "varchar",
        "roleID_fk": "integer",
        "dept_code": "varchar"
    }
}
```

When comparing NI-base to NI-MaxSim, it is interesting to point out the increased performance of the latter in the lower HR thresholds of 1 and 3. This aligns with our idea that ColBERT is able to match on a more granular level, as opposed to the former model which is still able to capture context, but not as finely. For instance, the following query resulted in 15^{th} and 1^{st} position, respectively: "*How many issues are assigned to ASSIGNEE_NAME 'Jane Smith'?*". The poorer result demonstrated by the NI-base search technique can most likely be attributed to the single-embedding approach where the finer details of the query and table are lost. The most crucial components of the query "*issues*" and "*ASSIGNEE_NAME*" are pooled into one embedding consisting of the other query components. Similarly, the columns of the query's TP provided in Listing 4.2 are also pooled into a single embedding, negating the importance of the query-table components which are related. Using the NI-MaxSim as the search technique, noisy terms are properly ignored, encouraging a stronger result.

Listing 4.2: PFX Integration Table

```
{
    "name": "pfx_integration_dxc",
    "body": {
        "PFX_ID": "varchar",
        "COMP_NAME": "varchar",
        "INT_TYPE": "varchar",
        "LABEL_LIST": "varchar",
        "PROJ_CODE": "varchar",
        "RESOL_STATUS": "varchar",
        "TEST_ID": "varchar",
        "AFFECTS_VER": "double",
        "FIX_VER": "double",
        "CREATED_ON": "date",
        "UPDATED_ON": "date",
        "PRIORIT_LVL": "varchar",
        "TESTER_NAME": "varchar",
        "ASSIGNEE_NAME": "varchar",
        "ISSUE_ID": "varchar",
        "REPORTER_NAME": "varchar"
    }
}
```

4.2 | Finetuned Models

When measuring using MRR, we found that our best result of 0.790 was obtained when using the NoInstruct base model trained using the weighted probability distributed dataset for five epochs. We termed this model as *NI_wpd_5-MaxSim*. In second is a model we titled *NI_ud_6-MaxSim*, which uses the same base NI model, but trained for six epochs using the uniformly distributed dataset. The $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2019) model lagged behind with its best MRR score of 0.749 being achieved in the third epoch. We found BERT to also underperform across all thresholds of HR, except for HR@10, where it equaled NI_wpd_5-MaxSim, but still fell short of the NI_ud_6-MaxSim model. We found the NI_ud_6-MaxSim to easily outperform across HR thresholds of 3, 5, and 10, but beaten by NI_wpd_5-MaxSim for HR@1. We provide the results for the three individual models across all training epochs in Tables 4.2, 4.3, and 4.4. We also demon-

strate side-by-side comparisons of model HR results by epoch in Figures 4.1, 4.2, 4.3, and 4.4. Compared with our base model results (4.1), we found that finetuning via the ColBERT (Khattab and Zaharia, 2020) method improved table selection across all metrics when measured with MRR and HR@k.

Table 4.2: Finetuned BERT Results (UD)

Epoch	MRR	HR@1	HR@3	HR@5	HR@10
1	0.726	61.47	80.99	86.80	91.29
2	0.746	63.29	83.34	89.57	93.52
3	**0.749**	**63.46**	83.95	90.08	94.23
4	0.746	62.67	**84.31**	90.77	94.27
5	0.739	61.80	83.89	90.10	94.43
6	0.740	61.74	83.83	90.59	94.49
7	0.745	62.36	**84.31**	**90.99**	**94.64**

The results of the $\text{BERT}_{\text{BASE}}$ model we finetuned over several epochs using our dataset built with uniform distribution (UD) negative sampling. Model weights were trained using the ColBERT method and results obtained using late interaction MaxSim.

Table 4.3: Finetuned NoInstruct Results (UD)

Epoch	MRR	HR@1	HR@3	HR@5	HR@10
1	0.748	63.98	83.64	88.78	93.19
2	0.765	65.64	85.58	90.30	94.49
3	0.774	66.57	86.23	91.29	95.32
4	0.781	67.46	86.90	91.78	95.61
5	0.782	67.51	87.33	92.21	95.53
6	**0.786**	**67.55**	**88.18**	92.73	**96.36**
7	0.784	67.32	88.06	**92.79**	96.32

The results of the base NoInstruct model we finetuned over several epochs using our dataset built with uniform distribution (UD) negative sampling. Model weights were trained using the ColBERT method and results obtained using late interaction MaxSim.

Table 4.4: Finetuned NoInstruct Results (WPD)

Epoch	MRR	HR@1	HR@3	HR@5	HR@10
1	0.731	63.09	80.75	85.51	89.78
2	0.772	67.51	85.04	89.60	93.16
3	0.775	67.95	85.22	89.86	93.20
4	0.789	69.27	86.72	91.11	94.29
5	**0.790**	**69.39**	86.94	91.03	94.23
6	0.788	69.02	**87.09**	**91.21**	**94.49**
7	0.787	69.02	86.92	91.19	94.45

The results of the base NoInstruct model we finetuned over several epochs using our dataset built with weighted distribution probability (WPD) negative sampling. Model weights were trained using the ColBERT method and results obtained using late interaction MaxSim.

Given $\text{BERT}_{\text{BASE}}$'s MTEB IR score[1] of only 10.59, it is natural to presume this to be the reason for the poorer performance scores when comparing to NoInstruct. Although we did not finetune BERT using the weighted probability distribution dataset, we believe this is a promising approach worth evaluating. It may outperform the version finetuned on uniform distribution and help reduce the performance gap with the NoInstruct models. The slightly better results of the second NoInstruct model when compared to the first NoInstruct model can be potentially attributed to the higher presence of hard negatives. This is due to the equally-weighted nature of uniform distribution's selection process, which may yield lesser quality negatives. Furthermore, the performance gap between the two models can be potentially increased if greater weights are used during the weighted probability distribution sampling, but this is not determined by our work. Listing 4.3 provides an example of negatives taken from our test datasets selected using the two negative sampling strategies. The weighted probability distribution strategy produced a harder negative by selecting a table with greater term overlap with the query. The uniform distribution strategy, on the other hand, selected a negative example which has little to no overlap.

[1] `https://huggingface.co/spaces/mteb/leaderboard`

Listing 4.3: Negative Sampling Comparison Example

```
QUERY
Can we identify any relationships between revenue and the
number of years retained?

UNIFORM DISTRIBUTION
ipr task scheduler TSK ID TSK Name TSK Desc TSK Priority
Level TSK Due Date TSK Assigned To TSK Status TSK Created On
TSK Updated On TSK Category TSK Parent Task TSK Recurring TSK
Recurring Pattern TSK Reminder TSK Reminder Date TSK Project
ID TSK is Archived TSK Notes

WEIGHTED PROBABILITY DISTRIBUTION
vwx business intelligence BI ID Data Source BI Customer ID
sales Region BI Product Category BI Revenue USD BI Cost USD
created Dt last Updated Dt
```

Our NoInstruct_1.6 model, which is initialized from NI_ud_6-MaxSim and further finetuned on the dataset consisting of weighted probability distributed negatives, outperformed all previous models by substantial margins. The model achieved a MRR score of 0.810, an increase of $+0.02$ over NI_wpd_5-MaxSim, and also surpassed it on all HR thresholds by $+1.76$, $+2.57$, $+2.51$, and $+2.77$ for the respective values of k. The model also outperformed NI_ud_6-MaxSim by lesser or even greater margins depending on the metric. Comparison results are provided in Table 4.7. We believe this model performed so strongly due to the dynamic characteristics of the negatives, which benefit from both uniform distribution and weighted probability distribution sampling. Furthermore, we believe that such a mixture of hard and non-hard negatives encouraged the trainer to reduce any potential overtraining caused by the initial finetuning on the uniform distribution data. While we do not finetune this model beyond a single epoch due to time constraints, we would encourage potential future research to investigate the effects of further training.

Most FPs produced by our NoInstruct_1.6 method were caused by poor data synthesis, where queries were either too vague or irrelevant. For example, the following is a vague query where its TP was ranked at the 138^{th} position: *"How do I insert a new record into this schema?"*. Such a query clearly lacks any useful entities and cannot be reliably mapped to a single table with confidence. The following query, on the other hand, shows an example of a query which is not relevant to its TP table shown in Listing 4.4:

"*Are there any data retention policies or automatic cleanup processes for old records?*". The informative parts of the query show no relationship to the TP table. As a result, the table ranked 159th. Besides poor data synthesis, the NoInstruct_1.6 method also underperformed for queries consisting of terminology that is commonly shared among several tables. For instance, the query "*What is the expiry date of the 'Standard Operating Procedure' for the Manufacturing process?*" resulted in its TP being ranked 133rd. This is most likely due to terms like "*procedure*" and "*process*", which are frequently occurring terms across multiple table definitions. Incorporating snippets of cell data into the table embeddings, as done by Jin et al. in their ODTQA paper, may help address this issue.

Listing 4.4: BWV API Gateway Table

```
{
    "name": "bwv_api_gateway",
    "body": {
        "apiKey": "varchar",
        "reqURL": "varchar",
        "resp_Status": "integer",
        "reqPayload": "text",
        "respPayload": "text",
        "authToken_Id": "varchar"
    }
}
```

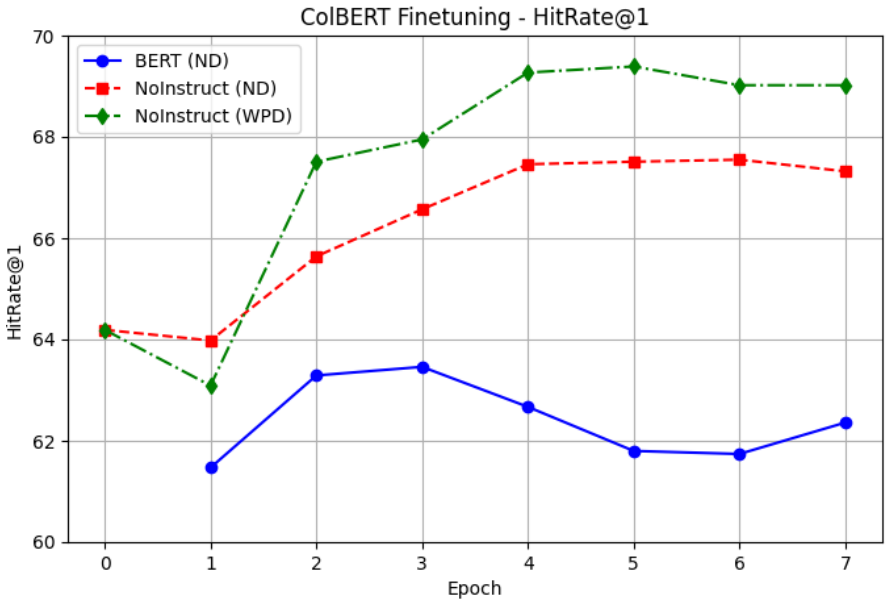


Figure 4.1: **The HitRate@1 performance across all finetuning epochs for the $\text{BERT}_{\text{BASE}}$ and NoInstruct models. *UD*=uniform distribution and *WPD*=weighted probability distribution, which represent the two datasets we curated.**

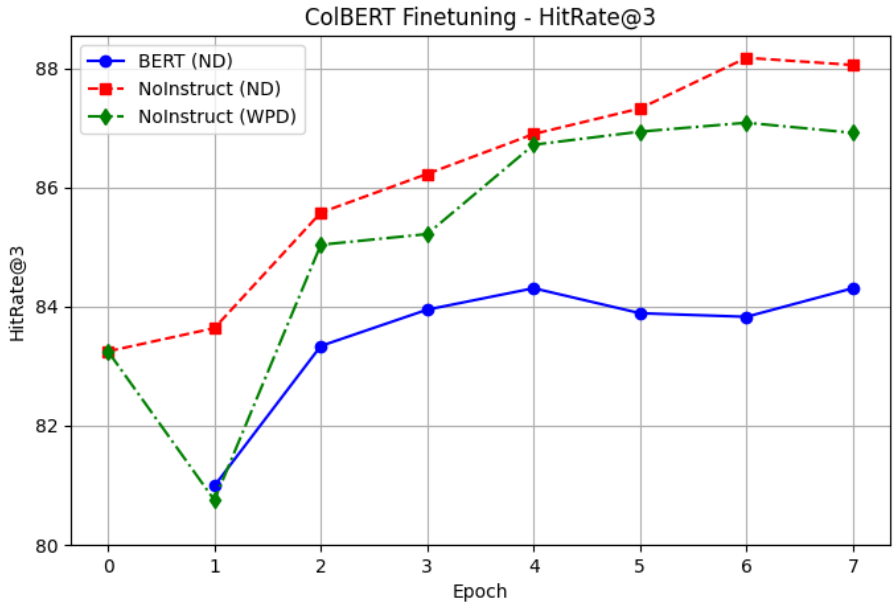


Figure 4.2: **The HitRate@3 performance across all finetuning epochs for the $\text{BERT}_{\text{BASE}}$ and NoInstruct models. *UD*=uniform distribution and *WPD*=weighted probability distribution, which represent the two datasets we curated.**

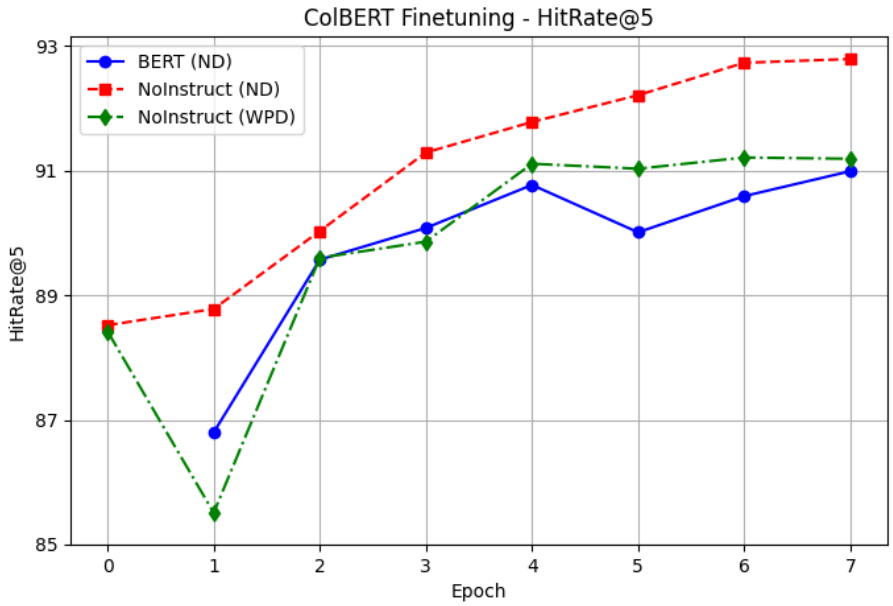


Figure 4.3: **The HitRate@5 performance across all finetuning epochs for the $\text{BERT}_{\text{BASE}}$ and NoInstruct models. *UD*=uniform distribution and *WPD*=weighted probability distribution, which represent the two datasets we curated.**

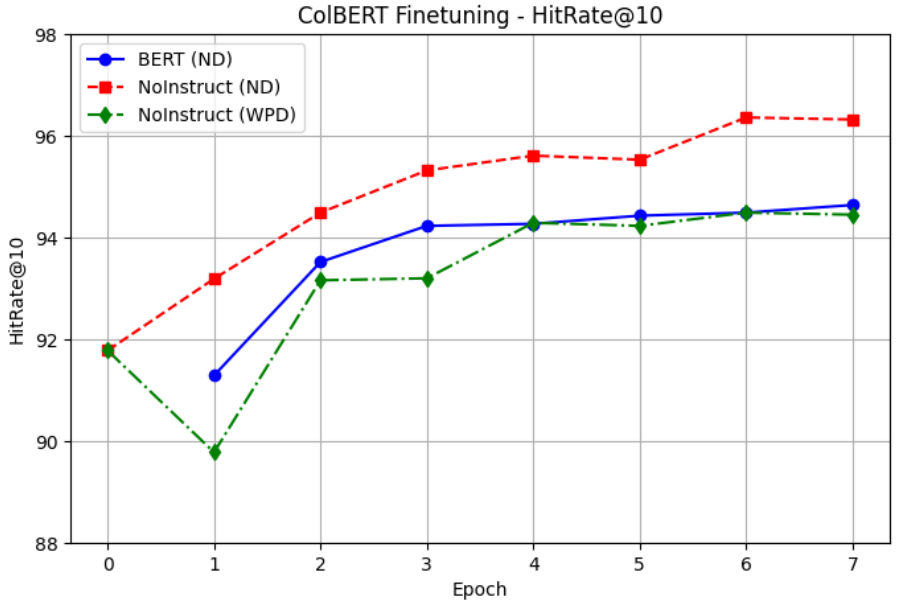


Figure 4.4: **The HitRate@10 performance across all finetuning epochs for the $\text{BERT}_{\text{BASE}}$ and NoInstruct models. *UD*=uniform distribution and *WPD*=weighted probability distribution, which represent the two datasets we curated.**

4.3 | Data Fusion

The results obtained using all data fusion techniques resulted in poorer performance, regardless of model combination or fusion technique. Our best performing approach, although still short of the previous isolated models, is termed *BM25_on-NI_6.1-CMNZ*. This approach uses BM25 and NoInstruct_6.1 as its retrievers, while utilizing CombMNZ (Fox and Shaw, 1993) for data fusion. When the superior single representation NoInstruct retriever was used instead of BM25, CombMNZ resulted in poorer results. This same behavior was also reflected when CombMNZ was swapped out for RRF (Cormack et al., 2009). On the contrary, Linear Combination (Wu et al., 2011) performed better with stronger baseline models across all three search sizes of 10k, 20, and 10. While Wu et al. report negligible impact when reducing the search size, we on the other hand found that smaller search sizes lead to respectable performance gain for our table selection task. All results are provided in Table 4.5.

Table 4.5: Data Fusion Results

Retriever	MRR	HR@1	HR@3	HR@5	HR@10
BM25_on-NI_6.1-RRF	0.780	67.61	85.91	91.15	96.13
BM25_on-NI_6.1-CMNZ	**0.787**	**67.95**	**87.79**	**92.91**	**96.60**
BM25_on-NI_6.1-LC@10k	0.651	55.48	72.69	77.23	81.42
BM25_on-NI_6.1-LC@20	0.689	56.96	77.53	83.81	90.45
BM25_on-NI_6.1-LC@10	0.691	57.18	77.67	84.11	90.67
NI-NI_6.1-RRF	0.776	66.15	87.25	92.29	96.40
NI-NI_6.1-CMNZ	0.756	64.06	84.45	90.71	95.24
NI-NI_6.1-LC@10k	0.717	60.12	80.08	87.39	92.49
NI-NI_6.1-LC@20	0.744	62.67	83.44	89.80	94.64
NI-NI_6.1-LC@10	0.746	62.79	83.54	89.90	94.68

A comparison of the best performing data fusion retrieval strategies. We compare Reciprocal Rank Fusion (RRF), CombMNZ, and Linear Combination (LC) using BM25 (entities extracted with odtqa nltk function), our NoInstruct_6.1 model, and NoInstruct single representation.

Overall, our data fusion results agree with the observation made by Beitzel et al. (2004), stating that data fusion has a tendency to degrade performance when employing retrieval strategies which are already strong in isolation. In our experiments, we used strategies of such a nature, with BM25 being the least performing strategy, scoring a strong HR@1 and HR@3 of 55.70 and 73.60, respectively. Initially, we anticipated that CombMNZ would outperform RRF due to the former's use of raw relevancy scores, but experimental results showed that for our use-case, both had similar performance.

We believe this to be due to the observations made by Beitzel et al.. Future experiments could compare the results of these two fusion techniques using worst performing retrieval strategies. Linear Combination's better performance when using stronger performing retrievers, on the other hand, is somewhat difficult to explain as it contradicts Beitzel et al.'s observations. This behavior could be due to the potential of a pretraining step, which Beitzel et al. do not account for when conducting their experiments. We believe that the process of learning weights aids in score calculation by adapting to the retrievers' respective base performance. Nevertheless, Beitzel et al.'s overall observation still holds true, as the NI-NI_6.1-LC@k retriever fails to improve upon the isolated NI_6.1 base retriever.

Analyzing the results produced by BM25_on-NI_6.1-CMNZ retriever, we can largely attribute the degradation in performance to BM25, whose influence diminishes the impact of the stronger NI_6.1 retriever. For example, the isolated BM25 retriever ranked the following query in 5^{th} position, whereas the isolated NI_6.1 retriever ranked it 2^{nd}: "*What is the next scheduled follow-up date for the lead from 'Social Media'?*". When using the BM25_on-NI_6.1-CMNZ retriever, the overall rank dropped to 5^{th}, which is 3 positions worse than the isolated NI_6.1 retriever. A similar effect is observed for the query "*Search for customers with a name containing the word 'Tech'*", where the rank decreased by 4 positions. The majority of the the poorer performing queries follow this same pattern, collectively lowering the overall performance of the retrieval technique. This reasoning also applies to the other data fusion techniques explored in this work, where the weaker retriever can be to seen to dampen the effectiveness of the stronger retriever.

Although the introduction of data fusion methods led to an overall degradation in performance, 3.52% of the test queries showed improved results across all methods when comparing to the isolated NI_6.1 retriever. The data fusion methods showed a performance gain for an average of 11.06% of queries, which indicates that data fusion techniques can be useful for some types of queries. Unfortunately, due to the diverse nature of the queries, it is difficult to discern a pattern indicating precisely which queries would benefit from data fusion. We believe this to be a strong area for future research concerning table selection. Despite this uncertainty, one potentially viable approach could be to fall back to a data fusion method when the strong isolated retriever fails to retrieve the TP. Table 4.6 lists the percentage of queries for each data fusion method which resulted in better performance when comparing to the isolated NI_6.1 retriever.

Table 4.6: Data Fusion Improvements

Retriever	Improvement
BM25_on-NI_6.1-RRF	11.13%
BM25_on-NI_6.1-CMNZ	12.31%
BM25_on-NI_6.1-LC@10k	10.28%
BM25_on-NI_6.1-LC@20	10.43%
BM25_on-NI_6.1-LC@10	10.47%
NI-NI_6.1-RRF	12.39%
NI-NI_6.1-CMNZ	11.32%
NI-NI_6.1-LC@10k	10.26%
NI-NI_6.1-LC@20	10.99%
NI-NI_6.1-LC@10	11.03%

The percentage of queries for each data fusion method which resulted in better performance when comparing to the isolated NI_6.1 retriever.

4.4 | Final Remarks

We illustrate our best performing models from each category in Table 4.7. Our NI_6.1 model greatly outperformed all other models across all HR@k threshold metrics, as well as the overall MRR metric with an increase of $+1.76$, $+1.33$, $+0.63$, $+0.40$, and $+0.02$, respective to the next best performing model. Figure 4.5 demonstrates a visual representation of the HR@k performance metrics for all evaluated models. In an effort to not overwhelm the graph, we only include the better performing LC@10 for Linear Combination. For similar reasoning, we also limit our finetuned models to the optimal epoch per group. Lastly, we do not include the unfinetuned $\text{BERT}_{\text{BASE}}$ model due to its poor performance resulting in outlier data points. Our NI_6.1 model (teal data point •) can be seen to consistently outperform all other methods across all thresholds of HR@k.

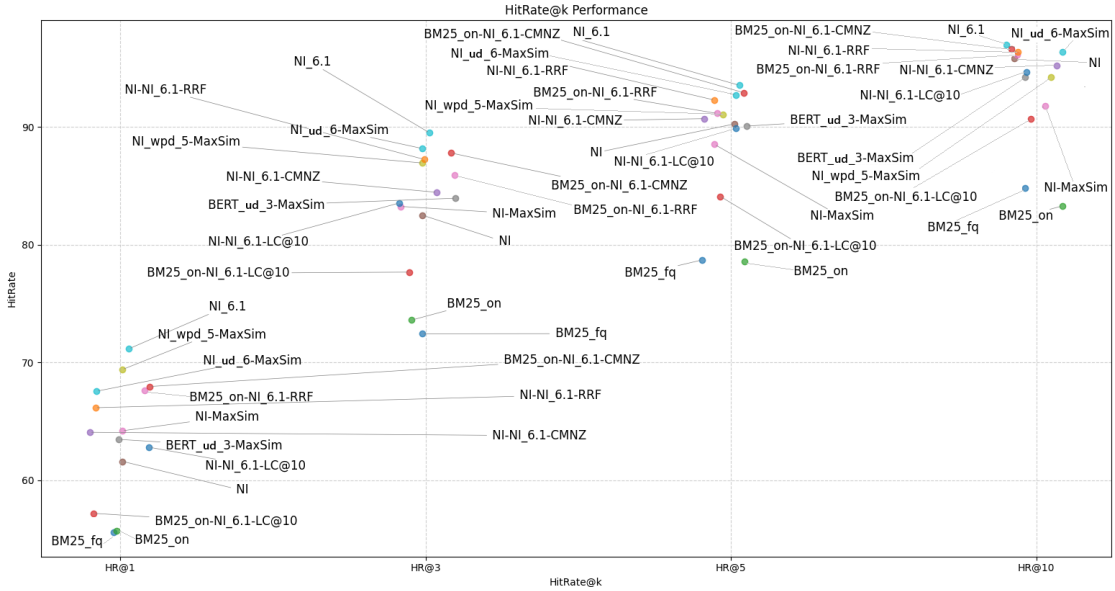


Figure 4.5: **A visual representation of the HitRate@k performance metrics for the evaluated models.**

Table 4.7: Result Comparison

Retriever	MRR	HR@1	HR@3	HR@5	HR@10
NI	0.748	64.19	83.25	88.52	91.78
NI_ud_6-MaxSim	0.786	67.55	88.18	92.73	96.36
NI_wpd_5-MaxSim	0.790	69.39	86.94	91.03	94.23
BM25_on-NI_6.1-CMNZ	0.787	67.95	87.79	92.91	96.60
NI_6.1	**0.810**	**71.15**	**89.51**	**93.54**	**97.00**

A comparison of the best performing models respective to their result groups. *ud_6* indicates the 6^{th} epoch trained using the *ud*=uniform distribution dataset. *wpd_5* indicates the 5^{th} epoch trained using the weighted probability distribution dataset. *on* indicates BM25 extracted entities using odtqa nltk function.

4.5 | Summary

In this section, we reported the results of the techniques and models employed for table selection. We've shown that finetuning via the ColBERT method substantially improves performance when compared to our unfinetuned base models, such as NoInstruct and $\text{BERT}_{\text{BASE}}$. Our best performing finetuned model, NoInstruct_6.1, greatly outperformed all other models and techniques demonstrated in this work for both the MRR and HR@k

metrics. Lastly, we've shown that for our setup, data fusion yielded no improvements over the individual models' performance, which is in agreement with the claims made by Beitzel et al. (2004).

5

Conclusions

This section revisits the initial objectives and evaluates whether they were achieved based on the experimentation and results of this work. We then discuss some of the potential limitations that should be considered by the reader. Finally, we discuss how future research may improve or expand upon this work by suggesting the areas we believe to be of interest.

5.1 | Achieved Objectives

The objectives outlined in this work served to establish a road map for building a system which given a natural query, is able to select the correct table to be used for further downstream tasks such as Text-to-SQL. The objectives of this work cover the curation of the training and test datasets, the finetuning techniques, and and the evaluation and comparison of the different search techniques.

5.1.1 | Objective 1

Our first objective was to select the LLM to use and identify the prompting technique to curate the training and test datasets. Due to the limitation of our company's subscription plan at the time, we decided to utilize Anthropic's Claude Sonnet 3 (Askell et al., 2021). While we could have used other equally viable LLMs such as those from the GPT series (Achiam et al., 2023) or that of Google's Gemini (Team et al., 2024), processing would have been slow due to the the limitations imposed on the publicly available free-tier. We also considered open-sourced models such as T5 (Raffel et al., 2020), but at the time of this writing, closed-source LLMs like Claude Sonnet 3 were the state of the art. For selecting our prompting technique, we followed the examples-first-instructions-last

methodology demonstrated by successful text-to-SQL research such as Gao et al. (2024). This technique resulted in datasets which were both satisfactory and diverse. Refer to Appendix A for data samples, and Appendix B for links to the source code housing the datasets themselves.

5.1.2 | Objective 2

The second objective aimed to select negative sampling techniques to include negative examples in our training set. We also aimed to compare them against one another. We selected two sampling techniques, uniform distribution and weighted probability distribution, both adopted from Solatorio (2024). While the author uses the latter technique to curate positives, we on the other hand used a variation of the technique to select the negatives. As demonstrated by our experiment's results, both sampling techniques lead to stronger contrastive learning for all models when evaluating the table selection IR task. We also noted stronger results when finetuning our models using the dataset built using weighted probability distribution. The link to the dataset of triplets (q, p^+, p^-) for each respective sampling technique is provided in Appendix B. Samples are provided in Appendix A.

5.1.3 | Objective 3

For the third objective, we aimed to select the base models to use for embedding our queries and tables. We also required these models to evaluate the training and negative sampling techniques selected for this work. When selecting the models, we compared three main attributes: parameter size, dimension size, and MTEB IR score. We gave preference to models having smaller parameter and dimension sizes due to the low cost and low latency requirements of the business use-case. From all embedding models considered, we selected the NoInstruct model, a followup to the first iteration model published by Solatorio (2024). Compared to other models, this model stood out with a strong MTEB IR score of 51.99, as well as having smaller parameter and dimension sizes of 33M and 384, respectively. As shown in the previous section (4), the NoInstruct model ultimately demonstrated stronger table selection retrieval results compared with the foundational BERT (Devlin et al., 2019) model.

5.1.4 | Objective 4

Our fourth objective aimed to select a technique to finetune the base NoInstruct and BERT (Devlin et al., 2019) embeddings models. When comparing approaches, we de-

cided to proceed with ColBERT (Khattab and Zaharia, 2020), which trains the models using contrastive learning through the MaxSim technique. This decision was taken with inspiration from the table selection research published by Jin et al. (2023), where a strong $\text{BERT}_{\text{BASE}}$ model was finetuned using a similar contrastive learning MaxSim approach as ColBERT. Compared with other approaches like SANTA (Li et al., 2023) and TaPas (Herzig et al., 2021), and others, we felt that ColBERT offered a more granular approach to the column components of the table data. For our task of table selection, the models we finetuned using the ColBERT method demonstrated a substantial increase in IR performance. We provide all implementation code at the online link provided in Appendix B.

5.1.5 | Objective 5

For the fifth objective, we aimed to finetune and compare models using different combinations of base embedding models and training sets using the ColBERT (Khattab and Zaharia, 2020) method. We finetuned the $\text{BERT}_{\text{BASE}}$ model using the synthetic dataset housing the uniformly distributed negatives for seven epochs. We also finetuned the NoInstruct model using the same dataset, as well as the additional dataset which was built using the weighted probability distribution sampling technique. This model was finetuned for seven epochs for each dataset. We further finetuned our already-finetuned NoInstruct model, which we describe in detail in the methodology section (3), to produce our best performing model we term NoInstruct_6.1. All model epochs were evaluated and results published in the previous section (4). We provide the trained weights for all individual model epochs at the online link found in Appendix B.

5.1.6 | Objective 6

For our final objective, we aimed to improve the results of the individual base techniques by incorporating a multi-retrieval approach, where the results of multiple retrieval techniques are combined. Overall, we compared two combinations of retrievers across three individual data fusion techniques. For the retriever combinations, we evaluated the BM25-NI_6.1 and NI-NI_6.1 pairs, while for the data fusion techniques we utilized RRF (Cormack et al., 2009), CombMNZ (Fox and Shaw, 1993), and Linear Combination (Wu et al., 2011). For Linear Combination, we experimented with three different search-size values: 10,000, 20, and 10. Ultimately, all data fusion techniques underperformed the previously established NI_6.1 model. The weights for the

Linear Combination data fusion technique can be found through the link provided in Appendix B.

5.2 | Critique & Limitations

While conducting our research, numerous limitations were noted. Firstly, and most importantly we believe, is the lack of real-world data and the purely synthetic nature of the datasets ultimately used. Our training sets were curated entirely through LLM text generation, so any potential biases within the model may have been transferred onto the data itself. Furthermore, while seemingly diverse at a first glance, the queries and table themselves may inherently follow some obscure pattern imposed by the LLM. Unfortunately, due to the large number of training pairs and lack of resources and time, it was not possible to verify every instance. A much smaller sample size of roughly fifty pairs were instead manually verified. Lastly, as a result of the instructions given to the LLM, the tables that were ultimately generated followed a business-oriented tone, frequently utilizing obscure or domain-specific terminology. While this was our intention given the use-case of our business, we believe that this has the potential to negatively skew performance where queries or tables refrain from following this pattern. Furthermore, our use of a fixed table column range of 6–20 may limit performance on tables with fewer or more columns. This is contrary to other works like Jin et al. (2023) and ColBERT (Khattab and Zaharia, 2020), who make use of publicly available and diverse data through datasets like WikiSQL (Zhong et al., 2017) and MS MARCO (Bajaj et al., 2018), respectively. It should also be noted that these datasets consist of millions of examples, while ours only contain around 5k.

Our datasets also lack examples of multi-table queries, where a query must be answered by consulting multiple tables through the use of SQL keywords like *JOIN*. While we purposely conducted our work in this manner due to the business use-case, we understand that this may pose as a limitation to use-cases looking to perform cross-table answering. While such a requirement can be fulfilled through similar LLM generation, it would undoubtedly add more complexity to the dataset curation process. Given the time constraints of our work, we decided to not include any examples which would not be required by our explicit use-case. It should be noted that the absence of multi-table queries is also a trait found in the work published by Jin et al., where the underlying datasets do not make use of any table joining. On the contrary, the table selection work by Chopra and Azam (2024), which utilize the Spider (Yu et al., 2018) dataset, do cross-table answering.

Finally, we would like to mention the shortcoming of the ODTQA NLTK function used to extract entities from the queries for our BM25 retrieval experiments. While this function's use demonstrates a strong performance in the work published by Jin et al., upon manual verification of the entities extracted, we noticed that the function did not always correctly select the appropriate noun phrases. Additionally, the function would sometimes split entities into multiple words or phrases, when it should not have. While these instance are for the most part less common, their presence cannot be ignored. Such behavior would undoubtedly skew the BM25 results reported in our experiments, which would include the baseline results (4.1), as well as data fusion combinations (4.3) which made use of the search technique. Unfortunately, to truly understand the impact of this behavior, sample sizes would need to be taken and manually verified within a controlled setting. We leave this task for potential future research.

5.3 | Future Work

Given the limited number of works covering table selection, as well as the somewhat novel approach of our implementation, we believe there to be several areas of interest for potential future research. Firstly, the datasets may be expanded upon to include non-business-like data to help the finetuned models adapt to other types of queries and tables. Secondly, in our work we only make use of the NoInstruct and $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2019) models, but future research may include other embedding models. We also only train our models for a total of seven epochs, training for further epochs may yield different results. We also believe that a finetuned single embedding approach, instead of MaxSim via the ColBERT (Khattab and Zaharia, 2020) method, is also worth comparing. For instance, using the training methods demonstrated by GIST (Solatorio, 2024) and SANTA (Li et al., 2023) are viable options to training single representations. Lastly, further negative sampling techniques could also be explored, especially those which function on-the-fly, where negatives are resampled immediately following each epoch by using the latest weights of the model currently undergoing finetuning. ANCE (Xiong et al., 2021) is one such example where hard negatives are sampled in this manner.

5.4 | Conclusion

In this work, we introduced the issue of table dependency in current text-to-SQL implementations and highlighted the need for further research in this area. We reviewed existing studies that directly address this challenge, as well as related works that, while

focused on different goals, employed tools and techniques relevant to our research. Our approach leveraged both sparse and dense retrieval methods, using BM25 for the former and ColBERT-style late interaction maximum vector similarity for the latter. We utilized two embedding models, $\text{BERT}_{\text{BASE}}$ and GIST-NoInstruct, and applied contrastive learning techniques inspired by ColBERT to enhance model performance. To support this, we curated training datasets from scratch using LLM-based prompting. We explored various negative sampling strategies, including uniform and weighted probability distributions. Additionally, we experimented with multiple data fusion techniques, such as RRF, CombMNZ, and linear combination to combine retrieval results from different methods like BM25 and MaxSim.

The strong results achieved in our experiments demonstrate that the table selection techniques employed are viable candidates for overcoming the table dependency that limits text-to-SQL solutions. The ColBERT training method proved effective in producing strong embedding models, especially when combined with negative sampling techniques such as uniform distribution and weighted probability distribution. Additionally, constructing in-house datasets via LLM prompting proved both time-efficient and cost-effective. Although not as successful as our standalone models, experiments with data fusion techniques, such as RRF, CombMNZ, and linear combination, also indicated potential for improvement in areas where other models fall short.

The strong results from our experiments demonstrate that the table selection techniques employed are viable for mitigating the table dependency that hampers text-to-SQL solutions. The ColBERT training method was particularly effective in producing robust embedding models, especially when paired with negative sampling strategies such as uniform and weighted probability distributions. Our best-performing model, NI_6.1, significantly outperformed both lexical search and dense retrieval baselines using BERT and base NoInstruct. Furthermore, generating in-house datasets through LLM prompting proved to be both time-efficient and cost-effective. While data fusion techniques, such as RRF, CombMNZ, and linear combination did not surpass our standalone models, they showed promise for addressing cases where individual models underperform.

While applying the ColBERT MaxSim approach using our NI_6.1 model proves to be a viable solution, it is important to consider the costs associated with an increasing number of tables, as each table column requires a separate embedding. Storing embeddings is memory-intensive, particularly when using the HNSW method, which can lead to higher computational costs. Additionally, performing embedding inference in cloud environments such as AWS may further increase expenses, as many cloud platforms charge per token inference. Latency is another critical factor, as this approach introduces a preliminary step to identify the target table. In our experiments, embedding the

query and selecting the relevant tables added between 1 and 3 seconds to the response time. Such delays may be unacceptable for certain text-to-SQL applications. However, this latency can potentially be reduced by running the embedding model locally rather than in the cloud.

Incorrect table selection can also cause significant issues for downstream tasks. For example, providing an incorrect table to the LLM may result in the model either raising an exception by refusing to generate the SQL, or, in the worst case, attempting to generate the equivalent SQL query based on the wrong table. This can lead to incorrect results being presented to the user. Such errors are often difficult to detect and may only be noticed by the end user. Relying solely on the user's observation is not reliable and can be dangerous, particularly if the user assumes the output is correct. Therefore, model interpretability must also be taken into account to mitigate these risks.

A

Sample Training Examples

In this section, we provide samples of example triplets (q, p^+, p^-) from our two datasets populated with negatives sampled using uniform distribution and weighted probability distribution, respectively.

A.0.1 | Uniform Distribution

Below are two examples taken from the training dataset populated with negatives sampled using uniform distribution. Each example starts with the query, followed by the positive table, and ends with the negative table.

Listing A.1: Uniform Distribution Example #1

```
What are the notification recipients for the deployment of
'My App'?

software deployment Deployment ID Application Name Version
Build Number Release Status Deployment Status Environment
Deployed By Approved By Repository URL Branch Commit Hash
Deployment Date Last Updated Rollback Available Rollback
Version Artifact Location Config File Path Database
Migration Migration Script Path Dependency List Security
Patch Level Supported OS Cloud Provider Instance Type
Containerized Container Image Helm Chart Version Kubernetes
Namespace Monitoring Enabled Log Location Error Logs
Performance Metrics Notification Recipients

qwk compliance audit QCK ID Business Unit Audit Type Audit
Ref ID Loc Code Fac Name Rev Num Ent Type Site Addr QCK Ref
ID1 QCK Ref ID10 QCK Ref ID9 Comp Status Audit Date Auditor
Name Findings Desc Risk Level
```

Listing A.2: Uniform Distribution Example #2

```
Get the full address for companies located in 'TX' with a
zip code starting with '75'.

rbd tax compliance Tax ID Country Code Tax Region Comp Name
Addr Line1 Addr Line2 City State Prov Zip Code Filing Status
Last Updated On

fgh mobile app analytics app Id app Name country Code
device Type OS Version install Date last Opened user Id
session Duration in App Purchases ad Revenue fgh Analytics
Id
```

A.0.2 | Weighted Probability Distribution

Below are two examples taken from the training dataset populated with negatives sampled using weighted probability distribution. Each example starts with the query, followed by the positive table, and ends with the negative table.

Listing A.3: Weighted Probability Distribution Example #1

```
Is there a way to check the API version being used at a
particular point in time?

vvv blockchain integration BC Tx Id BC Network BC Node Type
BC Consensus BC Block Height BC Block Hash BC Prev Block Hash
BC Merkle Root BC Nonce BC Timestamp BC Difficulty Target API
Endpoint API Version Integration Status Last Sync Time Err Log
Integration Notes

kpi data mining KPI ID Data Source Metric
Name Metric Alias Target Value Actual Value Status Calc Method
Freq Biz Unit Dept ID Last Updated Date is Active Flag Remarks
```

Listing A.4: Weighted Probability Distribution Example #2

```
What records have an uptime greater than a certain number of
hours?

vwx system diagnostics sys id diag time cpu util mem used disk
usage net traffic sys temp log level err count warn count last
reboot uptime hrs softwre Ver patch level node id cluster name
env details

dhcp ip address management IP Addr MAC Addr Subnet Mask
Gateway Lease Time DHCP Server Client Host Name Assigned
Date Expiry Date Client ID Is Reserved Vendor Class ID User
Class ID DHCP Pool
```

B

Source Code

The source code and datasets used for this research can be downloaded from our public Google Drive[1]. Table B.1 below provides further detail into the content of the downloadable directories. Unfortunately, the trained model weights could not be uploaded due to their disk size of 11 GiBs (compressed). For those wishing to implement last-meter/mile experiments, please reach out to us by sending an email.

[1] https://drive.google.com/drive/folders/1DeZOK148a4HOYzz2wKJ-e4Dv51f-z_5i?usp=drive_link

Table B.1: Directory Information

Path	Description
code/	All code
code/colbert_embed/	Code to embed text using the models trained via the ColBERT method
code/common/	Common code which is used by multiple classes
code/graphs/	Code to generate the metric visuals presented in this work
code/synthesize/	Code to synthesize the datasets
dataset/	All datasets
dataset/test/	The test datasets used to evaluate the retrievers
dataset/train/	The datasets used for training the models
dataset/train/linear_comb/	The datasets used to train the MLR model for Linear Combination data fusion
dataset/train/uniform_distribution/	The training dataset built using uniform distribution negative sampling
dataset/train/weighted_probability/	The training dataset built using weighted probability distribution negative sampling

A description of the directories found inside the downloadable content pertaining to the implementation of this research.

References

OpenAI Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haim ing Bao, Mo Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Made laine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Benjamin Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Sim'on Posada Fishman, Juston Forte, Is abella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Raphael Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Jo hannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Lukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Hendrik Kirchner, Jamie Ryan Kiros, Matthew Knight, Daniel Kokotajlo, Lukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Li, Rachel Lim, Molly Lin, Stephanie Lin, Ma teusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel P. Mossing, Tong Mu, Mira Murati, Oleg Murk, David M'ely, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Ouyang Long, Cullen O'Keefe, Jakub W. Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alexandre Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Pondé de Oliveira Pinto, Michael Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack W. Rae, Aditya Ramesh, Cameron Raymond, Francis Real,

Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario D. Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas A. Tezak, Madeleine Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cer'on Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll L. Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qim ing Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report. 2023. URL `https://api.semanticscholar.org/CorpusID:257532815`.

Akari Asai, Timo Schick, Patrick Lewis, Xilun Chen, Gautier Izacard, Sebastian Riedel, Hannaneh Hajishirzi, and Wen-tau Yih. Task-aware retrieval with instructions. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 3650–3675, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.225. URL `https://aclanthology.org/2023.findings-acl.225/`.

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Benjamin Mann, Nova Dassarma, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, John Kernion, Kamal Ndousse, Catherine Olsson, Dario Amodei, Tom B. Brown, Jack Clark, Sam McCandlish, Christopher Olah, and Jared Kaplan. A general language assistant as a laboratory for alignment. *ArXiv*, abs/2112.00861, 2021.

S. Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC/ASWC*, 2007. URL `https://api.semanticscholar.org/CorpusID:7278297`.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. Ms marco: A human generated machine reading comprehension dataset, 2018. URL `https://arxiv.org/abs/1611.09268`.

Steven M Beitzel, Eric C Jensen, Abdur Chowdhury, David Grossman, Ophir Frieder, and Nazli Goharian. Fusion of effective retrieval strategies in the same information retrieval system. *Journal of the American Society for Information Science and Technology*, 55(10):859–868, 2004.

Chandra Bhagavatula, Thanapon Noraset, and Doug Downey. Tabel: Entity linking in web tables. In *International Workshop on the Semantic Web*, 2015. URL `https://api.semanticscholar.org/CorpusID:14265783`.

Steven Bird. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL 2006 interactive presentation sessions*, pages 69–72, 2006.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss,

Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Ma teusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020.

Nuo Chen, Linjun Shou, Ming Gong, Jian Pei, Chenyu You, Jianhui Chang, Daxin Jiang, and Jia Li. Bridge the gap between language models and tabular understanding. *ArXiv*, abs/2302.09302, 2023.

Ankush Chopra and Rauful Azam. Enhancing natural language query to sql query generation through classification-based table selection. In Lazaros Iliadis, Ilias Maglogiannis, Antonios Papaleonidas, Elias Pimenidis, and Chrisina Jayne, editors, *Engineering Applications of Neural Networks*, page 152–165, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-62495-7.

Gordon V. Cormack, Charles L A Clarke, and Stefan Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, page 758–759, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605584836. doi: 10.1145/1571941.1572114. URL `https://doi.org/10.1145/1571941.1572114`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

Laura Dietz. Trec car y3: Complex answer retrieval overview. 2019. URL `https://api.semanticscholar.org/CorpusID:209513909`.

Haoyu Dong and Zhiruo Wang. Large language models for tabular data: Progresses and future directions. SIGIR '24, page 2997–3000, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704314. doi: 10.1145/3626772.3661384. URL `https://doi.org/10.1145/3626772.3661384`.

Edward A. Fox and Joseph A. Shaw. Combination of multiple searches. In *Text Retrieval Conference*, 1993. URL `https://api.semanticscholar.org/CorpusID:1309301`.

D. Frank Hsu and Isak Taksa. Comparing rank and score combination methods for data fusion in information retrieval. *Information Retrieval*, 8(3):449–480, jan 2005. ISSN 1573-7659. doi: 10.1007/s10791-005-6994-4.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation, 2023. URL `https://arxiv.org/abs/2308.15363`.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145, January 2024. ISSN 2150-8097. doi: 10.14778/3641204.3641221.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. SimCSE: Simple contrastive learning of sentence embeddings. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online

and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.552. URL https://aclanthology.org/2021.emnlp-main.552/.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. *ArXiv*, abs/2006.03654, 2020.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. TaPas: Weakly supervised table parsing via pre-training. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.398. URL https://aclanthology.org/2020.acl-main.398/.

Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian Eisenschlos. Open domain question answering over tables via dense retrieval. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 512–519, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.43. URL https://aclanthology.org/2021.naacl-main.43/.

Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. Next-generation database interfaces: A survey of llm-based text-to-sql, 2025. URL https://arxiv.org/abs/2406.08426.

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing - STOC '98*, Dallas, Texas, United States, 1998. ISBN 978-0-89791-962-3. doi: 10.1145/276698.276876. URL http://portal.acm.org/citation.cfm?doid=276698.276876.

Nengzheng Jin, Dongfang Li, Junying Chen, Joanna Siebert, and Qingcai Chen. Enhancing open-domain table question answering via syntax- and structure-aware dense retrieval. In Jong C. Park, Yuki Arase, Baotian Hu, Wei Lu, Derry Wijaya, Ayu Purwarianti, and Adila Alfa Krisnadhi, editors, *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 157–165, Nusa Dua, Bali, November 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.ijcnlp-short.18. URL https://aclanthology.org/2023.ijcnlp-short.18/.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550/.

Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 39–48, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380164. doi: 10.1145/3397271.3401075. URL https://doi.org/10.1145/3397271.3401075.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Skip-thought vectors. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 3294–3302, Cambridge, MA, USA, 2015. MIT Press.

Xianming Li and Jing Li. Aoe: Angle-optimized embeddings for semantic textual similarity. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1825–1839, 2024.

Xinze Li, Zhenghao Liu, Chenyan Xiong, Shi Yu, Yu Gu, Zhiyuan Liu, and Ge Yu. Structure-aware language model pretraining improves dense retrieval on structured data. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 11560–11574, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.734. URL https://aclanthology.org/2023.findings-acl.734/.

J. MacQueen. Some methods for classification and analysis of multivariate observations. 1967. URL https://api.semanticscholar.org/CorpusID:6278891.

Yu A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42 (4):824–836, April 2020. ISSN 0162-8828, 2160-9292, 1939-3539. doi: 10.1109/TPAMI.2018.2889473.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Neural Information Processing Systems*, 2013. URL https://api.semanticscholar.org/CorpusID:16447573.

Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. MTEB: Massive text embedding benchmark. In Andreas Vlachos and Isabelle Augenstein, editors, *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2014–2037, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.eacl-main.148. URL https://aclanthology.org/2023.eacl-main.148/.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. URL https://aclanthology.org/N18-1202.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in text-to-SQL. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=CvGqMD5OtX.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. 21(1), jan 2020. ISSN 1532-4435.

Petar Ristoski and Heiko Paulheim. Rdf2vec: Rdf graph embeddings for data mining. In *International Workshop on the Semantic Web*, 2016. URL `https://api.semanticscholar.org/CorpusID:35288341`.

Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *Nist Special Publication Sp*, 109:109, 1995.

Joshua Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. Contrastive learning with hard negative samples. *ArXiv*, abs/2010.04592, 2020.

Aivin V. Solatorio. Gistembed: Guided in-sample selection of training negatives for text embedding fine-tuning, 2024. URL `https://arxiv.org/abs/2402.16829`.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul R. Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, Jack Krawczyk, Cosmo Du, Ed Chi, Heng-Tze Cheng, Eric Ni, Purvi Shah, Patrick Kane, Betty Chan, Manaal Faruqui, Aliaksei Severyn, Hanzhao Lin, YaGuang Li, Yong Cheng, Abe Ittycheriah, Mahdis Mahdieh, Mia Chen, Pei Sun, Dustin Tran, Sumit Bagri, Balaji Lakshminarayanan, Jeremiah Liu, Andras Orban, Fabian Güra, Hao Zhou, Xinying Song, Aurelien Boffy, Harish Ganapathy, Steven Zheng, HyunJeong Choe, Ágoston Weisz, Tao Zhu, Yifeng Lu, Siddharth Gopal, Jarrod Kahn, Maciej Kula, Jeff Pitman, Rushin Shah, Emanuel Taropa, Majd Al Merey, Martin Baeuml, Zhifeng Chen, Laurent El Shafey, Yujing Zhang, Olcan Sercinoglu, George Tucker, Enrique Piqueras, Maxim Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez, Misha Khalman, Jakub Sygnowski, Alexandre Frechette, Charlotte Smith, Laura Culp, Lev Proleev, Yi Luan, Xi Chen, James Lottes, Nathan Schucher, Federico Lebron, Alban Rrustemi, Natalie Clay, Phil Crone, Tomas Kocisky, Jeffrey Zhao, Bartek Perz, Dian Yu, Heidi Howard, Adam Bloniarz, Jack W. Rae, Han Lu, Laurent Sifre, Marcello Maggioni, Fred Alcober, Dan Garrette, Megan Barnes, Shantanu Thakoor, Jacob Austin, Gabriel Barth-Maron, William Wong, Rishabh Joshi, Rahma Chaabouni, Deeni Fatiha, Arun Ahuja, Gaurav Singh Tomar, Evan Senter, Martin Chadwick, Ilya Kornakov, Nithya Attaluri, Iñaki Iturrate, Ruibo Liu, Yunxuan Li, Sarah Cogan, Jeremy Chen, Chao Jia, Chenjie Gu, Qiao Zhang, Jordan Grimstad, Ale Jakse Hartman, Xavier Garcia, Thanumalayan Sankaranarayana Pillai, Jacob Devlin, Michael Laskin, Diego de Las Casas, Dasha Valter, Connie Tao, Lorenzo Blanco, Adrià Puigdomènech Badia, David Reitter, Mianna Chen, Jenny Brennan, Clara Rivera, Sergey Brin, Shariq Iqbal, Gabriela Surita, Jane Labanowski, Abhi Rao, Stephanie Winkler, Emilio Parisotto, Yiming Gu, Kate Olszewska, Ravi Addanki, Antoine Miech, Annie Louis, Denis Teplyashin, Geoff Brown, Elliot Catt, Jan Balaguer, Jackie Xiang, Pidong Wang, Zoe Ashwood, Anton Briukhov, Albert Webson, Sanjay Ganapathy, Smit Sanghavi, Ajay Kannan, Ming-Wei Chang, Axel Stjerngren, Josip Djolonga, Yuting Sun, Ankur Bapna, Matthew Aitchison, Pedram Pejman, Henryk Michalewski, Tianhe Yu, Cindy Wang, Juliette Love, Junwhan Ahn, Dawn Bloxwich, Kehang Han, Peter Humphreys, Thibault Sellam, James Bradbury, Varun Godbole, Sina Samangooei, Bogdan Damoc, Alex Kaskasoli, Sébastien M. R. Arnold, Vijay Vasudevan, Shubham Agrawal, Jason Riesa, Dmitry Lepikhin, Richard Tanburn, Srivatsan Srinivasan, Hyeontaek

Lim, Sarah Hodkinson, Pranav Shyam, Johan Ferret, Steven Hand, Ankush Garg, Tom Le Paine, Jian Li, Yujia Li, Minh Giang, Alexander Neitz, Zaheer Abbas, Sarah York, Machel Reid, Elizabeth Cole, Aakanksha Chowdhery, Dipanjan Das, Dominika Rogozińska, Vitaliy Nikolaev, Pablo Sprechmann, Zachary Nado, Lukas Zilka, Flavien Prost, Luheng He, Marianne Monteiro, Gaurav Mishra, Chris Welty, Josh Newlan, Dawei Jia, Miltiadis Allamanis, Clara Huiyi Hu, Raoul de Liedekerke, Justin Gilmer, Carl Saroufim, Shruti Rijhwani, Shaobo Hou, Disha Shrivastava, Anirudh Baddepudi, Alex Goldin, Adnan Ozturel, Albin Cassirer, Yunhan Xu, Daniel Sohn, Devendra Sachan, Reinald Kim Amplayo, Craig Swanson, Dessie Petrova, Shashi Narayan, Arthur Guez, Siddhartha Brahma, Jessica Landon, Miteyan Patel, Ruizhe Zhao, Kevin Villela, Luyu Wang, Wenhao Jia, Matthew Rahtz, Mai Giménez, Legg Yeung, James Keeling, Petko Georgiev, Diana Mincu, Boxi Wu, Salem Haykal, Rachel Saputro, Kiran Vodrahalli, James Qin, Zeynep Cankara, Abhanshu Sharma, Nick Fernando, Will Hawkins, Behnam Neyshabur, Solomon Kim, Adrian Hutter, Priyanka Agrawal, Alex Castro-Ros, George van den Driessche, Tao Wang, Fan Yang, Shuo yiin Chang, Paul Komarek, Ross McIlroy, Mario Lučić, Guodong Zhang, Wael Farhan, Michael Sharman, Paul Natsev, Paul Michel, Yamini Bansal, Siyuan Qiao, Kris Cao, Siamak Shakeri, Christina Butterfield, Justin Chung, Paul Kishan Rubenstein, Shivani Agrawal, Arthur Mensch, Kedar Soparkar, Karel Lenc, Timothy Chung, Aedan Pope, Loren Maggiore, Jackie Kay, Priya Jhakra, Shibo Wang, Joshua Maynez, Mary Phuong, Taylor Tobin, Andrea Tacchetti, Maja Trebacz, Kevin Robinson, Yash Katariya, Sebastian Riedel, Paige Bailey, Kefan Xiao, Nimesh Ghelani, Lora Aroyo, Ambrose Slone, Neil Houlsby, Xuehan Xiong, Zhen Yang, Elena Gribovskaya, Jonas Adler, Mateo Wirth, Lisa Lee, Music Li, Thais Kagohara, Jay Pavagadhi, Sophie Bridgers, Anna Bortsova, Sanjay Ghemawat, Zafarali Ahmed, Tianqi Liu, Richard Powell, Vijay Bolina, Mariko Iinuma, Polina Zablotskaia, James Besley, Da-Woon Chung, Timothy Dozat, Ramona Comanescu, Xiance Si, Jeremy Greer, Guolong Su, Martin Polacek, Raphaël Lopez Kaufman, Simon Tokumine, Hexiang Hu, Elena Buchatskaya, Yingjie Miao, Mohamed Elhawaty, Aditya Siddhant, Nenad Tomasev, Jinwei Xing, Christina Greer, Helen Miller, Shereen Ashraf, Aurko Roy, Zizhao Zhang, Ada Ma, Angelos Filos, Milos Besta, Rory Blevins, Ted Klimenko, Chih-Kuan Yeh, Soravit Changpinyo, Jiaqi Mu, Oscar Chang, Mantas Pajarskas, Carrie Muir, Vered Cohen, Charline Le Lan, Krishna Haridasan, Amit Marathe, Steven Hansen, Sholto Douglas, Rajkumar Samuel, Mingqiu Wang, Sophia Austin, Chang Lan, Jiepu Jiang, Justin Chiu, Jaime Alonso Lorenzo, Lars Lowe Sjösund, Sébastien Cevey, Zach Gleicher, Thi Avrahami, Anudhyan Boral, Hansa Srinivasan, Vittorio Selo, Rhys May, Konstantinos Aisopos, Léonard Hussenot, Livio Baldini Soares, Kate Baumli, Michael B. Chang, Adrià Recasens, Ben Caine, Alexander Pritzel, Filip Pavetic, Fabio Pardo, Anita Gergely, Justin Frye, Vinay Ramasesh, Dan Horgan, Kartikeya Badola, Nora Kassner, Subhrajit Roy, Ethan Dyer, Víctor Campos Campos, Alex Tomala, Yunhao Tang, Dalia El Badawy, Elspeth White, Basil Mustafa, Oran Lang, Abhishek Jindal, Sharad Vikram, Zhitao Gong, Sergi Caelles, Ross Hemsley, Gregory Thornton, Fangxiaoyu Feng, Wojciech Stokowiec, Ce Zheng, Phoebe Thacker, Çağlar Ünlü, Zhishuai Zhang, Mohammad Saleh, James Svensson, Max Bileschi, Piyush Patil, Ankesh Anand, Roman Ring, Katerina Tsihlas, Arpi Vezer, Marco Selvi, Toby Shevlane, Mikel Rodriguez, Tom Kwiatkowski, Samira Daruki, Keran Rong, Allan Dafoe, Nicholas FitzGerald, Keren Gu-Lemberg, Mina Khan, Lisa Anne Hendricks, Marie Pellat, Vladimir Feinberg, James Cobon-Kerr, Tara Sainath, Maribeth Rauh, Sayed Hadi Hashemi, Richard Ives, Yana Hasson, Eric Noland, Yuan Cao, Nathan Byrd, Le Hou, Qingze Wang, Thibault Sottiaux, Michela Paganini, Jean-Baptiste Lespiau, Alexandre Moufarek, Samer Hassan, Kaushik Shivakumar, Joost van Amersfoort, Amol Mandhane, Pratik Joshi, Anirudh Goyal, Matthew Tung, Andrew Brock, Hannah Sheahan, Vedant Misra, Cheng Li, Nemanja Rakićević, Mostafa Dehghani, Fangyu Liu, Sid Mittal, Junhyuk Oh, Seb Noury, Eren Sezener, Fantine Huot, Matthew Lamm, Nicola De Cao, Charlie Chen, Sidharth Mudgal, Romina Stella, Kevin Brooks, Gautam Vasudevan, Chenxi Liu, Mainak Chain,

Nivedita Melinkeri, Aaron Cohen, Venus Wang, Kristie Seymore, Sergey Zubkov, Rahul Goel, Summer Yue, Sai Krishnakumaran, Brian Albert, Nate Hurley, Motoki Sano, Anhad Mohananey, Jonah Joughin, Egor Filonov, Tomasz Kępa, Yomna Eldawy, Jiawern Lim, Rahul Rishi, Shirin Badiezadegan, Taylor Bos, Jerry Chang, Sanil Jain, Sri Gayatri Sundara Padmanabhan, Subha Puttagunta, Kalpesh Krishna, Leslie Baker, Norbert Kalb, Vamsi Bedapudi, Adam Kurzrok, Shuntong Lei, Anthony Yu, Oren Litvin, Xiang Zhou, Zhichun Wu, Sam Sobell, Andrea Siciliano, Alan Papir, Robby Neale, Jonas Bragagnolo, Tej Toor, Tina Chen, Valentin Anklin, Feiran Wang, Richie Feng, Milad Gholami, Kevin Ling, Lijuan Liu, Jules Walter, Hamid Moghaddam, Arun Kishore, Jakub Adamek, Tyler Mercado, Jonathan Mallinson, Siddhinita Wandekar, Stephen Cagle, Eran Ofek, Guillermo Garrido, Clemens Lombriser, Maksim Mukha, Botu Sun, Hafeezul Rahman Mohammad, Josip Matak, Yadi Qian, Vikas Peswani, Pawel Janus, Quan Yuan, Leif Schelin, Oana David, Ankur Garg, Yifan He, Oleksii Duzhyi, Anton Älgmyr, Timothée Lottaz, Qi Li, Vikas Yadav, Luyao Xu, Alex Chinien, Rakesh Shivanna, Aleksandr Chuklin, Josie Li, Carrie Spadine, Travis Wolfe, Kareem Mohamed, Subhabrata Das, Zihang Dai, Kyle He, Daniel von Dincklage, Shyam Upadhyay, Akanksha Maurya, Luyan Chi, Sebastian Krause, Khalid Salama, Pam G Rabinovitch, Pavan Kumar Reddy M, Aarush Selvan, Mikhail Dektiarev, Golnaz Ghiasi, Erdem Guven, Himanshu Gupta, Boyi Liu, Deepak Sharma, Idan Heimlich Shtacher, Shachi Paul, Oscar Akerlund, François-Xavier Aubet, Terry Huang, Chen Zhu, Eric Zhu, Elico Teixeira, Matthew Fritze, Francesco Bertolini, Liana-Eleonora Marinescu, Martin Bölle, Dominik Paulus, Khyatti Gupta, Tejasi Latkar, Max Chang, Jason Sanders, Roopa Wilson, Xuewei Wu, Yi-Xuan Tan, Lam Nguyen Thiet, Tulsee Doshi, Sid Lall, Swaroop Mishra, Wanming Chen, Thang Luong, Seth Benjamin, Jasmine Lee, Ewa Andrejczuk, Dominik Rabiej, Vipul Ranjan, Krzysztof Styrc, Pengcheng Yin, Jon Simon, Malcolm Rose Harriott, Mudit Bansal, Alexei Robsky, Geoff Bacon, David Greene, Daniil Mirylenka, Chen Zhou, Obaid Sarvana, Abhimanyu Goyal, Samuel Andermatt, Patrick Siegler, Ben Horn, Assaf Israel, Francesco Pongetti, Chih-Wei "Louis" Chen, Marco Selvatici, Pedro Silva, Kathie Wang, Jackson Tolins, Kelvin Guu, Roey Yogev, Xiaochen Cai, Alessandro Agostini, Maulik Shah, Hung Nguyen, Noah Ó Donnaile, Sébastien Pereira, Linda Friso, Adam Stambler, Adam Kurzrok, Chenkai Kuang, Yan Romanikhin, Mark Geller, ZJ Yan, Kane Jang, Cheng-Chun Lee, Wojciech Fica, Eric Malmi, Qijun Tan, Dan Banica, Daniel Balle, Ryan Pham, Yanping Huang, Diana Avram, Hongzhi Shi, Jasjot Singh, Chris Hidey, Niharika Ahuja, Pranab Saxena, Dan Dooley, Srividya Pranavi Potharaju, Eileen O'Neill, Anand Gokulchandran, Ryan Foley, Kai Zhao, Mike Dusenberry, Yuan Liu, Pulkit Mehta, Ragha Kotikalapudi, Chalence Safranek-Shrader, Andrew Goodman, Joshua Kessinger, Eran Globen, Prateek Kolhar, Chris Gorgolewski, Ali Ibrahim, Yang Song, Ali Eichenbaum, Thomas Brovelli, Sahitya Potluri, Preethi Lahoti, Cip Baetu, Ali Ghorbani, Charles Chen, Andy Crawford, Shalini Pal, Mukund Sridhar, Petru Gurita, Asier Mujika, Igor Petrovski, Pierre-Louis Cedoz, Chenmei Li, Shiyuan Chen, Niccolò Dal Santo, Siddharth Goyal, Jitesh Punjabi, Karthik Kappaganthu, Chester Kwak, Pallavi LV, Sarmishta Velury, Himadri Choudhury, Jamie Hall, Premal Shah, Ricardo Figueira, Matt Thomas, Minjie Lu, Ting Zhou, Chintu Kumar, Thomas Jurdi, Sharat Chikkerur, Yenai Ma, Adams Yu, Soo Kwak, Victor Ähdel, Sujeevan Rajayogam, Travis Choma, Fei Liu, Aditya Barua, Colin Ji, Ji Ho Park, Vincent Hellendoorn, Alex Bailey, Taylan Bilal, Huanjie Zhou, Mehrdad Khatir, Charles Sutton, Wojciech Rzadkowski, Fiona Macintosh, Konstantin Shagin, Paul Medina, Chen Liang, Jinjing Zhou, Pararth Shah, Yingying Bi, Attila Dankovics, Shipra Banga, Sabine Lehmann, Marissa Bredesen, Zifan Lin, John Eric Hoffmann, Jonathan Lai, Raynald Chung, Kai Yang, Nihal Balani, Arthur Bražinskas, Andrei Sozanschi, Matthew Hayes, Héctor Fernández Alcalde, Peter Makarov, Will Chen, Antonio Stella, Liselotte Snijders, Michael Mandl, Ante Kärrman, Paweł Nowak, Xinyi Wu, Alex Dyck, Krishnan Vaidyanathan, Raghavender R, Jessica Mallet, Mitch Rudominer, Eric Johnston, Sushil Mittal, Akhil Udathu, Janara Christensen, Vishal Verma, Zach

Irving, Andreas Santucci, Gamaleldin Elsayed, Elnaz Davoodi, Marin Georgiev, Ian Tenney, Nan Hua, Geoffrey Cideron, Edouard Leurent, Mahmoud Alnahlawi, Ionut Georgescu, Nan Wei, Ivy Zheng, Dylan Scandinaro, Heinrich Jiang, Jasper Snoek, Mukund Sundararajan, Xuezhi Wang, Zack Ontiveros, Itay Karo, Jeremy Cole, Vinu Rajashekhar, Lara Tumeh, Eyal Ben-David, Rishub Jain, Jonathan Uesato, Romina Datta, Oskar Bunyan, Shimu Wu, John Zhang, Piotr Stanczyk, Ye Zhang, David Steiner, Subhajit Naskar, Michael Azzam, Matthew Johnson, Adam Paszke, Chung-Cheng Chiu, Jaume Sanchez Elias, Afroz Mohiuddin, Faizan Muhammad, Jin Miao, Andrew Lee, Nino Vieillard, Jane Park, Jiageng Zhang, Jeff Stanway, Drew Garmon, Abhijit Karmarkar, Zhe Dong, Jong Lee, Aviral Kumar, Luowei Zhou, Jonathan Evens, William Isaac, Geoffrey Irving, Edward Loper, Michael Fink, Isha Arkatkar, Nanxin Chen, Izhak Shafran, Ivan Petrychenko, Zhe Chen, Johnson Jia, Anselm Levskaya, Zhenkai Zhu, Peter Grabowski, Yu Mao, Alberto Magni, Kaisheng Yao, Javier Snaider, Norman Casagrande, Evan Palmer, Paul Suganthan, Alfonso Castaño, Irene Giannoumis, Wooyeol Kim, Mikołaj Rybiński, Ashwin Sreevatsa, Jennifer Prendki, David Soergel, Adrian Goedeckemeyer, Willi Gierke, Mohsen Jafari, Meenu Gaba, Jeremy Wiesner, Diana Gage Wright, Yawen Wei, Harsha Vashisht, Yana Kulizhskaya, Jay Hoover, Maigo Le, Lu Li, Chimezie Iwuanyanwu, Lu Liu, Kevin Ramirez, Andrey Khorlin, Albert Cui, Tian LIN, Marcus Wu, Ricardo Aguilar, Keith Pallo, Abhishek Chakladar, Ginger Perng, Elena Allica Abellan, Mingyang Zhang, Ishita Dasgupta, Nate Kushman, Ivo Penchev, Alena Repina, Xihui Wu, Tom van der Weide, Priya Ponnapalli, Caroline Kaplan, Jiri Simsa, Shuangfeng Li, Olivier Dousse, Fan Yang, Jeff Piper, Nathan Ie, Rama Pasumarthi, Nathan Lintz, Anitha Vijayakumar, Daniel Andor, Pedro Valenzuela, Minnie Lui, Cosmin Paduraru, Daiyi Peng, Katherine Lee, Shuyuan Zhang, Somer Greene, Duc Dung Nguyen, Paula Kurylowicz, Cassidy Hardin, Lucas Dixon, Lili Janzer, Kiam Choo, Ziqiang Feng, Biao Zhang, Achintya Singhal, Dayou Du, Dan McKinnon, Natasha Antropova, Tolga Bolukbasi, Orgad Keller, David Reid, Daniel Finchelstein, Maria Abi Raad, Remi Crocker, Peter Hawkins, Robert Dadashi, Colin Gaffney, Ken Franko, Anna Bulanova, Rémi Leblond, Shirley Chung, Harry Askham, Luis C. Cobo, Kelvin Xu, Felix Fischer, Jun Xu, Christina Sorokin, Chris Alberti, Chu-Cheng Lin, Colin Evans, Alek Dimitriev, Hannah Forbes, Dylan Banarse, Zora Tung, Mark Omernick, Colton Bishop, Rachel Sterneck, Rohan Jain, Jiawei Xia, Ehsan Amid, Francesco Piccinno, Xingyu Wang, Praseem Banzal, Daniel J. Mankowitz, Alex Polozov, Victoria Krakovna, Sasha Brown, MohammadHossein Bateni, Dennis Duan, Vlad Firoiu, Meghana Thotakuri, Tom Natan, Matthieu Geist, Ser tan Girgin, Hui Li, Jiayu Ye, Ofir Roval, Reiko Tojo, Michael Kwong, James Lee-Thorp, Christopher Yew, Danila Sinopalnikov, Sabela Ramos, John Mellor, Abhishek Sharma, Kathy Wu, David Miller, Nicolas Sonnerat, Denis Vnukov, Rory Greig, Jennifer Beattie, Emily Caveness, Libin Bai, Julian Eisenschlos, Alex Korchemniy, Tomy Tsai, Mimi Jasarevic, Weize Kong, Phuong Dao, Zeyu Zheng, Frederick Liu, Fan Yang, Rui Zhu, Tian Huey Teh, Jason Sanmiya, Evgeny Gladchenko, Nejc Trdin, Daniel Toyama, Evan Rosen, Sasan Tavakkol, Linting Xue, Chen Elkind, Oliver Woodman, John Carpenter, George Papamakarios, Rupert Kemp, Sushant Kafle, Tanya Grunina, Rishika Sinha, Alice Talbert, Diane Wu, Denese Owusu-Afriyie, Cosmo Du, Chloe Thornton, Jordi Pont-Tuset, Pradyumna Narayana, Jing Li, Saaber Fatehi, John Wieting, Omar Ajmeri, Benigno Uria, Yeongil Ko, Laura Knight, Amélie Héliou, Ning Niu, Shane Gu, Chenxi Pang, Yeqing Li, Nir Levine, Ariel Stolovich, Rebeca Santamaria-Fernandez, Sonam Goenka, Wenny Yustalim, Robin Strudel, Ali Elqursh, Charlie Deck, Hyo Lee, Zonglin Li, Kyle Levin, Raphael Hoffmann, Dan Holtmann-Rice, Olivier Bachem, Sho Arora, Christy Koh, Soheil Hassas Yeganeh, Siim Põder, Mukarram Tariq, Yanhua Sun, Lucian Ionita, Mojtaba Seyedhosseini, Pouya Tafti, Zhiyu Liu, Anmol Gulati, Jasmine Liu, Xinyu Ye, Bart Chrzaszcz, Lily Wang, Nikhil Sethi, Tianrun Li, Ben Brown, Shreya Singh, Wei Fan, Aaron Parisi, Joe Stanton, Vinod Koverkathu, Christopher A. Choquette-Choo, Yunjie Li, TJ Lu, Abe Ittycheriah, Prakash Shroff, Mani Varadarajan, Sanaz Bahargam, Rob Willoughby, David

Gaddy, Guillaume Desjardins, Marco Cornero, Brona Robenek, Bhavishya Mittal, Ben Albrecht, Ashish Shenoy, Fedor Moiseev, Henrik Jacobsson, Alireza Ghaffarkhah, Morgane Rivière, Alanna Walton, Clément Crepy, Alicia Parrish, Zongwei Zhou, Clement Farabet, Carey Radebaugh, Praveen Srinivasan, Claudia van der Salm, Andreas Fidjeland, Salvatore Scellato, Eri Latorre-Chimoto, Hanna Klimczak-Plucińska, David Bridson, Dario de Cesare, Tom Hudson, Piermaria Mendolicchio, Lexi Walker, Alex Morris, Matthew Mauger, Alexey Guseynov, Alison Reid, Seth Odoom, Lucia Loher, Victor Cotruta, Madhavi Yenugula, Dominik Grewe, Anastasia Petrushkina, Tom Duerig, Antonio Sanchez, Steve Yadlowsky, Amy Shen, Amir Globerson, Lynette Webb, Sahil Dua, Dong Li, Surya Bhupatiraju, Dan Hurt, Haroon Qureshi, Ananth Agarwal, Tomer Shani, Matan Eyal, Anuj Khare, Shreyas Rammohan Belle, Lei Wang, Chetan Tekur, Mihir Sanjay Kale, Jinliang Wei, Ruoxin Sang, Brennan Saeta, Tyler Liechty, Yi Sun, Yao Zhao, Stephan Lee, Pandu Nayak, Doug Fritz, Manish Reddy Vuyyuru, John Aslanides, Nidhi Vyas, Martin Wicke, Xiao Ma, Evgenii Eltyshev, Nina Martin, Hardie Cate, James Manyika, Keyvan Amiri, Yelin Kim, Xi Xiong, Kai Kang, Florian Luisier, Nilesh Tripuraneni, David Madras, Mandy Guo, Austin Waters, Oliver Wang, Joshua Ainslie, Jason Baldridge, Han Zhang, Garima Pruthi, Jakob Bauer, Feng Yang, Riham Mansour, Jason Gelman, Yang Xu, George Polovets, Ji Liu, Honglong Cai, Warren Chen, XiangHai Sheng, Emily Xue, Sherjil Ozair, Christof Angermueller, Xiaowei Li, Anoop Sinha, Weiren Wang, Julia Wiesinger, Emmanouil Koukoumidis, Yuan Tian, Anand Iyer, Madhu Gurumurthy, Mark Goldenson, Parashar Shah, MK Blake, Hongkun Yu, Anthony Urbanowicz, Jennimaria Palomaki, Chrisantha Fernando, Ken Durden, Harsh Mehta, Nikola Momchev, Elahe Rahimtoroghi, Maria Georgaki, Amit Raul, Sebastian Ruder, Morgan Redshaw, Jinhyuk Lee, Denny Zhou, Komal Jalan, Dinghua Li, Blake Hechtman, Parker Schuh, Milad Nasr, Kieran Milan, Vladimir Mikulik, Juliana Franco, Tim Green, Nam Nguyen, Joe Kelley, Aroma Mahendru, Andrea Hu, Joshua Howland, Ben Vargas, Jeffrey Hui, Kshitij Bansal, Vikram Rao, Rakesh Ghiya, Emma Wang, Ke Ye, Jean Michel Sarr, Melanie Moranski Preston, Madeleine Elish, Steve Li, Aakash Kaku, Jigar Gupta, Ice Pasupat, Da-Cheng Juan, Milan Someswar, Tejvi M., Xinyun Chen, Aida Amini, Alex Fabrikant, Eric Chu, Xuanyi Dong, Amruta Muthal, Senaka Buthpitiya, Sarthak Jauhari, Nan Hua, Urvashi Khandelwal, Ayal Hitron, Jie Ren, Larissa Rinaldi, Shahar Drath, Avigail Dabush, Nan-Jiang Jiang, Harshal Godhia, Uli Sachs, Anthony Chen, Yicheng Fan, Hagai Taitelbaum, Hila Noga, Zhuyun Dai, James Wang, Chen Liang, Jenny Hamer, Chun-Sung Ferng, Chenel Elkind, Aviel Atias, Paulina Lee, Vít Listík, Mathias Carlen, Jan van de Kerkhof, Marcin Pikus, Krunoslav Zaher, Paul Müller, Sasha Zykova, Richard Stefanec, Vitaly Gatsko, Christoph Hirnschall, Ashwin Sethi, Xingyu Federico Xu, Chetan Ahuja, Beth Tsai, Anca Stefanoiu, Bo Feng, Keshav Dhandhania, Manish Katyal, Akshay Gupta, Atharva Parulekar, Divya Pitta, Jing Zhao, Vivaan Bhatia, Yashodha Bhavnani, Omar Alhadlaq, Xiaolin Li, Peter Danenberg, Dennis Tu, Alex Pine, Vera Filippova, Abhipso Ghosh, Ben Limonchik, Bhargava Urala, Chaitanya Krishna Lanka, Derik Clive, Yi Sun, Edward Li, Hao Wu, Kevin Hongtongsak, Ianna Li, Kalind Thakkar, Kuanysh Omarov, Kushal Majmundar, Michael Alverson, Michael Kucharski, Mohak Patel, Mudit Jain, Maksim Zabelin, Paolo Pelagatti, Rohan Kohli, Saurabh Kumar, Joseph Kim, Swetha Sankar, Vineet Shah, Lakshmi Ramachandruni, Xiangkai Zeng, Ben Bariach, Laura Weidinger, Tu Vu, Alek Andreev, Antoine He, Kevin Hui, Sheleem Kashem, Amar Subramanya, Sissie Hsiao, Demis Hassabis, Koray Kavukcuoglu, Adam Sadovsky, Quoc Le, Trevor Strohman, Yonghui Wu, Slav Petrov, Jeffrey Dean, and Oriol Vinyals. Gemini: A family of highly capable multimodal models, 2024. URL `https://arxiv.org/abs/2312.11805`.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *ArXiv*,

abs/2302.13971, 2023.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation Learning with Contrastive Predictive Coding. *arXiv e-prints*, art. arXiv:1807.03748, July 2018. doi: 10.48550/arXiv.1807.03748.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. (arXiv:2101.12631), may 2021. doi: 10.48550/arXiv.2101.12631.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. Chain-of-table: Evolving tables in the reasoning chain for table understanding. *ArXiv*, abs/2401.04398, 2024.

Shengli Wu, Yaxin Bi, and Xiaoqin Zeng. The linear combination data fusion method in information retrieval. In Abdelkader Hameurlain, Stephen W. Liddle, Klaus-Dieter Schewe, and Xiaofang Zhou, editors, *Database and Expert Systems Applications*, page 219–233, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-23091-2.

Shitao Xiao, Zheng Liu, Yingxia Shao, and Zhao Cao. RetroMAE: Pre-Training Retrieval-oriented Language Models Via Masked Auto-Encoder. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 538–548, Abu Dhabi, United Arab Emirates, dec 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.35. URL `https://aclanthology.org/2022.emnlp-main.35/`.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-pack: Packed resources for general chinese embeddings. *arXiv*, 2309.07597, sep 2024. doi: 10.48550/arXiv.2309.07597.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=zeFrfgyZln`.

Kuan Xu, Yongbo Wang, Yongliang Wang, Zihao Wang, Zujie Wen, and Yang Dong. SeaD: End-to-end text-to-SQL generation with schema-aware denoising. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz, editors, *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 1845–1853, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-naacl.141. URL `https://aclanthology.org/2022.findings-naacl.141/`.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. TaBERT: Pretraining for joint understanding of textual and tabular data. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.745. URL `https://aclanthology.org/2020.acl-main.745/`.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1425. URL `https://aclanthology.org/D18-1425/`.

Shuo Zhang and Krisztian Balog. Ad hoc table retrieval using semantic similarity. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 1553–1562, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee. ISBN 9781450356398. doi: 10.1145/3178876.3186067. URL `https://doi.org/10.1145/3178876.3186067`.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Z. Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jianyun Nie, and Ji rong Wen. A survey of large language models. *ArXiv*, abs/2303.18223, 2023.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *ArXiv*, abs/1709.00103, 2017.