Demonstrating Saṃsāra: A Super-Optimizer for Multimodal Stream Processing

Alessandro Ferri*
Technical University of Darmstadt and DFKI
alessandro.ferri@dfki.de

Uélison Jean Lopes dos Santos*
Technical University of Darmstadt and DFKI
uelison.santos@dfki.de

Szilard Nistor
Technical University of Darmstadt and DFKI
szilard.nistor@dfki.de

Pratyush Agnihotri
Technical University of Darmstadt and DFKI
pratyush.agnihotri@dfki.de

Manisha Luthra†
Ruhr University Bochum and Technical University of Darmstadt
manisha.luthra@rub.de

Carsten Binnig
Technical University of Darmstadt, DFKI, and hessian.AI
carsten.binnig@tu-darmstadt.de

ABSTRACT

The growing prevalence of multimodal data streams—such as video, images, and text—poses new challenges for stream processing systems. Existing systems are designed for structured data and struggle to support multimodal workloads efficiently. While multimodal large language models (MLLMs) provide powerful semantic understanding, naively integrating them into streaming pipelines can incur excessive computation and degrade performance. This demonstration presents Saṃsāra, a novel *super-optimizer* that enables real-time processing of multimodal data streams by aggressively specializing query plans to the query and data stream. Through an interactive traffic monitoring scenario, the demo demonstrates how multimodal queries are optimized by transforming naive query plans into optimized query plans. Users can directly observe how *semantic*, *logical*, and *physical* optimizations interact, reduce redundant model invocations, and impact system behavior, including responsiveness and execution accuracy. The demonstration shows that efficient multimodal stream processing requires rethinking query optimization beyond naive MLLM integration.

PVLDB Reference Format:
Alessandro Ferri, Uélison Jean Lopes dos Santos, Szilard Nistor, Pratyush Agnihotri, Manisha Luthra, and Carsten Binnig. Demonstrating Saṃsāra: A Super-Optimizer for Multimodal Stream Processing. PVLDB, 19(12): 4666 - 4669, 2026.
doi:10.14778/3827998.3828092

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/DataManagementLab/Samsara-demo.

*Both authors contributed equally.
†Work done while at DFKI and TUDa.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828092

1 INTRODUCTION

The Multimodal Gap in Streaming Systems. Stream processing systems (SPSs) have become a fundamental component of modern data infrastructures, enabling continuous processing of data streams and powering a wide range of real-time applications. These systems excel at handling structured data and provide strong support for low-latency and high-throughput processing [1]. However, despite their success, existing SPSs are fundamentally limited in their ability to process unstructured and multimodal data. In practice, many emerging applications rely on continuous streams of images, video, and audio, yet current systems lack native support to query and reason over such data in real time.

Multimodality as the New Normal. Real-world data streams are inherently multimodal. From traffic monitoring with camera feeds to smart city infrastructures and autonomous systems, data is continuously generated across multiple modalities. Meanwhile, recent advances in multimodal large language models (MLLMs) have demonstrated strong capabilities in interpreting and reasoning over such data. This makes it a natural step to integrate MLLMs into streaming systems in order to support rich, continuous queries over multimodal streams.

Why Naive Integration Fails. A straightforward approach is to embed MLLMs as operators in a streaming query plan and apply them directly on incoming data. We illustrate this using an example query where a user aims to detect a stolen car at a toll booth using a camera stream (see Figure 1). This approach, however, quickly breaks down in streaming environments (roughly 5× less throughput and high costs). Unlike batch-oriented systems, streaming systems must continuously process data under strict latency and throughput constraints. Applying expensive multimodal inference to every data item leads to excessive overhead, making naive integration impractical for real-time applications.

Multimodal Stream Processing with a Super-Optimizer. To address these challenges, in this paper we demonstrate Saṃsāra [5], a new class of optimizers for multimodal streaming, referred to as a *super-optimizer*. The key idea is to aggressively specialize query execution to a given query and data stream. While traditional optimizers focus on generic execution plans, the super-optimizer exploits the long-running nature of streaming queries to enable deeper, query-specific optimization.

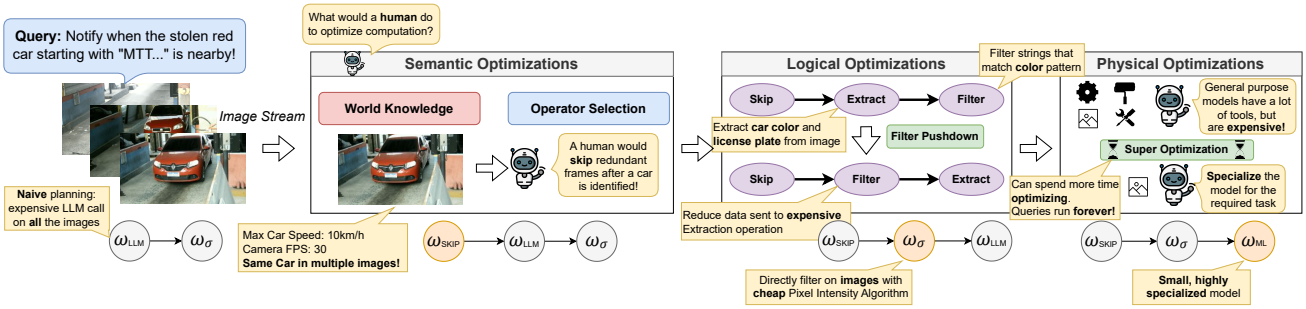


Figure 1: Overview of Saṃsāra, a super-optimizer to enable multimodal streaming systems which transforms a naive multimodal query plan into an optimized plan through a series of novel semantic, logical, and physical optimizations which aggressively optimize the plan for a particular long-running streaming query plan.

The super-optimizer extends classical optimization techniques with a novel *semantic optimization* phase, which incorporates contextual and domain knowledge into query planning. For example, in the toll booth scenario, the system can reason about temporal continuity of vehicles, skip redundant frames, and focus processing on relevant regions of the image. In addition, it applies logical and physical optimizations, such as pushing down filters by changing modality and selecting efficient models tailored to the query.

Interactive Demo. In this demo, we make these challenges tangible through a realistic traffic monitoring scenario, where a camera stream continuously monitors passing vehicles. Users can issue queries such as detecting a specific vehicle at real-time based on visual attributes and textual descriptions, highlighting the need to combine perception, reasoning, and continuous processing. Meanwhile, this setting exposes the inefficiencies of naïve approaches that treat multimodal models as black-box operators applied uniformly. The demo lets users explore how multimodal queries are executed and optimized in practice. Participants can issue queries over the same stream and observe how Saṃsāra transforms naïve plans into specialized optimized plans. Across these queries, we illustrate how semantic, logical, and physical optimizations interact to eliminate unnecessary computation.

2 SAṂSĀRA OVERVIEW

With Saṃsāra, our goal is to enable a new generation of stream processing systems that can natively understand and query across multiple modalities, including text, images, and audio. This requires integrating multimodal large language models (MLLMs) as first-class operators within streaming query plans, allowing users to express rich, continuous queries over unstructured data streams. However, approaches that directly embed MLLMs into query plans, as explored in batch-oriented systems such as CAESURA [7], ThalamusDB [2], Lotus [4], or DocETL [6], do not translate to streaming settings. In contrast to batch processing, streaming systems must operate under strict latency and throughput constraints, making naive multimodal query execution impractical. Moreover, edge streaming systems like NebulaStream [3] support multimodal data but lack optimization beyond naive structured processing.

Demonstrating a Super-Optimizer. In this demo, we present Saṃsāra, a first prototype that realizes the vision of multimodal streaming through *super-optimization*. The super-optimizer fundamentally rethinks query optimization by aggressively specializing query plans to a given query and data stream. Rather than relying on general-purpose plans, Saṃsāra generates deeply optimized execution strategies that minimize unnecessary multimodal inference while preserving query semantics.

The key insight is that streaming queries are long-running, enabling more expensive upfront optimization. In contrast to traditional optimizers that must produce plans quickly, the super-optimizer leverages this property to explore richer optimization opportunities and apply more comprehensive transformations across all stages of query planning.

Super Optimization Stages in Action. Figure 1 illustrates how the super-optimizer transforms a naive query plan into a specialized execution strategy in a toll booth scenario, where a continuous image stream is queried to detect a specific stolen car. The left side of the figure shows the naive approach, where a general-purpose MLLM is applied uniformly to all incoming images, resulting in unnecessary and repeated computation.

The first stage, *semantic optimization*, introduces a new class of transformations that leverage contextual and domain knowledge about the data stream. As illustrated, the optimizer first reasons about properties such as the maximum speed of a car and the frame rate of the camera, recognizing that the same vehicle appears across multiple consecutive frames. This allows the system to introduce operators such as frame skipping and to avoid redundant inference, mimicking what a human would do.

The second stage, *logical optimization*, extends classical query rewrites to the multimodal setting. As shown in the figure, operations such as filter pushdowns can fundamentally transform the way the filter operates, enabling application directly to the image stream, which results in early pruning of irrelevant data. For instance, instead of filtering over the color attribute (a string) extracted by the MLLM, lightweight image-based filters can be applied before invoking expensive extraction operators, thereby reducing the amount of data that needs to be processed by the MLLM.

Finally, *physical optimization* adapts the underlying execution by deriving and selecting specialized models tailored to the query and data stream. Instead of relying on generalized models, which

Table 1: Tollbooth dataset queries combining MLLM perception and stream processing operators.

Query ID	Description of Query	MLLM Tasks
Q1	Car brand recognition	Object detection
Q2	Car color recognition	Color recognition
Q3	License plate detection	Object detection, text extraction
Q4	Most popular brand & color	Color recognition, object detection, aggregation
Q5	Most popular brand	Color recognition, object detection, aggregation
Q6	Most popular color	Color recognition, aggregation
Q7	Repeated car detection	Object detection, text extraction, aggregation
Q8	Red stolen 'MTT' car	Color recognition, object detection, text extraction, filtering
Q9	Unique license plates	Object detection, text extraction, aggregation

are flexible but expensive, the system generates smaller, task-specific models that provide higher efficiency while retaining accuracy.

From Naive to Specialized Execution. The demo showcases how these optimization stages interact to transform naive query plans into highly specialized execution strategies. Users can explore different queries over the same traffic scenario and observe how the system adapts its execution plan, progressively refining it through semantic, logical, and physical optimizations. Leveraging a side-by-side comparison of the Naive and Optimized execution, the demo highlights how aggressive specialization of query plans is essential for enabling efficient multimodal stream processing.

3 DEMONSTRATION OVERVIEW

Scenario and System Setup. We demonstrate Saṃsāra using a traffic monitoring scenario in which a continuous image stream captures vehicles passing along a street. We deploy a physical setup using a Carrera Hybrid track to emulate a traffic environment. Users can control vehicles and vary their speed, trajectory, and distance from the camera, thereby generating diverse and realistic streaming conditions. The live video feed is captured using a Sony Alpha a6000 camera connected via an HDMI capture card.

The captured frames are continuously ingested into an Apache Flink streaming pipeline via Kafka, where queries are executed in real time. We construct a diverse set of queries covering operations such as *filtering* and *aggregation*, as illustrated in Table 1. Each query includes at least one multimodal LLM-based operator targeting different extraction functions, including *color recognition*, *object detection*, and *text extraction*.

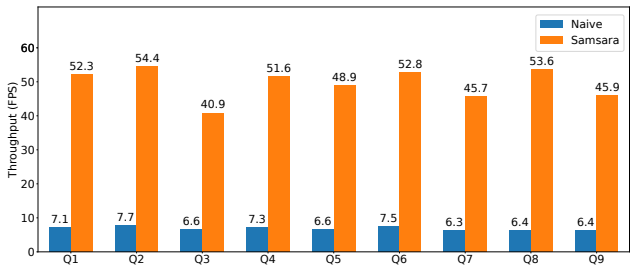


Figure 2: End-to-End gains on the tollbooth queries.

User Interaction. The demonstration provides an interactive environment that allows users to explore system behavior under different workloads. As shown in Figure 3, users can select from a predefined set of representative queries spanning detection, extraction, and aggregation tasks (e.g., detecting a stolen red car, extracting license plates, or counting the number of vehicles). Moreover, users can formulate custom queries through a natural language interface, enabling flexible interaction with the system. The system visualizes execution through synchronized views of the live stream, the optimized execution, and the naive execution (Figure 4 (a)). Frame indices are overlaid on each stream, indicating the current input frame and the most recently processed frames for each execution strategy. This makes the processing lag of each approach directly observable. A second screen (Figure 4 (b)) continuously reports runtime metrics, including throughput and end-to-end latency, allowing users to quantitatively assess performance.

Optimization in Action. A central aspect of the demonstration is to highlight how Saṃsāra transforms naive query plans into super optimized plans. In the naive approach, each incoming frame is processed by a multimodal model (Qwen-2.5-VL 7B), leading to repeated and unnecessary computation across the stream. In contrast, Saṃsāra applies a sequence of optimization stages that progressively refine execution. *Semantic optimization* leverages contextual knowledge about the stream, such as temporal continuity, to eliminate redundant computation (e.g., avoiding repeated processing of similar frames). *Logical optimization* restructures the query plan via filter and projection pushdown, reducing data processed by expensive operators. *Physical optimization* selects specialized models tailored to the query, reducing inference cost while retaining accuracy.

System Results and Observations. The effects of these optimizations are directly observable in the demonstration. As shown in Figure 4, naive execution exhibits a clear processing lag relative to the live stream, as it applies expensive multimodal inference uniformly to all frames. In contrast, optimized execution is able to keep up with the incoming stream more closely, reflecting the reduction of redundant computation and specialization of execution.

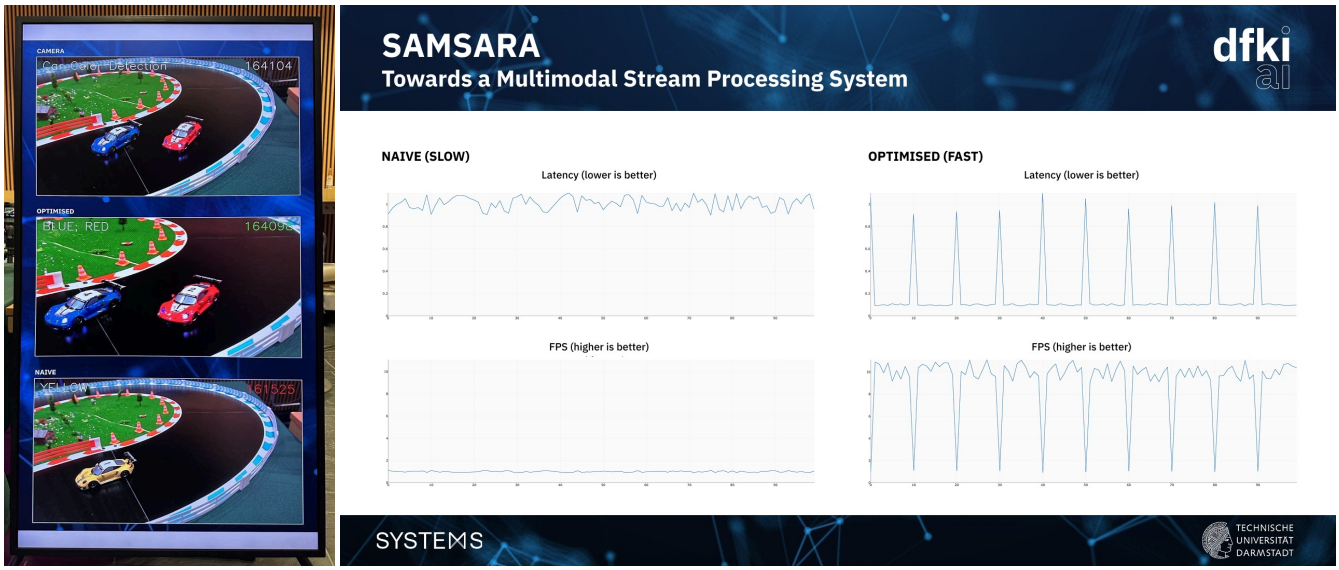
These observations hold across nine queries from the Rodosol-ALPR dataset (Table 1), which captures cars passing through a tollbooth. Figure 2 shows per-query performance. Across all workloads, naive execution processes frames at a lower rate due to repeated invocation of a MLLM. In contrast, optimized execution achieves higher throughput by reducing expensive model calls and tailoring execution to the query and data stream.

At the same time, the demonstration highlights the trade-offs introduced by aggressive optimization. Techniques such as frame skipping and region-based processing may introduce small inaccuracies. However, the observed degradation in accuracy remains limited, and most inaccuracies stem from mispredictions of the underlying models rather than from the optimization strategies themselves. This indicates that the super-optimizer improves performance while largely preserving correctness.

Overall, the demonstration shows that naive integration of multimodal models is insufficient for streaming settings. Instead, efficient multimodal stream processing requires aggressive, query-specific optimization across semantic, logical, and physical stages.

(a) (b)

Figure 3: (a) Main UI for selecting queries targeting tasks such as color detection, OCR, and aggregation. (b) Camera setup for the simulated street monitoring scenario, where live video streams are processed using naive and SAMSARA-optimized queries.



(a) (b)

Figure 4: (a) Execution of the selected query in three panes: live camera feed (top), optimized execution (middle), and naive execution (bottom). Frame indices show the current input (top) and most recently processed frames (middle, bottom), highlighting processing lag. (b) Live query performance in terms of latency and throughput.

REFERENCES

[1] Paris Carbone et al. 2015. Apache Flink: Stream and Batch Processing in a Single Engine. *IEEE Data Eng. Bull.* (2015).

[2] Saehan Jo and Immanuel Trummer. 2024. ThalamusDB: Approximate Query Processing on Multi-Modal Data. *Proc. ACM Manag. Data* 2, 3 (2024), 186.

[3] Adrian Michalke, Aljoscha Lepping, Volker Markl, Ricardo Martinez, Nils Schubert, Lukas Schwerdtfeger, Taha Tekdogan, Steffen Zeuch, Ariane Ziehn, Christoph Falkensteiner, et al. 2025. NebulaStream: an extensible, high-performance streaming engine for multi-modal edge applications. In *Companion of the 2025 International Conference on Management of Data*. 195–198.

[4] Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2025. Semantic operators and their optimization: Enabling llm-based data processing with accuracy guarantees in lotus. *Proceedings of the VLDB Endowment* 18, 11 (2025), 4171–4184.

[5] Uélison Jean Lopes dos Santos, Alessandro Ferri, Szilard Nistor, Riccardo Tommasini, Carsten Binnig, and Manisha Luthra. 2026. Towards Multimodal Stream Processing Systems. In *EDBT*. 627–633.

[6] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G. Parameswaran, and Eugene Wu. 2025. DocETL: Agentic Query Rewriting and Evaluation for Complex Document Processing. *Proceedings of VLDB* 18, 9 (2025), 3035–3048.

[7] Matthias Urban and Carsten Binnig. 2024. CAESURA: Language Models as Multi-Modal Query Planners. In *CIDR*.