applied sciences

Article

Modeling and Performance Analysis of a Notification-Based Method for Processing Video Queries on the Fly

Clayton Kossoski *, Jean Marcelo Simão and Heitor Silvério Lopes

Graduate Program in Electrical and Computer Engineering, Federal University of Technology—Paraná, Curitiba 80230-901, PR, Brazil
* Correspondence: claytonk@alunos.utfpr.edu.br

Abstract: With the rapid growth of video data, the search for content and events in videos is becoming increasingly relevant, and many challenges arise. Various approaches have been proposed to deal with many issues. However, many open questions are still related to computational cost and latency, especially for real-time applications. Considering the need for new and efficient solutions, the so-called NOP (Notification Oriented Paradigm) could be a suitable alternative. NOP introduced a new way of thinking and developing software in which small collaborative entities perform fact execution and logical decision processing based on precise notifications. Following these concepts and practical tools, this paper proposes a new querying processing method based on NOP, focusing on search and matching in a continuous flow context. Experiments on a labeled dataset demonstrated the suitability of the proposed method for low-latency processing with polynomial complexity. The results are better than the state of the art, which works at exponential cost.

Keywords: query processing method; notification-oriented paradigm; event stream processing; dataset; case study

Citation: Kossoski, C.; Simão, J.M.; Lopes, H.S. Modeling and Performance Analysis of a Notification-Based Method for Processing Video Queries on the Fly. *Appl. Sci.* **2024**, *14*, 3566. https://doi.org/10.3390/app14093566

Academic Editor: Andrea Prati, Božidar Šarler, Haiping Liu, Jian Zhang and Jiaqi Li

Received: 17 December 2023
Revised: 16 February 2024
Accepted: 18 March 2024
Published: 24 April 2024

1. Introduction

Computer vision has evolved rapidly due to recent advances in deep learning, development libraries, and GPU computing power, allowing large amounts of video to be processed more efficiently than before [1,2]. However, important work has pointed out difficulties in dealing with this type of unstructured data, especially for processing and retrieving events [3–6]. Therefore, to solve these problems, computer vision researchers have developed a large body of work [2,7–9].

Recently, video streaming queries have gained attention, mainly due to their wide scope of applicability. For example, traffic must be constantly supervised in smart cities [10,11]. In this context, traffic authorities may need a system that allows direct queries to detect high-volume traffic, locate vehicles by features (e.g., license plate, color, or model), or track suspicious vehicles that are traveling together, such as a criminal escort car, among others [4,5,11,12]. Despite the advances, several problems still need to be solved, such as the lack of a standard query language and an efficient engine, which requires a pragmatic solution, as happened with relational databases with ANSI SQL. Still, complex processing pipelines and the high latency between the user's search command and the video retrieval are relevant issues [4,10,13].

In addition, more attention should be given to the computational cost of applications [14]. Hence, even though there have been general technical advances in computing power (e.g., multicore processors and GPUs), graphics processing libraries (https://docs.opencv.org/4.x/index.html) (accessed on 16 December 2023) [15], and video databases [3,4], the state of the art is not yet able to provide adequate solutions to the problems mentioned above.

From an epistemological and ontological point of view, the origin of the problem would be beyond the current approaches, reaching its usual programming paradigms and related idiosyncrasies, such as the tendency to misuse processing capacity and program coupling [16]. The so-called paradigm unfitness would be a relevant and even primary cause of software development problems that hinder breakthrough solutions in the video query system as much as other domains involving distributed and high-demanding processing. Therefore, researchers have devised new development paradigms, such as the highlighted Notification Oriented Paradigm (NOP), created more than a decade ago and evolved in recent years [16–22].

In particular, this paradigm aims to promote better performance in system execution and easier construction of complex systems, especially to parallel and distributed systems [16]. NOP splits a computational system into collaborative and decoupled entities with minimal subentities. Their decoupled notifier subentities collaborate through a well-orchestrated and precise notification chain. It avoids redundancies and coupling, saves processing power, and allows processing distribution. Several researchers have highlighted the advantages of NOP for software development [23], programming language [24], computational parallelism [25], and low-cost operations [21], among others.

Inspired by the features and advantages of NOP and existing development tools, this paper proposes a new query-processing method that handles multiple video events and reacts with low latency when matches occur. A distinct conformation of the state-of-the-art NOP framework is used to query the system domain to avoid intensive database operations: storage, retrieval, and triggers. In addition, a use case with several experiments has been developed to show the feasibility of the proposed NOP-based solution for this domain.

This paper presents three main contributions, among others:

- The Notification-oriented Querying Method and Prototype (NOP Query) is a new querying method based on NOP concepts and its state-of-the-art framework.
- A realistic case study that systematically evaluates the proposed NOP querying method/prototype through CPU and memory usage experiments.
- A new large labeled traffic surveillance dataset used in this case study are available to the research community to enable reproducible experiments.

We also present and discuss some possible applications of the proposed method in related areas.

The next sections of this paper are organized as follows. Section 2 presents related works. Section 3 presents an overview of the NOP and its state-of-techniques development framework. Section 4 proposes and describes the NOP Query. Section 5 presents the case study which aims to show the suitability of the NOP Query. Section 6 discusses and compares the NOP Query with related technologies. Section 7 concludes the paper and discusses the promising results and future research directions.

2. Related Work

Historically, video query processing requires enormous computing power and many different technologies, from image acquisition to pre-processing, labeling, storage, and event processing. There are a large number of studies dealing with search engines and video queries from different aspects. The most relevant works are considered in this section.

2.1. Computer Vision Using GPUs and FPGAs

In the field of artificial intelligence, the convergence of computer vision, deep learning, and GPUs (Graphics Processing Units) has revolutionized the way that machines perceive and interpret visual information [26]. Computer vision is a multidisciplinary field that allows machines to interpret and understand visual information from images, videos, or other visual data. In turn, Deep Learning is a subset of machine learning that uses neural networks with multiple layers, called deep architectures [27,28]. These models learn hierarchical representations from raw data, automatically extracting relevant features. Still, GPUs speed up deep learning training and inference due to their parallel processing capabilities and ease with matrix operations, such as convolutions and matrix multiplication [29].

Examples of applications that benefit from these technologies together include autonomous vehicles, medical imaging, surveillance, robotics, augmented reality, and much more.

Specifically in the area of video queries, many authors have used GPUs to perform data ingestion, preprocessing, and object labeling [3–5,30–33], among others. Other interesting approaches have used GPUs to calculate string similarity metrics for large datasets [34], as well as to object tracking using the honeybee search algorithm taking advantage of parallel computing [35]. As matter of fact, string similarity assessment is a processing task applied to large volumes of data, often performed by various applications, such as search engines, biomedical data analysis, and even software tools to defend against viruses, spyware, or spam. In turn, object tracking refers to the relocation of specific objects in consecutive frames of a video sequence. These examples demonstrate how GPUs can be used to solve problems that especially need parallel computing for processing mathematical operations.

Besides, it is possible to use Field-Programmable Gate Arrays (FPGA) for deep learning tasks instead of GPUs [36]. Even if GPUs have been the dominant hardware platform for deep learning due to their highly parallel architecture and efficient matrix operations, FPGAs actually offer several advantages that make them attractive to certain applications mainly because their energy efficiency, reconfiguration flexibility, low latency, and customization at the hardware level. Some interesting uses of FPGAs include real-time detection of straight lines in static and low-definition images [37], robotic computing [38], and neural network optimizations [39], among others.

2.2. Computing Architectures for Video Querying

Computing architecture for video querying refers to the physical integration between producers, processors, and consumers. In general, there are three computing architectures: (a) compute everything on a powerful server, (b) compute where the data is produced, or (c) distribute computing across hierarchical nodes. Each approach has advantages and disadvantages.

The most common architecture used over time is the server-only architecture, also called client-cloud configuration or cloud computing. In this architecture, developers consider distributing a powerful server or cluster with many CPUs, GPUs, memory, and bandwidth across producers (e.g., mobile, IoT devices, video databases) and the cloud to process all data [3,4,30,40–43]. The advantages are simplicity, powerful hardware support, and low internal latency. However, the main disadvantage depends on the external Internet connection between the video source and the system, which can be very slow.

The edge-first architecture, also called client-first or server-less, is almost the opposite of server-only. In this architecture, the focus of query processing is on the edge devices and local edge servers rather than high-performance cloud systems [31,44–47]. This architecture is particularly suitable for applications such as autonomous vehicles and real-time monitoring that cannot tolerate internet latency and offloading of processing. The disadvantages are related to the hardware limitations of edge computing devices.

In the fully distributed architecture, all data flow is distributed through a three-stage pipeline coordinated by an extensive scheduling algorithm that determines when each task is processed [42,48]. For example, [49] proposed the mobile-cloudlet-cloud architecture, a proof-of-concept system to distribute the computing of the face recognition task on mobile devices, cloudlet server, and cloud in order to minimize the response time, given the diverse latencies and computing resources. Similarly, [42] focused on distributing DNN processes between edge devices and the cloud, considering online and offline contexts. Furthermore, [48] proposed an optimization model to maximize data throughput in a more extensive cluster system. Because of these characteristics, it is the most complex architecture, as it depends on many internal factors (e.g., scheduling) and external factors (e.g., network connectivity).

2.3. Video Query System

A video query system aims to find the most relevant videos given a user query [50]. Among the processing activities, the most important are ingesting and preprocessing, which require advanced computer vision algorithms to find and identify relevant content, such as shapes, objects, colors, and events [33]. Then, each object and event must be properly labeled and stored in a data structure or database. Finally, an engine is needed to retrieve videos and events given a user query. For this reason, video search frameworks are usually all-in-one solutions that handle data ingestion, preprocessing, object detection, labeling, storage, and retrieval [4–6,10,32,43,48,51–53].

From the user's perspective, few of the mentioned works provide a query language or human–computer interaction (HCI) [4,5,10,33,51,53,54]. Some current query languages extend traditional SQL with specific operators related to video events, such as objects (e.g., a car), attributes of objects (e.g., a red car), and spatiotemporal relationships (e.g., a car to the left of a bus, a truck that appears before the car) [10,13,53,54].

2.4. Video Databases

In recent decades, with the development of video recording devices with Internet access, the amount of video content has increased dramatically, requiring the support of a specific video database to facilitate storage and retrieval [10]. Over time, several studies have used relational databases to store and query video data using regular SQL, usually in offline mode (e.g., "after the fact") [4,5,32,43]. In some works, relational databases have used triggers to compute events [11,51,55].

Despite the great success of the SQL language for relational databases, they are not directly applicable to video databases because video data are dynamic and contain many "semantic elements", including spatiotemporal relationships between objects and events. Due to the disadvantages of relational databases, other types of databases have been used. For example, [6,54] used a graph database to process spatiotemporal relationships and perform queries. However, although graph databases can express objects and their relationships more easily, the recording process in streaming scenarios is quite expensive [56].

2.5. Video Querying Latency

Image and video processing require expensive data extraction and storage resources [4,54]. The most accurate modern extraction methods are based on convolutional neural networks (CNNs). Unfortunately, these methods require several milliseconds per image/frame when using modern GPUs. Therefore, the final cost of queries on a huge video corpus can be prohibitive [43].

Query latency refers to the computation time between the start of query processing and the retrieval of the desired image or video event. There are two main processing times: offline processing (or query time) and online processing (or ingest time). Offline processing has no latency restrictions and can take minutes, hours, or even days. It is a very popular approach due to its low cost, requiring powerful hardware only when the application has latency constraints [12]. On the other hand, for online processing, latency is crucial for detecting events in real-time. It requires modern GPUs in video ingestion, deep learning models for image processing, and complex event detection algorithms. These features make it more expensive and complicated than offline processing. For example, the authors in [5,12] used cheap CNNs at ingest time to create an approximate index of all possible object classes in each frame to process queries for each class in the future, balancing the use of cheap models and expensive convolutional networks at query time to improve latency and throughput as needed.

As far as the authors of this article are aware, the VidCEP work [6,13,53] and other related studies, including the recent PhD thesis [33], are the first to show efficient query processing with low latency (e.g., near real time). However, the approach uses intensive graph database operations and short-term window analysis, which significantly limits the size of the search range, and the processing time increases significantly with large window sizes. Currently, there is a gap in video querying with some real-time concerns, which promotes the need for new solutions, as explained in the following sections of this article.

3. The Notification-Oriented Paradigm

The NOP is an emerging paradigm that originated from a manufacturing discrete control solution [19], which later evolved into a general discrete control solution, an alternative inference solution, and even a computational paradigm [21]. As a main differential, NOP has two groups of distinct processing entities, namely fact–execution entities and logical–causal entities, which collaborate through punctual, precise notifications.

The fact–execution entities deal with NOP elements and the execution of procedures/actions that change their states, whereas the logical–causal entities decide which actions should be executed. In turn, the fact–execution entities are represented by Fact Base Elements (FBEs) with their *Attributes* and *Methods*. The logical–causal entities are represented by *Rules* elements with their *Conditions-Premises* and *Actions-Instigations* [16,57]. Still, *FBEs* and *Rules*, with their constituents, can be and usually are represented in a declarative way [58].

Figure 1 presents an example of the NOP entities for a given sensor application. The *FBE* named Sensor, somehow similar to an instance of a class in the Object-Oriented Paradigm (OOP), represents a sort of notifier object composed of two boolean *Attributes*, atIsRead and atIsActivated, and a *Method* named mtTrigger that sets the values of atIsRead to true and atIsActivated to false. In turn, the associated notifiable *Rule* rlSensorTrigger comprises a *Condition* related to *Premises* and an *Action* related to an *Instigation*. When both *Condition*'s *Premises* are true after notifications of the pertinent *Attributes*, it fires the *Rule* rlSensorTrigger, which causes the *Action*'s *Instigation* to start the *Method* mtTrigger from the *FBE* Sensor. As the example suggests, the *FBE* and *Rule* constituents collaborate through precise notifications.

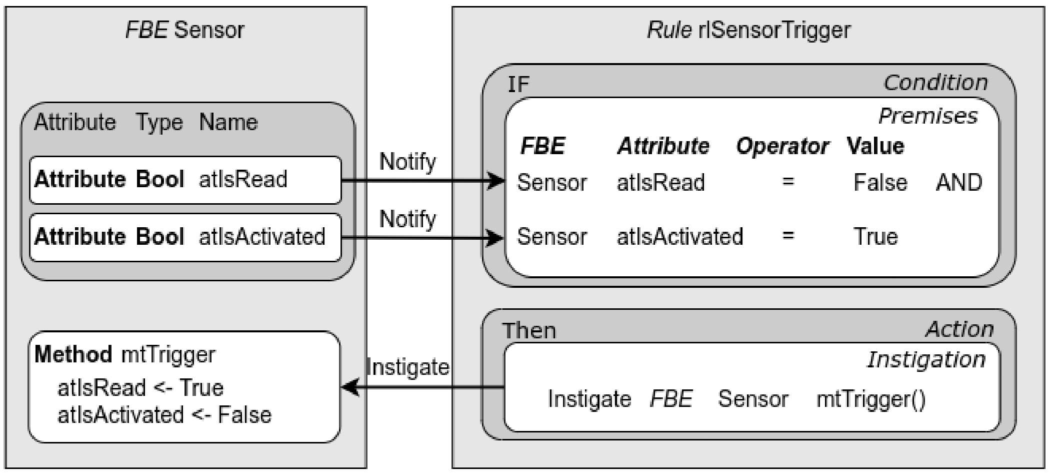

FBE Sensor
Attribute Type Name
Attribute Bool atIsRead
Attribute Bool atIsActivated
Method mtTrigger
atIsRead <- True
atIsActivated <- False
Notify
Notify
Instigate
Rule rlSensorTrigger
IF
Condition
Premises
FBE ***Attribute*** ***Operator*** **Value**
Sensor atIsRead = False AND
Sensor atIsActivated = True
Then
Action
Instigation
Instigate *FBE* Sensor mtTrigger()

Figure 1. Example of instances of FBE Sensor and Rule rlTriggerSensor with their constituents for a sensor application.

The NOP state of the art, called Notification-Oriented Paradigm Language (NOPL), combines a programming language and a compilation system for specific targets. Despite being very performant, the NOPL is yet to apply to large or industrial-level projects. For now, it is used to demonstrate NOP properties in academic and benchmark applications [59,60]. Alternatively, the NOP state-of-techniques is the so-called NOP C++ Framework 4.0/4.5 for large or industrial projects. This framework is an evolution of the previous ones and achieved maturity to be more extensively applied. As it is a notification-oriented framework over C++, changing the usual way that C++ operates, it is naturally less performant than the quite prototypal NOPL Technology with dedicated language and compilers. Nevertheless, this framework is still performant and stable for real NOP applications [59–61].

Considering the example of Figure 1, Algorithm 1 presents the corresponding code for NOPL, and Algorithm 2 presents the corresponding code using NOP Framework C++ 4.0. More details about the NOPL and the NOP Framework C++ can be found in [24,57,60,62]. The framework is in the final stage of the patent process, and it is not currently available to the community. However, its structure and details have been widely disclosed [22,57,59], including the works cited in this section.

Algorithm 1 Example of NOPL statements for *FBE* Sensor and *Rule* rlTriggerSensor presented in Figure 1

```
fbe Sensor
  private boolean atIsRead = false
  private boolean atIsActivated = false
  private method mtTrigger
    assignment
      this.atIsRead = true
      this.atIsActivated = false
    end_assignment
  end_method
  rule rlSensorTrigger
    condition
      subcondition
        premise prIsReadFalse
          this.atIsRead == false
        end_premise
        and
        premise prIsActivated
          this.atIsActivated == true
        end_premise
      end_subcondition
    end_condition
    action sequential
      instigation sequential
        call this.mtTrigger();
      end_instigation
  end_action
 end_rule
end_fbe
```

The core mechanism of the paradigm is the so-called NOP Inference Chain [63], which is modeled in Figure 2 and exemplified in Figure 3. Each NOP entity can send and/or receive precise and pertinent notifications, thereby allowing the evaluation of states only when a notification arrives. This collaborative notification-oriented behavior provides a new form to develop and execute software (and even hardware) applications based on small reactive notifiable entities [16,21].

Algorithm 2 Example of NOP Framework C++ statements for *FBE* Sensor and *Rule* rlTriggerSensor presented in Figure 1

```
/* Include the NOP Framework C++ 4.0 */
#include "libnop/framework.h"

/* Create the class Sensor */
class Sensor
{
 public:

  /* Create the Attribute atIsRead */
  NOP::SharedAttribute<bool> atIsRead
  { NOP::BuildAttribute<bool>(false) };

  /* Create the Attribute atIsActivated */
  NOP::SharedAttribute<bool> atIsActivated
  { NOP::BuildAttribute<bool>(false) };

  /* Create the Premise prIsRead */
  NOP::SharedPremise prIsRead
  { NOP::BuildPremise(this.atIsRead, false, NOP::Equal()) };

  /* Create the Premise prIsActivated */
  NOP::SharedPremise prIsActivated
  { NOP::BuildPremise(this.atIsActivated, true, NOP::Equal()) };

  /* Create the Condition cnRlSensorTrigger */
  NOP::SharedCondition cnRlSensorTrigger =
  NOP::BuildCondition<NOP::Conjunction>(prIsRead, prIsActivated);

  /* Create the Method mtTrigger */
  NOP::Method mtTrigger{
   this.AtIsRead = true;
   this.atIsActivated = false;
  };

  /* Create the Instigation inMtTrigger */
  NOP::SharedInstigation inMtTrigger =
  NOP::BuildInstigation(mtTrigger);

  /* Create the Action acMtTrigger */
  NOP::SharedAction acMtTrigger =
  NOP::BuildAction(inMtTrigger);

  /* Create the Rule rlSensorTrigger */
  NOP::SharedRule rlSensorTrigger =
  NOP::BuildRule(cnRlSensorTrigger, acMtTrigger);

/* End class */
};
```

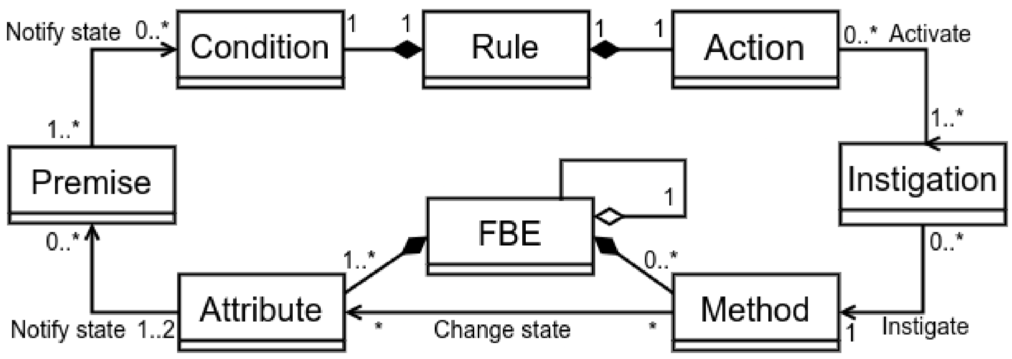


Figure 2. NOP inference chain.

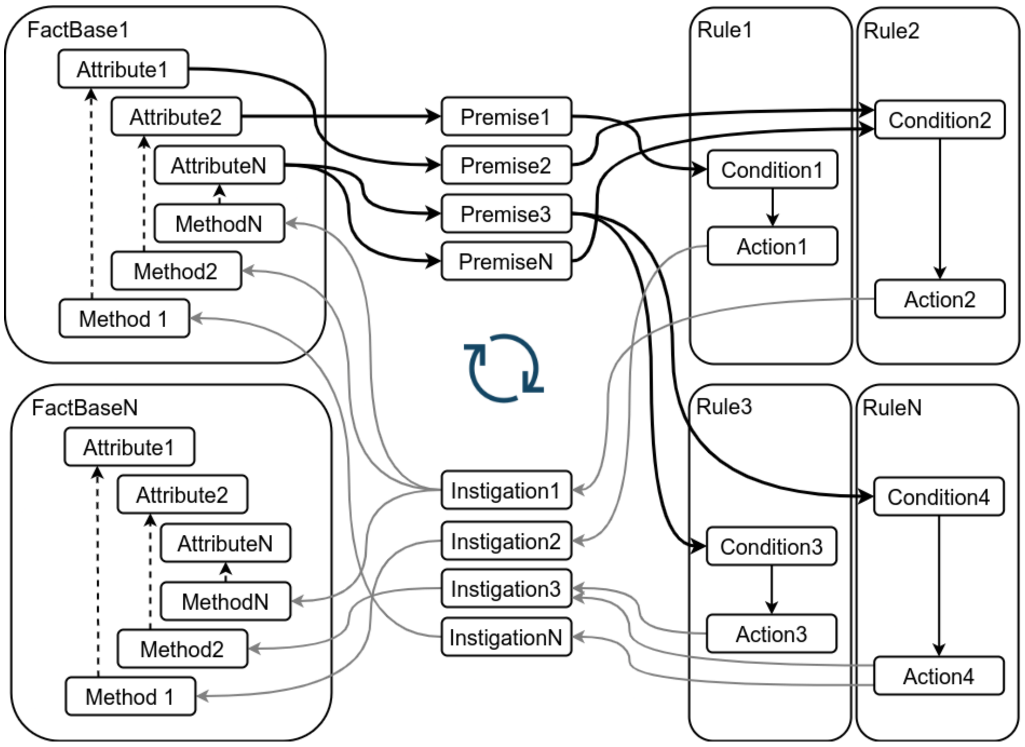


Figure 3. Notification flows.

3.1. NOP Inference Chain

The following items explain the NOP entities according to the inference chain, taking into account Figures 2 and 3:

- Fact Base Element (*FBE*): Entity (e.g., an instance of a type or class) that contains the notifier *Attributes* and inciteful *Methods*;
- *Attribute*: Entity that is responsible for storing a value that represents property states of an *FBE* and can precisely notify related *Premises* when their value changes;
- *Premise*: Entity that, after being notified by an *Attribute*, compares that *Attribute* with another *Attribute* or a constant via a logic-relational operator (e.g., $=$, $\neq$, $>$, $<$), and subsequently, notifies related *Condition* when its logical state changes;
- *Condition*: Entity that groups one or more *Premises* and, when notified by any of them, performs a logical operation (e.g., conjunction or a disjunction) over their states and notifies its concerned *Rule* if its logical value changes;
- *Rule*: Entity with a *Condition* and an *Action* that determines the execution of its *Action* when its *Condition* is approved (e.g., its logical state is true);
- *Action*: Entity that, when instigated by its *Rule*, activates its *Instigations* to execute its functions, properly parametrizing them if necessary;

- *Instigation*: Entity related to one or more *Actions* that, when instigated by an *Action*, instigates a set of *Method* entities to execute, properly parametrizing them if necessary;
- *Method*: Entity that, when instigated by an *Instigation*, executes a function or service of a *FBE* and, therefore, may change the states of *Attributes*, thereby feeding a notification inference cycle.

The NOP organization reduces or even eliminates some of the issues concerning classical development paradigms, such as the Imperative Paradigm (IP) and Declarative Paradigm (DP), which, respectively, and even notably, include OOP and Rule-Based Systems (RBS) [16,57]. Examples of those issues are the often strong coupling of code parts and the related structural and temporal redundancies concerning logical–causal evaluation processing [57].

Structural redundancies occur when a given logical and/or causal evaluation is unnecessarily repeated elsewhere in the code. In turn, temporal redundancies occur when a given logical and/or causal evaluation is unnecessarily re-executed over time, and the data involved in this calculation did not change since its last execution. These problems frequently happen, for example, in nested loops in IP languages that usually tend to cause code coupling, thereby making it more challenging to achieve processing parallelism and distribution [16,18,22].

The problems mentioned above do not happen in NOP since the precise notifications, e.g., between *Attributes* and *Premises*, avoid temporal redundancy. Furthermore, the *Conditions* sharing *Premise* collaborations avoid structural redundancy. In this sense, NOP differs from OOP approaches because it is not loop-oriented, and NOP differs from rule-based approaches because it is not based on a monolithic factual search-oriented inference engine [16,18,19,22].

To summarize, the NOP Inference Chain allows software development with three elementary properties [16]:

1. High-level rule-oriented modeling and programming, which helps easiness in some software/system development [59].
2. Avoidance of structural and temporal redundancies, which allows achieving proper system performance [64].
3. Implicit entity decoupling, which can favor parallel and/or distributed computing [65].

More detailed information about the NOP and its extensive research can be found in [16,25,58,66].

3.2. *The Historical Applicability of NOP*

Many efforts have been made to become NOP development more straightforward and popular. However, as NOP is a relatively new paradigm, some artifacts and materialization may still need to reach the maturity level necessary for developing a complete industry-level software project.

Besides, a gap exists concerning industrial-like projects using NOP C++ Framework 4.0. To the best of the authors' knowledge, only academic example problems were still approached to assess the feasibility of NOP C++ Framework 4.0 and even the NOP technologies in general. Even if some of those academic examples are relevant for NOP demonstrations, more complex applications are necessary to evaluate their potential in a real-world scenario [59–61].

In essence, new applications may contribute to the maturity and popularity of NOP as a general development paradigm, always regarding its philosophical and current technological foundations. Considering this motivation about NOP itself and the gaps mentioned above in video querying systems, this paper proposes an innovative online/streaming video query solution using the NOP C++ Framework 4.0.

4. Notification-Based Querying Method (NOP Query) Proposal

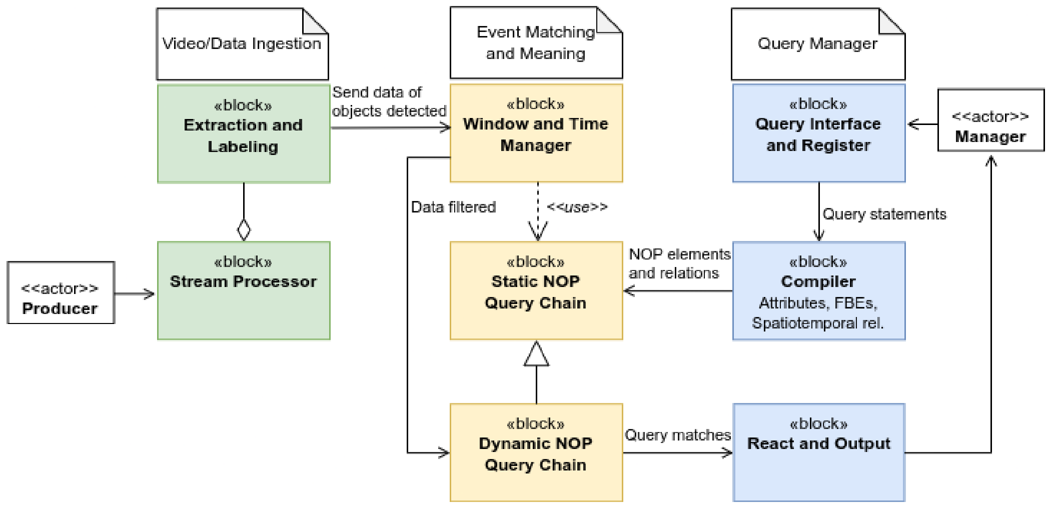
This section proposes and discusses the Notification-oriented Querying Method (NOP Query), a new processing method for streaming queries. NOP Query supports automatic query matching when data arrive in the system, thus avoiding intensive database operations.

4.1. NOP Query Overview

As shown in Section 3, the NOP inference chain is the "core of NOP", responsible for managing the notification mechanism when any *Attribute* value is changed. The proposed NOP Query inherits the advantages of this paradigm by using/adapting the features of the C++ 4.0 NOP Framework, naturally differing from existing video query methods. As the NOP Query is NOP-compatible, it is natively reactive. It avoids loop-based encoding and unnecessary logic evaluations, thus reducing wasted processing power and potentially allowing for fine-grained processing distribution. In fact, as a high-performance decoupling mechanism, NOP Query considers changes to *Attribute* to initiate the notification inference chain and promptly calculates alerts when the query matches.

Figure 4 provides an overview of the NOP Query method comprising three parts: Video/Data Ingestion, Event Matching and Meaning, and Query Manager. Video/Data ingestion is responsible for reading a video stream or a set of CSV data, extracting data, and defining features for all objects in the frame. Deep learning inference models can be used to detect objects and attributes. It calculates the frame number, object label, color, ID, and bounding box values (e.g., positions). In turn, the Query Manager handles queries from a user/manager and generates results when a query matches the event in the data/video stream. Finally, Event Matching and Meaning is the core of NOP Query and consists of three modules:

1. The Window and Time Manager receives input from the data ingestion and filters it according to the time restrictions and data values of the Static NOP Query Chain;
2. The Static NOP Query Chain (SQC) is responsible for receiving queries from the Window and Time Manager and creating a static structure that represents the manager's query following the NOP Chain schema;
3. The Dynamic NOP Query Chain (DQC) is responsible for receiving data from the Window and Time Manager, checking the data flow against the SQC, computing notifications, and sending warnings when the data match the SQC.



Video/Data Ingestion
Event Matching and Meaning
Query Manager
«block» Extraction and Labeling
Send data of objects detected
«block» Window and Time Manager
«block» Query Interface and Register
<<actor>> Manager
Data filtered
<<use>>
Query statements
«block» Stream Processor
<<actor>> Producer
«block» Static NOP Query Chain
NOP elements and relations
«block» Compiler Attributes, FBEs, Spatiotemporal rel.
«block» Dynamic NOP Query Chain
Query matches
«block» React and Output

Figure 4. Notification-oriented querying method overview.

The next two sections present more detailed information about SQC and DQC, since they are the essence of the NOP Query.

4.2. Static NOP Query Chain (SQC)

The SQC is a static schema of the query expression that describes the elements of the query and the corresponding values and associations between them, represented as an NOP Chain. Consider a hypothetical query that finds a "red car on the left below a white truck" expressed in the Listing 1 using the SQL-like format. Currently, the "NOP Query language" is a prototype under development.

Listing 1. Query example for finding a red car to the left below a white truck.

```
SELECT LEFT BELOW(Object[1], Object[2])
FROM DataStream
WHERE Object[1].label='car'
AND Object[1].color='red'
AND Object[2].label='truck'
AND Object[2].color='white'
```

Based on this query, Figure 5 presents an overview of the SQC creation process. Using the Symbol Table, the Lexical Analyzer examines the query expression and separates the elements and their corresponding values. The query can be represented in two predicates because two vehicles are recognized *FBEs*: "red car left below" and "white truck". Considering these elements, the resulting tokens belong to *FBEs* and their attributes, spatial relations, and the action "find". In the first step, the Notification Chain Assembler (NCA) assembles the Static *NOP* Query Chain with *Attributes* for object names/labels (car and truck) and colors (red, white).

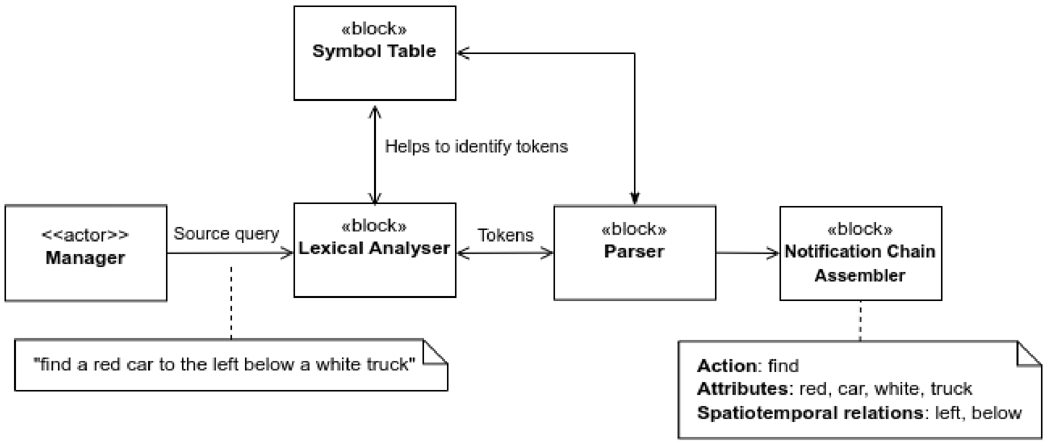

«block»
Symbol Table
Helps to identify tokens
<<actor>>
Manager
Source query
«block»
Lexical Analyser
Tokens
«block»
Parser
«block»
Notification Chain Assembler
"find a red car to the left below a white truck"
Action: find
Attributes: red, car, white, truck
Spatiotemporal relations: left, below

Figure 5. Overview of the process for creating the static NOP query chain.

In the second step, the NCA assembles the corresponding *Premises* for labels, colors, and position values (X and Y), following the scheme shown in Figure 6, using a 2D coordinate system of the computer vision:

- *Premise* 1: "car left truck", "*Attribute X* (*FBE* 1) $<$ *Attribute X* (*FBE* 2)";
- *Premise* 2: "car below truck", "*Attribute Y* (*FBE* 1) $>$ *Attribute Y* (*FBE* 2)";
- *Premise* 3: "*Attribute Color* (*FBE* 1) = red";
- *Premise* 4: "*Attribute Label* (*FBE* 1) = car";
- *Premise* 5: "*Attribute Color* (*FBE* 2) = white";
- *Premise* 6: "*Attribute Label* (*FBE* 2) = truck".

In the third step, the NCA creates the *Condition* with all these *Premises* and sets the conjunction operator. In the fourth step, the NCA creates the *Rule* with the associated *Condition*, *Instigation*, and *Action* elements. Finally, the *Method* is created to display a warning message if the query is matched. The result of the SQC process with all these entities and expected states required for query matches is shown in Figure 7.

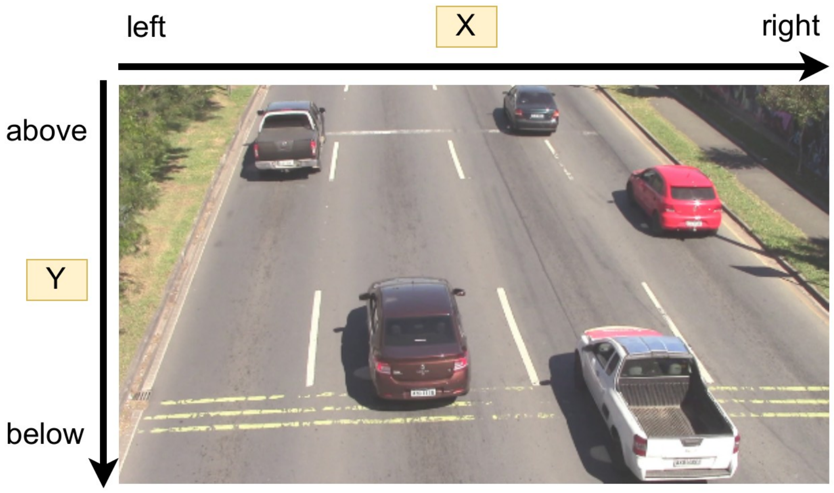


Figure 6. Spatial position coordinates perspective. Source: Original image of the UTFPR-HSD public dataset available at http://labic.utfpr.edu.br/datasets/UTFPR-HSD.html, accessed on 16 December 2023.

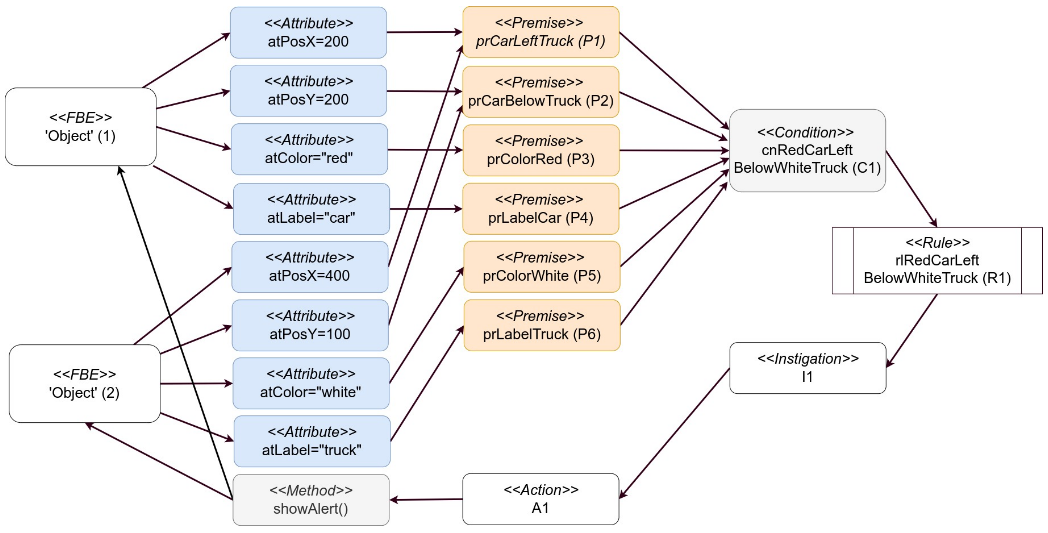

<<FBE>> 'Object' (1)
<<FBE>> 'Object' (2)
<<Attribute>> atPosX=200
<<Attribute>> atPosY=200
<<Attribute>> atColor="red"
<<Attribute>> atLabel="car"
<<Attribute>> atPosX=400
<<Attribute>> atPosY=100
<<Attribute>> atColor="white"
<<Attribute>> atLabel="truck"
<<Method>> showAlert()
<<Premise>> prCarLeftTruck (P1)
<<Premise>> prCarBelowTruck (P2)
<<Premise>> prColorRed (P3)
<<Premise>> prLabelCar (P4)
<<Premise>> prColorWhite (P5)
<<Premise>> prLabelTruck (P6)
<<Action>> A1
<<Condition>> cnRedCarLeft BelowWhiteTruck (C1)
<<Rule>> rlRedCarLeft BelowWhiteTruck (R1)
<<Instigation>> I1

Figure 7. Result of the static NOP query chain. The position values are suggested according to the query needs.

4.3. Dynamic NOP Query Chain (DQC)

DQC receives filtered Window and Time Manager data and checks each object according to the SQC structure. It is dynamic because the NOP chain connections between the elements are assembled as the data are ingested, and when a new ID/object appears and matches, even partially, the query. When an object matches all or part of the first or second predicate query, the DQC must be updated. The object ID is needed in the current version to match the elements and create the DQC connections.

For example, if an object/*FBE* with label or color matches the SQC structure (e.g., atLabel = "car" or atColor = "red"), the DQC adds that *FBE* to the DQC and connects it to all other existing *FBE* Trucks. In this way, each object relevant to the SQC belongs to a "dynamic network" with other related objects defined by the query. The following procedure is performed for each new *FBE* Car in the system:

- Check every *FBEs* "Truck" in the DQC;
- For each *FBE* Truck present, then connects it to the current FBE Car;
- If this connection exists, update the corresponding *Attributes* and start the notification mechanism.

A similar process occurs when the object belongs to the second predicate of the query, but the match is made in all the *FBEs* of the first predicate. When two *FBEs* are combined to form DQC once, the corresponding *Premises* P3 and P5 (color) and *Premises* P4 and P6 (label) are created (see Figure 7). In the next step, the corresponding *Condition*, *Rule*, *Instigation*, *Action*, and *Method* will be created. SQC occurs once and DQC occurs when each new relevant object arrives from the stream.

5. Case Study

We have developed a case study to allow empirical evaluation of the proposed NOP Query. It aims to evaluate the performance of the NOP Query, the memory usage, and the *Rules* produced, as well as to test and debug the prototypes. The tests were conducted carefully using empirical validation.

5.1. The NS100KCOLORID Dataset

In the literature, the Noscope (Jackson Town Square) (https://github.com/stanford-futuredata/noscope (accessed on 16 December 2023)) [3] is one of the most popular datasets used in query experiments about traffic surveillance in general. It contains 60 h of surveillance video and the respective labeled CSV dataset with annotations for each object (event) found in each frame. The CSV file contains 1,064,237 events (object detections) in 6,426,647 frames with the following fields: *frame*, *objectname*, *confidence*, *xmin*, *ymin*, *xmax*, and *ymax*.

The Noscope dataset is adequate for simple object-related queries and spatiotemporal attributes. However, the query expressed in Section 4.2 needs more information: the object's color and the ID of the objects. The last is required by the DQC (Section 4.3) to create the connections between *FBEs* dynamically. It is worth mentioning that other related works in the literature also use the ID and color of objects [33,53,54].

Therefore, a new dataset was created for this case study from around 10% of the original Noscope dataset. The new NS100KCOLORID dataset contains 102,835 labeled events in 858,806 frames. This subset was improved by including two more fields: the predominant color of the vehicle (red, blue, green, yellow, white, gray, black, pink, and teal) and a unique Id of each object obtained using the DeepSort algorithm [67]. Furthermore, aiming at improving the data quality, only the events with at least 50% confidence were added, following the literature approach to computing objects above this confidence in a video stream [54].

The NS100KCOLORID dataset offers more data to explore the capabilities of the NOP Query. Several tests can be performed focused on CPU usage, memory consumption, and reactivity to different streaming video queries. The dataset generated during the current study is available in the GitHub repository at the following link (https://github.com/bioinfolabic/NS100KCOLORID (accessed on 16 December 2023)). It is another contribution of this paper.

There are no other large, public datasets available in the related literature. Most of the authors used streams from surveillance cameras on Youtube that are no longer available.

5.2. Preparatory Steps

The experiments presented in this section were based on similar work in the literature [6,53,54]. These authors performed queries and tests with the operators *objects*, *conj*, *left*, *seq*, and *count*. Table 1 describes and exemplifies these operators using a SQL-like notation created by the authors above:

- The *object* operator retrieves the frame of interest within a time window. For instance, the query example selects/finds the frame of an object with the label "car" within a time window of 10 s, considering each second is composed of 30 frames. This operator is the most simple one and evaluates only one attribute.
- The *conj* or conjunction operator retrieves the frame of interest, evaluating objects and attributes within the time window. The query example selects the frame in which an

object appears with the label "car" and another object with the label "truck" within a 10 s time window.

- The $left$ operator retrieves the frame of interest, evaluating the spatial relationship "left" between objects in the same frame. The query example selects the frame in which an object with the label "car" appears at the left of the object with the label "truck" within a 10 s time window.
- The *seq* operator retrieves the frame of interest, evaluating their temporal relationship in a given time window interval. The query example alerts if an object with the label "car" and another with the label "truck" appear within the same time window of 10 s.
- The *count* operator is the most complex than the former ones, and it counts the number of objects, evaluating their temporal occurrence within the same time window interval. The count of objects should be consistent across several frames, denoted by each frame clause. The query example counts any objects that appeared within a time window of 10 s and alerts if the count is greater or equal to 5.
- The *window* operator also establishes a time interval scope in which the queries are performed based on frame numbers or timestamps. It is often used to limit the time interval to reduce the computational cost of searches. This operator restricts the time window size of the entire query.

Table 1. Operators and queries used in the experiments following SQL-like syntax.

Operator	**Query Example**
OBJECTS	$SELECT\ Object$ $FROM\ Data\_stream$ $WHERE\ Object.\ label\ =\ \text{“}car\text{”}$
CONJ	$SELECT\ Object[1],\ Object[2]$ $FROM\ Data\_stream$ $WHERE\ Object[1].\ label\ =\ \text{“}car\text{”}\ AND$ $Object[2].\ label = \text{“}truck\text{”}$
LEFT	$SELECT\ LEFT(Object[1], Object[2])$ $FROM\ Data\_stream$ $WHERE\ Object[1].\ label = \text{“}car\text{”}\ AND$ $Object[2].\ label\ =\ \text{“}truck\text{”}$ $WITHIN\_TIME\_WINDOW\ =\ 10\text{ s}$
SEQ	$SELECT\ SEQ(Object[1], Object[2])\ FROM$ $Data\_stream$ $WHERE\ Object[1].\ label\ =\ \text{“}car\text{”}\ AND$ $Object[2].\ label\ =\ \text{“}truck\text{”}$ $WITHIN\_TIME\_WINDOW\ =\ 10\text{ s}$
COUNT	$SELECT\ COUNT(Object)\ FROM\ Data\_stream$ $WHERE\ Object.\ label\ LIKE\ \text{“}\%\text{”}\ AND$ $COUNT(Object) >=\ 5$ $FOREACH\_FRAME$ $WITHIN\_TIME\_WINDOW\ =\ 10\text{ s}$

Below, we briefly explain each *Rules* required per query.

- The operator *objects* require one *Rule* per target object/attribute.
- Both operators, *left* and *conj*, require one *Rule* for each pair of target objects.
- Three *Rules* are needed for the *seq* operator. *Rule* (a) counts the number of times the objects appear within the time window. *Rule* (b) fires when the *seq* count is reached. *Rule* (c) resets the count when the *seq′ Rule* fires.
- The operator *count* also requires three *Rules*. *Rule* (a) counts the number of objects in the same frame. *Rule* (b) checks if the count is consistent for each frame and fires when it matches the number of objects in the time window. *Rule* (c) resets the count operator when the *count′ Rule* fires.

5.3. Experimental Results

The prototype system was implemented in the C++ programming language using the g++ 11 compiler, NOP C++ Framework 4.0, and Linux Ubuntu 20.04 Kernel 5.4.0-125-generic. The experiments were carried out on a computer with an Intel Core i7-8750H CPU and 16 GB of RAM. It is important to note that the GPU was not used in this current experiment version. GPUs were not used because our method only focuses on query processing, in which GPUs could be useful but are not fundamental. Actually, focusing just precisely on query processing, the activities of data ingestion and object detection were not considered, being by nature those activities the GPUs demand.

We use the operators mentioned earlier for two different experiments. The first is aimed at establishing some comparison with the literature. Therefore, it is limited to the scope and operators of the time window size found in [54]. In that work, the authors tested six different time windows (5 s, 10 s, 25 s, 1 min, 5 min, 10 min). The second experiment uses a broader range of time windows, from 10 min to 190 min, and it is aimed to estimate the computational complexity of the proposed approach experimentally.

5.3.1. Comparison with the Literature

Figure 8 shows the CPU processing time considering the same queries, operators, and window sizes common in the literature [54]. A total of 300 tests were carried out, and the results shown correspond to the average of 10 runs per query and different time windows. We used four trend curves for the left operator, which is more expensive: linear, polynomial, logarithmic, and exponential. The processing time considers the construction of the NOP query, the search time, and the reaction time when the query is matched.

Unfortunately, from these results, it is not possible to infer the computational cost because the time windows are short. On the other hand, the literature results of [54] show an exponential cost from 1 min onwards.

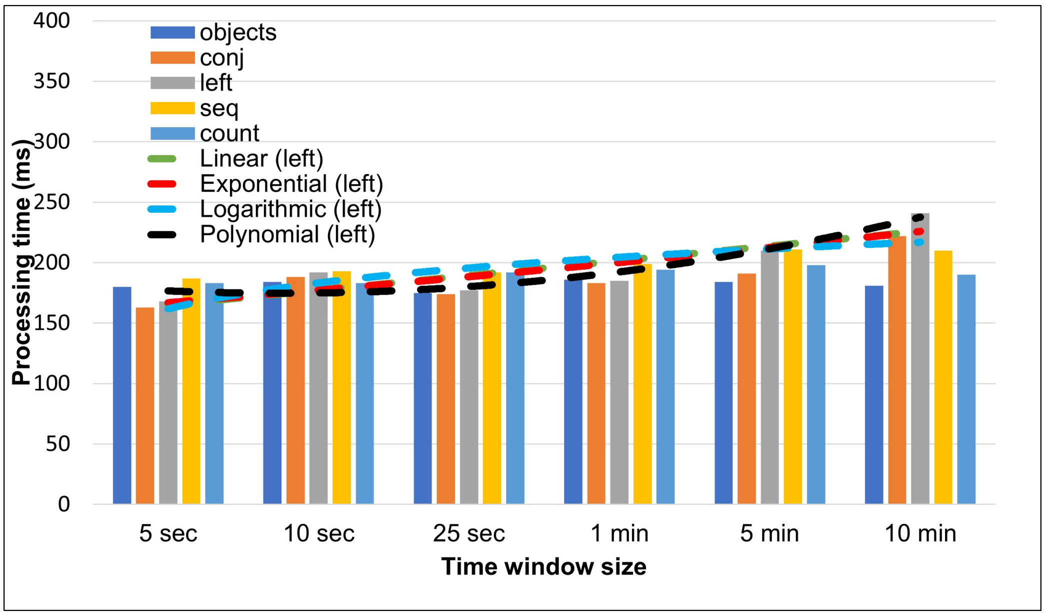


Figure 8. Processing time results considering query construction and search time using short time window sizes.

5.3.2. Estimation of the Computational Complexity

The second experiment set aims to estimate the computational complexity of CPU and memory usage as well as the number of *Rules* created considering large time window sizes, from 10 min to 190 min. A total of 950 runs were made, and the results shown correspond to the average of 10 runs per query and time window size. We also used four curve fits to estimate the computational cost: linear, polynomial, logarithmic, and exponential. As mentioned earlier, the *left* operator was also used to estimate the trend lines. In this version of the experiments, the operators involving more than one object are configured on all the objects that match the query, even partially. Therefore, the tests were extensive and explored all the combinations to find out their processing capacity at high workloads.

Figure 9 shows the results of the CPU processing time considering the construction of the NOP query and the search time. Please observe that the *left*, *conj*, and *seq* operators are expensive because they evaluate pairs of objects within the time window size. In particular, the *left* operator is more costly because it evaluates the spatial positions of the objects and their *Attribute* values. In this application, positioning changes are constant because the objects are moving. In turn, the *objects* and *count* operators are the least expensive because the *objects* operator only evaluates one object at a time, and the *count* operator only adds the counter when an object appears in the time window. These results suggest that the computational complexity grows polynomially. Therefore, this approach performs better than others in the literature.

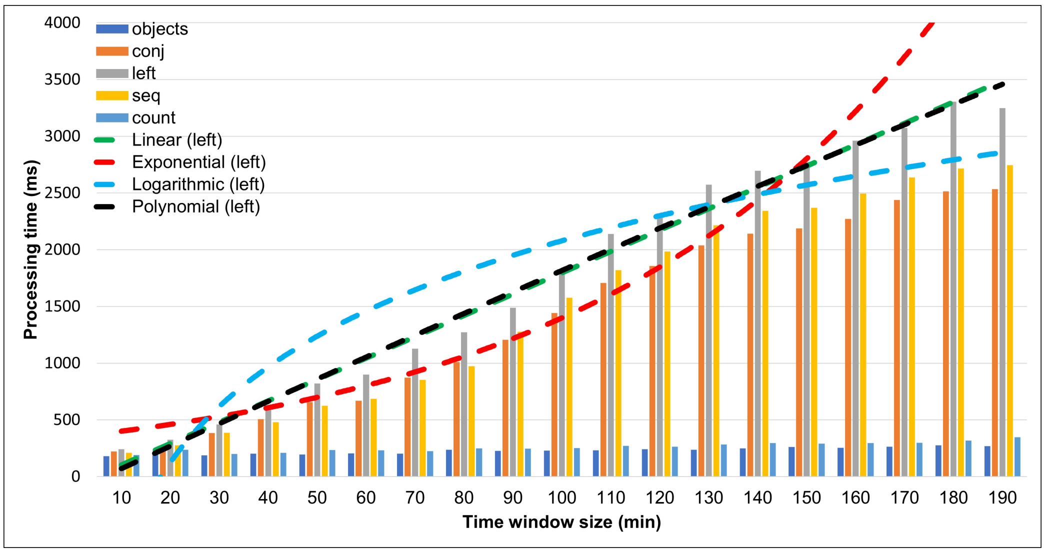


Figure 9. CPU processing time results considering the NOP Query construction and search time using large time window sizes.

Figure 10 presents the results regarding memory usage, keeping in mind that the current version of the experiment does not have any garbage collector when the objects and the respective NOP elements are not used. This strategy was defined to test the application in a CPU-intensive and memory-consuming setting. It was observed that the *left*, *conj*, and *seq* operators require more memory because they evaluate pairs of objects. Once again, it can be observed that the *objects* and *count* operators perform with constant computational cost because the dataset contains a similar distribution of objects across time window sizes.

Figure 11 presents the amount of *Rules* produced. The *Rules* are created when a new object matches, even partially, with the query expression. It means that the number of *Rules* created was high due to the lack of a garbage collector. If some strategy for clearing unused or "old" NOP elements from memory were used, the final number of *Rules* would be smaller, but the real potential would not be known. In this way, it is possible to see how the system behaves with a large mass of data processed.

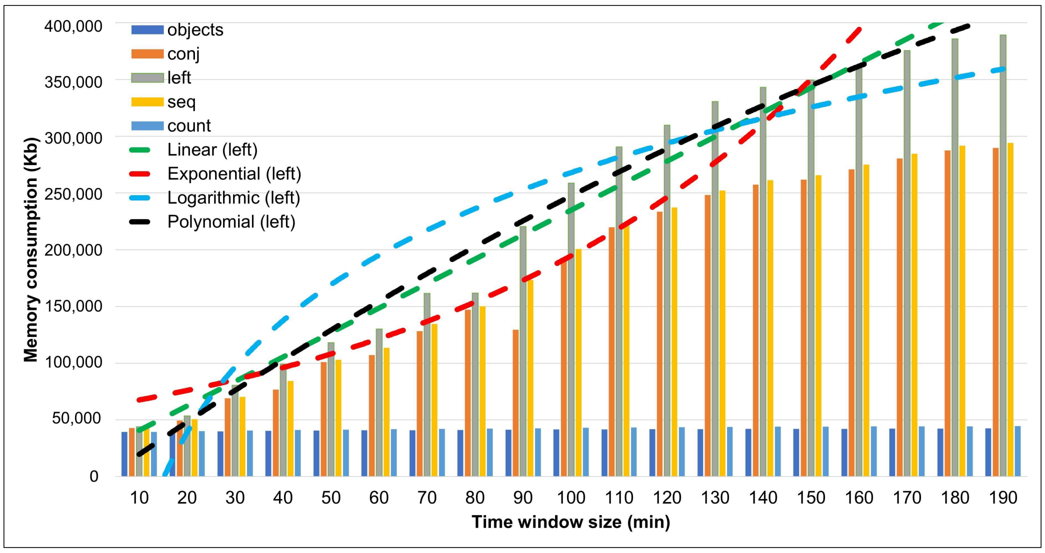


Figure 10. Memory usage results considering large time window sizes.

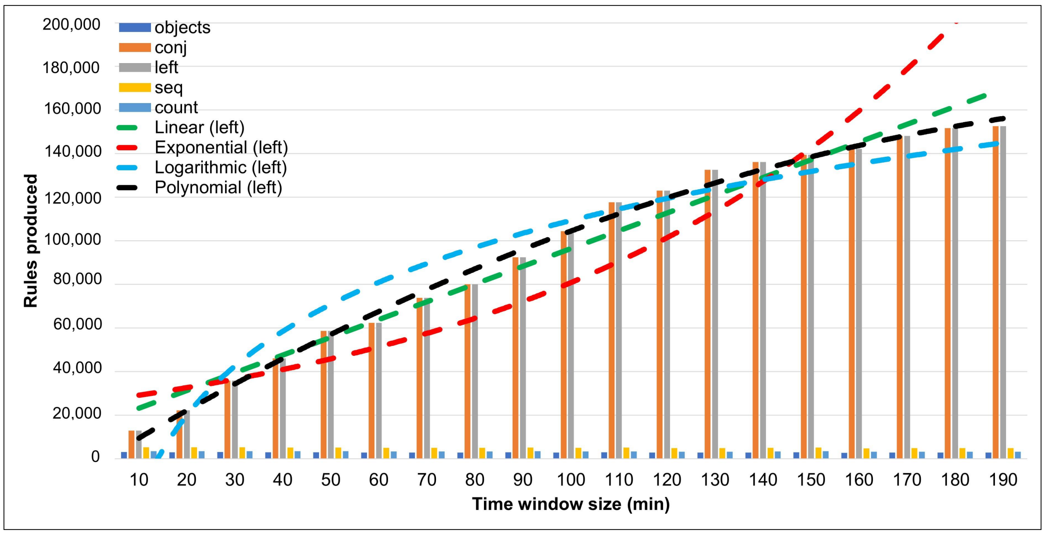


Figure 11. Number of *Rules* created considering large time window sizes.

5.4. Discussion of Experimental Results

In Section 3.1, we highlight the features and advantages of NOP in avoiding structural and temporal redundancies to reduce wasted processing and unnecessary evaluations. The promising results in the case study corroborate the theoretical concepts of NOP, suggesting that the proposed approach can be very suitable for surveillance and query applications regarding computational cost and latency. Furthermore, the current version of the experiment runs on a server-only architecture. Still, the distributed properties of NOP elements may allow edge-first or even fully distributed architectures in future versions, further optimizing the available resources.

It is worth noting that the current version of this project does not use multithreads or multiprocessing. However, the results are very proper already with single threads. The results of our experiments show the computational complexity curves as the workload increases. As the graphs show, even if the computational cost does not tend to be linear, it is polynomial and not exponential, thereby being treatable. Still, as mentioned above, the results obtained in the literature have an exponential computational cost. Therefore, our approach is better than those from the literature.

Besides, by induction and by the NOP literature, if more processing cores are available, the results tend toward logarithmic complexity. In turn, logarithmic complexity grows much more slowly than current polynomial complexity. Thus, a priori, the results could still be better. Regardless, the use of multithreads or multiprocessing with our approach is envisaged as future work.

The current performance limiting factor is the memory consumption required to connect all the NOP entities for each identified object. In this sense, Figure 2 (NOP Inference Chain) and Figure 7 (Result of the Static NOP Query Chain) show how the connection between the NOP entities occurs. Each entity is an object relating to other objects, and this requires memory space. Still, as mentioned in the paper, the current version of this work does not have the garbage collector feature precisely in order to test the "worst" possible scenario. Even so, the results are very promising, as stated in the experiment results in the paper. In any case, the literature approaches also use a lot of memory due to their idiosyncrasies and much more processing resources due to their computational paradigm orientation.

Furthermore, our approach offers manageable performance even when dealing with a lot of data/object movement by taking advantage of the NOP properties in the terms explained in above. As a matter of fact, this is another advantage over literature.

Moreover, the analysis of the experimental results suggests the following highlights:

- The computational complexity of the proposed approach does not grow exponentially, and it is treatable between polynomial and logarithmic.
- Only when some *Attribute* value changes (e.g., label, color, or positions), the corresponding *Premise* is notified to reevaluate its state and fire a notification if the new state is different. Because of such behavior, the system performance is high, and the latency is low.
- Unlike the literature, the latency is low for short and long window sizes.
- It is possible to develop complex event queries using NOP *Rules*.
- The NOP Query works as an active database without permanent triggers. It is the foundation of a new technology called "Reactive database based on NOP" under development.
- This querying method is focused on real-time processing.

6. Comparison of the NOP Query with Related Technologies and Possible Applications

The best way to discuss NOP Query in terms of its concepts and advantages is to compare it with related technologies so that it is possible to foresee possible future applications. A priori, NOP Query can be used in any domain where the system receives inputs and needs to process them with low latency to compute event/semantic information or inference.

6.1. Active Database × NOP Query

According to [68], traditional relational databases are passive, providing data only when explicitly requested by users or applications. In contrast, active databases consist of a series of triggers that are fired when certain conditions occur. They were developed to overcome the limitations of relational databases. Interestingly, most modern relational databases include active database features in the form of database triggers. However, this approach is challenging to maintain due to the complexity of understanding the effects of these triggers. Otherwise, the NOP Chain, the core of the NOP Query, is succinct, easy to understand, and easily maintainable. The chain of notifications starts when some *Attribute* changes its value and notifies its related *Premise*(s), feeding the notification chain. The NOP Query can be an engine for a new type of (Re)active Database in which the Rules and related NOP elements are the central focus instead of triggers.

6.2. Knowledge Graph × NOP Query

Knowledge graphs are a structured representation of facts consisting of entities, relations, and semantic descriptions. Entities can be real objects or abstract concepts, edges represent the relationship between entities and semantic descriptions contain properties with well-defined meanings [69]. They are frequently used in various domains because of their rich structured knowledge. The technology most closely related to NOP Query is the knowledge graph, as shown in [53,54]. Both technologies can express objects and relationships through entities/nodes and connections. Future work includes extending NOP Query to function as a new type of database system and comparing its performance with other related technologies in terms of latency, especially in real-time, distributed, and Big Data contexts.

6.3. Information Retrieval × NOP Query

Information retrieval aims to find relevant documents in a large collection, minimizing manual effort [70]. Meanwhile, NOP Query also performs data/event retrieval following a different method from the existing literature, especially for applications where the entries are regular, as the start of the notification chain depends on the change in the values of *Attribute*(s). The NOP query can also support information retrieval offline, e.g., without regular entries, but requires further experimentation to adapt to this context.

6.4. Big Data × NOP Query

In the literature, video streaming applications also belong in the big data context due to the high speed with which videos are generated, the large video files stored, and the variety of video formats and sources [7]. NOP Query's features are suitable for the big data context due to the reduced computational cost, the distributed elements (e.g., parallelizable), and the fact that they avoid intensive writing to and reading from secondary memory, following current trends in big data frameworks, such as Apache Spark (https://spark.apache.org/ (accessed on 16 December 2023)) and Apache Storm (https://storm.apache.org/ (accessed on 16 December 2023)).

7. Conclusions

Streaming video queries have been one of the most challenging research areas in computer vision, software engineering, and big data. In recent years, several approaches to video queries have emerged. However, none have become popular due to many limitations related to the high latency between the user's query and information. These approaches also require intensive CPU and memory usage, several database operations per second, complex data structures, and high network usage. In addition, the related solutions are usually created as monolithic structures. Although GPU processing enables fast computing, existing video query approaches are still expensive for real-time processing. Considering these aspects, this work presents the following contributions.

First, a new lean query processing method called NOP Query (Notification-oriented Querying Method) was presented as a straightforward and intuitive approach to dealing with streaming video queries. As far as we know, this is the pioneering work based on NOP for this context. NOP Query inherits important features from NOP, such as formalism and tools that are constantly being improved. Additionally, this work has demonstrated the viability of NOP Query for applications that require resource optimization and fast processing, especially in the context of video queries and event matching.

Next, a new dataset based on the popular Noscope has been developed by adding two more fields (color and ID), making it more suitable for this type of experiment. The NS100KCOLORID dataset is available to the community to promote further research and comparisons.

Then, we carried out systematic experiments and compared the results with the literature. In this way, we evaluated the suitability of the NOP Query method and its potential for processing large amounts of data and different types of queries. These experiments can be useful for future work and comparisons with other emerging technologies.

Finally, NOP Query is a prominent application that creates and manages thousands of *Rules*, simultaneously demonstrating the NOP Framework C++ 4.0's ability to process complex queries.

Future Work

As far as we know, this work is the first to focus solely on video event data and query processing without considering tasks such as pre-processing, convolutional neural network costs, and database operations. Therefore, the proposed solution and results show the real computational cost without the burden of third-party algorithms. Future work will consider testing the NOP Query on a real-world surveillance system with many video streams, using an entire computer vision pipeline, including GPU processing. NOPL support will be added, and the results will be compared with the C++ 4.0 NOP Framework results. An interesting extension of the current work includes the development of a new "NOP-based (re)active database", which avoids permanent triggers. It also includes the addition of other query operators, such as the one recently published in [4,6,32]:

- The *selection* operator allows the selection of particular objects or events of interest. For example, users may search for instances of people.
- The *aggregation* operator allows computing some statistics over the video frames. Common aggregations include count, sum, average, maximum, and minimum operators. For example, a city planner may be interested in computing the average number of cars per frame or counting the number of pedestrians that cross in front of a camera.
- The *limit* operator allows finding a cardinality-limited number of events occurring within some time interval. For example, a city planner may search for ten instances of buses at stop signs.
- The *similatiry* operator allows searching for portions of the video similar to an input frame or video clip. For example, a football analyst may input a frame or video clip of a goal of a soccer player. Such queries often involve iterative, ad hoc analysis to arrive at the final query.
- The *join* operator allows performing a join and, subsequently, a *selection*, *aggregation*, or *limit* query. For example, an amber alert application may join extracted license plates with an external data source.
- The *projection* operator extracts only a part of the information to compute video events.

In addition, other possible use cases for NOP Query include the domain of Complex Event Processing (CEP) systems, also known as event stream processing. This technology is used to query data before storing them in a database or, in some cases, without storing them. The main objective of CEP is to identify significant events, such as opportunities or threats, in real-time situations and quickly respond to them. More details about CEP systems can be found at [68,71]. Examples of CEP systems include:

- Stock market trading systems: real-time analysis of market data;
- Mobile devices: handling incoming streams of events;
- Fraud detection: identifying suspicious patterns;
- Business activity monitoring: monitoring business processes;
- Security monitoring: detecting anomalies;
- Transportation industry: real-time tracking;
- Governmental intelligence gathering: analyzing diverse data sources.

Author Contributions: Conceptualization, C.K., J.M.S. and H.S.L.; methodology, C.K., J.M.S. and H.S.L.; software and experiments, C.K.; validation, C.K. and H.S.L.; formal analysis, C.K. and H.S.L.; investigation, C.K.; resources, C.K.; data curation, C.K.; writing—original draft preparation, C.K.; writing—review and editing, C.K., H.S.L. and J.M.S.; visualization, C.K.; supervision, H.S.L.; project administration, C.K.; funding acquisition, H.S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The dataset generated during the current study is available in the GitHub repository at the following link https://github.com/bioinfolabic/NS100KCOLORID (accessed on 16 December 2023).

Acknowledgments: This study was financed in part by the Coordination for the Improvement of Higher Education Personnel–Brazil (CAPES).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Mittal, S.; Vaishay, S. A survey of techniques for optimizing deep learning on GPUs. *J. Syst. Archit.* **2019**, *99*, 101635. [CrossRef]
2. Dong, S.; Wang, P.; Abbas, K. A Survey on Deep Learning and Its Applications. *Comput. Sci. Rev.* **2021**, *40*, 100379. [CrossRef]
3. Kang, D.; Emmons, J.; Abuzaid, F.; Bailis, P.; Zaharia, M. NoScope: Optimizing Neural Network Queries over Video at Scale. *Proc. VLDB Endow.* **2017**, *10*, 1586–1597. [CrossRef]
4. Kang, D.; Bailis, P.; Zaharia, M. BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics. *Proc. VLDB Endow.* **2019**, *13*, 533–546. [CrossRef]
5. Hsieh, K.; Ananthanarayanan, G.; Bodik, P.; Venkataraman, S.; Bahl, P.; Philipose, M.; Gibbons, P.B.; Mutlu, O. Focus: Querying large video datasets with low latency and low cost. In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, Carlsbad, CA, USA, 8–10 October 2018; pp. 269–286.
6. Yadav, P.; Curry, E. VEKG: Video event knowledge graph to represent video streams for complex event pattern matching. In Proceedings of the IEEE First International Conference on Graph Computing, Laguna Hills, CA, USA, 25–27 September 2019; pp. 13–20.
7. Alam, A.; Ullah, I.; Lee, Y.K. Video Big Data Analytics in the Cloud: A Reference Architecture, Survey, Opportunities, and Open Research Issues. *IEEE Access* **2020**, *8*, 152377–152422. [CrossRef]
8. Pouyanfar, S.; Sadiq, S.; Yan, Y.; Tian, H.; Tao, Y.; Reyes, M.P.; Shyu, M.L.; Chen, S.C.; Iyengar, S.S. A Survey on Deep Learning: Algorithms, Techniques, and Applications. *ACM Comput. Surv.* **2018**, *51*, 1–36. [CrossRef]
9. Aafaq, N.; Mian, A.; Liu, W.; Gilani, S.Z.; Shah, M. Video description: A survey of methods, datasets, and evaluation metrics. *ACM Comput. Surv.* **2019**, *52*, 1–37. [CrossRef]
10. Lu, C.; Liu, M.; Wu, Z. SVQL: A SQL extended query language for video databases. *Int. J. Database Theory Appl.* **2015**, *8*, 235–248. [CrossRef]
11. Stonebraker, M.; Bhargava, B.; Cafarella, M.; Collins, Z.; McClellan, J.; Sipser, A.; Sun, T.; Nesen, A.; Solaiman, K.; Mani, G.; et al. Surveillance video querying with a human-in-the-loop. In Proceedings of the Workshop on x Human-in-the-Loop Data Analytics, Portland, OR, USA, 14–19 June 2020; pp. 14–19.
12. Hsieh, K. Machine Learning Systems for Highly-Distributed and Rapidly-Growing Data. Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2019.
13. Yadav, P. High-performance complex event processing framework to detect event patterns over video streams. In Proceedings of the 20th International Middleware Conference Doctoral Symposium, Davis, CA, USA, 9–13 December 2019; pp. 47–50.
14. Hwang, J.; Kim, M.; Kim, D.; Nam, S.; Kim, Y.; Kim, D.; Sharma, H.; Park, J. CoVA: Exploiting Compressed-Domain Analysis to Accelerate Video Analytics. In Proceedings of the USENIX Annual Technical Conference, USENIX Association, Carlsbad, CA, USA, 11–13 July 2022; pp. 707–722.
15. Farhadi, A.; Redmon, J. Yolov3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767.
16. Linhares, R.R.; Pordeus, L.F.; Simão, J.M.; Stadzisz, P.C. NOCA A Notification-Oriented Computer Architecture: Prototype and Simulator. *IEEE Access* **2020**, *8*, 37287–37304. [CrossRef]
17. Simão, J.M. A Contribution to the Development of a HMS Simulation Tool and Proposition of a Metal-Model for Holonic Control. Ph.D. Thesis, Centro Federal de Educação Tecnológica do Paraná (CEFET-PR) and Henri Poincaré University (UHP), Curitiba, Brazil, 2005.
18. Simão, J.M.; Tacla, C.A.; Stadzisz, P.C. Holonic control metamodel. *IEEE Trans. Syst. Man Cybern.-Part A Syst. Hum.* **2009**, *39*, 1126–1139. [CrossRef]
19. Simão, J.M.; Stadzisz, P.C. Inference based on notifications: A holonic metamodel applied to control issues. *IEEE Trans. Syst. Man Cybern.-Part A Syst. Hum.* **2008**, *39*, 238–250. [CrossRef]
20. Belmonte, D.L.; Ronszcka, A.F.; Linhares, R.R.; Banaszewski, R.F.; Tacla, C.A.; Stadzisz, P.C.; Batista, M.V. Notification-oriented and object-oriented paradigms comparison via sale system. *J. Softw. Eng. Appl.* **2012**, *5*, 695–710.
21. Simão, J.M.; Tacla, C.A.; Stadzisz, P.C.; Banaszewski, R.F. Notification oriented paradigm (NOP) and imperative paradigm: A comparative study. *J. Softw. Eng. Appl.* **2012**, *5*, 402–416. [CrossRef]
22. Ronszcka, A.F.; Banaszewski, R.F.; Linhares, R.R.; Tacla, C.A.; Stadzisz, P.C.; Simão, J.M. Notification-oriented and Rete network inference: A comparative study. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Hong Kong, China, 9–12 October 2015; pp. 807–814.
23. Neves, F.S. Framework PON C++ 4.0: Contribuição para concepção de aplicações no Paradigma Orientado a Notificações por meio de programação genérica. Master's Dissertation, Federal University of Technology-Paraná (UTFPR), Curitiba, Brazil, 2021. (In Portuguese)

24. Negrini, F.; Ronszcka, A.F.; Linhares, R.R.; Fabro, J.A.; Stadzisz, P.C.; Simão, J.M. NOPL-Erlang: Programação multicore transparente em linguagem de alto nível. *Cad. Do IME-Série Informática* **2019**, *43*, 70–74. (In Portuguese)
25. Schütz, F.; Fabro, J.A.; Ronszcka, A.F.; Stadzisz, P.C.; Simão, J.M. Proposal of a declarative and parallelizable artificial neural network using the notification-oriented paradigm. *Neural Comput. Appl.* **2018**, *30*, 1715–1731. [CrossRef]
26. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
27. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]
28. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
29. Chollet, F. *Deep Learning with Python*; Manning: Shelter Island, NY, USA, 2021.
30. Jiang, J.; Ananthanarayanan, G.; Bodik, P.; Sen, S.; Stoica, I. Chameleon: Scalable adaptation of video analytics. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication, Budapest, Hungary, 20–25 August 2018; pp. 253–266.
31. Canel, C.; Kim, T.; Zhou, G.; Li, C.; Lim, H.; Andersen, D.G.; Kaminsky, M.; Dulloor, S.R. Scaling Video Analytics on Constrained Edge Nodes. In Proceedings of the 2nd SysML Conference, Stanford, CA, USA, 31 March–2 April 2019.
32. Kang, D.; Romero, F.; Bailis, P.; Kozyrakis, C.; Zaharia, M. VIVA: An End-to-End System for Interactive Video Analytics. In Proceedings of the 12th Conference on Innovative Data Systems Research (CIDR), Chaminade, CA, USA, 9–12 January 2022.
33. Yadav, P. Query-Aware Adaptive Windowing for Spatiotemporal Complex Video Event Processing for Internet of Multimedia Things. Ph.D. Thesis, University of Galway, Galway, Ireland, 2021.
34. Baloi, A.; Belean, B.; Turcu, F.; Peptenatu, D. GPU-based similarity metrics computation and machine learning approaches for string similarity evaluation in large datasets. *Soft Comput.* **2023**, *28*, 3465–3477. [CrossRef]
35. Perez-Cham, O.E.; Puente, C.; Soubervielle-Montalvo, C.; Olague, G.; Aguirre-Salado, C.A.; Nuñez-Varela, A.S. Parallelization of the honeybee search algorithm for object tracking. *Appl. Sci.* **2020**, *10*, 2122. [CrossRef]
36. Wu, R.; Guo, X.; Du, J.; Li, J. Accelerating Neural Network Inference on FPGA-Based Platforms—A Survey. *Electronics* **2021**, *10*, 1025. [CrossRef]
37. Lu, X.; Song, L.; Shen, S.; He, K.; Yu, S.; Ling, N. Parallel Hough Transform-based straight line detection and its FPGA implementation in embedded vision. *Sensors* **2013**, *13*, 9223–9247. [CrossRef] [PubMed]
38. Wan, Z.; Yu, B.; Li, T.Y.; Tang, J.; Zhu, Y.; Wang, Y.; Raychowdhury, A.; Liu, S. A Survey of FPGA-Based Robotic Computing. *IEEE Circuits Syst. Mag.* **2021**, *21*, 48–74. [CrossRef]
39. Ayachi, R.; Said, Y.; Ben Abdelali, A. Optimizing neural networks for efficient FPGA implementation: A survey. *Arch. Comput. Methods Eng.* **2021**, *28*, 4537–4547. [CrossRef]
40. Zhang, H.; Ananthanarayanan, G.; Bodik, P.; Philipose, M.; Bahl, P.; Freedman, M.J. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation, Boston, MA, USA, 27–29 March 2017; pp. 377–392.
41. Pakha, C.; Chowdhery, A.; Jiang, J. Reinventing Video Streaming for Distributed Vision Analytics. In Proceedings of the 10th USENIX Workshop on Hot Topics in Cloud Computing, USENIX Association, Boston, MA, USA, 9 July 2018.
42. Ran, X.; Chen, H.; Zhu, X.; Liu, Z.; Chen, J. Deepdecision: A mobile deep learning framework for edge video analytics. In Proceedings of the IEEE Conference on Computer Communications, Honolulu, HI, USA, 16–19 April 2018; pp. 1421–1429.
43. Anderson, M.R.; Cafarella, M.; Ros, G.; Wenisch, T.F. Physical representation-based predicate optimization for a visual analytics database. In Proceedings of the International Conference on Data Engineering, Macao, China, 8–11 April 2019; pp. 1466–1477.
44. Yi, S.; Hao, Z.; Zhang, Q.Q.; Zhang, Q.Q.; Shi, W.; Li, Q. LAVEA: Latency-Aware video analytics on edge computing platform. In Proceedings of the International Conference on Distributed Computing Systems, Atlanta, GA, USA, 5–8 June 2017; pp. 2573–2574.
45. Yang, P.; Lyu, F.; Wu, W.; Zhang, N.; Yu, L.; Shen, X.S. Edge Coordinated Query Configuration for Low-Latency and Accurate Video Analytics. *IEEE Trans. Ind. Inform.* **2020**, *16*, 4855–4864. [CrossRef]
46. Tchaye-Kondi, J.; Zhai, Y.; Shen, J.; Lu, D.; Zhu, L. Smartfilter: An edge system for real-time application-guided video frames filtering. *IEEE Internet Things J.* **2022**, *9*, 23772–23785. [CrossRef]
47. Khani, M.; Ananthanarayanan, G.; Hsieh, K.; Jiang, J.; Netravali, R.; Shu, Y.; Alizadeh, M.; Bahl, V. {RECL}: Responsive {Resource-Efficient} Continuous Learning for Video Analytics. In Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation, Boston, MA, USA, 17–19 April 2023; pp. 917–932.
48. Qin, A.; Xiao, M.; Wu, Y.; Huang, X.; Zhang, X. Mixer: Efficiently understanding and retrieving visual content at web-scale. *Proc. VLDB Endow.* **2021**, *14*, 2906–2917. [CrossRef]
49. Soyata, T.; Muraleedharan, R.; Funai, C.; Kwon, M.; Heinzelman, W. Cloud-Vision: Real-time Face Recognition Using a Mobile-Cloudlet-Cloud Acceleration Architecture. In Proceedings of the IEEE Symposium on Computers and Communications, Cappadocia, Turkey, 1–4 July 2012; pp. 59–66.
50. Hussain, A.; Ahmad, M.; Hussain, T.; Ullah, I. Efficient content-based video retrieval system by applying AlexNet on key frames. *ADCAIJ Adv. Distrib. Comput. Artif. Intell. J.* **2022**, *11*, 207–235. [CrossRef]
51. Collins, Z. Active Database Interface for Video Search. Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2020.
52. Xu, T.; Botelho, L.M.; Lin, F.X. VStore: A data store for analytics on large videos. In Proceedings of the 14th EuroSys Conference, Dresden, Germany, 25–28 March 2019; pp. 1–17.

53. Yadav, P.; Curry, E. VidCEP: Complex event processing framework to detect spatiotemporal patterns in video streams. In Proceedings of the IEEE International Conference on Big Data, Los Angeles, CA, USA, 9–12 December 2019; pp. 2513–2522.
54. Yadav, P.; Salwala, D.; Das, D.P.; Curry, E. Knowledge graph driven approach to represent video streams for spatiotemporal event pattern matching in complex event processing. *Int. J. Semant. Comput.* **2020**, *14*, 423–455. [CrossRef]
55. Sipser, A. Video Ingress System for Surveillance Video Querying. Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2020.
56. Pokornỳ, J. Graph databases: Their power and limitations. In Proceedings of the IFIP International Conference on Computer Information Systems and Industrial Management, Warsaw, Poland, 24–26 September 2015; pp. 58–69.
57. Ronszcka, A.F.; Valenca, G.Z.; Linhares, R.R.; Fabro, J.A.; Stadzisz, P.C.; Simão, J.M. Notification-oriented paradigm framework 2.0: An implementation based on design patterns. *IEEE Lat. Am. Trans.* **2017**, *15*, 2220–2231. [CrossRef]
58. Neves, F.d.S.; Simão, J.M.; Linhares, R.R. Application of generic programming for the development of a C++ framework for the Notification Oriented Paradigm. In Proceedings of the 11th International Conference on Information Society and Technology, Kopaonik, Serbia, 7–10 March 2021; Volume 1, pp. 56–61.
59. Babu, M.A.A. Notification Oriented Paradigm as a Green Technology: Development of a Simulated Sensor Correlation Application with NOP C++ Framework 4.0 and Comparing Green Aspects with Usual OOP Languages. Master's Thesis, Luleå University of Technology, Department of Computer Science, Electrical and Space Engineering, Luleå, Sweden, 2022.
60. Fabro, J.A.; Santos, L.A.; Freitas, M.D.d.; Ronszcka, A.F.; Simão, J.M. NOPL-Notification Oriented Programming Language—A New Language and Its Application to Program a Robotic Soccer Team. In Proceedings of the EPIA Conference on Artificial Intelligence, Virtual Event, 7–9 September 2021; pp. 445–455.
61. Pordeus, L.F.; Linhares, R.R.; Stadzisz, P.C.; Simão, J.M. NOP-DH–Evaluation Over Bitonic Sort Algorithm. *Microprocess. Microsyst.* **2021**, *85*, 104314. [CrossRef]
62. Ronszcka, A.F.; Ferreira, C.A.; Stadzisz, P.C.; Fabro, J.A.; Simão, J.M. Notification-oriented programming language and compiler. In Proceedings of the VII Brazilian Symposium on Computing Systems Engineering, Curitiba, Brazil, 6–10 November 2017; pp. 125–131.
63. Peters, E.; Jasinski, R.P.; Pedroni, V.A.; Simão, J.M. A new hardware coprocessor for accelerating notification-oriented applications. In Proceedings of the IEEE International Conference on Field-Programmable Technology, Seoul, Republic of Korea, 10–12 December 2012; pp. 257–260.
64. Liao, Y.; Panetto, H.; Stadzisz, P.C.; Simão, J.M. A notification-oriented solution for data-intensive enterprise information systems—A cloud manufacturing case. *Enterp. Inf. Syst.* **2018**, *12*, 942–959. [CrossRef]
65. Kerschbaumer, R.; Linhares, R.R.; Simão, J.M.; Stadzisz, P.C.; Erig Lima, C.R. Notification-oriented paradigm to implement digital hardware. *J. Circuits Syst. Comput.* **2018**, *27*, 1850124. [CrossRef]
66. Simão, J.M.; Renaux, D.P.; Linhares, R.R.; Stadzisz, P.C. Evaluation of the notification-oriented paradigm applied to sentient computing. In Proceedings of the IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, Reno, NV, USA, 10–12 June 2014; pp. 253–260.
67. Wojke, N.; Bewley, A.; Paulus, D. Simple Online and Realtime Tracking with a Deep Association Metric. In Proceedings of the IEEE International Conference on Image Processing (ICIP), Beijing, China, 17–20 September 2017; pp. 3645–3649.
68. Cugola, G.; Margara, A. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv. (CSUR)* **2012**, *44*, 1–62. [CrossRef]
69. Ji, S.; Pan, S.; Cambria, E.; Marttinen, P.; Yu, P.S. A Survey on Knowledge Graphs: Representation, Acquisition, and Applications. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *33*, 494–514. [CrossRef]
70. Unterkalmsteiner, M.; Gorschek, T.; Feldt, R.; Lavesson, N. Large-scale information retrieval in software engineering-an experience report from industrial application. *Empir. Softw. Eng.* **2016**, *21*, 2324–2365. [CrossRef]
71. Cugola, G.; Margara, A. The complex event processing paradigm. In *Data Management in Pervasive Systems*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 113–133.