

# MVGPT: Generative Materialized View Forecasting

Yue Han

Tsinghua University, China  
han-y19@mails.tsinghua.edu.cn

Guoliang Li

Tsinghua University, China  
liguoliang@tsinghua.edu.cn

Wenchun Xu

Alibaba Group, China  
wenchun.xwc@alibaba-inc.com

Xianglei Ran

Alibaba Group, China  
ranxianglei.rxl@alibaba-inc.com

Zeya Gong

Alibaba Group, China  
zeya.gzy@alibaba-inc.com

Wei Guo

Alibaba Group, China  
lengchuan.gw@alibaba-inc.com

Guang Qiu

Alibaba Group, China  
guang.qiug@alibaba-inc.com

Bo Zheng

Alibaba Group, China  
bozheng@alibaba-inc.com

**Abstract**—Existing materialized view (MV) selection methods rely on historical queries to select MVs. However, queries are dynamically and continually changing, and it is ineffective to generate MVs based only on historical queries. To address this limitation, we propose a novel MV forecasting framework, MVGPT, which utilizes large language models (LLMs) to predict effective MVs for evolving query workloads. First, we fine-tune an LLM for MV forecasting, retrieve relevant events to incorporate real-time event knowledge to instruct the query workload evolution inference. We then propose a predicate-level MV model based on the Bayesian network to generate valid and effective MVs. Finally, we propose a variable set representation method as the interactive interface between the LLM and the MV model to address the risk that the LLM generates incorrect results. We have implemented our system MVGPT and deployed it into Alibaba’s advertising analysis platform, and experimental results on real datasets show that our method outperforms existing state-of-the-art approaches.

**Index Terms**—database, autonomous materialized view, large language model

## I. INTRODUCTION

In Database Management Systems (DBMS), analytical query workloads often contain complex but repetitive sub-queries, leading to redundant computation and slow queries. The materialized view (MV) is a commonly used technique in DBMS to avoid these redundant computations, which pre-computes an intermediate result derived from a frequent sub-query for answering subsequent queries. MVs will be refreshed when the source table changes, to ensure the result is up-to-date. In this way, users can utilize MVs instead of the original large table, achieving faster query processing.

Existing MV generation methods select MVs from historical query workloads according to varied strategies, including rule-based and learning-based techniques [1], [3], [9], [10], [12], [13], [16]–[18], [21], [22], [24], [34]–[36], [39], [40], [56]. Specifically, given a query workload, they select a set of MVs from common subqueries in the query workload and

Guoliang Li is the corresponding author. This work was supported by National Key R&D Program of China (2023YFB4503600), NSF of China (62525202, 62232009), Shenzhen Project (CJGJZD20230724093403007), Science and Technology Research and Development Plan of China Railway & National Engineering Research Center of System Technology for High-Speed Railway and Urban Rail Transit (L2024W001), Science and Technology Project of the East China Branch of State Grid (52992425001D), Zhongguancun Lab, and Huawei, and Beijing National Research Center for Information Science and Technology (BNRist).

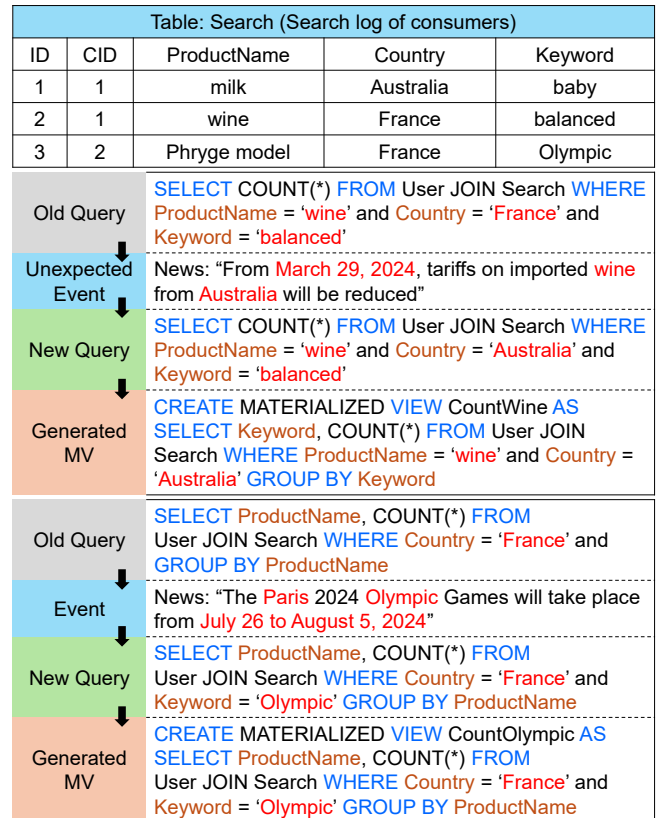


Fig. 1: Evolving query workload examples.

use these MVs to rewrite and answer queries to minimize the total latency of the query workload with an acceptable overhead of MVs. These methods are suitable for static or periodic query workloads where future queries highly overlap with past queries. However, analytical query workloads in real scenarios have two problems that challenge existing MV generation methods: (1) Query workload distribution often changes due to unexpected external events, called evolving query workloads. Traditional methods primarily rely on historical queries, which may implicitly capture certain event effects, but lack mechanisms to explicitly model and incorporate event knowledge. (2) New query statements, with new combinations of predicates and joins, do not appear in past queries. Existing MV generation methods that extract MVs from past subqueries

struggle to cover these new queries.

Evolving query workloads increasingly include new queries, making old MVs ineffective. For example, in e-commerce advertising analysis scenarios, like the data-management platform of Alibaba where advertisers conduct audience analysis queries for advertisement placement, we observe that the hit rate of old MVs drops by 30% after 30 days. As shown in Figure 1, the unexpected event of “reducing tariffs on wine from Australia” triggers more sales and new queries on Australian wines. Thus, creating corresponding MVs for these queries in advance will significantly improve the query performance. However, existing methods cannot anticipate the workload changes to generate MVs for unseen queries. Moreover, existing query workload forecasting methods also cannot predict the predicate-level new queries. Therefore, it is necessary to study the event-aware query workload forecasting and predicate-level MV generation.

**Our proposal.** To address the problem of unexpected events changing query workloads and introducing new query statements, we propose a generative end-to-end MV forecasting method. There are two key innovations. (1) Generative MV model. To address the new query statements problem, we decompose the query at the predicate level and model the conditional probability distribution (CPD) of predicates, aggregations, and group by clauses in historical queries. Then we recompose new MVs with new combinations of predicates. (2) Event-aware query workload change inference. To address the unexpected events problem, we explicitly extract and incorporate real-time event information to improve inference of query workload changes and generate MV seeds. As new events have various types and are mainly described in natural language, we use language models, especially large language models (LLMs) [4], [31], [47], [57] with stronger reasoning ability and the retrieval-augmented generation (RAG) mechanism [23], to infer how events change query workloads.

**Challenges.** There are four challenges of forecasting MVs in evolving workloads. **C1:** Evolutionary query workloads are hard to accurately predict based on historical queries because they are related to unexpected external events. **C2:** The causal relationship between events and high-utility MVs is difficult to estimate, and there is a lack of ground truth and training data. **C3:** LLMs suffer risks and unreliability when inferring query changes and generating results, *e.g.*, generate syntactically invalid SQL statements, lack awareness of the frequency of query predicates within workloads, and have high computational costs. **C4:** The diversity of generated MVs is hard to guarantee, leading to overlapping and low total utility.

To address these challenges, we propose an LLM-based materialized views forecasting framework **MVGPT** to actively generate MVs for future query workloads. For **C1**, we fine-tune an LLM with an event-query-MV dataset to learn the relationship between events, query workload change, and high-utility MVs (Section IV). For **C2**, we build a knowledge base to manage real-time events to improve query workload change inference and MV matching (Section IV-A). For **C3**, we learn a Bayesian network (BN) based on our MV variable set

representation to model the predicate conditional probability and quickly generate MVs with ensuring the SQL statement correctness and diversity (Section V). For **C4**, we propose a pipeline of using LLM to instruct the BN to effectively generate thousands of MVs for future queries (Section V-C).

**Contributions.** We make the following contributions.

- (1) We propose a generative-model-based framework for effectively forecasting future materialized views.
- (2) We design a query workload evolution-related event knowledge base and a relevant event retrieval method to improve long-term materialized view forecasting.
- (3) We design a prompt and fine-tuning method for LLMs on materialized views forecasting task.
- (4) We propose an expressive graph structure for materialized views’ features representation and high-utility materialized views generation method using the BN.
- (5) We have implemented and deployed our system into Alibaba, and evaluated our method in a real-world e-commerce advertising platform. Experimental results show that **MVGPT** significantly outperforms existing approaches.

## II. BACKGROUND AND RELATED WORK

We first introduce the *evolving workloads* and then discuss the necessity of MV forecasting and why existing MV selection methods cannot adapt to evolving query workloads.

### A. Evolving Query Workloads

Evolving query workloads refer to a scenario in database systems where query workloads continually evolve due to changing user demands, typically over days, months, or even years. Changes in query workloads cover aspects such as query predicates, aggregations, group by clauses, and join structures.

We introduce evolving query workloads with an example of advertising analysis, as shown in Figure 1. The database table “Search” contains consumers’ search history, including the product name, country of origin, and other keywords. Advertisers query this table to analyze consumers’ interests for targeted advertisement placements, *e.g.*, counting the number of people searching for their products. The queries usually change because of various events. First, we define the event:

*Definition 1 (Event):* An event is a pair of two elements, occurrence time and description, represented as  $(t, g)$ , extracted from news, websites, activities, and so on. The description contains not only the original text but also the source, region, and category information. The description is used to identify the influence scope of the event, whether and how the event will change the queries, and the time indicates when the event will change the queries. The set of events is denoted as  $(Date T, Knowledge G)$ .

We use two examples to illustrate how unexpected events lead to completely new queries and what the MVs look like. For instance, after the news “Tariffs on imported wine from Australia will be reduced from March 29, 2024” released on March 28, users suddenly submitted 300 new analytical queries on Australian wine products on April 1, as shown in Figure 2a. The example query is “SELECT COUNT(\*)

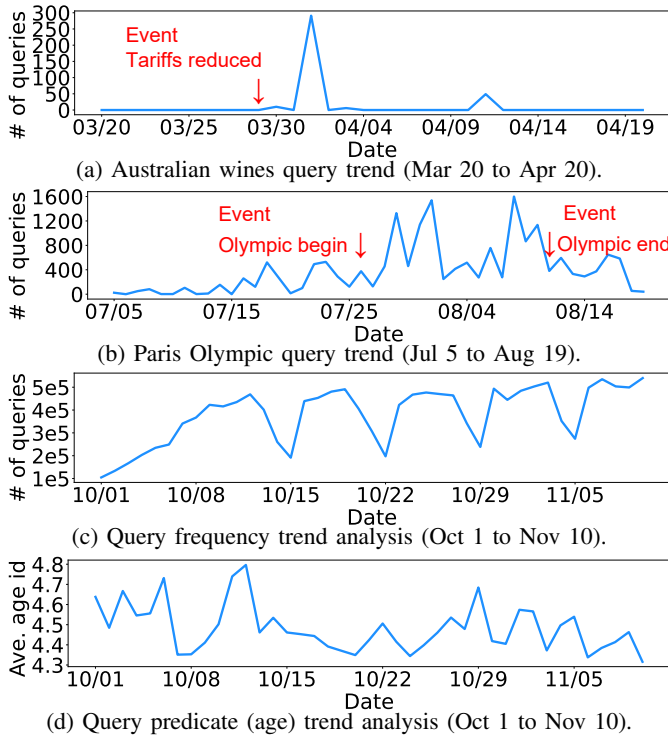


Fig. 2: Evolutionary query trend analysis

FROM User JOIN Search WHERE ProductName = ‘wine’ and Country = ‘Australia’ and Keyword = ‘balanced’” which aims to analyze how many consumer searches are about wines from Australia. Thus, if we collect events daily and extract the date and description information, we can create the corresponding MVs in advance, *e.g.*, “CREATE MATERIALIZED VIEW CountWine AS SELECT Keyword, COUNT(\*) FROM User JOIN Search WHERE ProductName = ‘wine’ and Country = ‘Australia’ GROUP BY Keyword”, to optimize the new queries. Another example is the worldwide famous event “The Paris 2024 Olympic Games” which takes place from July 26 to August 11, 2024, and more consumers search for Olympic souvenirs such as the Phryge model from France. Accordingly, advertisers query about the search frequency of Olympic products from France to place more advertisements on hot products. To verify this, we analyze the queries each day from July 5 to August 19, 2024, as shown in Figure 2b. We find that the number of queries on “Olympics” significantly increases during the Olympic Games. Therefore, creating MV of “CREATE MATERIALIZED VIEW CountOlympic AS SELECT ProductName, COUNT(\*) FROM User JOIN Search WHERE Country = ‘France’ and Keyword = ‘Olympic’ GROUP BY ProductName” will significantly optimize the queries. In addition to the examples above, many other events, such as thousands of product launches, product promotions, and festivals, lead to new queries and change the query distribution. Thus, it is necessary to collect the new events daily and learn how the events change the queries.

### B. Query Workload Trend Analysis

Capturing different query change characteristics is important for MV forecasting. Firstly, we define the query workload

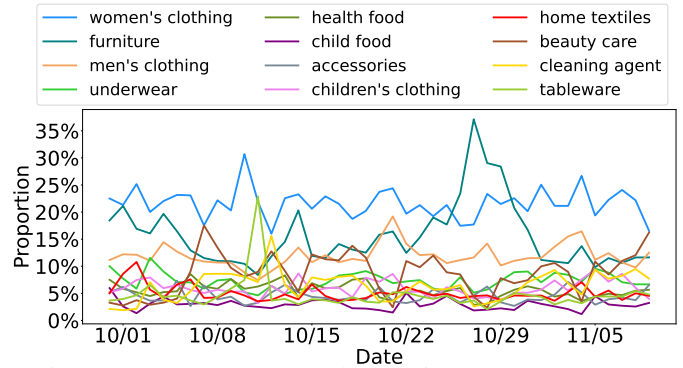


Fig. 3: Query category trend analysis (Oct 1 to Nov 10).

trend as the changes in query predicate, aggregations, group by clauses, and join structures occurrence frequency and conditional probability distribution (CPD). To clearly illustrate the evolutionary trends in query workloads, we visualize the workload changes from aspects of daily query frequencies, query predicates, and proportions of query categories. We use the audience analysis query workload of 40 days before a marketing event on November 11, 2023, for visualization. First, we observe a gradual increase in query frequency as shown in Figure 2c. This indicates that advertisers begin to make more queries to prepare for the marketing event. On the other hand, there is a periodic pattern in the query frequency: during weekends and holidays (the first seven days in the data). Second, Figure 2d shows a slightly gradual decrease in the consumer age predicate. This suggests that advertisers expect young people to participate more in marketing events.

Fourth, Figure 3 shows the proportion of several query categories. We observe that the proportion of furniture queries peaked on October 28, 2023, and subsequently declining. The proportion of beauty care queries gradually increased as the event approached, specifically rising during the five days leading up to the event. Meanwhile, queries for essential items such as underwear remained relatively stable. This indicates that the shifting interests of users change the proportion of different query categories within the query workload.

### C. Related Work

**MV generation methods.** MV generation aims to find the optimal MV set for a given query workload within a specified storage budget which is NP-hard [11]. This problem can be modeled as a 0-1 integer linear programming (ILP) problem [18], [53]. However, the complexity of the ILP solver approach is  $\mathcal{O}(2^n)$  for large workloads. Thus, various heuristic methods [1], [3], [9], [10], [16]–[18], [22], [40] including genetic algorithms [16], [17], reefs optimization [3], and particle swarm optimization [22], have been proposed. These heuristic methods, however, are based on assumptions about data and query distributions, limiting their generalizability to other workloads and datasets. To overcome this limitation, learning-based methods [13], [24], [56] have been proposed. **AutoView** [13] encodes features of queries and MVs and utilizes an RNN model [5] to estimate the benefits of MVs. **RLView** [56] uses reinforcement learning (RL) to select MVs.

TABLE I: Advantages of LLM/BN

Models	LLM	BN
Align with user intent	✓	✗
Long-term inference	✓	✗
SQL validity	✗	✓
Training costs	✗	✓
Parameter interpretability	✗	✓

**MV dynamic management.** The performance of MVs decreases in the dynamic query workload scenario because query patterns and data distributions shift over time. This necessitates the regular maintenance of MV sets, *i.e.*, create new ones and evict low-utility ones. Traditional approaches like **WATCHMAN** [39], **DynaMat** [21], and **HAWC** [34] manage MVs based on quality scores, but often suffer from inaccurate quality estimations, result in suboptimal solutions. **DQM** [24] uses an RL model trained on real runtime statistics to select and evict MVs, but its lack of query feature encoding leads to unstable score estimations.

**Query workload forecasting.** Query workload forecasting methods are usually used for query optimization, tuning, or database resource adjustment [28], [29], [54]. The researchers [15], [33] predict the next query using Markov models. QueryBot 5000 [25] predicts the arrival rate of the clusters’ queries by applying the logical composition method. However, the forecast queries are primarily based on historical queries without explicitly modeling external events, and it is hard to cover new queries triggered by certain events.

**Large language models (LLMs).** LLMs have strong abilities in understanding and generating text, thereby well handling tasks like question answering [4], [43], [48], [49], [58], NL2SQL [8], and code generation [31]. For the MV forecasting task, the model learns three tasks: SQL modeling, MV generation, and reasoning about how events change queries, which is hard for smaller models. Thus, we fine-tune LLMs for MV forecasting. However, LLMs have limitations as shown in Table I, including “hallucination”, high computation cost of training, and weak interpretability, as discussed in Section IV-F. The parameters of BN are the CPD tables among elements, which are easier to explain or manually edit.

**Bayesian networks (BN).** BN offer a powerful framework for knowledge representation and reasoning. The structure learning methods [20], [30] of BN include heuristic searches and model structure scoring methods to determine the edges and CPDs of the graph according to the training data. BN can be used to synthesize data, *e.g.*, population synthesis [42], [55]. However, as shown in Table I, BN are unable to capture event knowledge, leading to their inability to forecast future materialized views for specific events.

Existing MV generation methods struggle to generate high-utility MVs for evolving query workloads because they rely on historical data without considering external events influencing query evolution. Thus we need a novel approach for evolving query workloads via integrating event knowledge. To this end, we consider leveraging the strengths of both LLMs and BN.

### III. MVGPT FRAMEWORK

We first introduce the overall framework of **MVGPT**. As shown in Figure 4, **MVGPT** consists of three modules: generative MV model, event-aware query workload change inference, and MV generation. At a high level, the generative MV model decomposes MVs in the predicate space and learns the distribution of high-utility MVs’ predicates. With this model, we compose new high-utility MVs that can optimize future unseen queries. However, the model cannot perceive the future changes of queries caused by real-time events. Thus, the event-aware query workload change inference module collects new events daily relevant to the workload and analyzes the event information with an LLM to adjust the predicate distribution with prior conditions. Finally, based on the adjusted predicate distribution, the MV generation module selects predicates to generate MVs that are tailored to specific events and offer high utility. Next, we overview the design of the generative MV model and query workload change inference module.

#### A. Generative MV model

The main challenge of MV generation at the predicate level is how to find high-utility MVs in exponential combinations of predicates. To address this challenge, we propose the generative MV model to learn the high-utility predicates.

First, we decompose MV statements into units of query template, predicate, aggregation, and other clauses. An MV template is “CREATE MATERIALIZED VIEW <MV name> AS SELECT [aggregations, columns] FROM <tables> WHERE [predicates] GROUP BY [columns]” which covers most of the analytical queries. The predicate units are represented as triplets (identifier, operator, value). For example, the WHERE clause “WHERE ProductName IN (‘wine’, ‘milk’)” will be decomposed to “ProductName = ‘wine’” and “ProductName = ‘milk’”. Aggregations and group by clauses are represented as (agg, expression) and (group by, identifier) such as “COUNT \*” and “group by Keyword”. We also collect the utility and category of MVs as attribute units to help generate high-utility and category-specified MVs. For example, “hit rate = 0.8” indicates that the MV optimizes more queries. The full set of the units above is denoted as  $C$ .

High-utility MVs typically consist of predicate units that are commonly queried together, so our goal is to identify which subsets of these units frequently co-occur. We learn the CPDs between units from historical MVs using the BN. We first construct units in  $C$  into a DAG, where edges represent relationships between units. We then search the structure of the DAG and learn the parameters of CPDs to make the BN best describe the historical MVs. For example, because France is famous for its wine, users often query about French wine. Thus, its MVs have higher utility. Accordingly, the edge between “ProductName = ‘wine’” and “Country = ‘France’” will have a higher weight. Moreover, we also learn the CPDs between the “hit rate” unit and predicates to figure out what predicates are queried frequently.

With the well-learned generative MV model, we generate MVs by sampling and composing the units according to MV

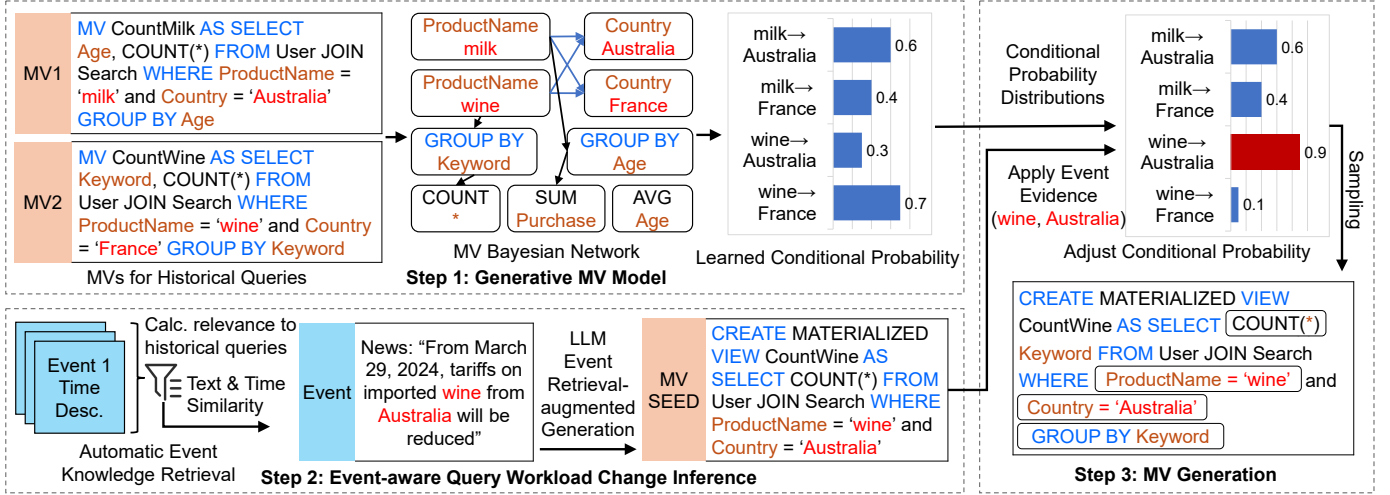


Fig. 4: Overall Framework of MVGPT.

templates. To improve the utility and diversity of generated MVs, we control the probability weight of the “hit rate” unit and predicates by giving specific evidence to the BN.

### B. Event-aware query change inference

In evolving query workloads, the probability distribution of predicates will be changed by unexpected and unseen events as discussed in Section II. Thus, the learned MV model from past data does not perform well for future query workloads. A straightforward method is to first predict future queries and then select MVs. However, existing forecasting methods are not designed for MV selection, so the predicted queries are inefficient for selecting MVs. Therefore, we incorporate real-time event information to directly adjust MVs’ predicate CPDs and control MV generation. The main challenge is finding the events that are the most related to the query workloads and identifying how events change the CPDs.

First, as there are many new events coming up daily, we use the language model to automatically select relevant events and extract information to store in the knowledge base. The event type includes news, festivals, holidays, and others. The event information is formed in date and description pairs (*Date T, Knowledge G*), e.g., (“March 29, 2024”, “Tariffs on imported wine...”) with text embedding indexes generated by a text embedding model. To filter out irrelevant events, we design a prompt to use LLM to estimate the similarity between the event description and the query workload context, which includes query values, source region, user information, and other characteristics depending on the workload. For example, a tariff adjustment announcement from the government of South Africa will not affect the sale in China and this event will be filtered out because the query source region does not match the region in the event description even though the query statements contain the keyword “wine”. Note that vector databases, used as the storage of the event knowledge base, are scalable and can easily store and index billions of entries [26], [51]. Instead, the scale of event knowledge base depends on the efficiency of event selection based on LLM.

Then, we design a retrieval algorithm to automatically search top-*k* relevant events with the scoring function based on the semantic similarity and temporal proximity to augment the generation of MVs. We feed the event description text as additional context into the LLM prompt to infer the query changes, e.g., “users will query more about Australian wine”. To enable the LLM to infer the query changes at the predicate level, we fine-tune the LLM with the domain-specific knowledge, i.e., query and MV’s SQL statements. With the fine-tuned model, we generate future sample MVs with input of the event information. We use these sample MVs as seeds to adjust the predicate CPDs of the BN model. Specifically, we extract predicate/agg/group by units from the MVs seeds as the evidence of the predicate graph in the MV generation phase to generate many more event-specified high-utility MVs.

## IV. QUERY EVOLUTION FORECASTING

This section introduces how to predict future query workloads with consideration of event influence. The challenges include: (1) How to detect and classify the varied evolutionary trend of query workloads. (2) How to model the influence of events on query changes. (3) LLMs are not pre-trained on the MV task. To address these challenges, we first build an event knowledge base related to the query workload based on multiple sources. We then define the pattern of query changes and induce the evolutionary trend. Next, we propose an event retrieval method based on semantic similarity and temporal proximity to determine the related events for queries. Finally, we fine-tune an LLM to learn the relationship between events and query changes from large training data and design a pipeline to predict future MV seeds.

### A. Event Knowledge Base Construction and Maintenance

In evolving query workloads, queries shift over time due to new analytical requirements arising from external events, resulting in queries that cannot be found in historical queries. It is essential to gather and analyze the correlation and influence of events related to queries to inform the forecasting of future MVs. Therefore, we propose an automatic construction

method including an event relevance analysis and description extraction method using LLMs. The event knowledge base is built by collecting event records from multiple sources, including news, activities, festivals, and domain-specific sources. We employ search engines to retrieve websites related to events, such as news. For example, festival events have a significant impact on tourism. For specific query workloads, domain-specific knowledge will be useful. For example, in the case of advertising analysis scenarios, promotions and marketing events strongly impact ad placement strategies. Moreover, news also has an impact on the query workload as discussed in Section II-A. However, most of these event sources are unstructured text, so we instruct LLMs to extract the event date and description from the source text to form event tuples  $(t, g)$ , which are then stored in the event knowledge base.

To ensure the quality, the event knowledge base is updated daily to incorporate new events, and outdated, irrelevant, or duplicate events are removed. We consider two aspects: relevance and hotness [60]. We find that the relevance of events to a query is not only related to the query statement itself but is also closely tied to the query’s context, *e.g.*, query source region and product category features. For instance, while summer clothing is popular in northern hemisphere countries, winter clothing is more favored in southern countries during the same period. Similarly, sports souvenir shops are influenced by sporting events more than temperature changes. Therefore, considering query contexts helps estimate events’ relevance. We sample the queries’ context information and statements to represent the query workload. We input the query workload information and event description into the LLM and output whether the event is relevant. To measure the hotness, we use the hot topic detection method [60], which is based on the TF-IDF [19] (term frequency and inverse document frequency) algorithm that is widely used in information retrieval to measure the importance of words in documents. Higher hotness means that the event has a greater impact. Finally, we filter out events that lack both hotness and query relevance.

### B. Related Event Matching

We incorporate event information to improve the inference of query workload changes, which requires an event matching algorithm to find the most related event in the event knowledge base for the target query, formally speaking, matching events  $(t_k, g)$  that are related to a given query  $q$  at time  $t_j$ . Thus, we rank events based on their semantic similarity and temporal proximity to the query. To estimate the semantic similarity, we use a text embedding model [32], [45] to transform the event description into embedding vectors in the semantic space. We utilize the Euclidean distance of embeddings to measure the similarity, where a smaller distance means higher similarity:

$$Dis(\mathbf{x}_g, \mathbf{x}_q) = \sqrt{\sum_{i=1}^d (x_{g_i} - x_{q_i})^2} \quad (1)$$

Where  $d$  is the number of dimensions of the embedding,  $\mathbf{x}_g$  is the event knowledge embedding, and  $\mathbf{x}_q$  is the query

embedding. Notice that event knowledge in natural language is significantly different from queries of SQL, we enrich SQL statements with their interpretation and context. Specifically, we first revert predicates in the query to its textual representation (*e.g.*, mapping “age = 2” to “age = ‘10~19 years old’”) and then utilize the LLM to interpret the query  $q$  into natural language, facilitating a more effective matching within the event knowledge base. For the temporal proximity function, we require it to decay with increasing temporal distance. Therefore, we utilize a normal distribution with the query time  $t_j$  as the mean and  $\sigma^2$  as the variance.

$$Sim(t_k, t_j) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(t_k - t_j)^2}{2\sigma^2}\right) \quad (2)$$

Observing that events significantly impacting queries are more likely to occur within a week before or after the query time, we set  $\sigma$  to 7 days. This choice results in the area under the curve of the temporal proximity function, within the interval  $(t_j - 7 \text{ days}, t_j + 7 \text{ days})$ , being 0.683. As the value of  $t_k$  ranges from  $t_j$  to  $t_j + 7$  days, the value of the temporal proximity function decays from 0.057 to 0.035. A higher value of the temporal proximity function indicates a greater likelihood of relevance between the query and the event.

Finally, we design a scoring function to estimate the likelihood of event knowledge influencing a query by combining semantic similarity and temporal proximity:  $Sim(t_k, t_j)/(Dis(\mathbf{x}_g, \mathbf{x}_q) + 1)$ . Based on this scoring function, we fetch top- $k$  event knowledge records for a given query as the potentially related events for later query change reasoning. We empirically found that setting  $k = 3$  provides a good balance between coverage and precision, which means fetching more events often introduces less relevant events without significantly improving the MV performance, while fetching fewer ones often misses the most relevant one.

### C. Query Evolutionary Trend Modeling

We model the evolutionary trends of queries by essentially modeling the CPDs within the sequence of queries, as expressed in the following formula.

$$P(W) = \prod_{j=1}^n P(q_j | q_1, q_2, \dots, q_{j-1}, g) \quad (3)$$

Where  $W$  denotes the query workload,  $q_j$  denotes the  $j$ th query in the workload,  $n$  denotes the total number of queries, and  $g$  denotes the event that influences the query evolution. However, given millions of queries, it is challenging to learn the aforementioned CPDs. Thus, we utilize the Markov assumption [27], approximating the CPDs of the entire sequence based on the distribution between two queries:  $P(q_j | q_i, g), \forall i \leq j$ . This means that the model learning through query pairs. An advantage is that it enables the model to perform temporal reasoning on different time scales (with timestamps attached to  $q_i$  and  $q_j$ ), whereas traditional methods mostly infer on a fixed time scale at each step. Further, considering that the number of MVs is significantly smaller

than the number of queries, we directly predict MVs because their CPDs are easier to learn:  $P(v_j | q_i, g), \forall i \leq j$ .

#### D. Training Data

We introduce how to construct high-quality training samples, which are crucial for fine-tuning performance. Training samples are tuples of (event, past query, future query, future MVs), indicating how the query changes under the impact of an event and what MVs can effectively optimize the future queries. Two examples are shown in Figure 1, including MVs for the Australian wine and French Olympic souvenir queries. Samples are selected by three principles: (1) predicates of future query and MVs should relate to the event; (2) the past query and future query have similar join structure with changed predicates; (3) the future MVs can effectively optimize the future query. We build training samples in three steps: (1) sample query pairs (past query, future query); (2) retrieve query-pair-related events; (3) match effective MVs from the MV dataset. We then introduce the details of these steps and methods to improve the data quality.

(1) Query pairs sampling. We sample query pairs from the workload, forming pairs of past query  $(t_i, q_i, f)$  and future query  $(t_j, q_j, f)$ , where  $t$  denotes the timestamp,  $q$  denotes the SQL, and  $f$  denotes the query context features. However, randomly sampling often yields irrelevant pairs. To address this, we propose clustering queries, assuming that query changes usually occur within similar query groups. First, we extract query features such as query templates, query fields, join structures, predicates, aggregation fields, timestamp, user ID, and user category. These features are grouped into evolution and clustering types. We cluster queries by query templates, join structure, and user category, and then sample query pairs in each cluster where queries have higher correlation.

(2) Query-pair-related event retrieval. We search the knowledge base for events that drive the query change. However, the query change is implicitly indicated by the difference between the two queries. To improve the retrieval result, we identify and encode the differing fields and predicates between  $q_i$  and  $q_j$  as the query change features, and then use the event retrieval method to obtain event knowledge records  $g$  related to  $q_j$  and the query change feature.

(3) MV matching. The quality of “ground truth” future MVs is very important for the training samples. The mapped MV should be optimal for the past query and event. To this end, we first build a high-quality MV dataset from the query workload using the MV selection method, which will be introduced in Section V-A. Next, because the future query  $q_j$  is the potential changed query of  $q_i$ , we match MVs in the MV dataset that can optimize  $q_j$  and estimate the corresponding benefit. We then enumerate combinations of matched MVs and select ones that offer the maximum benefit to  $q_j$ , denoted as  $v_j$ . To improve the efficiency of matching a large MV dataset, we create indices on tables and fields to prune irrelevant MVs.

Following the steps of query pair sampling, event knowledge retrieval, and MV matching, we obtain the query pairs  $(t_i, q_i, f)$  and  $(t_j, q_j, f)$ , event knowledge  $(t_k, g)$ , and MVs

$(t_j, v_j, f)$ , where  $t_i < t_j \approx t_k$ . We form training samples as  $(g, q_i, q_j, v_j)$ , where the past query  $q_i$  changes to future query  $q_j$  under impact of the event  $g$ , and can be optimized by MVs  $v_j$ . We use these tuples as data to train LLMs for evolutionary trend forecasting and MV seed generation.

#### E. Event Retrieval-Augmented Generation

To address the challenge that LLMs are not pre-trained with the complex objectives of evolutionary trend forecasting, we proposed to fine-tune the LLM on our evolutionary trend dataset with a specially designed prompt.

Given the training dataset, we design the prompt to instruct the LLM to predict evolutionary trends and generate MV seeds according to the query and event knowledge. The prompt is the input of the LLM, consisting of three parts: instructions, contexts, and the response. In the instructions part, we outline the task objectives, describe the data format, and provide guidance on the steps for forecasting query evolutionary trends. In the context part, we offer an introduction to MV’s concept to assist the LLM’s understanding, along with future event knowledge. In the response part, we present the solving process and target MV seeds. The prompt design is as follows:

**Instructions:** Predict the query evolutionary trend and generate future MV seeds given the query  $q_{\text{past}}$  and the future event knowledge  $g$ . Follow the step-by-step inference: (1) Understand user intent. Explain  $q_{\text{past}}$  into natural language descriptions, deduce the underlying user intent driving it, and classify it into a specific query category. (2) Integrate event knowledge. Incorporate event knowledge  $g$  to assess how it influences  $q_{\text{past}}$ . (3) Predict the evolutionary trend of queries. Predict the evolved query  $q_{\text{future}}$  of  $q_{\text{past}}$  considering the integrated event knowledge  $g$  and the inferred user intent. Then suggest potential MV seeds  $v_{\text{future}}$  for  $q_{\text{future}}$ .

**Contexts:** (1) MV’s concept. (2) Event knowledge:  $\langle g \rangle$ . (3) The past query from the query pair:  $\langle q_{\text{past}} \rangle$ .

**Response:** (1)  $\langle \text{The explanation of } q_{\text{past}} \rangle$  and  $\langle \text{the query category} \rangle$ . (2) The event knowledge  $\langle g \rangle$  relevant to the query. Under the influence of the event,  $q_{\text{past}}$  evolves into  $\langle q_{\text{future}} \rangle$ . (3) To optimize  $q_{\text{future}}$ , we suggest MV seeds  $\langle v_{\text{future}} \rangle$ .

**Fine-tuning and evaluation.** We use a supervised fine-tuning approach that is to incrementally train a pre-trained LLM on the tasks of forecasting evolutionary trends. Fine-tuning the large model is necessary to internalize the knowledge of query evolutionary trends and MVs into model parameters. We design an MV-task criterion for the generated results, and it differs from those of classical Q&A tasks and query workload forecasting tasks. The most important evaluation metric for MVs is the query optimization effect. Thus, we test the hit rate of the MV seeds as the criterion. The model parameters update via back propagation [37] based on this criterion signal.

**RAG during inference.** Given a past query, we retrieve related events, which are semantically similar to the query and close in time to the present, from the event knowledge database. We also retrieve similar historical queries and MVs as examples. These retrieved records are concatenated with the context

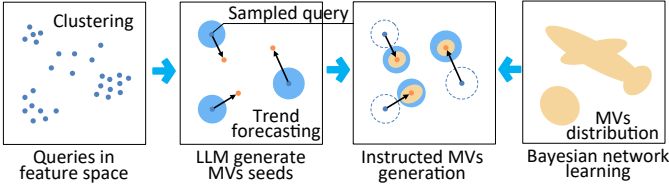


Fig. 5: Evolving trend forecasting and MVs generation.

in the prompt for LLM inference, in line with RAG. This improves the query change inference and MV quality.

#### F. Limitations and Risks of LLMs

While LLMs enable our system to incorporate external event knowledge and reason about unseen queries, they also introduce well-known challenges regarding controllability and reliability. These risks include: (1) generation of syntactically invalid or semantically incorrect SQL statements, (2) hallucination of events or predicates not grounded in the query workload, and (3) high computational costs that limit frequent retraining and deployment. These issues could reduce the practicality of our approach in production systems.

We analyze these risks and propose the following solutions. First, we do not directly execute the MVs generated by LLMs, because syntactically invalid SQL will lead to errors in the database, which is unacceptable in production systems. Instead, we constrain the role of the LLM to high-level reasoning and seed generation, and propose a BN model to explicitly formalize the MV generation (introduced in Section V), avoiding SQL execution errors. Second, LLM "hallucinating" and linking irrelevant events will lead to useless MVs that bring additional maintenance costs while providing no benefits. To address this, we verify the optimization effect of MVs online and remove useless ones from the database and training set to mitigate their impact on performance. Third, while the high computational cost of LLM inference limits the MV seeds generation speed, the BN can perform multiple inferences quickly for each MV seed, supporting scenarios that require a large number of MVs. We acknowledge that these risks are not yet fully solved and require broader effort, but our design choices (LLM only for reasoning + BN for controlled MV generation) offer a practical and reliable pathway.

#### V. MV GENERATION USING BN

The MV seeds cannot be used for query optimization directly due to the LLMs' limitations, while the BN can synthesize data efficiently and its parameters are interpretable and modifiable. Therefore, we propose the pipeline of two steps: forecasting evolutionary trends (via LLMs) and generating high-utility MVs (via BN). As shown in Figure 5, we use LLMs to predict the trends of representative queries selected from the query workload, thereby generating MV seeds, and use BN to model the probability distribution of predicates and generate high-utility MVs based on the MV seeds. The main challenges are (1) how to represent MVs in a graph structure of BN; (2) hard to learn the CPDs of a large number of predicates. Next, we introduce our method to generate varied, high-utility, and syntactically correct MVs.

MV1: ... COUNT(\*) FROM user WHERE age IN (1,2,3) and crowd IN (343, 596)  
 MV2: ... COUNT(\*), age FROM user WHERE age IN (2, 3) GROUP BY age

	hit rate	cate	time_seg	tmpl	age 1	age 2	age 3	crowd 343	crowd 596	COUNT *	GROUP BY age
MV1	0.4	2	'6-11'	2	1	1	1	1	1	1	0
MV2	0.6	4	'12-17'	4	0	1	1	0	0	1	1

MV hit rate & features                      MV statement template & conditions

Fig. 6: Variable set representation of MVs.

#### A. MV Data Generation

We first generate ground truths of high-utility MVs. Given a query workload  $Q$ , we parse the queries into join structures and predicates. For each query, we enumerate subsets of the predicates to form subqueries and generate MV candidates [13]. Since the number of predicate combinations grows exponentially, we reduce the complexity by grouping the predicates by tables. We propose a utility function to measure the value of MV candidates, using the hit rate of MV candidates to queries as a measurement. Given the query workload within the time interval  $[t_1, t_{|Q}]$ , we simulate the query execution process of the query workload and evaluate the hit rate of each MV. To ensure a fairer comparison, we normalize MV hit rates by the length of their life cycle. Moreover, we multiplied the utility of MVs by a time-decaying factor to emphasize the urgency of optimizing recent queries.

#### B. MVs' Graph Representation

We introduce the variable set representation for MVs, as shown in Figure 6. We first decompose the MV statements into templates, predicates, aggregations, and group by clauses. Templates are indexed by the variable  $var_{tmpl} \in [1, |templates|]$ . The basic MV template is "CREATE MATERIALIZED VIEW <MV name> AS SELECT [aggregations, columns] FROM <tables> WHERE [predicates] GROUP BY [columns]". We also enumerate the clauses to generate other templates. Next, we represent predicates as tuples. For equation-type predicates, we convert them into triplets (identifier, operator, value). For example, "price > 10" will be represented as  $var_{(price, >, 10)} \in \{0, 1\}$ . Predicates containing IN clauses are split into multiple equations. For example, "age IN (2,3)" is split into  $var_{(age, =, 2)}$  and  $var_{(age, =, 3)}$ . Moreover, aggregations and "group by" clauses are also represented as variables to indicate their existence. The value of  $\{0, 1\}$  indicates the absence or presence of the predicate or clause. We also use variables to represent the features of MVs, such as hit rate, category, and the time segment are denoted as  $var_p \in \mathbb{R}$ ,  $var_{ctg} \in [1, |categories|]$ , and  $var_{time\_seg} \in \{0-5, "6-11", "12-17", "18-23\}$ , respectively. Finally, we build the variable set  $C$ .

**Variable pruning.** Converting all predicates into BN variables leads to an unmanageable number, e.g., 2 million queries yield 800,000 distinct variables, greatly increasing the BN learning cost. However, state-of-the-art learning methods support only up to hundreds of thousands of variables [38], [59]. Fortunately, we find that a small set of high-frequency variables makes the most contribution, which means we can cover most

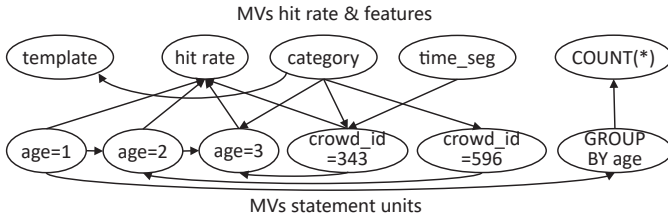


Fig. 7: The DAG structure of BN.

queries with a smaller variable set. Thus, we evaluate the performance of MVs generated on top-k frequent variables (Section VI-C), finding that about the top 1/40 (20,000) variables can optimize about 50% of queries. Therefore, we select these high-frequency variables in practice to balance computation cost and performance. This pruning inevitably leaves a portion of queries that are not optimized. But it will not lose much benefit because the overhead of creating MVs to optimize extremely rare queries outweighs their benefits. We note this as a scalability limitation of BN structure learning. To overcome this, future directions include (1) hierarchical BN structures that partition variables by table or predicate type, (2) replacing BN with other scalable graphical models. Next, we introduce how to build the graph of variables and train BN.

### C. BN Learning & MV Generation

We transform high-utility MVs in the MV dataset into sets of variables as training samples of the BN. When training [20], [30] the BN, we have two primary objectives: (1) Model the CPDs of features and query predicates of MVs. (2) Generate high-utility, diverse and valid MVs. The BN is a DAG where nodes indicate variables and edges indicate the relationship. For example, as shown in Figure 7, if the predicates “age = 1” and “age = 2” often co-occur, they will have higher conditional probability  $P(\text{var}_{(age,=,2)} \mid \text{var}_{(age,=,1)})$ .

First, we identify the most effective network structure that describes the dataset, including edge connections and node CPDs. Popular structure learning methods [20], [30] include PC algorithm [46], hill-climbing, Chow-Liu Algorithm [6], exhaustive search, and model structure scoring methods. We chose the PC algorithm, which gradually removes edges from a complete graph via conditional independence tests, for its stability and efficiency. Although exact structure learning is NP-hard, the PC algorithm is asymptotically consistent under assumptions of causal sufficiency (all common causes are measured) and faithfulness (conditional independencies in the distribution reflect the BN graph). Thus, with infinite data, it recovers the true graph. However, with finite data, errors in conditional independence tests can lead to suboptimal graphs. Its worst-case time complexity is exponential, but we find that causal relationships of predicates in the MV dataset are sparse, reducing the runtime to polynomial. We further reduce computational cost by two heuristics: (1) constraining the maximum degree of each node to prevent overly complex CPD tables and ensure the sparsity; (2) pre-screening correlations to remove edges between variables with weak correlation, thereby reducing the candidate edge set.

TABLE II: Attributes of datasets

Datasets	# of queries	# of tables	Duration
<b>Ad-summer</b>	2,926,599	690	3 months (Jun~Aug)
<b>Ad-festival</b>	3,235,692	690	3 months (Sep~Nov)
<b>XuetangX</b>	35,749	14	2 months (Aug~Oct)

TABLE III: Example of Query and MV

Query example	SELECT COUNT(*) FROM User JOIN Search WHERE keyword IN ('wine', 'imported', 'Australia') and period <= 30
MV example	CREATE MATERIALIZED VIEW CountWine AS SELECT COUNT(*), channel_id FROM User JOIN Search WHERE keyword IN ('wine', 'imported', 'Australia') and period <= 30 GROUP BY channel_id

With the learned BN, we sample the predicate variables to compose MV statements with given predicate evidence extracted from MV seeds generated by LLM based on events. We also give high-value evidence for the hit rate variable to control the MV utility. We then traverse the graph along the edges, sampling variable values according to their CPDs and previously sampled variables. Once all variable values are sampled, we construct the MV statement with templates. Since the CPDs are learned from the training data, the generated MVs follow the same distribution. When sampling the network multiple times, the generated MVs tend to overlap with each other, which contributes less to the total utility. Thus, to increase the MV diversity, we enumerate the values of the category, time\_seg, and template variables as evidences during sampling to ensure the MVs cover more queries.

**MV Selection & Creation.** MV creation is scheduled daily during idle periods (mostly at night) to minimize impact on database performance. Although BN can quickly generate many MV candidates, not all of them are materialized because of MVs’ creation and maintenance cost that depends on data volume, update frequency, and MV refresh strategy. Thus, we select MVs by estimated benefit versus cost using existing dynamic management methods [13], which creates MVs whose benefit (hit rate  $\times$  MV latency - cost) exceeds a given threshold. We omit its details because MV cost estimation and selection are not the core contribution of this paper. Moreover, the MV refresh strategy also influences the maintenance cost. Using a daily scheduled MV refresh strategy for batch analytical queries makes the cost more controllable.

## VI. EXPERIMENT

We conduct extensive experiments to evaluate the effectiveness of the **MVGPT**. First, we evaluate the overall performance of the generated MVs on the evolving query workload. Next, we examine the contributions of each component of **MVGPT** with ablation studies and case studies. We also evaluate the impact of hyper-parameter choices.

### A. Experimental Settings

**Dataset.** We use three real-world datasets, including two advertising analysis datasets, **Ad-summer** (Advertising-summer) and **Ad-festival** (Advertising-festival), and a massive open online courses (MOOC) platform query dataset **XuetangX** [41],

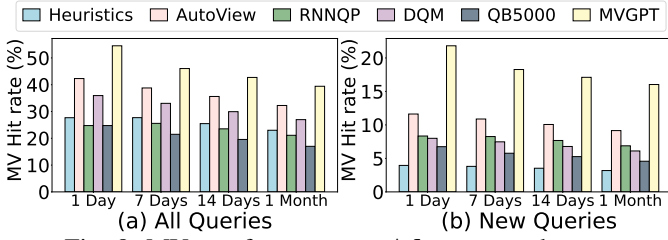


Fig. 8: MVs performance on **Ad-summer** dataset.

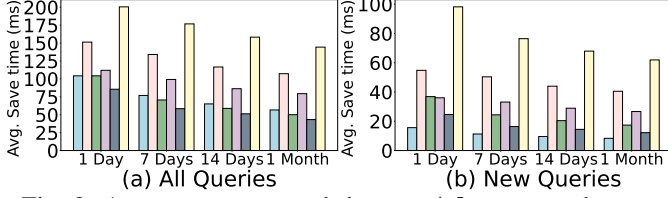


Fig. 9: Average query saved time on **Ad-summer** dataset.

[50]. Queries in these datasets are naturally assigned timestamps. **Ad-summer** and **Ad-festival** are from the scenario of the advertising data management platform [2], [52] (DMP) of Alibaba, where we implement and deploy our system. On this platform, advertisers use target user behavior to segment audiences and deliver personalized advertisements, thereby improving click-through conversion rates. The system handles over 500 QPS, with daily requests reaching the million level, and is deployed on cloud servers. We automatically recommend MVs daily to optimize user query latency to cope with query load peaks such as the Double Eleven Shopping Festival. As shown in Table II, **Ad-summer** and **Ad-festival** contain 690 tables with 2,926,599 and 3,235,692 queries respectively. Table III shows an example of a query and MV. We focus on this basic MV structure rather than complex forms like nested subqueries. Additionally, both datasets include the “User ID” and “User category” features per query. Since data changes have a limited impact on MV performance, we emphasize query changes and omit data updates. Moreover, this paper focuses on MV generation, while the MV maintenance and discarding which can be handled by existing dynamic management methods [12], [34]. To build the event knowledge base for the advertising, we collect the news, season, festival, and marketing events, including 4 seasons, 24 solar terms, 158 festivals, 6 promotional events, and 5,428 news items. The **XuetangX** dataset captures student activities such as course searches, comments, and homework. It contains 14 tables and 35,749 slow queries (latency > 1s) spanning two months. We collect course events for the knowledge base.

**Environment.** We use a machine with an Intel(R) Xeon(R) 6242R CPU @ 3.10GHz and 256GB RAM. For fine-tuning the LLM, we use 16 NVIDIA A100 GPUs, each with 80GB RAM. The LLM we selected is LLaMA [44]. We use the last month of the workload for testing and the rest for training.

**Evaluation metrics.** We evaluate the performance by the utility of generated MVs. We first use each method to generate the same amount of MVs, and then compare their hit rates and saved execution time on the test set. Assuming the test set size is  $|Q|$ , and the number of queries that can be optimized (hit)

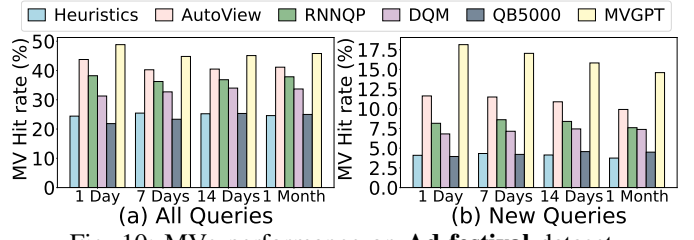


Fig. 10: MVs performance on **Ad-festival** dataset.

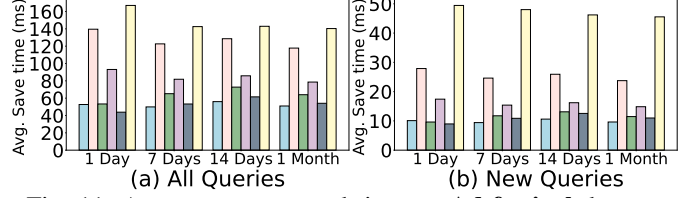


Fig. 11: Average query saved time on **Ad-festival** dataset.

by the generated MVs is  $|Q_{hit}|$ , we calculate the hit rate of the MVs as  $|Q_{hit}|/|Q|$ , where  $|Q| > 0$ . The saved execution time is the difference between the query and the optimized query. We averaged the time savings to avoid the effect of the number of queries:  $\sum_{q \in Q} (time_q - time_{q'})/|Q|$ .

### B. MVs Performance Evaluation

We first evaluate the performance of MVs generated by **MVGPT** on real-world workloads. Given the training query workload, we generate 1,000 MVs and examine their hit rate on the testing workload. We then evaluate **MVGPT**’s ability to respond to external events, adjust the MV generation, and perform long-term forecasting. Evaluations are conducted across different time scales (from 1 day to 1 month). Additionally, to verify the model’s performance on new queries, we also measure the hit rate and reduced execution time of MVs on new queries appearing only in the test set.

**Baselines.** We compare **MVGPT** with the following baselines.

- (1) **Heuristics**: It is a greedy algorithm that extracts and sorts the MV candidates by their hit rate in descending order and prioritizes those with higher hit rates.
- (2) **AutoView** [13]: It is an MV selection method that uses DL to estimate MV benefits and uses RL to select MVs.
- (3) **RNNQP**: It first uses a GRU [7] model to predict future queries. We use it to predict a number of future queries and then generate MVs by selecting frequent subqueries.
- (4) **DQM** [24]: It is an RL-based method that learns MV scores from database runtime statistics and updates the MV scores after each query execution.
- (5) **QB5000** [25]: It predicts the arrival rate of different types of queries in the future by applying the logical composition method on historical queries. We then select MVs from the predicted query workload.

After training the model on the training set, we evaluate the hit and saved execution time of generated MVs on the testing query set across different time intervals, as shown in Figure 8, 9, 10, 11, 12, 13. First, on evolutionary query workloads, most methods show declining MV effectiveness over time, while **MVGPT** maintains high performance by

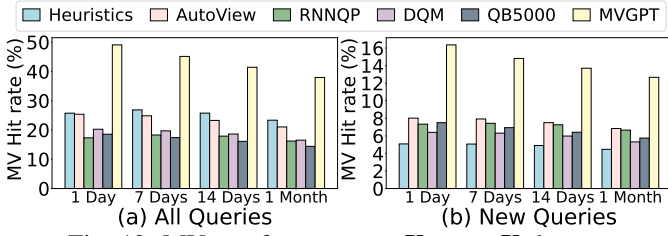


Fig. 12: MVs performance on **XuetangX** dataset.

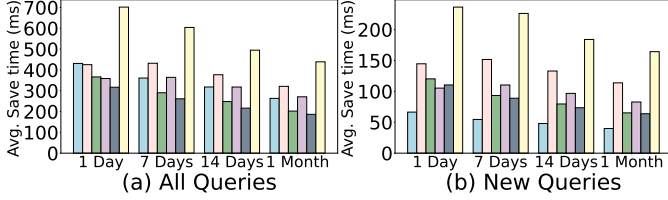


Fig. 13: Average query saved time on **XuetangX** dataset.

leveraging future event knowledge to pre-generate useful MVs. This highlights its capability for long-term MV prediction. We then analyze the model’s ability to respond to external events. The hit rate of **Heuristics** drops several times due to limited MV diversity and poor adaptability to query changes. In contrast, **MVGPT** incorporates recent events into MV generation, enabling event-targeted responses. Next, we assess the performance on new queries in the query workload, which often come from user manual input rather than scheduled scripts. Optimizing latency for these queries is critical for user experience. **Heuristics**, **AutoView**, and **RNNQP** achieve lower hit rates, as they primarily select or predict MVs based on historical workloads, resulting in a lower generalization capability. In contrast, by deriving MVs from predicate CPDs, **MVGPT** achieves better optimization of new queries.

As shown in Figure 8, 9, on dataset **Ad-summer**, **MVGPT** achieves a hit rate 364% higher than **Heuristics**. **Heuristics** tends to select MVs from periodic query workloads, exhibiting high stability but weaker generalization capability. In contrast, **MVGPT** generalizes well because the BN learns the CPDs of varied MVs, thus generating more diverse MVs.

**AutoView** and **DQM** both use RL to select MVs. **AutoView** learns the MV benefit offline while **DQM** learns online. As shown in Figure 10, on dataset **Ad-festival**, **MVGPT** achieves a hit rate 48% higher than **AutoView** and 96% higher than **DQM** in the evaluation of optimizing new queries. This is because **AutoView** selects MVs from historical queries, but new queries in evolving workloads rarely match them. **DQM** updates MVs scores at runtime, which causes a delay when the query changes. As shown in Figure 11, **DQM** has higher improvement than **RNNQP** on execution time, although **DQM**’s hit rate is lower. This is because **DQM** and **AutoView** learn the MV utility from the actual execution time. However, learning-based methods are less efficient because they use deep neural networks to estimate MV benefits and use RL to select MVs, both of which are computationally slow. Similarly, **MVGPT** has relatively high hardware requirements.

**RNNQP** and **QB5000** are query workload forecasting methods, with the former predicting at the predicate level and the

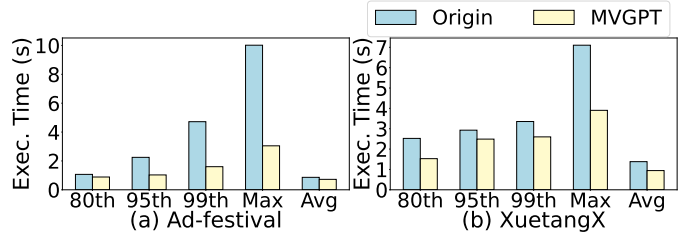


Fig. 14: Query execution time improvements.

TABLE IV: Training and Inference Overhead

Methods	Training time	Inference time/MV
<b>AutoView</b>	1.37 hrs	65 ms
<b>RNNQP</b>	1 hrs	33 ms
<b>DQM</b>	-	599 ms
<b>QB5000</b>	0.05 hrs	80 ms
<b>MVGPT</b>	8.8 hrs	30 ms

latter at the query level. As shown in Figure 12, 13, on dataset **XuetangX**, **MVGPT** achieves a hit rate 134% higher than **RNNQP** and 157% higher than **QB5000**. **RNNQP** performs worse because long-term RNN predictions accumulate errors, as each step depends on the previous query. The prediction error in query workload prediction reduces the performance of MV generation. **QB5000** captures implicit workload trends but fails to cover new queries. Moreover, because query workload forecasting methods are not designed for MV selection, most predicted queries are unrelated to MVs.

We evaluate the improvement on queries of different execution time. As shown in Figure 14a, on **Ad-festival**, MVs reduce the average, 95th, 99th, and max execution time by 16.16%, 54.17%, 66.07%, and 69.59% respectively. As shown in Figure 14b, on **XuetangX**, MVs reduce that by 31.74%, 14.98%, 22.50%, and 45.04% respectively. It shows that MVs are more effective in optimizing slow queries.

### C. Resource Efficiency

Next, we compare the computational costs of **MVGPT** with other learning-based methods in fine-tuning and inference, as shown in Table IV. The results show that **MVGPT** has relatively low inference time but high fine-tuning time. The MV inference time of the BN is 30 ms/MV, that is 2.17x faster than **AutoView**, which uses heavier neural networks, and 20x faster than **DQM**, which uses RL to learn online. For fine-tuning the LLM, we utilize distributed-data-parallel (DDP) training methods with 16 GPUs, taking about 8 hours for 3 epochs on 54K training samples. During inference for MV seeds, the LLM averages about 10 seconds per MV. The BN takes 0.8 hours to train on the high-utility MV dataset, but can generate 10,000 MVs within 5 minutes. Thus, LLMs alone cannot efficiently produce large numbers of MVs, necessitating efficient models like BN for MV generation. On the other hand, when changing to workloads of different domains, it needs corresponding query feature engineering and incremental learning of the BN. We will continually explore applying deep learning methods more efficiently.

**Size of BN.** In Section V, we mention that the number of variables in BN affects both performance and training time.

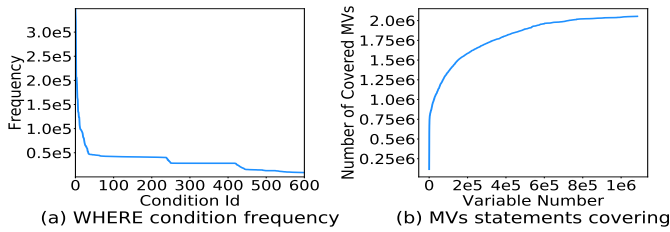


Fig. 15: MV generation analysis for variable selection.

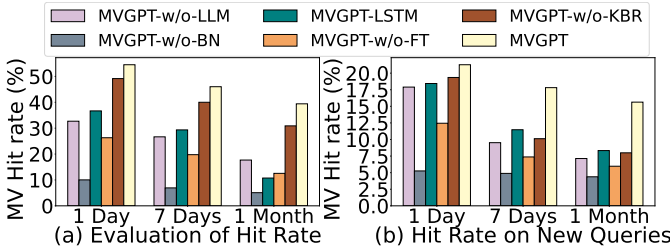


Fig. 16: MVs performance contribution of components.

Thus, it is essential to choose an appropriate number of variables to balance both aspects. To this end, we evaluate the performance and efficiency under different numbers of variables. As shown in Figure 15, the frequency of variables shows a long-tail distribution, which means that a few frequent variables contribute most to the query optimization. Beyond a threshold, adding more variables yields diminishing gains. Therefore, we select the top 20,000 frequent variables that can cover approximately 50% of the queries.

#### D. Ablation Study

To evaluate the individual contributions of the LLM, BN, and event knowledge base, we separately remove each component and conduct evaluations of MVs’ hit rates on the **Adsummer** dataset, as shown in Figure 16. Baselines:

- (1) **MVGPT-w/o-LLM**: It removes the LLM, and directly generates MVs from the BN’s CPDs learned from past MVs. In other words, it does not use the LLM to infer the event and adjust the CPDs of BN so that it cannot perceive the event.
- (2) **MVGPT-w/o-BN**: It removes the BN, and directly uses the MVs seeds output of the LLM as the final MVs instead of generating from the BN. However, as the LLM cannot ensure output correctness, we remove invalid MV output (about 5%).
- (3) **MVGPT-LSTM**: It replaces the LLM with Long Short-Term Memory (LSTM) [14] which is another widely used neural network architecture of language models.
- (4) **MVGPT-w/o-FT**: It directly uses the pre-trained LLM (no fine-tuning) to infer events and generate MV seeds.
- (5) **MVGPT-w/o-KBR**: It removes the event knowledge base retrieval algorithm to verify retrieval’s contributions. Because it is too large to input all event descriptions into the model, we randomly select a batch of events.

**Evaluate LLM.** As shown in Figure 16b, the hit rate of **MVGPT-w/o-LLM** stays high on the first day but drops quickly at 7 days (from 17.90% to 9.52%) because its CPDs are learned from historical data and cannot adapt to workload evolution. Replacing the LLM with an LSTM, *i.e.*, **MVGPT-**

**LSTM**, yields a 20.48% higher hit rate than **MVGPT-w/o-LLM** on new queries. However, with fewer parameters, it performs worse than LLM, especially on long-term event inference (46.71% lower at 1 month). Moreover, as shown in Figure 16a, inaccurate event inference by LSTM reduces the hit rate on the entire workload (39.29% lower at 1 month).

**Evaluate BN.** As shown in Figure 16a, we observe that the hit rate of **MVGPT-w/o-BN** decreases to 5%. This is because the LLM lacks the perception of the high-frequent query predicate distribution, making it hard to generate high-utility MVs.

**Evaluate fine-tuning.** As shown in Figure 16a, the hit rate of **MVGPT-w/o-FT** is 51.75% lower than **MVGPT** and 19.54% lower than **MVGPT-w/o-LLM** on day 1. This is because LLMs lack domain-specific knowledge, including predicate values and schema information in the database, resulting in generating invalid MV seeds. For example, in the database, “WHERE age IN (1,2)” means the age of consumers is in the intervals of 10~19 and 20~29 years old. However, the LLM without domain-specific knowledge generates “WHERE age BETWEEN 10 AND 25” which cannot be utilized.

**Evaluate event knowledge base.** As shown in Figure 16b, the hit rate of **MVGPT-w/o-KBR** is 48.89% lower than **MVGPT**. It shows that event retrieval has a higher influence on new queries because new events lead to more new queries. The event retrieval method can effectively extract relevant information from daily collected new events.

#### E. Case Study

We count the number of used MVs and event facts out of 1000 queries to verify how many MVs and events make the key contribution. For 1000 MVs, 127 events are retrieved, including 1 season, 2 solar terms, 19 festivals, and 105 news events. This shows that news events play an important role in query change. 564 out of 1000 queries are optimized, where the top 20 MVs account for 194 hits, indicating a long-tail distribution. Specifically, the top 3 MVs are “CREATE ... WHERE cate1\_id = ‘\*\*\*’ and channel\_id = ‘1000’” (23 uses); “CREATE ... WHERE gender = ‘0’” (17 uses); “CREATE ... WHERE cate1\_id = ‘\*\*\*’ and ds <= ‘20230924’ and ds >= ‘20230627’ and shop\_id = ‘\*\*\*’ and trd\_cnt > 0” (14 uses). For example, on April 1, 2024, 291 queries newly appear due to the news on March 28 “Tariffs on imported wine from Australia will be reduced from March 29, 2024”. These queries match our generated MVs instead of the baseline methods.

## VII. CONCLUSIONS

We have studied the problem of MV forecasting on evolving workloads and proposed the **MVGPT** framework. We presented a fine-tuning method for LLMs in the context of query evolutionary trend forecasting tasks. We proposed an event knowledge retrieval method and modeled the events’ impact on future queries. We proposed to use a BN to efficiently generate valid and high-utility MVs. We introduced a joint inference pipeline that integrates the strengths of LLMs and the BN. Experimental results on real datasets showed that our method significantly outperformed state-of-the-art approaches.

## VIII. AI-GENERATED CONTENT ACKNOWLEDGMENT

This article uses large language models to refine grammar, improving the clarity and readability of certain sentences.

## REFERENCES

- [1] R. Ahmed, R. G. Bello, A. Witkowski, and P. Kumar. Automated generation of materialized views in oracle. *Proc. VLDB Endow.*, 13(12):3046–3058, 2020.
- [2] Alibaba DMP. <https://dmp.alimama.com/>, 2007.
- [3] H. Azgomi and M. K. Sohrabi. A novel coral reefs optimization algorithm for materialized view selection in data warehouse environments. *Appl. Intell.*, 49(11):3965–3989, 2019.
- [4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.
- [5] K. Cho, B. van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *ACL*, pages 1724–1734, 2014.
- [6] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Trans. Inf. Theory*, 14(3):462–467, 1968.
- [7] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.
- [8] J. Fan, Z. Gu, S. Zhang, Y. Zhang, Z. Chen, L. Cao, G. Li, S. Madden, X. Du, and N. Tang. Combining small language models and large language models for zero-shot NL2SQL. *Proc. VLDB Endow.*, 17(11):2750–2763, 2024.
- [9] A. Gosain and K. Sachdeva. Handling constraints using penalty functions in materialized view selection. *Int. J. Nat. Comput. Res.*, 8(2):1–17, 2019.
- [10] H. Gupta. Selection of views to materialize in a data warehouse. In F. N. Afrati and P. G. Kolaitis, editors, *ICDT*, volume 1186 of *Lecture Notes in Computer Science*, pages 98–112. Springer, 1997.
- [11] H. Gupta and I. S. Mumick. Selection of views to materialize under a maintenance cost constraint. In C. Beeri and P. Buneman, editors, *ICDT 1999*, volume 1540, pages 453–470. Springer, 1999.
- [12] Y. Han, C. Chai, J. Liu, G. Li, C. Wei, and C. Zhan. Dynamic materialized view management using graph neural network. In *ICDE*, 2022.
- [13] Y. Han, G. Li, H. Yuan, and J. Sun. An autonomous materialized view management system with deep reinforcement learning. In *ICDE*, pages 2159–2164. IEEE, 2021.
- [14] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [15] M. Holze and N. Ritter. Autonomic databases: Detection of workload shifts with n-gram-models. In P. Atzeni, A. Caplinskas, and H. Jaakkola, editors, *Advances in Databases and Information Systems*, pages 127–142. Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [16] J. Horng, Y. Chang, and B. Liu. Applying evolutionary algorithms to materialized view selection in a data warehouse. *Soft Comput.*, 7(8):574–581, 2003.
- [17] J.-T. Horng, Y.-J. Chang, B.-J. Liu, and C.-Y. Kao. Materialized view selection using genetic algorithms in a data warehouse system. In *Computation-CEC99*, volume 3, pages 2221–2227. IEEE, 1999.
- [18] A. Jindal, K. Karanasos, S. Rao, and H. Patel. Selecting subexpressions to materialize at datacenter scale. *PVLDB*, 11(7):800–812, 2018.
- [19] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *J. Documentation*, 60(5):493–502, 2004.
- [20] D. Koller and N. Friedman. Probabilistic graphical models: Principles and techniques - adaptive computation and machine learning. 2009.
- [21] Y. Kotidis and N. Roussopoulos. Dynamat: A dynamic view management system for data warehouses. In A. Delis, C. Faloutsos, and S. Ghandeharizadeh, editors, *SIGMOD 1999*, pages 371–382. ACM Press, 1999.
- [22] A. Kumar and T. V. V. Kumar. Materialized view selection using self-adaptive perturbation operator-based particle swarm optimization. *Int. J. Appl. Evol. Comput.*, 11(3):50–67, 2020.
- [23] P. S. H. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.
- [24] X. Liang, A. J. Elmore, and S. Krishnan. Opportunistic view materialization with deep reinforcement learning. *CoRR*, abs/1903.01363, 2019.
- [25] L. Ma, D. V. Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon. Query-based workload forecasting for self-driving database management systems. In G. Das, C. M. Jermaine, and P. A. Bernstein, editors, *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 631–645. ACM, 2018.
- [26] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(4):824–836, 2020.
- [27] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *ICLR*, 2013.
- [28] B. Mozafari, C. Curino, A. Jindal, and S. Madden. Performance and resource modeling in highly-concurrent OLTP workloads. In K. A. Ross, D. Srivastava, and D. Papadias, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 301–312. ACM, 2013.
- [29] D. Narayanan, E. Thereska, and A. Ailamaki. Continuous resource monitoring for self-predicting dbms. In *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 239–248, 2005.
- [30] R. E. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2003.
- [31] E. Nijkamp, H. Hayashi, C. Xiong, S. Savarese, and Y. Zhou. Codegen2: Lessons for training llms on programming and natural languages. *ICLR*, 2023.
- [32] OpenAI. <https://openai.com/blog/new-and-improved-embedding-model>, 2021.
- [33] A. Pavlo, E. P. C. Jones, and S. B. Zdonik. On predictive modeling for optimizing transaction execution in parallel OLTP systems. *Proc. VLDB Endow.*, 5(2):85–96, 2011.
- [34] L. L. Perez and C. M. Jermaine. History-aware query optimization with materialized intermediate views. In *IEEE*, pages 520–531. IEEE Computer Society, 2014.
- [35] T. Phan and W. Li. Dynamic materialization of query views for data warehouse workloads. In G. Alonso, J. A. Blakeley, and A. L. P. Chen, editors, *ICDE*, pages 436–445. IEEE Computer Society, 2008.
- [36] K. A. Ross, D. Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. In H. V. Jagadish and I. S. Mumick, editors, *SIGMOD*, pages 447–458. ACM Press, 1996.
- [37] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.
- [38] M. Scanagatta, A. Salmerón, and F. Stella. A survey on bayesian network structure learning from data. *Prog. Artif. Intell.*, 8(4):425–439, 2019.
- [39] P. Scheuermann, J. Shim, and R. Vingralek. WATCHMAN : A data warehouse intelligent cache manager. In *VLDB 1996*, pages 51–62. Morgan Kaufmann, 1996.
- [40] M. K. Sohrabi and V. Ghods. Materialized view selection for a data warehouse using frequent itemset mining. *J. Comput.*, 11(2):140–148, 2016.
- [41] J. Sun, J. Zhang, Z. Sun, G. Li, and N. Tang. Learned cardinality estimation: A design space exploration and A comparative evaluation. *Proc. VLDB Endow.*, 15(1):85–97, 2021.
- [42] L. Sun and A. Erath. A bayesian network approach for population synthesis. *Transportation Research Part C: Emerging Technologies*, 61:49–62, 2015.
- [43] Z. Sun, X. Zhou, G. Li, X. Yu, J. Feng, and Y. Zhang. R-bot: An llm-based query rewrite system. *Proc. VLDB Endow.*, 18(12):5031–5044, 2025.
- [44] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez,

- A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models, 2023.
- [45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.
- [46] T. Verma and J. Pearl. Equivalence and synthesis of causal models. In P. P. Bonissone, M. Henrion, L. N. Kanal, and J. F. Lemmer, editors, *UAI '90: Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence, MIT, Cambridge, MA, USA, July 27-29, 1990*, pages 255–270. Elsevier, 1990.
- [47] J. Wang and G. Li. Aop: Automated and interactive llm pipeline orchestration for answering complex queries. *CIDR*, 2025.
- [48] J. Wang, G. Li, and J. Feng. idatalake: An llm-powered analytics system on data lakes. *IEEE Data Eng. Bull.*, 49(1):57–69, 2025.
- [49] J. Wang, Y. Li, J. Wu, S. Xu, and G. Li. Unify: A system for unstructured data analytics. *Proc. VLDB Endow.*, 18(12):5287–5290, 2025.
- [50] XuetangX. <https://www.xuetangx.com/global/>.
- [51] N. Yadav, N. Monath, M. Zaheer, R. Fergus, and A. McCallum. Adaptive retrieval and scalable indexing for k-nn search with cross-encoders. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.
- [52] S. Yan, B. Ding, W. Guo, J. Zhou, Z. Wei, X. Jiang, and S. Xu. Flashp: An analytical pipeline for real-time forecasting of time-series relational data. *Proc. VLDB Endow.*, 14(5):721–729, 2021.
- [53] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *VLDB*, pages 136–145, 1997.
- [54] D. Y. Yoon, N. Niu, and B. Mozafari. Dbshero: A performance diagnostic tool for transactional databases. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD '16*, page 1599–1614, New York, NY, USA, 2016. Association for Computing Machinery.
- [55] J. Young, P. Graham, and R. Penny. Using bayesian networks to create synthetic data. *Journal of Official Statistics*, 25:549–567, 12 2009.
- [56] H. Yuan, G. Li, L. Feng, J. Sun, and Y. Han. Automatic view generation with deep learning and reinforcement learning. In *ICDE*, pages 1501–1512, 2020.
- [57] X. Z. J. W. G. L. Zhaoyan Sun, Jiayi Wang. Data agent: A holistic architecture for orchestrating data+ ai ecosystems. *IEEE Data Engineering Bulletin*, 2025.
- [58] X. Zhou, Z. Sun, and G. Li. DB-GPT: large language model meets database. *Data Sci. Eng.*, 9(1):102–111, 2024.
- [59] R. Zhu, A. Pfadler, Z. Wu, Y. Han, X. Yang, F. Ye, Z. Qian, J. Zhou, and B. Cui. Efficient and scalable structure learning for bayesian networks: Algorithms and applications. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, pages 2613–2624. IEEE, 2021.
- [60] Z. Zhu, J. Liang, D. Li, H. Yu, and G. Liu. Hot topic detection based on a refined TF-IDF algorithm. *IEEE Access*, 7:26996–27007, 2019.