VIDEX: A Disaggregated and Extensible Virtual Index for the Cloud and AI Era

Rong Kang
Shuai Wang
ByteDance Inc.
Beijing, China
kangrong.cn@bytedance.com
wangshuai.will@bytedance.com

Tieying Zhang*
ByteDance Inc.
San Jose, USA
tieying.zhang@bytedance.com

Xianghong Xu
Linhui Xu
Zhimin Liang
ByteDance Inc.
Beijing, China
xuxianghong@bytedance.com
xulinhui@bytedance.com
liangzhimin@bytedance.com

Lei Zhang
ByteDance Inc.
Chengdu, China
zhanglei.michael@bytedance.com

Rui Shi
ByteDance Inc.
Beijing, China
shirui@bytedance.com

Jianjun Chen
ByteDance Inc.
San Jose, USA
jianjun.chen@bytedance.com

ABSTRACT

Virtual indexes play a crucial role in database query optimization. However, with the rapid advancement of cloud computing and AI-driven models for database optimization, traditional virtual index approaches face significant challenges. Cloud-native environments often prohibit direct conducting query optimization process on production databases due to stability requirements and data privacy concerns. Moreover, while AI models show promising progress, their integration with database systems poses challenges in system complexity, inference acceleration, and model hot updates. In this paper, we present VIDEX, a three-layer disaggregated architecture that decouples database instances, the virtual index optimizer, and algorithm services, providing standardized interfaces for AI model integration. Users can configure VIDEX by either collecting production statistics or loading from a prepared file, enabling high-accuracy what-if analysis using virtual indexes that yield query plans identical to production instances. Additionally, users can freely integrate new AI-driven algorithms into VIDEX. VIDEX has been deployed at ByteDance, serving thousands of MySQL instances daily and over millions of SQL queries for index optimization tasks.

PVLDB Reference Format:
Rong Kang, Shuai Wang, Tieying Zhang, Xianghong Xu, Linhui Xu, Zhimin Liang, Lei Zhang, Rui Shi, and Jianjun Chen. VIDEX: A Disaggregated and Extensible Virtual Index for the Cloud and AI Era . PVLDB, 18(12): XXX-XXX, 2025.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/bytedance/videx.

*Tieying Zhang is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 18, No. 12 ISSN 2150-8097.
doi:XX.XX/XXX.XX

1 INTRODUCTION

Virtual index mechanisms facilitate hypothetical "what-if" analysis in a cost-efficient manner for query optimization evaluations, allowing database administrators to simulate the performance impact of potential indexes without actually creating them. This capability is implemented in most modern Database Management Systems (DBMSs). With the rapid advancement of cloud databases and AI-driven optimization techniques, two key challenges have emerged for database optimization systems: (1) Cloud-native environments require strict isolation between production instances and optimization components to ensure stability and data security; (2) AI-driven optimization approaches need flexible and standardized interfaces to integrate effectively with database systems. However, current virtual index implementations in popular DBMSs cannot address these requirements simultaneously, limiting their effectiveness in cloud-native environments and AI-driven optimization scenarios.

However, existing DBMSs struggle to meet modern cloud-native and AI-driven requirements, with key limitations summarized in Table 1. First, disaggregation from production databases—the ability to conduct query optimization using only statistical replicas—is crucial for security and scalability. While commercial systems like Oracle and SQL Server support this, their closed-source nature hinders custom extensions. Although PostgreSQL's HypoPG[1] is open-source, it is tightly integrated with the database, compromising true separation because certain statistics must be computed directly from the underlying source data[2] rather than being available in the statistical tables. Second, regarding virtual index support, the widely-used MySQL is the only evaluated system that completely lacks this functionality. Third, extensible model interfaces for integrating AI-driven models are paramount, yet among the evaluated systems, only PostgreSQL offers limited interfaces, which are not available on-demand [3]. This landscape reveals a pressing need for a new, fully disaggregated and extensible approach.

[1] https://github.com/HypoPG/hypopg (Accessed: 2025-07-14)
[2] https://www.postgresql.org/docs/current/catalog-pg-statistic.html (2025-07-14)

Table 1: Comparison of VIDEX for MySQL with popular DBMSs regarding key features, where ✓ represents satisfaction, ✗ indicates dissatisfaction, and ✙ denotes partial satisfaction.

Feature	Oracle	SQL Server	PostgreSQL	MySQL	VIDEX for MySQL
Disaggregated from Production	✓	✓	✗	✗	✓
Enable Virtual Index	✓	✓	✓	✗	✓
Extensible Model Interfaces	✗	✗	✙	✗	✓
Open-Source Availability	✗	✗	✓	✓	✓

To this end, we introduce VIDEX[3], a **VI**rtual index engine with **D**isaggregated and **EX**tensible properties, tailored for addressing database optimization challenges in the cloud and AI era. VIDEX presents a universal three-layer architecture applicable to various database systems, while our implementation specifically targets MySQL to demonstrate its effectiveness and simultaneously address the current absence of virtual index capabilities in this widely-used DBMS. We present these layers as follows: (1) The production database layer manages the online databases and maintains the statistical metadata. We demonstrate how the maintained statistical data enables secure query optimization operations by eliminating the need to access source data, thus significantly enhancing both security and scalability in cloud environments. (2) The VIDEX-optimizer layer functions as a "what-if" analysis tool using the statistical metadata. We demonstrate that it can generate query plans identical to those of the original online instances, thereby significantly reducing optimization overhead. (3) The VIDEX-Statistic-Server layer facilitates the integration and application of AI-driven database optimization models. We demonstrate that it can flexibly invoke different user-defined models for query optimization, such as estimation models for cardinality or number of distinct values (NDV), thus substantially accelerating the adoption of AI-driven approaches in database optimization.

VIDEX for MySQL has been deployed in ByteDance's large-scale production environments, optimizing thousands of MySQL instances and hundreds of thousands of SQL templates daily, where its reliability and practicality have been validated in an enterprise setting. VIDEX for MySQL has been open-sourced, and we hope it can deliver more efficient database optimization techniques and boost AI-driven optimization applications deployed into database production environments.

2 SYSTEM DESIGN

2.1 Architecture

Figure 1 illustrates the three-layer disaggregated architecture design of VIDEX, which consists of three major components:

Production database: The production database manages large volumes of real data and typically maintains several existing indexes. The instance maintains both schema and statistical data. Schema information defines the table structure, indexes, and integrity constraints. Statistical metadata includes table rows, disk size, page numbers, histograms, and NDV. This statistical data may be automatically updated in real-time during database operations or gathered when users initiate collection processes. Each type of

[3]A demonstration video is available at: https://youtu.be/Cm5O61kXQ_c (2025-07-14)

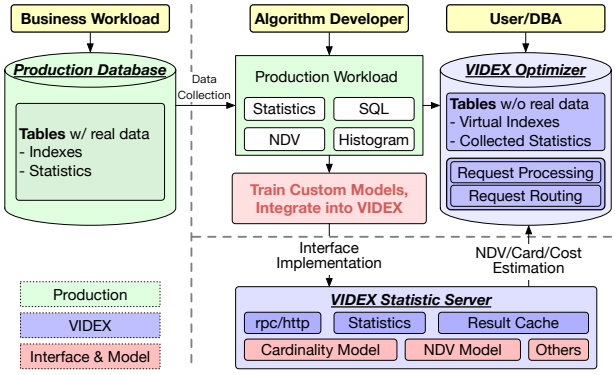

Figure 1: Three-layer, disaggregated VIDEX architecture

database employs a specified cost model within its query optimizer. Upon receiving a query, the database synthesizes a query plan using schema information, statistical metadata, and system configuration (such as `io_block_read_cost` in MySQL).

VIDEX-Optimizer: The VIDEX-Optimizer is designed to simulate the query optimization process of production databases without requiring access to the actual data. However, achieving accurate simulation poses two major challenges:

(1) Optimizer Logic Replication: Database optimization tasks commonly use external third-party query optimizers, like Apache Calcite [2], to simulate database optimizers like MySQL and PostgreSQL [1, 4]. However, third-party optimizers often do not perfectly reproduce the target production instances due to differing optimization rules and cost models, which can adversely affect the accuracy of what-if analyses.

(2) Statistical Data Replication: The statistical information exerts a decisive influence on making query plans. We aim to create validation instances in an extremely lightweight manner, by synchronizing statistical information without creating data replicas. Although some commercial databases enable flawless replication of statistics from online production instances to offline validation instances, it remains a non-trivial task for many popular databases, such as MySQL. Currently, optimization service providers often create replicas of online databases for experimentation, but the synchronization of substantial data incurs additional costs.

We design the VIDEX-Optimizer component to address these challenges effectively. The VIDEX-Optimizer reutilizes the native optimizer of the target database and intercepts its requests for underlying statistical information. During its operation, it constructs virtual tables, referred to as "VIDEX Tables," with schemas identical

to those of the production instance. When SQL queries are directed towards the VIDEX-Optimizer, it sequentially performs logical and physical optimizations, both of which may require statistical information. For simpler statistics, such as table row counts, the VIDEX-Optimizer directly injects pre-collected statistics. For more complex requests, like cardinality estimation, the optimizer forwards these requests to the disaggregated VIDEX-Statistic-Server, which will be introduced subsequently.

VIDEX-Statistic-Server: VIDEX decouples complex, precision-sensitive estimation requests from the VIDEX-Optimizer, such as cardinality estimation, which are particularly suitable for AI optimization. These requests are forwarded to a dedicated algorithmic service, the VIDEX-Statistic-Server, via REST or RPC protocols. The VIDEX-Statistic-Server stores data collected from the production environment, including statistical data and historical queries, which may be utilized by query-driven methods [3]. Based on the above data, the integrated AI algorithms can provide more accurate estimations. The VIDEX-Statistic-Server also supports response and model caching because we observe that different SQL queries often request repeated cardinality results, while some generalized pre-trained models only need to be loaded once to respond to multiple requests from different tables.

The design of disaggregated statistic server brings two benefits:

(1) Customizable AI-Driven GPU Optimization: Decoupling enables the VIDEX-Statistic-Server to run on dedicated GPU clusters, leveraging hardware acceleration and allowing AI models to be re-trained and hot-updated independently of the database.

(2) Extensible Algorithm Framework and Flexibility: The VIDEX-Statistic-Server offers standardized interfaces that simplify the algorithm integration, allowing researchers to use collected statistics for efficient model training and rapid deployment.

2.2 Workflow and Deployment

When a user initiates an analytical task for a specific production database, VIDEX first collects the necessary metadata, including schema, system variables, and statistical data, either through authorized production APIs or from a user-prepared metadata file. Subsequently, a VIDEX optimizer instance is chosen to create schema-identical, dataless tables. The collected metadata is then imported into an assigned VIDEX Statistics Server instance. After these preparations, users can connect to their optimizer instance to conduct index modification operations and use SQL EXPLAIN to generate query plans highly similar to those in the production environment.

The VIDEX Optimizer is deployed as a standalone service, providing what-if analysis for large-scale production instances. Given that the VIDEX Optimizer requires only schema and statistical information, it is extremely lightweight. Consequently, a single VIDEX Optimizer instance can manage hundreds of analytical tasks from various business workloads. Furthermore, by leveraging the elasticity of cloud-native environments, the VIDEX Optimizer can be dynamically scaled to accommodate an increasing number of analytical tasks.

The VIDEX Statistics Server is also deployed as a standalone service and handles requests from the VIDEX Optimizer. All requests generated by the same task are routed to the same statistics server instance. If multiple tasks correspond to the same database, requests are routed to the instance that has already loaded the relevant statistical data and models, thus minimizing redundant data imports and model loads.

```python
class RangeCond:
  """Represents a single-column range condition"""
  col_name                  # Column name
  data_type                 # Data type
  min_value, max_value      # None, or boundary value
  min_operator, max_operator  # "<", "<=", ">", ">="

class VidexModelBase(ABC):
  @abstractmethod
  def cardinality(self, range_cond: List[RangeCond]):
    pass # Estimates number of rows matching conditions

  @abstractmethod
  def ndv(self, column_list: List[str]):
    pass # Estimates number of distinct values
```

Listing 1: VIDEX Statistic Server Interfaces

Statistic Freshness: As production data evolves, statistical information can become outdated. Currently, VIDEX delegates statistic freshness to the user. This design is practical because refreshing a VIDEX environment is extremely lightweight. Users can import fresh statistics on-demand with minimal overhead. For future work, we plan to explore a mechanism to automatically detect statistical drift by comparing query plans from VIDEX with those from the production environment, which would proactively alert users when an update is advisable.

2.3 Algorithm Interface

We provide easy-to-understand and standard interfaces for custom algorithm integration. As shown in Listing 1, VIDEX defines two core interfaces: `cardinality` estimation and `NDV` estimation. The cardinality interface takes structured range conditions as input and returns the estimated row count, while the NDV interface accepts the column name list, returning the estimated NDV. This clean abstraction enables researchers to focus on their estimation algorithms without the need to navigate complex database internals. Utilizing this framework, developers can seamlessly integrate various heuristic or machine learning-based approaches.

3 IMPLEMENTATION

VIDEX-Optimizer: Since MySQL lacks a native virtual index mechanism and restricts schema modifications, we implemented the VIDEX-Optimizer with an underlying layer named VIDEX-Engine. This engine interacts with MySQL's query optimizer and index operations. We meticulously reviewed over 90 engine interfaces and implemented essential interfaces for query optimization and index management. The VIDEX-Engine formats requests related to cardinality and NDV estimation in a user-friendly manner and forwards them to the VIDEX-Statistic-Server

VIDEX-Statistic-Server: We developed two solutions for metadata collection and cost estimation. **1). Statistic Fetching**: This method leverages MySQL's system tables to gather basic statistics data. It collects single-column distinct values (NDV) and histograms via "UPDATE HISTOGRAM" and "SELECT COUNT DISTINCT", and leverages the column independence assumption to estimate

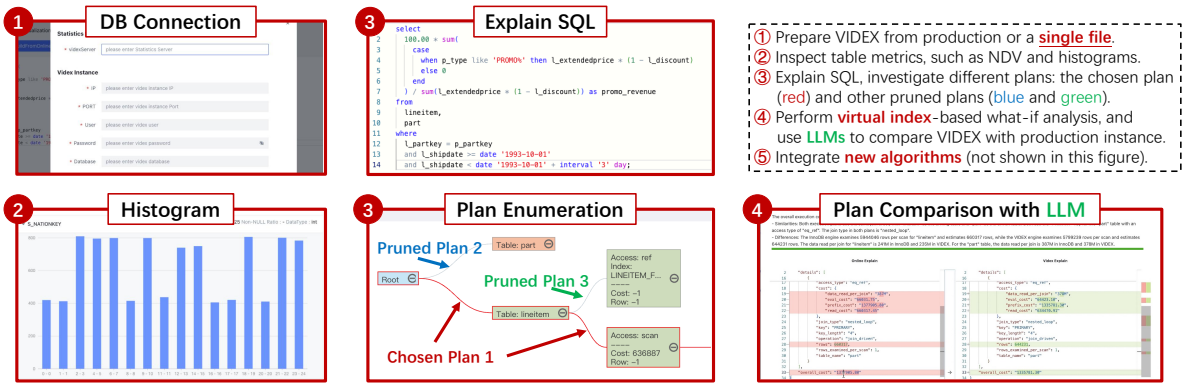

Figure 2: The illustration of the VIDEX Workflow.

multi-column join statistics. **2). Data Sampling**: This method employs authorized sampling APIs to collect a subset of data (up to 10^5 rows). It generates multi-column histograms to estimate the joint cardinality from sampling data, and employs an pre-trained AI model AdaNDV [5] to estimate the joint NDV.

Cost Model: VIDEX does not introduce a new cost model but is architected to reuse the native query optimizer of the target database, ensuring high-fidelity simulation. In the MySQL case, the optimizer enumerates potential join orders and access paths, calculating the cost for each candidate plan based on the statistics (NDV, histogram, etc.) provided by the VIDEX-Statistic-Server. This approach allows VIDEX to accurately simulate how the query plan changes in response to virtual indexes or modified data distributions, as the core cost calculation logic remains identical to that of the production database.

Notably, VIDEX can be engineered to extend to other databases such as PostgreSQL. VIDEX-Optimizer can utilize PostgreSQL's flexible hook mechanism and rich plugins, while the VIDEX-Statistic-Server is database-agnostic and interfaces with various database optimizers. This will be our future work.

4 EXPERIMENTS

We evaluated VIDEX's simulation accuracy under the premise that its algorithms can produce statistics identical to native MySQL. As shown in Table 2, VIDEX perfectly replicates query plans, achieving identical join orders and index selections across the TPC-H, TPC-H-Skew, and JOB benchmarks. The row count estimates are also highly precise, with an average q-error per plan operator (max ratio between VIDEX and native MySQL estimated rows) of less than 1.1x. This high fidelity is also maintained in our ByteDance production environment, even with imperfectly synchronized statistics.

Table 2: Simulation Accuracy: VIDEX vs. Native MySQL

Benchmark	#SQL	Join Order	Index Select	Row Q-Error
TPC-H	22	Identical	Identical	1.08
TPC-H-Skew	22	Identical	Identical	1.05
JOB	103	Identical	Identical	1.09

5 DEMONSTRATION SCENARIO

This section demonstrates VIDEX's key capabilities, including its disaggregated architecture, accurate plan simulation, what-if analysis, and extensibility, as illustrated in Figure 2.

① ∼ ② **Prepare VIDEX and Inspect Table Metrics**. We first prepare the VIDEX environment for a TPC-H benchmark. This involves collecting schema and statistics from a production MySQL instance (or a file) and loading them into the VIDEX-Optimizer and VIDEX-Statistic-Server. Users can then inspect table metrics like NDV and histograms.

③ **SQL explain and detailed analysis**. We execute an SQL EXPLAIN on TPC-H Q14 to review its query plan. The Web GUI offers an interactive query plan tree that visualizes all candidate plans enumerated by the optimizer. This feature allows users to explore the chosen plan (highlighted in red) and pruned alternatives, revealing the optimizer's cost-based decision-making process.

④ **What-if analysis via virtual index and compare with production database**. We demonstrate a what-if analysis by creating a virtual index in VIDEX, which leads to a significantly better query plan. Creating the corresponding real index on the production MySQL instance confirms the performance improvement. The GUI also compares plans from VIDEX and the production instance, using LLMs to explain any differences, showcasing VIDEX's high-fidelity simulation.

⑤ **Conveniently add a new cardinality algorithm**. Finally, we showcase the framework's extensibility by integrating a new, user-defined estimation algorithm implemented in Python.

REFERENCES

[1] [n.d.]. Alibaba Cloud: Database Autonomy Service. https://www.alibabacloud.com/en/product/das Accessed: 2025-03-26.

[2] [n.d.]. Apache Calcite: Dynamic data management framework. https://calcite.apache.org Accessed: 2025-03-26.

[3] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. 2021. Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. *VLDB* 15, 4 (Dec. 2021), 752–765.

[4] Zhaodonghui Li, Haitao Yuan, Huiming Wang, Gao Cong, and Lidong Bing. 2024. LLM-R2: A Large Language Model Enhanced Rule-based Rewrite System for Boosting Query Efficiency. *Proc. VLDB Endow.* 18, 1 (2024), 53–65.

[5] Xianghong Xu, Tieying Zhang, Xiao He, Haoyang Li, Rong Kang, Shuai Wang, Linhui Xu, Zhimin Liang, Shangyu Luo, Lei Zhang, and Jianjun Chen. 2025. AdaNDV: Adaptive Number of Distinct Value Estimation via Learning to Select and Fuse Estimators. *Proc. VLDB Endow.* 18, 4 (2025), 1104–1117.