nsDB: Architecting the Next Generation Database by Integrating Neural and Symbolic Systems

Ye Yuan
Beijing Institute of Technology
Beijing, China
yuan-ye@bit.edu.cn

Bo Tang
Southern Univ. of Sci. and Tech.
Shenzhen, China
tangb3@sustech.edu.cn

Tianfei Zhou
Beijing Institute of Technology
Beijing, China
ztfei.debug@gmail.com

Zhiwei Zhang
Beijing Institute of Technology
Beijing, China
zwzhang@bit.edu.cn

Jianbin Qin
Shenzhen University
Shenzhen, China
qinjianbin@szu.edu.cn

ABSTRACT

In this paper, we propose nsDB, a novel neuro-symbolic database system that integrates neural and symbolic system architectures natively to address the weaknesses of each, providing a strong database capable of data managing, model learning, and complex analytical query processing over multi-modal data. We employ a real-world NBA data analytical query as an example to illustrate the functionality of each component in nsDB and highlight the research challenges to build it. We then present the key design principles and our preliminary attempts to address them.

In a nutshell, we envision that the next generation database system nsDB integrates the complex neural system with the simple symbolic system. Undoubtedly, nsDB will serve as a bridge between databases with AI models, which abstracts away the AI complexities but allows end users to enjoy the strong capabilities of them. We are in the early stages of the journey to build nsDB, there are many opening challenges, e.g., in-database model training, multi-objective query optimization, and database agent development. We hope the researchers from different communities (e.g., system, architecture, database, artificial intelligence) could tackle them together.

PVLDB Reference Format:
Ye Yuan, Bo Tang, Tianfei Zhou, Zhiwei Zhang, and Jianbin Qin. nsDB: Architecting the Next Generation Database by Integrating Neural and Symbolic Systems. PVLDB, 17(11): 3283 - 3289, 2024.
doi:10.14778/3681954.3682000

1 INTRODUCTION

On one hand, either traditional relational database systems (e.g., PostgreSQL [17], MySQL [15]) or modern big data systems (e.g., Spark [7], Flink [5], Hive [6]) employs symbolic system (a.k.a. algebraic computation [19]) as the building brick in the system architecture. In particular, the complex data processing procedure in them is transferred to exact computation with expressions containing variables and are manipulated as symbols, i.e., relational algebra. The major advantage of symbolic system is that it provides exact computation and its computation procedure is step-by-step and explicit. On the other hand, both machine learning and deep learning in the field of artificial intelligence utilize mathematical models (a.k.a neuro system [37]) to learn from data and generalize to unseen data, and thus perform tasks without explicit instructions. The representative mathematical models are statistical algorithms and artificial neural networks. In recent years, the neuro system brings huge attention as its success in natural language processing (NLP), computer vision (CV), speech recognition, etc. The most important properties of neuro system are intuitive and unconscious. In recent years, many applications in various domains [25, 38, 57] have been emerged, which cannot be efficiently processed by either symbolic-based data management system or neuro-based artificial intelligence system independently.

Figure 1: User query in video database

Example. Considering the illustrated example in Figure 1, the data analysts in NBA marketing team want to advertise NBA all-star game by promoting the NBA super star "Lebron James" [1]. Hence, they want to find the clips from the NBA data repository such that Lebron James is dunking in these games when his team is "Los Angeles Lakers" and he scored at least 30 points. Inherently, it is not trivial to answer by either symbolic-based databases or neuro-based AI systems as it includes two fundamental tasks: (i) identify the specific frames from video database, i.e., the frames Lebron James is dunking; and (ii) finding all these frames in a large video database with attribute constraints, e.g., scored at least 30 points and in Los Angeles Lakers.

A straight forward idea to address the above query is combining the abilities of both symbolic system and neuro system. In the literature, integrating ML tasks into database system has been studied

Proceedings of the VLDB Endowment, Vol. 17, No. 11 ISSN 2150-8097.
doi:10.14778/3681954.3682000

at the beginning of 2000 [43]. Many techniques have been proposed over these years in both academia [10, 28, 33, 42, 45, 55, 57] and industry [2, 3, 11, 12, 16, 18]. In particular, these system architectures can be classified into three categories: AI-centric, UDF-centric, and relation-centric. Zhou et al. [57] proposed a novel RDBMS by seamlessly integrating these three architectures. However, none of these existing solutions can **natively** process the analytical query in the above example. The core reason is that the result accuracy of the AI models is ignored among them as they assume the used AI models are given and well-trained. For example, during the above query processing, none of them take the accuracy of different dunking action recognition models into account.

The nsDB **vision.** To overcome the limitations of existing solutions, in this work, we envision a novel type of neuro-symbolic database system nsDB to process these new emerged queries. It integrates neural with symbolic systems to address the weaknesses of each, providing a strong database capable of data managing, model learning, complex and multi-model analytical query processing. Specifically, nsDB abstracts away the complexities of AI models, and allows end users to build AI projects and use them for their individual upstream applications, even they are without any code skills, AI expertise and system developing experiences. To achieve that, the neuro system is abstracted as a native-supported module in nsDB, and the result accuracy and processing latency are considered simultaneously during query optimization. However, it is not trivial to achieve the above goal as the implicit property of the neuro system compromises accuracy and performance inherently. For example, the more accurate of the dunk action detection model, the higher the model inference latency.

The rest of the paper is organized as follows. We briefly analyze the unique aspects of nsDB within the context of extensive ongoing work in Section 2. In Section 3, we first introduce the system architecture of nsDB, then highlight the research challenges of each component, last present our design paradigms and preliminary ideas to address them. We discuss the generality of nsDB in Section 4 and conclude this vision paper in Section 5.

2 RELATED WORK

In this section, we differentiate our proposal nsDB from the most relevant systems and techniques in the literature.

Neuro-Symbolic database system. Numerous researches [2, 3, 10–12, 16, 18, 28, 28, 33, 39, 42, 44, 45, 55, 57] have been studied to integrate DB and AI workloads in both academia and industry since 2000. The architecture of existing solutions can be classified into three representative categories: (i) AI-centric, (ii) UDF-centric, and (iii) relation-centric. To overcome the limitation of the solutions in each category, a mixed solution was proposed [57] which integrates the above three architecture categories. All these solutions (including of our nsDB) provide AI model inferences for various analytical tasks. However, none of the existing solutions have emerged as the *de-facto* standard until now. The major reasons can be summarized by three aspects: (i) model training, (ii) performance goal, and (iii) optimization strategy, as shown in Table 1.

(I) Model training: Almost all existing AI and DB integrated systems assume the underlying AI models are well-trained and the integrated systems are designed for efficient model inferences. However, the fact is that model training cannot be ignored in real-world applications. To make the matter worse, model training is not trivial to support by the above integrated systems as they are designed to provide excellent model inferring performance. Existing systems in AI-centric category (e.g., Google Big Query [11], Amazon Redshift ML [3]) train these models by offloading to the underlying DL systems (e.g., PyTorch, Tensorflow). Obviously, it is not efficient as the training data should be pre-prepared and it relies on other systems.

Table 1: Comparison of AI and DB integration solutions

Architecture category	Model training	Performance goal	Optimization strategy
AI-centric [33]	Yes	Latency-only	Symbolic
UDF-centric [35]	No	Latency-only	Symbolic
Rel.-centric [55]	No	Latency-only	Symbolic
Mixed sol. [57]	No	Latency-only	Neuro-symbolic
Our nsDB	**Yes**	**Latency-accuracy**	**Neuro-symbolic**

(II) Performance goal: The performance goal of existing AI and DB integrated systems is only the query processing latency as the underlying AI models are well-trained, which means the accuracy of these AI models are fixed. However, the same task can be processed by multiple AI models. Moreover, different models have different result accuracy for the same task. For example, ArcFace [29], FaceNet [49], and EigenFace [52] are the typical models for face recognition task, and the result accuracy of them are different.

(III) Optimization strategy: Existing systems [2, 3, 10–12, 16, 18] with AI-centric architecture offload the inference computation to the decoupled AI runtimes. Thus, their query optimizer only use symbolic rules to process the predicates in the analytical query and ignore the optimization of the complex neuro-based computations. The UDF-centric systems [28, 35, 39] use UDF to model the neuro-based computations, and apply the symbolic-based optimization strategies on the UDF-based logical plan. The relation-centric systems [42, 55] employ the relations to represent the model parameter tensor and extend the traditional relational algebra to tensor relation algebra and optimize them in a holistic symbolic manner. A simple co-optimization idea (i.e., devising novel query transformation rules) of symbolic and neuro operators in the complex analytical queries has been proposed in [57]. However, it cannot achieve low latency and high accuracy goal simultaneously.

In this work, we envision the next generation database system nsDB, which provides in-database model training, and a novel neuro-symbolic query optimizer is devised in it to co-optimize the performance latency and result accuracy of the complex analytical query processing. The last row of Table 1 shows the unique aspects of nsDB w.r.t. the existing DB and AI integrated systems.

Query processing over multi-modal data. Conducting complex query over multi-modal data is an active research topic [24, 30, 32] in database community in recent years. The general idea of them to process complex query on multi-modal data is decomposing the query into several subqueries and executing them to different systems [24, 32]. Our nsDB differs from them in two ways: (i) it integrates both symbolic and neural operators to process the different tasks in the complex query over multi-modal data,

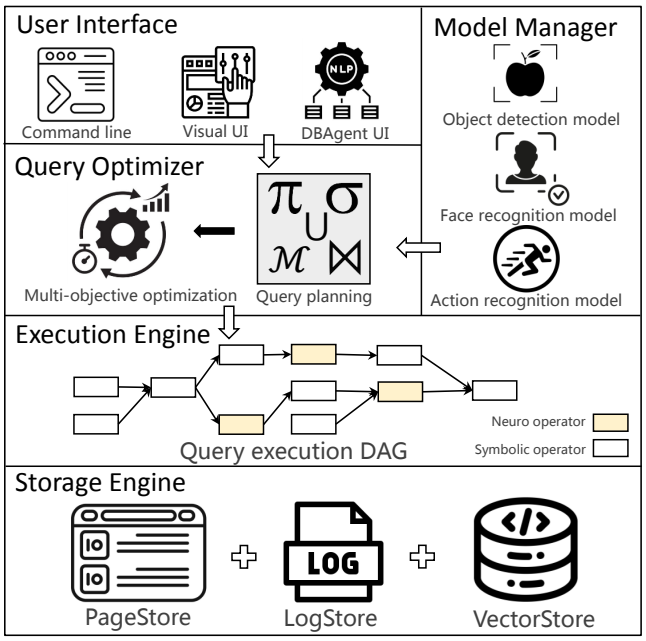

Figure 2: The architecture overview of nsDB

which is optimized in a holistic manner; and (ii) it considers the heterogeneity-native query execution and boosts the performance from the scratch. Moreover, nsDB does not replace these proposed systems, it offers an alternative solution for end-users.

Model management approach. Recently, several AI model management systems have been designed, e.g., ModelDB [14], AWS SageMaker [4], and learnware [58]. However, the fundamental principle of the model manager in nsDB is different from them. First, ModelDB is designed as a central repository for model tracking but lacks advanced capability such as model creation, training, and deployment. Second, while AWS SageMaker extends capabilities to encompass model creation, training, and deployment, it remains a ML platform, and the entire architecture lacks extensive interactions between AI models and query processing in database systems. Third, learnware [58] is a recently introduced model management tool that helps users to identify suitable models from a large repository to address various tasks. However, it neither involves model training and inference processes nor has the capability to generate the optimal execution plans for the complex queries. In contrast, the model manager is an integrated component within nsDB, engaging closely with other components to fulfill query tasks. It is designed to handle the entire life cycle of AI models, from training to finetuning to deployment, within a unified database framework.

3 THE SYSTEM ARCHITECTURE OF NSDB

Figure 2 depicts the system architecture of nsDB. It consists of 5 major components: (i) user interface, (ii) model manager, (iii) query optimizer, (iv) execution engine, and (v) storage engine. Each component in nsDB should be loosely decoupled to support diverse analytical queries efficiently, in contrast to a monolithic system where a fixed number of computing devices on the same server.

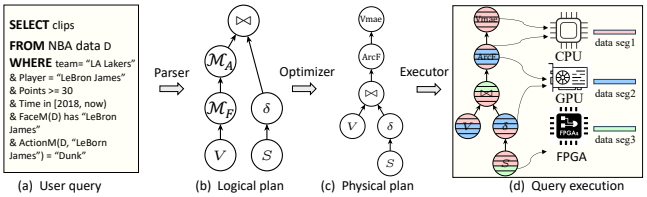

Figure 3: Query processing in nsDB

3.1 User Interface

Users submit their queries to nsDB via the user interface, e.g., the data analysts could submit their analytical query via command line or visual UI. nsDB plans to support advanced methods: end-users can express their query intention via natural language, and the large language model-based DBAgent of nsDB could convert it to the corresponding complex analytical SQL query. We will explain the concept of DBAgent shortly.

3.2 Model Manager

As shown in Figure 3, the user query first is parsed to the logical plan. One of the major characteristics of the logical plan in nsDB is that it includes: (i) the traditional symbolic SQL operators, see δ and $\bowtie$ on the relational NBA data table S in Figure 3(b); and (ii) the neuro AI operators, e.g., the face recognition model $\mathcal{M}_F$ and action recognition model $\mathcal{M}_A$ on the NBA video data **V** in Figure 3(b). nsDB utilizes the strong capability of neuro operators to process the unstructured data of various modalities, such as images, videos, texts, and audio. Thus, the first key challenge is building a suitable **model manager** in nsDB as it plays a vital role to process the complex queries on multi-modal data.

Traditional symbolic-based databases are known for their user-friendliness and ease to use, which can handle all kinds of queries by combining a limited number of SQL operators. In contrast, the neuro system lacks such high-level abstractions, primarily due to the inherent complexity of AI model. For example, developing and deploying AI models are complicated, involving various stages such as data collection, network design, model training, efficient inference, and many more. Properly navigating through these stages demands expertise in AI, posing a barrier to the widespread applicability of neuro models among the end users, which probably are not familiar with AI techniques. In this work, we combine symbolic operators and neural AI models via *model manager* component in nsDB. This combination facilitates end-users to engage with AI models through simple AI operators and empower users to process complex queries (e.g., the above analytical query involving multi-modal data) by harnessing the strengths of both realms.

A crucial component in our model manager is an AI agent, named as DBAgent. Powered by large language models (LLMs), DBAgent possesses advanced reasoning and planning capabilities, and offers key benefits to nsDB: (1) DBAgent provides a declarative approach for users to interact with nsDB. Users can describe their queries via natural language or high-level programs, and DBAgent is able to decompose them into a stream of sub-tasks, and translate them into queries compatible with nsDB. (2) DBAgent allows nsDB to efficiently support large-scale systems that potentially interleave

varied modalities of data. For optimal performance, different modalities might be processed by different programming languages, posing great challenges for existing data management platforms. DBAgent tackles this by reprogramming heterogeneous components via a unified language. This makes nsDB a highly scalable framework for real-world, complex systems. (3) Furthermore, the strong decision-making capability of DBAgent will enhance the autonomy of the model manager in nsDB through autonomously triggering, e.g., network pruning, model compression, or model retraining, if existing models do not meet the query criteria.

Abstracting away the intricacies of AI model interaction is a notable challenge in building nsDB. To tackle it, we intend to introduce AI-Tables, AI-Operators and AI-Models into the model manager. First, the introduction of AI-Tables is intended to store essential metadata of available AI models, including model identifiers, functional description, inference speed, accuracy, network layers/sizes, versioning information, etc. Th above abstraction of physical AI models serve as essential input for the query planning and optimizing. These metadata can be user-provided, automatically collected based on released information for public models, or obtained during private model training by evaluation on held-out validation samples. Second, the model manager extends traditional relational operators with a suite of AI-Operators at varying levels of granularity. These range from coarse-grained operators like `train`, `inference`, `finetune`, which encapsulate high-level operations of AI models, to the fine-grained ones like `pooling-layer`, `activation-layer`, which are abstractions of basic building blocks of an AI model. This will render a complete AI-to-SQL translation, ensuring the completeness of AI operators. These operators will enable users to smoothly interact with AI models via user-friendly SQL. Thus, the extended operator set in nsDB can be used to process complex analytical queries, i.e., generating the logical plan which includes both symbolic-based SQL operators and neuro-based AI operators. Third, the model manager will include commonly-used public AI models like those for face recognition or object detection as the model basis. However, in practice scenarios, public models might not fully align with query requirements. nsDB addresses this by offering customization options. On one hand, nsDB facilitates creating models from scratch. On the other hand, it allows for specializing models from public ones. For example, while VideoMAEV2 [54] shows state-of-the-art accuracy in recognizing human actions, it is limited in the high inference cost. For these cases, nsDB will allow for the distillation of knowledge from big models to small counterparts [34]. This offers an alternative to training from scratch and mitigate the difficulties associated with data collection and labelling. New models will be stored as separate records in AI-Tables.

3.3 Query Optimizer

Returning to the query processing example, with the supporting of AI-Tables and AI-Operators in model manager, the logical plan is derived by query planning in the query optimizer in nsDB. The query optimizer then optimizes it to the physical plan, see Figure 3(c). The second key challenge of nsDB is **multi-objective optimization** as it considers both latency and accuracy when it is finding the most efficient symbolic- and neuro-mixed physical execution plan. In traditional symbolic-based database system, the optimization goal of user query is only minimizing the query latency.

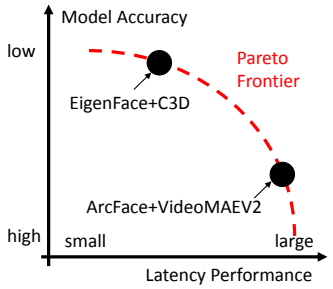

Figure 4: Pareto frontier of the user query in Figure 1

In existing DB and AI integrated systems, their performance goal also is only the query latency as they assume the underlying AI models are fixed. However, the neuro AI operators in nsDB is not exact computation, it always compromises the accuracy and latency. For example, to detect the frames which has Lebron James, ArcFace [29] is more accurate than EigenFace [52], but the inference cost of ArcFace is larger than EigenFace. Thus, the optimization goals in nsDB are two-fold: (i) query latency and (ii) result accuracy. Moreover, the fine-grained AI operators (e.g., `pooling-layer`, `activation-layer`) of the selected AI models will be pipelined with the symbolic SQL operators to derive the optimal physical plan. For ease of presentation, we use coarse-grained AI operators (e.g., `inference` of AI-models) in the examples.

We next elaborate the complexity of the query optimizer in nsDB by utilizing the analytical query in Figure 1. Suppose the logical plan of the user query includes two AI operators, as the face recognition model $\mathcal{M}_F$ and action recognition model $\mathcal{M}_A$ shown in Figure 3(b). To optimize the logical plan, there are three open questions: (1) there are various models for each AI operator. For example, ArcFace [29], FaceNet [49], and EigenFace [52] are the representative solutions for face recognition task; VideoMAEV2 [54], SlowFast [31], and C3D [51] are commonly used for action recognition task in different scenarios. The first opening question is *which one should be used in the physical execution plan?* (2) Similar to the join order selection problem in query optimizer of traditional symbolic-based databases, the second opening problem is *how to determine the model execution order?* Returning to the user query in Figure 1, two possible execution orders are: (a) it first finds all clips which have Lebron James then identifies these clips when he is dunking; and (b) it first identifies all dunking clips, then extracts a subset from them which includes Lebron James. (3) The third opening question is *how to cascade multiple models to achieve better performance or accuracy?* For example, a possible physical execution plan is employing a fast face recognition model first to find these frames which probably includes Lebron James, then applying a slow action recognition model to identify these dunking clips, last utilizing a slow-yet-accurate face recognition model to refine the final result. To make the matter worse, the above three opening questions are not independent, they are intricately connected.

With the above multi-objective optimization problem in mind, the query optimizer in nsDB addresses it by deriving the Pareto frontier of the underlying optimization problem efficiently and accurately. The Pareto frontier indicates the optimal performance

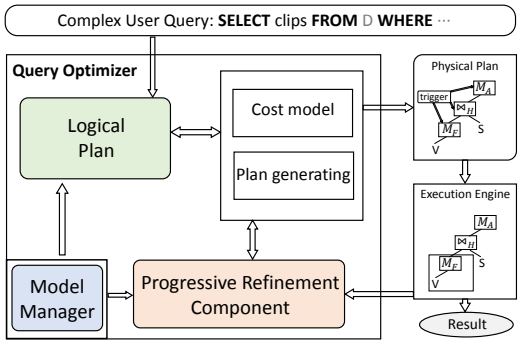

Figure 5: The architecture of query optimizer in nsDB, adapted from our preliminary work [53]

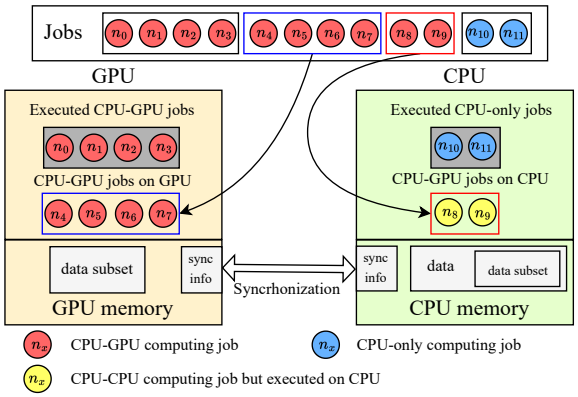

Figure 6: The execution engine in nsDB, adapted from our preliminary work [46]

theoretically. It can be used to guide the selection of underlying AI models and the generation of physical execution plan. As illustrated in Figure 4, it shows the Pareto frontier of the user query in Figure 1. Unfortunately, identifying the Pareto frontier of the complex user query by the query optimizer in nsDB is naturally complex. Logically, the hardness of identifying the Pareto frontier is obvious not easier than obtaining the accurate cardinality estimations in traditional query optimizer in symbolic-based databases. Thus, the design principle of the query optimizer in nsDB is *"achieving both performance and accuracy goals via progressive query optimization"*.

The architecture of the query optimizer in nsDB is illustrated in Figure 5. Specifically, we follow the query re-optimization paradigm to implement the design principle of progressive query optimization. The core reason is that it is almost impossible to obtain the optimal query execution plan at the first glance even in traditional symbolic-based database systems. As shown in Figure 5, the query optimizer of nsDB works as follows. It first utilizes the metadata of AI operators, which is provided by model manager (as we elaborated in Section 3.2), the cost model and the plan generator to generate the initial physical execution plan. During the query execution, the *trigger* observes the intermediate result and identifies the Pareto-frontier obtained so far. The progressive refinement component refines the execution plan by taking both the runtime statistics and metadata of the AI models into consideration. Thus, the execution plan will be adjusted progressively and dynamically. There are two possible approaches for the progressive refinement component: (i) symbolic-based and (ii) neural-based. For (i), the intermediate runtime statistics are used as optimization constrains. For (ii), these statistics are used as training data to improve the underlying decision AI model in the progress refinement component.

There are many research problems and technical challenges (e.g., cost model, query compilation) to build the query optimizer of nsDB. Our general idea is exploiting the successful existing techniques [9] and avoiding to reinvent the wheels. For example, the number of floating point values should be used as one of the cost metrics in the cost model nsDB by following the idea in [55]. The query compilation and optimization framework in [36] and the unified intermediate representation in [57] also will be considered in nsDB.

3.4 Heterogeneity-native Query Execution

Back to the query processing procedure in nsDB, the last step is executing the optimized physical plan in the execution engine of nsDB, as shown in Figure 3(d). Specifically, the optimized physical plan is a directed acyclic graph (DAG), which consists of a set of both symbolic- and neuro- operators. The execution engine schedules them to the heterogeneous computing units (e.g., CPU, GPU, FPGA), and executes them by accessing the data in storage engine.

In nsDB, we advocate to architect a *heterogeneity-native execution engine* as the heterogeneity is obvious in modern data-intensive systems. In particular, many database systems [23, 26, 27, 41] are designed to exploit the heterogeneous hardware, e.g., CPU, GPU and FPGA. The specialized computing hardware (e.g., TPU, DPU) also have been widely emerged and utilized in AI systems. The third key challenge of nsDB is architecting **heterogeneity-native query execution**. It is not trivial as the physical plan includes both symbolic SQL operators and neuro AI operators.

Abstractly, given a set of operators in the physical plan DAG and a set of available computing units, how to execute them efficiently is an opening question. The technical challenges to address it are from three aspects. First, it is *assigning right operator to the right computation unit*. Specifically, there are four operators in the physical plan of the user query, as shown in Figure 3(c). It is not straight-forward to efficiently execute them via the available computing units, i.e., CPU, GPU, and FPGA in Figure 3(d). Second, it is *guaranteeing the load balance among all these heterogeneous computation units*. In particular, the execution engine should maximize both intra- and inter- parallelism during the physical plan execution. It only can be achieved by balancing the workload in each computing unit. Third, it is *facilitating the execution by careful-designed data layouts in storage engine*. For example, the columnar data layout boosts the analytical query processing in traditional OLAP systems. However, the execution logic of the neuro AI operators are more complex than analytical queries. It is an opportunity to improve its execution performance by specific data layout in storage engine.

Hence, we plan to address the above challenges by following the design principle "*decomposition facilitates allocation*" in nsDB. From the operating system perspective, existing execution unit is abstracted by thread. The major limitation of this abstraction is that the context switch overhead is expensive when executing the physical DAG. To overcome it, several lightweight user-space execution units (e.g., Arachne [47], boost fiber [8], libco [13]) have been proposed to provide low latency and high throughput for the

upstream applications. Motivated by them, we plan to decompose each operator in the physical execution DAG into a set of fine-grained execution units in nsDB. These fine-grained execution units either enjoy low latency and high throughput or facilitate the allocation and execution [40]. In particular, we abstract the atomic execution unit "job" for each operator in the physical plan DAG, i.e., "n_x" shown in Figure 6. The requirements of the fine-grained "n_x" are: (i) run-to-completion, which guarantees atomic and facilitates execution; and (ii) supporting code variants, which is key to build heterogeneity-native execution engine. For example, it should have GPU- and FPGA- based code implementations for the matrix multiply operator if it may be executed on GPU or FPGA. The executing types of the job n_x are determined by its available code variants. As the CPU-GPU heterogeneous execution example shown in Figure 6, the jobs of some operators in the DAG are n_0 to n_{11}. The jobs n_0 to n_9 can be executed by both CPU or GPU (see red nodes). However, n_{10} and n_{11} only can be executed by CPU (see blue nodes). During the execution, all ready jobs are scheduled to the heterogeneous hardware and be executed simultaneously. For example, n_0 to n_3 are executed on GPU, and n_{10} and n_{11} are executed in CPU. To ensure the load balance among different computing units, the executor will take the runtime utilization of the hardware into consideration. For example, n_8 and n_9 are scheduled and executed in CPU even though it can execute on GPU. The reason is that when GPU is processing n_4 to n_7, CPU is ideal then it is used to execute n_8 and n_9. It is worth to point out the generic of the above example can be achieved by replacing the GPU in the left to any other computing units, e.g., FPGA or others.

3.5 Storage Engine

The design paradigm of storage engine in nsDB is that *"data layout boosts query processing"* as the data accessing cost becomes obvious in modern data-intensive systems. In traditional symbolic-based database systems, both row-based and column-based data layout have been used for various applications, e.g., transaction processing and analytical processing. The core idea is that the data layout improves the data access and cache locality efficiency during query processing. In nsDB, we follows the same idea to design the PageStore of the storage engine. For example, it is quite common that there exists dependencies and similarities among the large scale of multi-model data, e,g., different frames of NBA game video always share the same background. To improve the query processing among the video data, we plan to use the tile data layout [56] in the PageStore of nsDB. In particular, it splits the video into non-overlapping $N \times M$ tile sequences, and each sequence supports independent decoding, where N and M are the number of rows and columns. To minimize the cost of AI models (e.g., dunking action recognition) on these videos, the optimal tile layout can be constructed by modeling the cost of data loading, region of interest detecting, and tile decoding. In addition, we introduce VectorStore in the storage engine of nsDB to store, manage, and index the embedding vectors of multi-modal data.

4 DISCUSSION OF NSDB

We briefly discuss the generality of nsDB by elaborating how to clean the real-world dirty data in it. Many data cleaning systems [20, 21, 48] have been proposed in the literature. The methodologies of these systems can be divided into two categories: (i) logic rules, and (ii) machine learning (ML). Rock [22] unifies ML and logic deduction in a system to address data quality issues, e.g., entity resolution and conflict resolution. Fortunately, nsDB can achieve the same goal of Rock by including these ML classifiers into its model manager. In particular, the proposed extension of Entity Enhancing Rules REE^{++} will be divided into two parts in nsDB: (i) the traditional logic rules in REE^{++} are the symbolic predicates; and (ii) the embedded ML models (e.g., similarity checking, link prediction) are used as neural operators. Thus, the end users could conduct error detection on various data from different applications (e.g., sales and logistics) by writing simple SQL queries on nsDB.

5 CONCLUSION

In this work, we envision nsDB, which integrates both neural- and symbolic- systems and provides the capabilities of AI model management and complex query processing. Specifically, it consists of three important components: (i) robust model manager, (ii) multi-objective optimization, and (iii) heterogeneity-native execution. In the past few years, our team is working on several projects [41, 46, 50, 53, 56], which are the preliminary attempts to address these technical challenges in nsDB. For example, video data management [56], heterogeneous computing [41, 46], AI-enhanced query optimization [53], query performance diagnosing [50]. We believe nsDB is the next generation database system, which can be used for various applications, e.g., video analysis, chemistry and natural sciences. This vision paper serves as a research road map for building it and highlights the research challenges in the journey. While we are in the early stages of the journey, we are excited about the promising future of nsDB. Specifically, it not only enhances the usability of database in various new emerged applications, but also removes the barriers to build AI projects, and allow anyone without expertise could use AI models in nsDB as we use SQL in database today. There are many opening problems to build nsDB, e.g., query compilation and optimization, the sufficiency of VectorStore for bridging AI and DB systems. We hope the researchers from different communities (e.g., OS, DB, AI, CHI) could tackle them together.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments. Ye Yuan is supported by the National Key R&D Program of China (Grant No.2022YFB2702100) and the NSFC (Grant Nos. 61932004, 62225203, U21A20516). Bo Tang is supported by the Guangdong Province Key Laboratory (Grant No. 2020B121201001) and the Shenzhen Fundamental Research Program (Grant No. 20220815112848002). Zhiwei Zhang is supported by the National Key R&D Program of China (No.2021YFB2700700), and the NSFC (Grant No.U23B2019, No.62072035). Jianbin Qin is supported by National Key R&D program of China (No. 2021YFB3301500), the Guangdong Province Key Laboratory of Popular High Performance Computers 2017B030314073; Shenzhen University Research Fund 2023YQ017 and 20220810142731001; and CCF-Tencent Fund RAGR20230126. Bo Tang is also affiliated with the Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology, Shenzhen, China.

REFERENCES

[1] 2024. *2023 NBA All-Star Game Promo.* https://www.nba.com/watch/video/2023-nba-all-star-game-promo.

[2] 2024. *AI Functions on Databricks.* https://docs.databricks.com/en/large-language-models/ai-functions.html.

[3] 2024. *Amazon Redshift ML.* https://aws.amazon.com/blogs/aws/amazon-redshift-ml-is-now-generally-available-use-sql-to-create-machine-learning-models-and-make-predictions-from-your-data/.

[4] 2024. *Amazon SageMaker.* https://aws.amazon.com/sagemaker/.

[5] 2024. *Apache Flink.* https://flink.apache.org/.

[6] 2024. *Apache Hive.* https://hive.apache.org/.

[7] 2024. *Apache Spark.* https://spark.apache.org/.

[8] 2024. *Boost Fiber.* https://www.boost.org/doc/libs/1_83_0/libs/fiber/doc/html/fiber/overview.html.

[9] 2024. *DAPHNE: Integrated Data Analysis Pipelines for Large-Scale Data Management, HPC, and Machine Learning.* https://daphne-eu.github.io/.

[10] 2024. *EvaDB: Database system for AI-powered apps.* https://github.com/georgia-tech-db/evadb.

[11] 2024. *Introduction to AI and ML in BigQuery.* https://cloud.google.com/bigquery/docs/bqml-introduction.

[12] 2024. *Large Language Models for sentiment analysis with Amazon Redshift ML.* https://aws.amazon.com/blogs/big-data/large-language-models-for-sentiment-analysis-with-amazon-redshift-ml-preview/.

[13] 2024. *libco.* https://github.com/Tencent/libco.

[14] 2024. *ModelDB: An open-source system for Machine Learning model versioning, metadata, and experiment management.* https://github.com/VertaAI/modeldb.

[15] 2024. *MySQL.* https://www.mysql.com/.

[16] 2024. *PostgresML.* https://postgresml.org/.

[17] 2024. *PostgreSQL.* https://www.postgresql.org/.

[18] 2024. *SQL-only LLM for text generation using Vertex AI model in BigQuery.* https://cloud.google.com/blog/products/ai-machine-learning/llm-with-vertex-ai-only-using-sql-queries-in-bigquery.

[19] 2024. *Symbolic System.* https://en.wikipedia.org/wiki/Computer_algebra.

[20] Arvind Arasu, Christopher Ré, and Dan Suciu. 2009. Large-scale deduplication with constraints using dedupalog. In *ICDE*. 952–963.

[21] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent query answers in inconsistent databases. In *PODS*. 68–79.

[22] Xianchun Bao, Zian Bao, Binbin Bie, Qingsong Duan, Wenfei Fan, Hui Lei, Daji Li, Wei Lin, Peng Liu, Zhicong Lv, Mingliang Ouyang, Shuai Tang, Yaoshu Wang, Qiyuan Wei, Min Xie, Jing Zhang, Xin Zhang, Runxiao Zhao, and Shuping Zhou. 2024. Rock: Cleaning Data by Embedding ML in Logic Rules. In *SIGMOD*.

[23] Nils Boeschen and Carsten Binnig. 2022. GaccO-A GPU-accelerated OLTP DBMS. In *SIGMOD*. 1003–1016.

[24] Zui Chen, Zihui Gu, Lei Cao, Ju Fan, Sam Madden, and Nan Tang. 2023. Symphony: Towards natural language query answering over multi-modal data lakes. In *CIDR*. 8–151.

[25] Dawei Cheng, Sheng Xiang, Chencheng Shang, Yiyi Zhang, Fangzhou Yang, and Liqing Zhang. 2020. Spatio-temporal attention-based neural network for credit card fraud detection. In *AAAI*. 362–369.

[26] Monica Chiosa, Thomas B Preußer, Michaela Blott, and Gustavo Alonso. 2023. AMNES: Accelerating the computation of data correlation using FPGAs. *PVLDB* 16, 13 (2023), 4174–4187.

[27] Periklis Chrysogelos, Panagiotis Sioulas, and Anastasia Ailamaki. 2019. Hardware-conscious query processing in gpu-accelerated analytical engines. In *CIDR*.

[28] Francesco Del Buono, Matteo Paganelli, Paolo Sottovia, Matteo Interlandi, and Francesco Guerra. 2021. Transforming ML predictive pipelines into SQL with MASQ. In *SIGMOD*. 2696–2700.

[29] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. 2019. Arcface: Additive angular margin loss for deep face recognition. In *CVPR*. 4690–4699.

[30] Wenfei Fan, Ping Lu, Kehan Pan, Ruochun Jin, and Wenyuan Yu. 2024. Linking entities across relations and graphs. In *TODS*. 634–647.

[31] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. 2019. Slowfast networks for video recognition. In *ICCV*. 6202–6211.

[32] Vijay Gadepally, Peinan Chen, Jennie Duggan, Aaron Elmore, Brandon Haynes, Jeremy Kepner, Samuel Madden, Tim Mattson, and Michael Stonebraker. 2016. The BigDAWG polystore system and architecture. In *HPEC*. 1–6.

[33] Apurva Gandhi, Yuki Asada, Victor Fu, Advitya Gemawat, Lihao Zhang, Rathijit Sen, Carlo Curino, Jesús Camacho-Rodríguez, and Matteo Interlandi. 2023. The tensor data platform: Towards an ai-centric database system. *CIDR*.

[34] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

[35] Dimitrije Jankov, Shangyu Luo, Binhang Yuan, Zhuhua Cai, Jia Zou, Chris Jermaine, and Zekai J Gao. 2020. Declarative recursive computation on an RDBMS: or, why you should use a database for distributed machine learning. *ACM SIGMOD Record* 49, 1 (2020), 43–50.

[36] Michael Jungmair, André Kohn, and Jana Giceva. 2022. Designing an open framework for query optimization and compilation. *PVLDB* 15, 11 (2022), 2389–2401.

[37] Md Kamruzzaman Sarker, Lu Zhou, Aaron Eberhart, and Pascal Hitzler. 2021. Neuro-Symbolic Artificial Intelligence: Current Trends. *arXiv e-prints*, arXiv–2105.

[38] Daniel Kang, Francisco Romero, Peter D Bailis, Christos Kozyrakis, and Matei Zaharia. 2022. VIVA: An End-to-End System for Interactive Video Analytics. In *CIDR*.

[39] Konstantinos Karanasos, Matteo Interlandi, Doris Xin, Fotis Psallidas, Rathijit Sen, Kwanghyun Park, Ivan Popivanov, Supun Nakandal, Subru Krishnan, Markus Weimer, et al. 2020. Extending Relational Query Processing with ML Inference. *CIDR*.

[40] Haotian Liu, Runzhong Li, Ziyang Zhang, and Bo Tang. 2025. Tao: Improving Resource Utilization while Guaranteeing SLO in Multi-tenant Relational Database-as-a-Service. In *SIGMOD*.

[41] Haotian Liu, Bo Tang, Jiashu Zhang, Yangshen Deng, Xiao Yan, Xinying Zheng, Qiaomu Shen, Dan Zeng, Zunyao Mao, Chaozu Zhang, Zhengxin You, Zhihao Wang, Runzhe Jiang, Fang Wang, Yiu Man Lung, Huan Li, Mingji Han, Qian Li, and Zhenghai Luo. 2022. GHive: accelerating analytical query processing in apache hive via CPU-GPU heterogeneous computing. In *SoCC*. 158–172.

[42] Shangyu Luo, Zekai J Gao, Michael Gubanov, Luis L Perez, and Christopher Jermaine. 2018. Scalable linear algebra on a relational database system. *ACM SIGMOD Record* 47, 1 (2018), 24–31.

[43] Amir Netz, Surajit Chaudhuri, Jeff Bernhardt, and Usama Fayyad. 2000. Integration of data mining and relational databases. In *VLDB*. 285–296.

[44] Beng Chin Ooi, Shaofeng Cai, Gang Chen, Kian Lee Tan, Yuncheng Wu, Xiaokui Xiao, Naili Xing, Cong Yue, Lingze Zeng, Meihui Zhang, et al. 2024. NeurDB: An AI-powered Autonomous Data System. *arXiv preprint arXiv:2405.03924* (2024).

[45] Kwanghyun Park, Karla Saur, Dalitso Banda, Rathijit Sen, Matteo Interlandi, and Konstantinos Karanasos. 2022. End-to-end optimization of machine learning prediction queries. In *SIGMOD*. 587–601.

[46] Cui Pengjie, Liu Haotian, Tang Bo, and Yuan Ye. 2024. CGgraph: An Ultra-fast Graph Processing System on Modern Commodity CPU-GPU Co-processor. *PVLDB* 17, 6 (2024), 1405–1417.

[47] Henry Qin, Qian Li, Jacqueline Speiser, Peter Kraft, and John Ousterhout. 2018. Arachne: Core-Aware thread management. In *OSDI*. 145–160.

[48] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* 10, 11.

[49] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *CVPR*. 815–823.

[50] Qiaomu Shen, Zhengxin You, Xiao Yan, Chaozu Zhang, Ke Xu, Dan Zeng, Jianbin Qin, and Bo Tang. 2024. QEVIS: Multi-grained Visualization of Distributed Query Execution. *TVCG*, 153–163.

[51] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. 2015. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*. 4489–4497.

[52] Matthew A Turk and Alex P Pentland. 1991. Face recognition using eigenfaces. In *CVPR*. 586–587.

[53] Fang Wang, Xiao Yan, Man Lung Yiu, Shuai LI, Zunyao Mao, and Bo Tang. 2023. Speeding Up End-to-end Query Execution via Learning-based Progressive Cardinality Estimation. *SIGMOD* 1, 1 (2023), 1–25.

[54] Limin Wang, Bingkun Huang, Zhiyu Zhao, Zhan Tong, Yinan He, Yi Wang, Yali Wang, and Yu Qiao. 2023. Videomae v2: Scaling video masked autoencoders with dual masking. In *CVPR*. 14549–14560.

[55] Binhang Yuan, Dimitrije Jankov, Jia Zou, Yuxin Tang, Daniel Bourgeois, and Chris Jermaine. 2021. Tensor relational algebra for distributed machine learning system design. *PVLDB* 14, 8 (2021).

[56] Tianxiong Zhong, Zhiwei Zhang, Guo Lu, Ye Yuan, Yu-Ping Wang, and Guoren Wang. 2024. TVM: A Tile-based Video Management Framework. *PVLDB* 17, 4, 671—-684.

[57] Lixi Zhou, Qi Lin, Kanchan Chowdhury, Saif Masood, Alexandrem Eichenberger, Hong Min, Alexander Sim, Jie Wang, Yida Wang, Kesheng Wu, Binhang Yuan, and jia Zou. 2024. Serving Deep Learning Models from Relational Databases. In *EDBT*. 717–724.

[58] Zhi-Hua Zhou and Zhi-Hao Tan. 2024. Learnware: Small models do big. *Science China Information Sciences* 67, 1 (2024), 112102.