The University of Melbourne

COMP90050, Winter term, 2023

A Survey on Self-Driving Databases
Group 23

Yuan Cao - 1067709
Jiahao Chen - 1118749
Zhiquan Lai - 1118797
Mingyang Liu - 1113531

Abstract

As the Big Data era continues to evolve, so does the complexity of database management systems (DBMS) and the need for automated configuration, management and optimisation, leading to the emergence of self-driving databases. This paper surveys recent advances in self-driving databases from 2017 to 2023, with a particular focus on architecture, workload prediction, behavioural modelling, physical design tuning and other optimisation approaches. We discuss several self-driving database management system architectures, such as Peloton, Generalized Self-Driving Framework, and NoisePage. we also provide insights into current trends and approaches for workload prediction and behavioural modelling, such as QB5000, DBMind, and MB2. In addition, we examine physical strategies and AI models for design tuning and other optimisations, including storage structure selection, Explain-Tun, external and internal machine learning agents, and training data extraction methods such as TScout. Finally, we discuss future directions for self-driving databases, highlighting the increasing convergence of AI and machine learning techniques and the quest for further performance optimisation. Our survey aims to provide a comprehensive view of the future of self-driving databases, offering insights into current strategies and potential future advances in the field.

July 23, 2023

1 Introduction

In the era of big data, the application of DBMS is common and necessary as it enables developers to generate queries on desired data while ensuring optimal efficiency in query execution [1]. Due to the increasing complexity of modern DBMSs, database administrators (DBAs) need to devote considerable time to manually fine-tuning systems for optimal performance [2, 3], which has given rise to the concept of self-driving databases that aims to automate the process of configuration, management and optimization [4]. Tuning involves various aspects of databases, such as physical database design (indexes, partitioning, views), physical resources allocation (CPU, memory, disk, machine, network bandwidth), knob configuration (concurrency mechanisms, buffer pool size, caching policies), query planning (join orderings), etc [3, 4, 5, 6]. Optimally tuning these aspects can be challenging, such as deciding how to apply changes. There exists a trade-off in allocating the appropriate amount of resources for tasks such as index creation, where minimizing contention and maximizing query acceleration should be carefully balanced [7]. Generally, the tuning techniques can be mainly categorized into two classes: rule-based approaches that require developers to design specific rules to modify system behaviour, and machine learning methods [8].

The advancement of storage and computational hardware nowadays [2, 3], the achievements in the area of machine learning [2], together with the widespread cloud technologies [3], have paved the way for developing fully autonomous self-driving databases. Currently, the research trend in this field is on utilizing ML techniques to optimize the tuning process, and such methods can be integrated into DBMSs either externally or internally, depending on the software design and specific requirements [5]. The typical process involves predicting future workload as the initial step, followed by utilizing a behaviour model to estimate the cost of potential actions and an optimal action plan is finally generated based on the estimations [2, 4, 5, 9]. The training process for workload prediction and behaviour modelling can be categorized as either offline or online [9]. The difference is that in the offline approach, the data is sourced from past workloads, while in the online one, data is gathered in real-time [9]. It is also important to make the DBMS autonomously monitor and diagnose itself when encountering abnormal activities [10].

Similar to self-driving vehicles, DBMSs can be classified into 5 levels [4] as Table 1 presents.

Table 1: Levels of autonomy in DBMSs [4]

Level	Name	Description
0	Manual	Without autonomy
1	Assistant	Suggestions to assist the user
2	Mixed	Concurrent user and system control
3	Local	Independent components
4	Directed	Semi-autonomous with user guidance
5	Self-Driving	Completely autonomous without guidance

This paper presents the findings of a conducted survey that explores the recent developments and progress achieved in the field of self-driving databases, covering the period from 2017 to 2023. A variety of frameworks and algorithms will be introduced and briefly discussed, along with an evaluation of their advantages and disadvantages. Furthermore, the paper will put forward our insights and reflections on the future directions and potential advancements in this domain.

2 Related Work and Analysis

2.1 Architecture

In general, a DBMS architecture is the overall design and structure of a database management system that enables basic functions of a database such as storage, organisation, retrieval and management of data. DBMS architecture will aim to be effective, efficient, secure and reliable. Three self-driving architectures have been explored in this essay, Peloton, Generalized Self-Driving Framework and NoisePage. Most self-driving databases consist of similar main components including workload prediction and behaviour modelling.

2.1.1 Peloton [1]

Pavlo et al.[1] introduces the Peloton architecture for self-driving database management systems, which is the first architecture in this area. The goal of self-driving systems is to automate traditionally labour-intensive and expertise-required tasks, such as tuning and physical design.

The Peloton architecture includes 3 main parts, Workload Classification, Workload Forecasting and Action Planning & Execution Workload Classification involves classifying incoming queries into workload types by machine learning algorithm. Then the workload history is used to forecast future query behaviours. Finally, a proper action is executed based on the prior classification and forecasting.

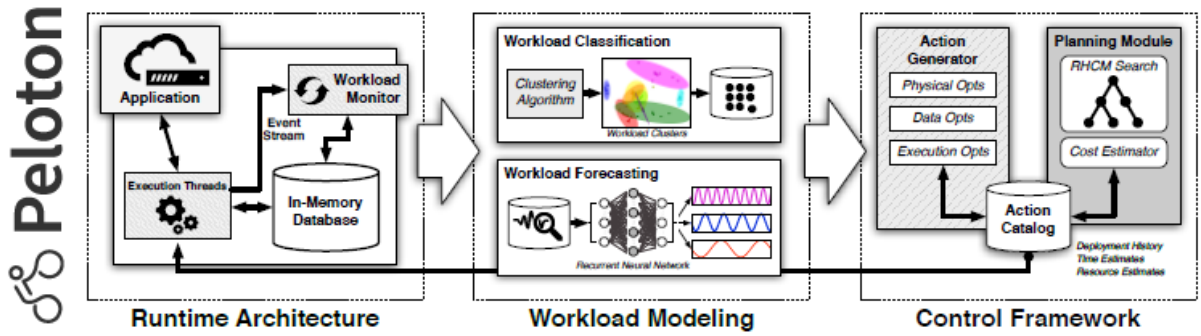


Figure 1: Workflow of Peloton Architecture [1]

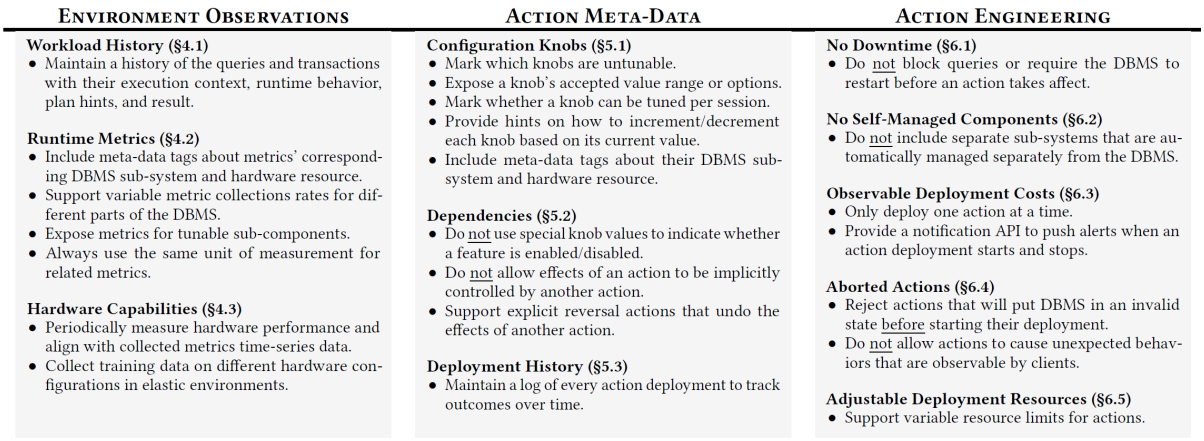
2.1.2 Generalized Self-Driving Framework [3]

The Generalized Self-Driving Framework is proposed by Dr Jan Kossmann[3], which is dedicated to solving the increase of complexity, shift to the cloud database and increased Number of Tuning Dimensions. This framework is designed to solve three challenges for the Self-Driving database, tuning, feedback loop and forecasting with an organizer, tuner and workload Analyzer. The framework is implemented in the hyrise[2] relational database system and has inherent high flexibility from hyrise.

The workflow starts with feeding query plans into the Query Plan Cache by the Query Optimizer. Subsequently, the Workload Analyzer, utilizing data derived from the Query Plan Cache, generates workload forecasts. The forecast results are then used by Organizer and the Tuner. The organiser is for controlling and overseeing the tuning processes. Tuner has three processes, the Elevator evaluates all potential actions and selects the most beneficial action after evaluation, and then an executor will execute the selected action.

2.1.3 NoisePage

Pavlo et al. proposed a self-driving database architecture named NoisePage, which consists of three main components: workload forecasting, behaviour modelling, and action planning [4], which is similar to the logic of self-driving vehicles and potentially has level 5 of DBMS autonomy. Figure 2 presents the summarised design principles of self-driving DBMS proposed by them [4].



Environment Observations

Workload History (§4.1)
- Maintain a history of the queries and transactions with their execution context, runtime behavior, plan hints, and result.

Runtime Metrics (§4.2)
- Include meta-data tags about metrics' corresponding DBMS sub-system and hardware resource.
- Support variable metric collections rates for different parts of the DBMS.
- Expose metrics for tunable sub-components.
- Always use the same unit of measurement for related metrics.

Hardware Capabilities (§4.3)
- Periodically measure hardware performance and align with collected metrics time-series data.
- Collect training data on different hardware configurations in elastic environments.

Action Meta-Data

Configuration Knobs (§5.1)
- Mark which knobs are untunable.
- Expose a knob's accepted value range or options.
- Mark whether a knob can be tuned per session.
- Provide hints on how to increment/decrement each knob based on its current value.
- Include meta-data tags about their DBMS sub-system and hardware resource.

Dependencies (§5.2)
- Do not use special knob values to indicate whether a feature is enabled/disabled.
- Do not allow effects of an action to be implicitly controlled by another action.
- Support explicit reversal actions that undo the effects of another action.

Deployment History (§5.3)
- Maintain a log of every action deployment to track outcomes over time.

Action Engineering

No Downtime (§6.1)
- Do not block queries or require the DBMS to restart before an action takes affect.

No Self-Managed Components (§6.2)
- Do not include separate sub-systems that are automatically managed separately from the DBMS.

Observable Deployment Costs (§6.3)
- Only deploy one action at a time.
- Provide a notification API to push alerts when an action deployment starts and stops.

Aborted Actions (§6.4)
- Reject actions that will put DBMS in an invalid state before starting their deployment.
- Do not allow actions to cause unexpected behaviors that are observable by clients.

Adjustable Deployment Resources (§6.5)
- Support variable resource limits for actions.

Figure 2: Design principles of Self-driving DBMS [4]

The workload forecasting component predicts future queries using forecast models generated by workload traces and database statistics (QB5000) [2]. Arrival rates and parameter samples are recorded when clients send workloads to NoisePage and the forecast models are periodically updated using these data [4]. Other methods such as query templatization and arrival-rate-pattern clustering are integrated into the system to enhance its efficiency [4]. To keep track of the past workload, sampling and aggregation can be applied to reduce the overhead [4].

Behaviour models are used to evaluate a potential action's impact on the system, which face the following challenges [4]:

- high dimensionality of inputs
- handling concurrent operations
- difficulties in training
- generalizability and explainability

Noise page separates different components in a DBMS into operating units (OUs) which are independent of each other and have a small size (ModelBot2) [8] to the dimension of each behaviour model. Besides, an interference model is applied to help resolve the resource competition between concurrent OUs and a couple of off-line runners are customized to provide OUs with training data [4]

Based on the predicted workload and the behaviour estimations, the executions of actions need to be properly planned. The plan should consider both the present and future workloads, meet the system constraints and be able to explain the planned actions, which allows for examination, auditing and debugging [4]. NoisePage uses receding horizon control (RHC) together with Monte Carlo tree search (MCTS) to balance planning cost and quality [4].

2.1.4 Summary

The Peloton was one of the earliest prototypes for self-driving database systems, which established a foundation for further developments. NoisePage serves as a subsequent and more advanced self-driving DBMS, which provided detailed implementations of various components and established comprehensive design principles. Additionally, NoisePage incorporates other recently developed strategies, such as the utilization of QB5000 for workload forecasting and ModelBot2 for behaviour modelling. The classification process in Peloton was eliminated because of the use of ModelBot2, which will be discussed in the behaviour modelling section. Generally speaking, NoisePage is a more robust infrastructure compared to others, which has paved the way for future research in this field.

2.2 Workload Prediction

Workload forecasting enables the DBMS to gain insights into potential workloads and make necessary preparations using prediction results obtained from pre-trained forecasting models. In this section, we will introduce QB5000, which is applied in the NoisePage architecture.

2.2.1 QueryBot5000 [2]

QueryBot5000(QB5000) is proposed by the research team from Carnegie Mellon University[2]. It is an innovative workloads prediction system, which helps to conquer the workload prediction. There are three main parts in QB5000, pre-processor, cluster and forecaster.

QB5000 begins with DBMS sending queries to the system and passing received queries into Pre-Processor. Pre-Processor works as workload template identifiers and records arrival rate history. Then pre-processed queries are passed into the Clusterer which clusters templates with similar arrival rate patterns. Finally, the clustered templates are processed by the Forecaster to build a prediction model for the future arrival rate[2].

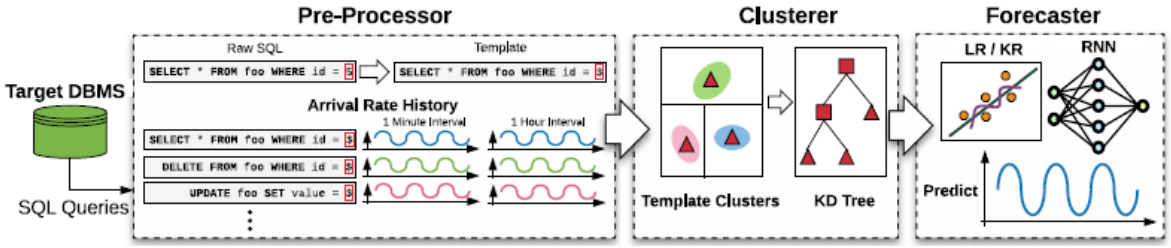


Figure 3: Workflow of QB5000 [2]

Researchers conduct seven experiments to show the effectiveness and improvement in different aspects including modelling ability, accuracy, spike prediction, prediction interval, overhead of computation & Storage, Index Selection and the clustering feature. The results show that QB5000 has the best performance so far.

Researchers from Carnegie Mellon University made one step toward the autonomous database by proposing a more efficient workload prediction system. The QB5000 make the further implementation of self-driving DBMS possible.

2.3 Behaviour Modelling

Behaviour modelling helps to estimate a potential action's cost, which is the basis of action planning. This section will introduce two approaches for creating behaviour models.

2.3.1 DBMind [10]

From the aspect of Query processing in self-driving databases. The process involves the execution of queries, which can be optimized through various techniques. The paper "Self-driving Database Management Systems: Forecasting, modelling, and Planning" discusses the development of a query processing framework that uses machine learning to predict the cost of executing a query plan [7]. This allows the system to select the most efficient query plan, reducing the overall execution time and improving the performance of the database. In DBMind system, it is designed to deliver high performance and low latency for OLTP (Online Transaction Processing) scenarios. It supports multi-core parallel computing and column-store indexes to improve query performance. It uses an LSTM model to profile the database status and detect anomalies such as slow queries and IO contention [10]. This system also uses prediction algorithms to anticipate future risks, such as slow queries, resource anomalies, and performance degradation.

2.3.2 MB2 [8]

ModelBot2 (MB2) is a behaviour modelling framework proposed by Ma et al., which is designed for self-driving DBMSs [8] and includes two primary considerations: (1) MB2 generates models offline in a workload and dataset-independent manner to make the process less time-consuming and costly, (2) MB2's models are debuggable, explainable, and adaptable, reducing development complexity and providing insight into DBMS action choices [8]. MB2 decomposes the DBMS into independent operating units (OUs) as Figure 4 presents. Each of them is corresponding to a DBMS task (e.g. index build, join hash table build) and is paired with an OU-runner [8].

MB2 uses concurrent runners to execute end-to-end workloads, which aims to simulate concurrent environments and orchestrate data collection [8]. Interference models are generated using the training data to estimate the impacts of resource competition, cache locality, and internal contention among concurrent OUs, which can serve as a simulator for a self-driving DBMS by providing estimations of runtime behaviour [8]. Given the forecasted workload and potential action inputs, MB2 uses the models to predict the behaviour of each OU, adjust for the impact of concurrent OUs, and guide the DBMS's planning system [8]. Compared to a single monolithic model for the entire DBMS, OU-models have the following advantages [8]:

- Have smaller input dimensions
- Require less training time
- Provide performance insight to each DBMS component

The framework was integrated into the NoisePage [4] DBMS and the results indicated that the models could support both OLTP and OLAP workloads with minimal loss of accuracy and outperform a deep learning-based external modelling approach [8].

Meanwhile, the limitations of this framework are listed as follows [8]:

- Does not support disk-oriented systems

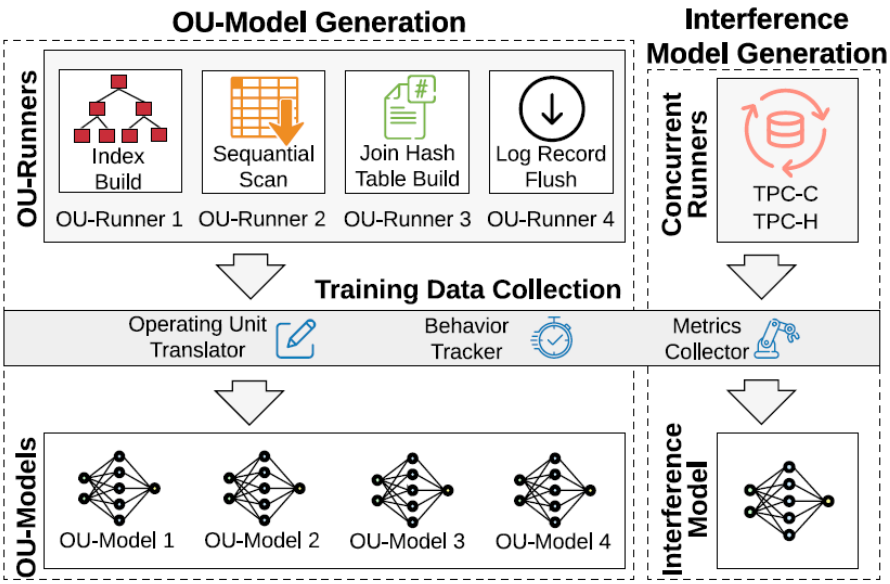


Figure 4: Structure of MB2 [8]

- Difficult to predict ad-hoc queries it has not seen before
- Hard to model OLTP queries due to high variances in short query execution times
- Doesn't model transaction aborts due to difficulties in predicting overlapping queries.
- No inclusion of specific hardware and environment features for different hardware

2.3.3 Summary

Compared to other approaches, it is highlighted that MB2 adopts the principle of "divide and conquer", which partitions the entire system into small components and hugely improves the pre-training time as well as the system's efficiency. Meanwhile, it is notable that DBMind has a good performance in OLTP environments.

2.4 Physical Design Structure

Tuning the physical design structure (PDS) of databases helps to optimize the way to store and organise data, which could significantly increase the system's efficiency. This section will focus on several approaches to PSD tuning in self-driving databases, with an emphasis on index selection.

2.4.1 DBMind Index Selection[10]

Kossmann and Schlosser [11] are among the pioneers who have delved into this topic and presented a conceptual framework for self-driven database systems that focuses on automating tasks typically performed by database administrators. Their research puts a special emphasis on index selection, proposing an automated method that uses machine learning to identify the indexes that are best suited to improve performance, which include access tracking, cost modelling and benefit estimation, workload modelling and prediction, as well as tuning algorithms. They introduce an LP approach to determine the order of multiple tuning features based on the pairwise dependencies between different options. The pa-

per ”DBMind: A Self-Driving Platform in OpenGauss” discusses the implementation of a self-driving database system that uses machine learning to automate database administration tasks, including index selection [10], based on OpenGauss, an open-source relational database management system (RDBMS) developed by Huawei, which is originally based on PostgreSQL and has been enhanced with additional features to meet the needs of enterprise-level applications. The paper ”Self-driving Database Systems: A Conceptual Approach” also discusses the importance of index selection in self-managing database systems [11]. Both papers highlight the complexity of the selection process and the trade-offs between finding optimized solutions, achieving low computation times for the optimization, and the robustness of the chosen candidates.

2.4.2 DBA Bandits [12]

Perera et al. proposed a novel approach that performs online index selection for database systems under unpredictable ad-hoc analytical workloads [12]. They delved into the field of reinforcement learning and implemented multi-armed bandit (MAB) learning to deal with this problem, which is a form of the Markov decision process [12]. Compared to traditional methods based on possibly inaccurate cost models which were pre-trained offline, MABs learn from real-time performance observations and have regret bounds to ensure the suitability of dynamically proposed indices [12]. The trade-off between exploring untried actions and exploiting actions that maximize the observed rewards is balanced [12]. The key contributions of this approach are listed as follows:

- Multi-armed bandit leads to practical and competitive solutions.
- Provides a worst-case safety guarantee against any optimal fixed policy using the C^2UCB bandit for regret analysis.
- Outperforms the state-of-art commercial tool by having up to 75% speed-ups under dynamic, analytical workloads.

2.4.3 HMAB [13]

HMAB is an advanced version of MAB proposed by Perera et al., which is the application of contextual combinatorial multi-armed bandits (MABs) to tackle the complexities of optimising multiple physical design structures at once [13]. It is a hierarchical structure for MABs that comprises two layers [13]:

- Layer 1: responsible for candidate physical design structure (PDS) selection
- Layer 2: selects the final configuration considering all the candidate structures together

HMAB combines the advantages of integrated and iterative search approaches for PDS tuning and is capable of handling large action spaces and supporting parallel features [13]. The key contributions of this approach are listed as follows [13]:

- The first online learned approach to tune multiple PDS in an integrated search space
- Introducing a new bandit version that builds a hierarchy of bandits to handle large action spaces
- Combining optimiser knowledge and execution statistics (58% reduction in PDS creation time)
- Up to 96% speed-up over a state-of-the-art commercial PS design tool under dynamic workloads
- Proving HMAB’s superiority over nine other index-tuning solutions

2.4.4 Storage Structure Selection[14]

Researchers from the Harbin Institute of Technology introduced an automatic machine-learning algorithm for selecting the storage structure of the database system. Wang et al. addressed the importance of selecting the optimal storage structure in the self-tuning process and split the algorithm-picking process into three stages: Pruning, generating candidates and evaluation. To be more precise, the pruning process will collect performance data based on the current workload and then the model will generate candidates by traversing search space and enumerating solutions based on the current workload, followed by evaluating and applying the optimal storage plan through comparisons by costs [14]. In order to get high-quality performance data, system-level features that can affect the query cost like runtime features and workload features should be selected [14]. Researchers emphasize that three criteria of cost function should be met: comparability, compatibility, and workload-oriented. The proposed system has proven to perform efficient storage structure selection in various circumstances.

2.4.5 Explain-Tun [15]

Ouared et al. proposed a new and comprehensible AI model called Explain-Tun in selecting the physical structure (PS) of the database system. It is noteworthy that the self-tuning process is a hard task in reality due to the complexity of the database system in all aspects and it is important to understand the motivation of these tuning decisions. As a result, Providing explanations of tuning decisions can help human experts to optimize and refine the machine learning models [15]. This model is designed based on Decision-Tree (DT) and Random-Forest Algorithms (RF). The Explain-Tun model consists of three phases. Firstly, The Explain-Tun model will utilise machine-learning techniques to extract useful information from historical data and then it will analyse the data by exploring various input parameters, which is followed by editing and refining the existing machine learning models [15]. The report then evaluates the performance of the DT and RF model in PS selection according to the metrics of F-measure and accuracy. The final result shows the reliability of the Explain-Tun of PS prediction and provides suggestive advice on how to achieve query optimization by changing PS of the database.

2.4.6 Summary

Online index tuning strategies, such as HMAB, appear to offer huge improvements over offline methods, which makes this a promising and innovative area for further exploration. Nonetheless, if workloads are relatively consistent and stable, the enhancement might not be as obvious as stated. The selection of methods should still adhere to the specific system design requirements. For example, if a system normally has a stable workload and requires a fast response time, offline methods might become a more suitable option.

In addition, recent research has put more emphasis on using various machine learning algorithms to optimize the physical structure, which has also demonstrated remarkable performance outcomes.

2.5 Other Optimisations

2.5.1 External & Internal [5]

Pavlo et al. [5] provide an in-depth analysis of external and internal ML agents used for DBMS tuning. The machine learning agents can be divided into two types, external and internal.

External agents have advantages such as the ability to use DBMS's existing APIs and infrastructure. However, external agents are facing three challenges, unavailability during configuration changes, limitations in performance metrics collection, and the necessity for human knowledge to set certain knobs.

Internal agents integrate directly within the DBMS which supports one or more agents running inside. The internal agents are advantageous in more exposed information and more low-level operations but it is hard to capture the dependencies between existing components.

2.5.2 Replicated Training [9]

Machine learning has been a crucial part of the self-driving database system. An ML agent can predict the runtime behaviour of DBMSs and automate tuning system configurations. To train a machine learning agent for a self-driving database system, a vast amount of data is required. However, the data collected by the DBMS is often biased and might cause the model to overfit toward one configuration. Pavlo and Andersen [9] proposed a method called Replicated Training to balance the overhead of training data generation across the entire distributed system.

Replicated Training is a technique that uses replica nodes to collect the metrics and allows for the offloading of metrics collection to replica nodes. By using replica nodes, a self-driving DBMS can avoid the performance overhead of metrics collection on the master node. Pavlo and Andersen[9] implemented replicated training in NoisePage and shows potential in the use of real-world cases.

2.5.3 Column Compression [16]

Fehér et al. introduce a concept of adaptive segment encoders and evaluates the performance with other segment compressors. Compression methods play a crucial role in meeting the demand for storing and analysing large datasets, as they enhance storage capacity and reduce operational costs for in-memory databases [16]. However, decompression costs should also be taken into consideration as it requires query processing and impacts the query performance. Column-oriented databases are more suitable for compression tasks than traditional row stores mainly because values within the same column typically share the same type [16].

Generalized deduplication is a compression framework that consists of two stages: applying transformation functions to convert input value into a base-and-deviation pair followed by base deduplication and determination of base index[16]. This widely-used approach enhances the retrieval of the original value. Four types of GD segments are shown in the reports with distinct ways of storing deviation and base index. Considering the query performance including two operations: random access and table scans among them, GD segment 3 and 4 outperform the other two segments due to their unique logic of deviation reconstruction [16]. Another factor that affects query performance is the choice of deviation size, which is suggested to be determined automatically from a range of possible values. This can be managed properly by a self-driving system which could also select the optimal configuration or encoder to be applied based on the evaluating usage patterns for each segment and performance measurements

[16]. Specifically, a self-driving system that aims to achieve automated maintenance and optimisation tasks can perform an estimation on whether re-encoding is efficient to discover the best strategy.[16] However, the best choice may be derived from arbitrary manual selection depending on how much they want to trade speed with compression cost.

2.5.4 TScout [6]

Butrovich et al. demonstrate an efficient and accurate way of extracting training data set which can be utilized further by machine learning models to achieve automation of the database management systems. Self-driving database systems utilize the forecasting system and behaviour model to predict future workloads of the application and its runtime behaviour respectively, while the planning system selects action to improve its performance [8]. Then two methods to collect metrics from the database system are shown as user-space and kernel-space. The former uses basic code-based functions to retrieve metrics while the latter runs an ancillary program inside the OS kernel [6]. Although the kernel-based approach is more efficient than the user-space method, it shows some potential problems in the long-term run. This is followed by a discussion of offline and online training data. Online training data is more accurate in predicting system workloads and query performance than offline training data as it could reflect changes in the current workload of the database and the system itself while offline [6]. They also introduce a new training data collection framework called TScout which could enhance the recording and processing of OU-granular training data from the multi-threaded database system. To be more specific, TScout utilises kernel-level programs to capture metrics of systems like CPU, memory etc. and generate training data for machine-learning models [6]. Deployment occurs in two stages [6]:

- Setup Phase for annotating DBMS source code with markers
- Runtime Phase for extracting markers and collecting training data

Finally, the evaluation of the TScout framework shows the reliability in collecting better training data which leads to better model performance.

3 Discussions and Future Directions

Table 2 summarises the main findings of this survey.

Table 2: Summary of main findings [4]

Approach	Focus	Methodology
QB5000	Workload prediction	Ensemble learning and kernel regression
DBMind	Behaviour modelling and PDS tuning	LSTM, LP
MB2	Behaviour modelling	OU-units, ML
DBA	Online PDS tuning	Multi-armed bandit
HMAB	Online PDS tuning	Hierarchical multi-armed bandits
Explain-Tun	PDS tuning	Decision tree and random forest
Column compression	Storage optimization	Generalized deduplication
TScout	Query optimization	ML, kernel-level programs

Self-driving databases as the latest technology to automate database management systems is still in the process of development. Future directions mainly involve the integration of AI technology, performance optimisation and other aspects like extending more complex database features. We aim to

research deeper into how to utilise machine learning models and AI to better understand workload patterns, predict future workload and optimize resource allocation accordingly. Moreover, automatic tuning configurations, indexing strategies and query plans are also important aspects to achieve the automation and functionality of self-driving systems. Moving on to improvements on performance optimisation, works including exploring automatic tuning decisions, storage schemes selection and predictive analysis offerings will be discovered. In the future, the database system should not only adapt dynamically to current workloads or other cost models but also provide predictive analysis and insights to assist technicians in making decisions. Besides, we would also continue to explore more into other variations to optimisation techniques like replicated training and plan to implement them in the future.

4 Conclusions

In conclusion, as the complexity of DBMS increases, human database administrators get pushed to their limit and Manual maintenance and tuning become time-consuming, even impossible. Therefore, The self-driving database system is essential to replace traditionally labour-intensive and expertise-required tasks. During our survey, we explored 5 aspects of self-driving database management systems: architecture, workload prediction, behaviour modelling, physical design tuning and optimisations. Three main architectures are explored, including. NoisePage is an improved version of Peloton with the latest workload prediction and behaviour modelling methods. Then we further dig into the architectures, in most of the architectures, Workload prediction and Behaviour modelling are the most important elements. The latest workload prediction system is QB5000, which offers the best overall performance so far. In terms of behaviour modelling, MB2 is adapted to the latest system and divide it into small components, which has improved pre-trained time. Furthermore, Physical design tuning is key for system efficiency. Online index tuning offers more improvements than offline strategies but the consistency and stability of the workload should still be considered. Finally, we explored four optimisation methods for different aspects, including the type of machine learning agents, metrics collection, and compression training data extraction. In the future, research direction could focus on the integration of AI and ML technology, and performance optimisation to achieve a more efficient and autonomous database management system.

5 Group Reflection

We mainly relied on Google Scholar and the University of Melbourne's library to search for relevant references. Firstly, we considered sources that were peer-reviewed and published in top conferences, which guarantees the reliability of our survey. Some recent materials may still be unpublished, and ArXiv is also a valuable platform for accessing such resources. Furthermore, we created a shared Google Drive for efficient resource management and scheduled Zoom meetings for regular communication.

In our initial selection phase, we mainly looked at the abstract and introduction of each potential paper, which narrow down to the most relevant sixteen research papers. Then, each team member was assigned approximately four papers to study thoroughly and required to summarize the main findings, contributions and potential limitations of our respective papers. Finally, we exchanged ideas and made comparisons between different approaches found within the literature.

Reflecting upon this process, our team greatly appreciates the balance of individual responsibility

and collaborative dialogue. The strategies and tools we employed streamlined our survey, ensuring that we efficiently and effectively made the most of a broad range of academic resources. If we are to conduct another survey, the approach may align with the previous methodology. Given the time constraints of this project, it may not be feasible to delve into deeper analysis. If time allows, it would be beneficial to explore the referenced materials in the selected papers to establish a more comprehensive understanding of this field.

6 Individual Reflection

6.1 Yuan Cao - 1067709

I am responsible for the following reference: [6], [14], [15], [16]. The most representative paper among these four reports would be "An Adaptive Column Compression Family for Self-Driving Databases". This report provides a comprehensive introduction to a new adaptive compression framework called generalized deduplication which aims to optimise storage usage and enhance query performance. With the dramatically increasing demand for storage space, data compression is essential for large-scale data storage and can be applied in many areas, like cloud computing. In a data set, a column usually contains a similar data type and repeated value, which makes the column compression method preferable to row-based compression. Compared with the traditional compression strategy which could not achieve memory usage reduction and query speed at the same time, this report discussed various adaptive encoding compressors with a balancing trade-off point of these two aspects. Besides, the author also provides us with an insight into how to achieve automation of the database system. This is implemented by actions like automated deviation size selection and self-encoding tuning both based on query performance. We can learn the methodology of optimizing both data storage optimization and query performance and also further extend the concept into other real-life applications.

6.2 Jiahao Chen - 1118749

I am responsible for the following references: [4], [8], [12], [13]. The paper "Make Your Database System Dream of Electric Sheep: Towards Self-Driving Operation" [4] proposed by Pavlo et al. serves as an optimal selection if only one is allowed, which introduces the NoisePage architecture. The researchers from Carnegie Mellon University are pioneers in this area, who also proposed the first conceptual architecture of self-driving databases in 2017 (Peloton) [1]. This work provides a comprehensive explanation of the overall structure of a fully-autonomous DBMS, together with corresponding design principles. For novices in the field, this is an excellent foundational resource to help understand the basic components and mechanisms of self-driving databases. Moreover, it references numerous related works, assisting readers in broadening their knowledge in the area. For example, it integrates ModelBot2 [8] for behaviour modelling. In contrast, other references focus on specific components in this area, such as physical structure tuning, which are more valuable for in-depth research. It is beneficial to first have a fundamental understanding and broad overview of the field before diving into these specific aspects.

6.3 Zhiquan Lai - 1118797

I have endeavoured to early resources on self-driving DBMS [1], [2], [3], [5], [9]. If only a single reference is allowed, it would be ”Self-driving database management systems”[1]. This paper is chosen for its coverage of the motivation for building self-driving DBMS, the challenges it needs to address, and the architecture of Peloton, a pioneering architecture for self-driving DBMS.

The motivation, as Pavlo et al[1]. argued, is the need for self-driving DBMS arises from the limitations of existing tools which are labour-intensive and unable to keep up with the big data era. Then challenges that self-driving DBMS needs to solve are stated, understanding an application's workload, forecasting resource utilisation trends, and identifying and implementing optimal actions at the right time. The paper also presents the architecture of Peloton which is a foundational architecture in this area. This earliest architecture provides valuable insights for later research and serves as a basis for subsequent systems like NoisePage

With the exploration of these three aspects, this paper can offer readers a solid understanding of the basics of self-driving databases.

6.4 Mingyang Liu - 1113531

I am responsible for the following references: [7], [10], [11]. Jan Kossmann's [11] work give me a basic view of how self-driving database work and some general approaches, and propose a component-based framework that enables database integration and the development of self-managing functionality with low overhead by relying on the separation of concerns. By keeping the components of the framework reusable and exchangeable, experiments are simplified, promoting further research in this area. Also, I investigated more deeper in the workload prediction section. After investigating DBMind [10] and QueryBot5000 [7]for workload prediction, I was intrigued by the innovative use of machine learning techniques in predicting future workload patterns, a key aspect of optimising database management. I appreciated the unique approach of both models, with the QB5000 excelling in adaptability and accuracy, and DBMind being robust in handling complex workloads. Furthermore, since DBMind is based on OpenGauss, an enterprise-level database used by Huawei, I learnt more about the application of self-driving database points in today's enterprises and why it is used Understanding the complexity of these algorithms was challenging but rewarding, highlighting the need for continuous learning as the field rapidly evolves. This experience enriched my understanding of workload prediction in self-driven databases and strengthened my belief in the potential of machine learning to shape the future of database management systems.

References

[1] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. C. Mowry, M. Perron, I. Quah, *et al.*, “Self-driving database management systems.,” in *CIDR*, vol. 4, p. 1, 2017.

[2] L. Ma, D. Van Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon, “Query-based workload forecasting for self-driving database management systems,” in *Proceedings of the 2018 International Conference on Management of Data*, pp. 631–645, 2018.

[3] J. Kossmann, “Self-driving: From general purpose to specialized dbmss.,” in *PhD@ VLDB*, 2018.

[4] A. Pavlo, M. Butrovich, L. Ma, P. Menon, W. S. Lim, D. Van Aken, and W. Zhang, "Make your database system dream of electric sheep: towards self-driving operation," *Proceedings of the VLDB Endowment*, vol. 14, no. 12, pp. 3211–3221, 2021.

[5] A. Pavlo, M. Butrovich, A. Joshi, L. Ma, P. Menon, D. Van Aken, L. Lee, and R. Salakhutdinov, "External vs. internal: an essay on machine learning agents for autonomous database management systems," *IEEE bulletin*, vol. 42, no. 2, 2019.

[6] M. Butrovich, W. S. Lim, L. Ma, J. Rollinson, W. Zhang, Y. Xia, and A. Pavlo, "Tastes great! less filling! high performance and accurate training data collection for self-driving database management systems," in *Proceedings of the 2022 International Conference on Management of Data*, pp. 617–630, 2022.

[7] L. Ma, *Self-Driving Database Management Systems: Forecasting, Modeling, and Planning*. PhD thesis, Carnegie Mellon University, 2021.

[8] L. Ma, W. Zhang, J. Jiao, W. Wang, M. Butrovich, W. S. Lim, P. Menon, and A. Pavlo, "Mb2: decomposed behavior modeling for self-driving database management systems," in *Proceedings of the 2021 International Conference on Management of Data*, pp. 1248–1261, 2021.

[9] G. E. A. Mezerhane, *Replicated Training in Self-Driving Database Management Systems*. PhD thesis, Carnegie Mellon University Pittsburgh, PA, 2019.

[10] X. Zhou, L. Jin, J. Sun, X. Zhao, X. Yu, J. Feng, S. Li, T. Wang, K. Li, and L. Liu, "Dbmind: A self-driving platform in opengauss," *Proceedings of the VLDB Endowment*, vol. 14, no. 12, pp. 2743–2746, 2021.

[11] J. Kossmann and R. Schlosser, "Self-driving database systems: a conceptual approach," *Distributed and Parallel Databases*, vol. 38, pp. 795–817, 2020.

[12] R. M. Perera, B. Oetomo, B. I. Rubinstein, and R. Borovica-Gajic, "Dba bandits: Self-driving index tuning under ad-hoc, analytical workloads with safety guarantees," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 600–611, IEEE, 2021.

[13] R. M. Perera, B. Oetomo, B. I. Rubinstein, and R. Borovica-Gajic, "Hmab: self-driving hierarchy of bandits for integrated physical database design tuning," *Proceedings of the VLDB Endowment*, vol. 16, no. 2, pp. 216–229, 2022.

[14] H. Wang, Y. Wei, and H. Yan, "Automatic single table storage structure selection for hybrid workload," *Knowledge and Information Systems*, pp. 1–27, 2023.

[15] A. Ouared, M. Amrani, and P.-Y. Schobbens, "Explainable ai for dba: Bridging the dba's experience and machine learning in tuning database systems," *Concurrency and Computation: Practice and Experience*, p. e7698, 2023.

[16] M. Fehér, D. E. Lucani, and I. Chatzigeorgiou, "An adaptive column compression family for self-driving databases," *arXiv preprint arXiv:2209.02334*, 2022.