Turning Databases Into Generative AI Machines

Alekh Jindal, Shi Qiao, Sathwik Reddy Madhula, Kanupriya Raheja, Sandhya Jain
research@smart-apps.ai
SmartApps Inc.
Bellevue, USA

ABSTRACT

Data is no more the commodity oil. Today, it is an asset for any enterprise. However, turning data into intelligence remains a challenge for most people. In this paper, we explore whether databases can be turned into generative AI machines that can talk to anyone. We identify three core challenges when applying generative AI on data, namely accuracy, scale, and privacy, and show how a generative large data model could solve all of these. We describe our conceptual framework of generative AI on databases, *the GOD machine*, and ground it in production workloads at SmartApps. Our results promise new directions in fusing AI with data.

ACM Reference Format:
Alekh Jindal, Shi Qiao, Sathwik Reddy Madhula, Kanupriya Raheja, Sandhya Jain. 2024. Turning Databases Into Generative AI Machines. In *Proceedings of Conference on Innovative Data Systems Research (CIDR'24).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Data is the differentiator for modern businesses and new age databases such as Snowflake, BigQuery, Synapse, RedShift, and Databricks offer highly sophisticated data processing on demand. However, turning data from these databases into insights involves a jungle of tools for data modeling, pipelining, dashboarding, and so on – a mountain of complex, manual, yet tedious work that requires a lot of time and expertise. It gets even worse with increasingly popular ELT architectures where data is loaded quickly, while leaving the transformations for the data analysts or the analytics engineers to pick up later. As a result, end users wait for weeks and months before they get the insights to make data-driven decisions, an unsustainable situation as the gap between experts and end users continues to grow. Just as cloud democratized the data infrastructure, it's time to democratize data intelligence as well.

Generative AI has shown a lot of promise in automating tedious manual tasks, such as writing copy and code or building images and videos. For data analytics, a lot of focus has been on generating SQL queries from natural language, i.e., text-to-SQL [16]. However, this just scratches the surface to understand user questions and not the underlying data. More importantly, it is prone to errors, with accuracy rates between 50-85%, that are increasingly hard to spot and need an expert to validate the generated SQL statements anyways. Consequently, text-to-SQL has seen little adoption beyond academic prototypes.

Providing more context is a popular approach to improve understanding and reduce errors with language models. This is motivated by larger context lengths supported in newer language models, e.g., 32K in GPT-4 and 100K in Claude [1]. As a result, many newer tools, such as Open AI Code Interpreter, provide data files as input to the LLM. However, the context is still limited to few 100s of MBs of data, which is peanuts compared to the size of enterprise databases. Retrieval augmented generation (RAG) [7] overcomes the scale challenge partially by fetching relevant context using vector search. However, it embeds the physical data for retrieval, thus scaling linearly with the database size. RAG is also limited to filtering relevant rows and does not understand other data transformations like joins or aggregate that could provide valuable context. In addition to the prompt engineering involved, context-based approaches also end up sending actual data to the language models, making them hard to work with third party LLMs.

The extreme approach to truly understand private data is to build custom LLMs, either via fine-tuning or building from scratch. BloombergGPT with 50 billion parameters is one example of building domain specific LLM built from scratch. However, this involves a non-trivial amount of engineering effort and costs running into millions of dollars for each run, a repeatedly incurring cost as new data comes in. Finetuning also requires carefully crafted set of instructions, e.g., the 15K instructions gathers from all Databricks employees for Dolly [5]. Finally, while custom LLMs address the privacy problem, they still suffer from hallucination. Recent studies from Open AI show GPT models having accuracy ranging between 50-80% [14]. Instead, database applications are business critical, and they demand 100% accurate answers.

In this paper, we ask the question whether we can turn any given database into a generative AI machine – one that is accurate, fast, and secure. Our goal is to overcome the practical challenges involved and make generative AI work for billions of professionals, helping them leverage data without being bottle-necked on a handful of experts. The key insight we have is that language models must be coupled with the right data models for answers to be truthful, i.e., hallucination-free. Given that data models are logical, they can scale to large data without revealing the physical values to third party LLMs. We introduce *Large Data Model* to generate semantically relevant data models in response to natural language queries. We retrieve the generated data model efficiently with automatic materialization and cache. Furthermore, since data models are logical, we can do one-time fine tuning of open-source models and match the performance with GPT. Finally, we introduce *DataChains* for building end to end applications on top of LDM.

In summary, we make the following key contributions:

CIDR'24, January 14–17, 2024, Chaminade, USA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

- We discuss the current landscape of generative AI for data and dig into why the current approaches of text-to-SQL, context building, and finetuning fall short. (Section 2)
- We present a brand-new plug-n-play approach to build generative AI on database (GOD) machines. (Section 3)
- We introduce Large Data Model (LDM) that can explore the space of data transformations and generate high quality data models to answer user questions. (Section 4)
- We present a scalable *retrieval augmentation* approach that gets relevant transformations, generated by the large data model, as context from a database of any size. (Section 5)
- We describe smaller open-source language models that could be hosted privately and pre-fine-tuned for database questions to match the performance of GPT-3.5. (Section 6)
- We define the notion of DataChain, a sequence to steps that translate the results from a data talk into end-to-end tasks for businesses. (Section 7)
- We show a case study from an Edtech customer looking to transform their business with generative AI. (Section 8)

2 CURRENT LANDSCAPE

In this section, we discuss the three primary approaches that are applicable for generative AI on data.

2.1 Natural Language Interface

Artificial intelligence began with the question whether machines can think, but soon the question also evolved into whether machines can talk. People wondered whether natural language is unnatural for machines [13]. In fact, one of the earliest natural language database systems, the English querying system (EQS), was developed at MIT in 1978 [12]. Today, many startups, such as text2sql.ai, seek.ai, defog.ai, and NLSQL.com, are working to help people converse with a database in natural language. Even more established players have introduced capabilities to make the database query interface simpler, e.g., ThoughtSpot Sage, Microsoft Synapse Fabric, Amazon QuickSight, and Databricks English SDK [17].

The challenge with natural language interfaces is the accuracy of translation from natural language to SQL. To illustrate, consider Spider, a popular leaderboard for semantic parsing and text-to-SQL challenge from Yale [18]. The leading model on Spider currently has an accuracy of 85.3%. Likewise, Bird is another benchmark for text-to-SQL over large-scale databases [8], and currently GPT-4 leads the benchmark with 54.89% accuracy. The reasons for this inaccuracy range from lack of perfect schema to work on, or the right data models to reason on, or the underlying data characteristics to infer from. Overall, inaccuracy is unacceptable for business applications that expect guaranteed correctness.

2.2 Contextual Generation

Retrieval-augmented generation (RAG) is a context-oriented approach to reduce hallucination on knowledge-intensive tasks. The core idea is to first retrieve text documents from a knowledge source and then generate the output based on those documents. Thus, RAG can provide more precise answers and the provenance to those answers. Examples include FinGPT [11] for financial data and RETRO [2] that matches GPT-3 despite being just 4% of its size. Furthermore, FLARE [6] performs a set of retrieval iteratively, while LlamaIndex [9] provides a "data framework" to apply RAG on private data. Specific approaches for prompt engineering, such as chain of thought or tree of thought, or even how to interleave them with retrieval are orthogonal to the retrieval process itself.

The retrieval augmentation approach requires creating embeddings on the physical data and later retrieve relevant portions using vector search. Unfortunately, this is hard to scale with large databases with billions of rows to embed and retrieve. Moreover, as data changes, the embeddings also need to be constantly updated. This is equivalent to always maintaining the perfect indexes – a hard problem that has challenged database researchers for decades. Even if we were to magically retrieve all relevant data, there are limits to the size of the context, e.g., 16k tokens with GPT-3.5 and 32k tokens with GPT-4. Larger context length is also leading to new problems such as ordering of data within the context [10]. Essentially, we may end up building another data processor outside of the database to do what databases already do.

Finally, retrieval augmentation adds sensitive private data to the context and hence unlikely to be used with third party LLMs. In fact, recent survey shows privacy and accuracy as the top barriers for leaders to adopt generative AI.

2.3 Fine-tuned Models

Organizations with massive resources are tempted to create in-house language models on their private data. Estimates indicate GPT models to cost between 4 to 100 hundred millions of dollars. Bloomberg's BloomberGPT, for instance, is estimated to have taken 1.3 million GPU hours for training. Such massive investments need strategic planning and long-term resourcing to maintain the models over time. The bigger challenge still is model accuracy and how to avoid it degrading over time. For instance, recent study shows ChatGPT's behavior degrading over time [4]. Given the costs involved and the expertise required to train inhouse language models, it is unlikely to go beyond niche use cases.

Approach	Accuracy	Speed	Cost	Privacy
Text-to-SQL	Medium	Low	Low	Medium
Contextual Generation	High	Medium	Medium	Low
Finetuning	Low	High	High	High
Ideal	**100%**	**<2s**	**Managed**	**Private**

Table 1: Trade-offs in approaches for generative AI on data.

Table 1 summarizes the competitive landscape for generative AI on data. While Text-to-SQL approach suffers from inaccuracy, slow speed (direct queries to databases), and privacy, it does provide a low-cost solution. Context generation, on the other hand improves accuracy at the expense of privacy. It can also suffer from speed and cost issues depending on how much effort and expertise is available. Finetuning preserves privacy and can be fast at the expense of hallucination and very high costs.

3 THE GOD MACHINE

Figure 1 shows the architecture of our proposed generative AI on databases, the GOD machine. It starts with existing databases and

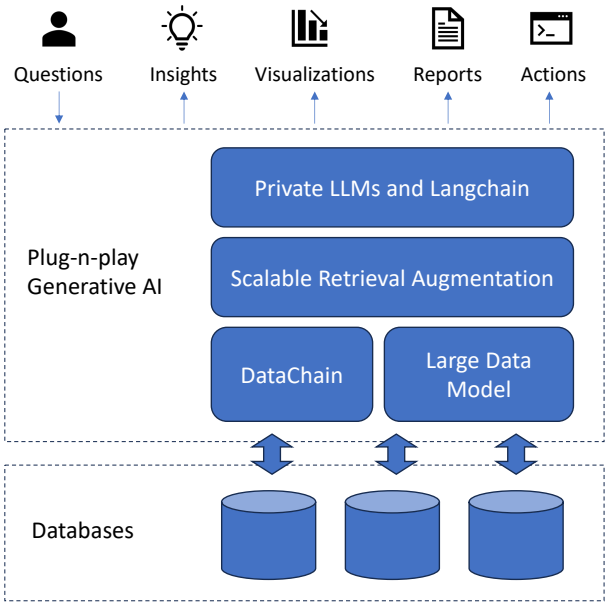

Figure 1: Generative AI on database (GOD) machine.

adds a plug-n-play generative AI layer on top. This layer combines the ability to generate arbitrary data models, retrieve them efficiently, and brings private language models to reason upon the user questions. Users can see insights, analyze visualizations, generate reports, or take actions. A DataChain combines a series of steps derived from the user actions for end-to-end applications. Users get started with the GOD machine by simply connecting their databases and wait for up to 30 minutes, for one-time training and embedding, before they start asking questions. Retraining/re-embedding is only needed when the database schema or distribution changes significantly. The GOD machine has the following salient features.

- All answers are generated only based on the underlying database that is connected.
- Users can trace back every answer to a concrete data model.
- All data transformations that are needed to generate the data model are applied behind the scenes.
- Likewise, all data pipelines that are needed to keep the data models refreshed are generated automatically.
- The machine has an intelligent cache to for an interactive querying experience.
- The machine provides efficient retrieval for context augmentation using vector search.
- The system maintains high levels of relevance to user queries.
- It can retrieve arbitrary data transforms that could be then refined into the final data model.
- The machine provides fully managed open source LLMs that match the accuracy of OpenAI.
- The system automatically handles all prompt engineering transparent to the users.
- The web interface provides automatic visuals for users to make the judgement on their own.
- Users can also drill down and chat with any of the generated data model.

Figure 2: Comparing default and filtered space of data models with LDM.

- Finally, users can apply one or more actions to make their insight actionable.

Below we describe each of the four key components in the GOD machine, namely the large data model, scalable retrieval augmentation, private language models, and the DataChain.

4 LARGE DATA MODEL

Data modeling is the process of transforming data from a database so that it can be used in business applications. Typically, it involves identifying the clean data, combining different tables, building aggregations, identifying meaningful dimensions and measures, identifying dynamic filters to slice data interactively, selecting the right subset of the data to present, and deciding the right visualizations for the resulting data at hand.

While traditionally data modeling depends on the domain and the business scenario, many of the transformation tasks are often repetitive and time consuming. Large data model (LDM) is a hierarchical mixture model that learns what kind of transformations make sense for different kinds of data distributions. It consists of thousands of parameterized rules to answer the following questions:

(1) Which part of the data is clean?
(2) Which tables are valid and useful to join?
(3) Which parts of the data to filter?
(4) Which grouping and aggregations should be built?
(5) Which columns are potential dimensions?
(6) Which columns are potential measures?
(7) Which columns can be interactive filters?
(8) Which models are of high-quality?
(9) Which subsets of the rows are interesting to analyze?
(10) Which visualizations are interesting?
(11) What can be generalized across data sources?

We can see there is a massive space of possible transformations, which takes users a lot of time (and expertise) to figure out manually on any given database. The goal of LDM, therefore, is to narrow down this space to a much smaller one and to make it searchable to users in real time. This means users do not have start from raw databases and apply mundane transformations repeatedly. Instead,

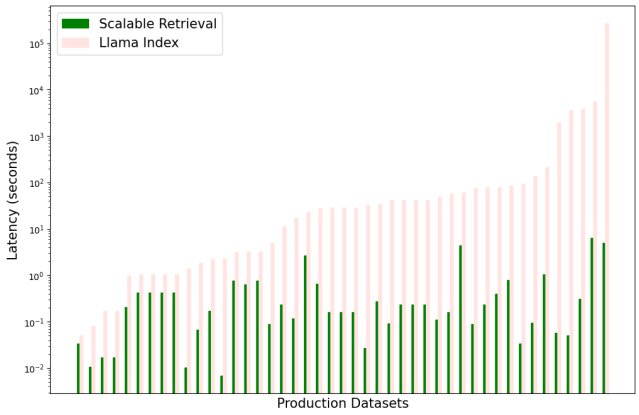

Figure 3: Comparing traditional retrieval using LlamaIndex with scalable retrieval.

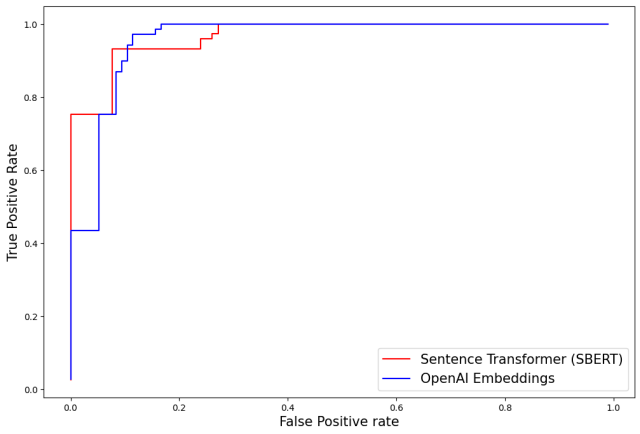

Figure 4: Comparing OpenAI with pre-tuned SBERT model.

they search the most relevant data model for their analysis and tune it further. Figure 2 illustrates the space of all valid and high-quality models over 50 production databases at SmartApps. We can see that while the total number of models can exceed 10s of millions on these databases, the actual high-quality models remain within 10s of thousands only – a three orders of magnitude reduction in the search space, helping users focus on the high-quality data models.

The key advantages of LDM are as follows. First, it eliminates hallucination by rooting all responses in well-defined and high-quality data models. Second, LDM scales to arbitrary databases since it operates on logical data model space. Third, since LDM decouples logical and physical data, it leverages existing database techniques such as materialization and caching to speed up the data models. And finally, LDM can work in combination with any LLM on top without any finetuning required on the physical data.

5 SCALABLE RETRIEVAL AUGMENTATION

A key component in the GOD machine architecture is scalable retrieval augmentation that retrieves relevant data to answer user questions from database of any size. The popular approach is to use a vector database to embed the physical data and retrieve the semantically relevant portions of it at query time. For databases, this means embedding all rows from all tables in the database, which is not feasible. Therefore, we embed logical data models, instead of the physical data, and retrieve relevant ones in response to user queries. As a result, vector search now scales with the number of data models, which have already been filtered by the LDM, and not the number of rows in the database.

Figure 3 shows how retrieval on logical data in our approach scales compares to retrieval on physical data in LlamaIndex [9] over 50 production databases at SmartApps. While LlamaIndex can easily have retrieval time run into 100*s* to 1000*s* of seconds, our scalable retrieval approach keeps it under 1 second in most cases and 10 seconds in the extreme cases. Thus, we see two orders more scalablity, making interactive performance (<2*s*) possible. Furthermore, while the retrieval latencies of physical data embedding approaches continue to grow with the number of rows, the latencies with logical data embeddings are bounded by the schema sizes.

We consider various options when generating the embeddings for semantic search. Every data model is represented by a SQL expression, which could be embedded directly. Alternatively, we could convert the SQL expression into a description, or a business topic, with few shot prompting for the data model to capture more general meaning. Currently, we use OpenAI for the embeddings and pgvector for vector search. We could as well replace them with other language models (e.g., Llama, T5, etc.) and vector databases (e.g., pg_embedding, Qdrant, etc.). To speed up the performance, we speculatively cache interesting data models in memory (Redis). This could be based on model quality, usage, recency, and so on.

Note that vector search only gives data sorted by similarity score (cosine similarity by default). However, we still need to decide whether it is relevant to the user query and not add irrelevant information as context. To do this, we determine a *similarity threshold* to decide whether a retrieved data model is relevant or not. We train this threshold by crafting irrelevant questions on a database and computing their similarity scores with valid data models on that database. We observe a strict similarity threshold of 0.851, where no irrelevant models are retrieved. But that also misses some relevant models. Therefore, we pick a threshold 0.784 that gives a true positive rate of 0.971 and false positive rate of 0.125. Overall, this results in a relevance of 85% to user queries.

In summary, we scale retrieval augmentation by operating on logical data model instead of the physical data. Such an approach scales in the size of schema (and data characteristics), without exploding with the number of rows in the database. We further cache the data models aggressively to hide the load latencies (overall 1.2s on average). We also determine a similarity threshold to decide whether the retrieved data model is relevant or not. Together, we provide a fast and effective retrieval for database of any size.

6 PRIVATE LANGUAGE MODELS

Third party models, such as GPT, are a privacy risk for many enterprises. An information leak could impact intellectual property, violate regulatory laws, or simply loose the competitive edge if the data is used by the third-party to train itself. Additionally, the general-purpose models contain a lot of redundant parameters and

are not trained for data analytics specifically. Therefore, the question is whether we can train a smaller inhouse model that matches the performance of OpenAI model. Below we compare OpenAI with open-source models on our two primary tasks: (1) embedding, and (2) text generation.

Embeddings. Recall that we embed high-quality models from the LDM and store them in pgvector database for scalable retrieval. We then also embed user questions and find the relevant data models with the highest cosine similarity scores.

For OpenAI, we train the embeddings model by fine tuning GPT 3.5 with a siamese objective of moving similar sentence embeddings closer to each other. However, this requires sharing the database schema with the OpenAI endpoint. Also, since the initial pre-trained model is decoder only transformer, the embeddings are not evenly distributed in the embedding space. In fact, when determining similarity threshold, we noted that relevant models all lie in the narrow range of 0.75 to 0.85.

For open source, we use the Sentence Transformer (SBERT-all-mpnet-base-v2) with 109.5 million parameters [15]. The Sentence transformer uses a pre-trained encoder only transformer and fine-tuned with the siamese objective. Unlike OpenAI embeddings, the embeddings produced by the all-mpnet-base-v2 model are distributed evenly across the embedding space, i.e, the similarity scores of relevant and non-relevant models are spread evenly in the range from 0 to 1. This helps distinguish the relevant models better and thus determine a crisper similarity threshold.

We evaluate the relevance of OpenAI and SBERT models by manually classifying the top 5 data models for 33 queries on our production databases as relevant or not. Figure 4 shows the ROC (receiver operating characteristic) analysis for both OpenAI and SBERT embeddings. For OpenAI, we can observe that the similarity threshold of 0.784 allows us to distinguish the relevant and non-relevant models with a true positive rate of 97.1% and false positive rate of 12.5%. For SBERT, a similarity threshold of 0.317 allows us to distinguish between relevant and non-relevant models with a true positive rate of 93.2% and false positive rate of 7.6%. From the ROC graph, we can also observe that the area under the curve (AUC) for SBERT embeddings (0.958) is higher than OpenAI (0.948), indicating that SBERT embeddings are more useful at ranking the most relevant data models than OpenAI. One caveat to note is that OpenAI has a lot more parameters storing a lot of information from many domains. This means that for very niche domains, SBERT may not perform very well since it may not be trained in that domain. Analyzing this further will be part of future work.

Text generation. We generate text from SQL to create better embeddings for semantic search. The generated text also helps users to understand the model better and to use it as context when answering their questions. Third party LLM end points, such as ChatGPT, expose the data model definitions which is not acceptable to many organizations. Therefore, we have fine-tuned the open source T5(base) model of 220 million parameters using our custom dataset curated from test workloads. This inhouse model generated text with almost the same quality as OpenAI GPT 3.5-Turbo model (154 billion), even though it has very few parameters. To evaluate the performance, we compare the similarity score of text generated by our fine-tuned T5 model with GPT-3.5 Turbo. The results show a similarity score of 93.6% with the OpenAI embeddings (similarity threshold of 0.784), and a similarity score of 0.77 with SBERT embeddings (similarity threshold of 0.317). We can see that the text generation quality of our custom model matches very closely to that of OpenAI on either embeddings.

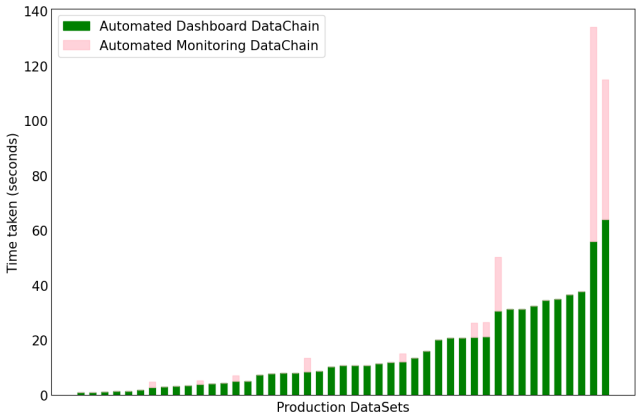

Figure 5: DataChain execution times for different production databases.

In summary, we can successfully train open-source language models that are much smaller in size and yet match or outperform OpenAI in terms of relevance and accuracy. Thus, we can build generative AI on data without compromising on privacy.

7 DATACHAIN

The GOD machine helps find truthful answers rooted in data models. It goes even further to help combine data tasks for an end-to-end application and takes care of running them autonomously. We refer to such chain of tasks as *DataChain* and it is like Langchain [3]. Below we describe four concrete scenarios of DataChain and show how they can be accomplished with the GOD machine.

Automated Dashboard. Data analysts spend a lot of time building dashboards that need to be kept refreshed. They need to write data pipelines and materialize aggregated data before it could be displayed on the dashboard. Most BI tools allow users to define the refresh intervals or to specify which columns to check for changes. Many also have limits, e.g., Power BI limits scheduled refresh to 8 refresh per day. All this manual tedious work can be automated with DataChains. Once users retrieve the relevant data model and tune it to their needs, they can pin it to a dashboard. Behind the scenes, the GOD machine creates an Airflow job (shared with all pinned) to keep data models refreshed. Thus, not only are users relieved from writing tedious data pipelines manually, but the system can also share computations and reduce cost more effectively.

Automated Monitoring. Cloud operators are often worried about things going wrong, and they want to stay on top when that happens. However, with the growing footprint of cloud services, it is extremely hard for them to analyze, visualize, and spot issues from the mountain of monitoring data that gets collected. Most monitoring tools allow users to create custom rules to manually define the abnormal and get alerted if that happens. However, given the number of possibilities, it is very hard for users to know what is

"normal" and catch all possible scenarios. DataChains can automate this manual monitoring effort by keeping track of how the data models behave over time. Users can mark which data models need monitoring or they can also monitor all high-quality models on a database. By batching the monitoring tasks into shared Airflow jobs, the system can scale efficiently, while relieving the user.

Automated Reports. Decision makers need periodic reporting on the state of affairs. However, current tools limit them to canned reports, and any customization requires falling back to the data experts to get the appropriate data. They also spend a lot of time interpreting the results and putting it in context for their business, e.g., with respect to external data. DataChain can help them generate reports from one or more data models and leverage LLMs for descriptive external data enabled analysis. Users can identify the data models they need to report on and choose the analysis they want to be included in their report. The system then generates tailored reports for them in a self-serve fashion.

Automated Q&A. Business analysts provide actionable insights to the stakeholders. However, they typically rely on data analysts to build a dashboard before they can see the data. Typically, this process gets plagued with back and forth on requirements followed by long delays in delivery. As a result, the business analyst needs to weigh in carefully before making any requests. With DataChain, the business analyst can "talk" to the data directly before coming up with concrete requirements. They can ask natural language questions and dig into a Q&A with any data model to ask more pointed questions. This can help them understand their scenario better and define the requirements with a concrete example. Behind the scenes, the GOD machine loads the data model into the LLM and runs real-time data science operations in response to user questions.

Figure 5 shows the automated dashboard and automated monitoring DataChains over the 50 production databases at SmartApps. Depending on the size of the database, the Airflow jobs underlying the DataChains can take from 10s to 100s of seconds. However, all those Airflow jobs are automatically generated, and their execution fully managed by the GOD machine. Thus, DataChains helps users accomplish sophisticated end-to-end analytics tasks with ease.

8 CASE STUDY

We now describe a case study of using the GOD machine for an Edtech company that runs 100*s* of courses over 1000*s* of students with 10,000*s* of online meetings every year. The key business questions that they are looking to answer are three-fold: (i) non-payment and refund issues, (ii) punctuality and engagement in classes, (iii) customer acquisition and churn. Currently, they employ engineers to answer these questions, and it takes anywhere from a couple of days to a week for each new question. This is both tedious and costly. As a result, they miss critical insights and potential actions they could take.

With GOD machine, they can search insights over all their data in natural language. They can visualize the insights, refine the data models, and pin the relevant ones to the dashboard. At any time, they can download a report to summarize their business insights. The GOD machine generated insights helped the company identify 10% leakage in revenue, and provided pointed actions to take.

9 CONCLUSION

Databases have long mastered the art of storing and processing data. However, taking data from a database into something insightful and actionable remains a challenge for many people. In this paper, we described turning databases into generative AI machines, automating many of the manual tedious tasks involved. We introduced large data model to generate relevant data models for user questions and presented a scalable retrieval approach for database of any size. We demonstrated the use of inhouse language models for privacy and described the notion of DataChains to put end-to-end applications together. To conclude, we believe generative AI can be a big leap for databases, and this paper is a step in that direction.

REFERENCES

[1] Anthropic. 2023. *Introducing 100K Context Windows.* Retrieved July 31, 2023 from https://www.anthropic.com/index/100k-context-windows

[2] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. 2022. Improving language models by retrieving from trillions of tokens. arXiv:2112.04426 [cs.CL]

[3] Harrison Chase. 2022. *LangChain.* https://github.com/hwchase17/langchain

[4] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. How is ChatGPT's behavior changing over time? arXiv:2307.09009 [cs.CL]

[5] Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. 2023. *Free Dolly: Introducing the World's First Truly Open Instruction-Tuned LLM.* Retrieved July 31, 2023 from https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm

[6] Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active Retrieval Augmented Generation. arXiv:2305.06983 [cs.CL]

[7] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401 [cs.CL]

[8] Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. arXiv:2305.03111 [cs.CL]

[9] Jerry Liu. 2022. *LlamaIndex.* https://doi.org/10.5281/zenodo.1234

[10] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the Middle: How Language Models Use Long Contexts. arXiv:2307.03172 [cs.CL]

[11] Xiao-Yang Liu, Guoxuan Wang, and Daochen Zha. 2023. FinGPT: Democratizing Internet-scale Data for Financial Large Language Models. arXiv:2307.10485 [cs.CL]

[12] William A. Martin. 1978. Some Comments On EQS, A Near Term Natural Language Data Base Query System. In *Proceedings of the 1978 Annual Conference (ACM '78).* Association for Computing Machinery, 156–164.

[13] Christine A. Montgomery. 1972. Is Natural Language an Unnatural Query Language?. In *Proceedings of the ACM Annual Conference - Volume 2 (ACM '72).* Association for Computing Machinery, 1075–1078.

[14] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[15] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv:1908.10084 [cs.CL]

[16] Ruoxi Sun, Sercan O. Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. 2023. SQL-PaLM: Improved Large Language Model Adaptation for Text-to-SQL. arXiv:2306.00739 [cs.CL]

[17] Gengliang Wang, Xiangrui Meng, Reynold Xin, Allison Wang, Amanda Liu, and Denny Lee. 2023. *Introducing 100K Context Windows.* Retrieved July 31, 2023 from https://www.databricks.com/blog/introducing-english-new-programming-language-apache-spark

[18] 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. 2019. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. arXiv:1809.08887 [cs.CL]