Report on Indexing and Query Optimization of Video Content Based on Text Labels

Zhao Jin, *Master, NCU,*

***Abstract*—As video content surges in digital media, the demand for efficient video content analysis and retrieval technologies continues to grow. This research utilizes advanced object detection technologies and optimized query processing methods, particularly employing the YOLOv5 object detection algorithm and MySQL database system, combined with the Elasticsearch search engine and NLP technology, aimed at enhancing the efficiency and accuracy of video content retrieval. Through systematic theoretical study, technology selection, environment setup, functionality implementation and optimization, and comprehensive testing and validation, this project has successfully developed a system capable of quickly and accurately retrieving video content. The system performs excellently, handling large-scale data sets and returning precise search results in almost real-time, significantly improving the performance and user experience of video retrieval. Future work will continue to optimize query performance and the user interface to meet higher performance requirements and improve user satisfaction.**

***Index Terms*—YOLOv5, Video Retrieval, MySQL Database, Elasticsearch.**

I. Introduction

With The rapid development of digital media technology, video content has become one of the fastest-growing types of data on the internet. The widespread application of these contents has led to an urgent demand for efficient video content analysis and retrieval technologies, enabling users to quickly find the information they need from massive video materials. Although traditional video retrieval technologies offer basic search functionalities, they often fail to meet the dual demands of efficiency and accuracy when dealing with high concurrency and large datasets.

This study focuses on utilizing advanced object detection techniques and optimized query processing methods to address the key challenges in video content retrieval. Initially, the `YOLOv5` [1] object detection algorithm was adopted due to its efficiency and accuracy, which have gained wide applications in real-time video analysis. Compared to traditional object detection algorithms such as `SSD` [2] and `Faster R-CNN` [3], `YOLOv5` offers faster processing speeds and better performance, making it particularly suitable for scenarios that require rapid processing of large volumes of video data.

In terms of data management, the `MySQL` database system was chosen to store and manage video content indexes. To further enhance system performance and query response speed, optimizations were made to the `MySQL` database, including the use of batch processing and precompiled statements, as well as indexing key fields. These optimizations significantly improved the database's operational efficiency and query speed.

To further optimize the querying process, this research combined `NLP` [4] [5] natural language processing techniques with an inverted index mechanism and introduced the `Elasticsearch` [6] [7]search engine, significantly enhancing the query's response speed and accuracy. This query optimization scheme not only improved the efficiency of data retrieval but also enhanced the system's capability to handle high concurrency queries, allowing users to retrieve accurate video content in almost real-time.

By constructing a video content indexing and retrieval system integrated with efficient object detection and advanced query technologies, this study aims to provide a fast and reliable solution to meet the growing needs for video data analysis. The report will detail the system's technology selection, design process, environment setup, performance optimization measures, and demonstrate the system's effectiveness and potential in practical applications through experimental validation.

II. Technology Selection and Theoretical Study

A. Technology Selection

In this project, my primary task was selecting an appropriate object detection algorithm. After extensive literature research and technical evaluation, I chose `YOLOv5` [1] as my main object detection framework. Compared to other algorithms such as `SSD` [2] and `Faster R-CNN` [3], `YOLOv5` not only demonstrated higher processing speeds but also exhibited superior detection accuracy. This is particularly critical for real-time video data processing, where `YOLOv5` achieves faster frame rates. Additionally, its architecture is particularly suited for single GPU acceleration, enhancing video processing performance. While `Faster R-CNN` may have advantages in accuracy, its slower inference speed makes it unsuitable for real-time analysis needs. `SSD`, although balanced in speed, falls behind `YOLOv5` in detecting small objects. Considering both speed and accuracy requirements, `YOLOv5`'s significant advantages in processing speed and real-time performance make it the ideal choice. Furthermore, optimizations in `YOLOv5`'s architecture have reduced false positive rates and increased accuracy, making it the preferred choice for real-time video processing.

In terms of development frameworks, I opted for `PyTorch` [8] [9] due to its friendliness in research and prototype development, flexibility, and robust GPU support. The active `PyTorch` community [10] and abundant tutorial resources

greatly accelerated the development process. For video file processing, I chose the `OpenCV` library [11]. As a mature open-source computer vision library, `OpenCV` provides extensive video and image processing capabilities, enabling me to efficiently implement video reading, frame processing, and display, which are crucial aspects of video content analysis.

B. Environment Setup

To ensure consistency and reproducibility of the development environment, I chose to conduct all development work within a Python virtual environment. By using a virtual environment, I can independently manage dependencies and avoid conflicts with other Python projects in the system. Specifically, I created an isolated environment using the `conda create -n xxx` command and installed PyTorch, OpenCV, and other necessary libraries via `pip`.

C. Database Selection and Performance Analysis

For effective management of video indexing, I opted for the MySQL database, considering its widespread use and stable, mature technical support. Additionally, I have mastered the use of `Apache JMeter` [12] for database performance testing through study and experimentation. This tool has aided in evaluating the database's performance under various query loads, validating the system's performance in practical applications.

D. Theoretical Study

During the theoretical study phase, I systematically read foundational and advanced papers, as well as technical blogs on video content analysis, video retrieval, and object detection. By accessing platforms such as `CSDN` and `GitHub`, I not only learned about the most advanced technological implementations but also kept up with the latest research findings in the field. This provided valuable theoretical support and practical techniques for the project, ensuring its scientific rigor and foresight. Additionally, I systematically studied programming techniques for video processing and object detection using the `PyTorch` framework [10]. Combined with my previous programming experience, this laid a solid foundation for system construction and prepared me adequately for the subsequent implementation phase.

III. Solution

A. Design Phase

1) Data Model Design: To achieve efficient video content retrieval, I have defined the following structure for MySQL database indexing:

- `label` (text type): Represents the label of objects identified in each video frame. These labels correspond to entities in the video, such as "bird," "horse," etc., and are crucial for subsequent video content querying and retrieval.
- `frame_index` (integer type): Identifies the frame number in the video. This is the index of the specific frame where each object is recognized, used to quickly pinpoint the exact location within the video.
- `time_stamp` (float type): Records the exact time each object appears in the video (in seconds). This timestamp is used to precisely jump to specific scenes during video playback.
- `confidence` (float type): The label confidence calculated by the object detection algorithm. This metric is used to assess the reliability of the identification results and is used to rank search results, ensuring the most credible results are displayed first.

Data is stored both in the MySQL database and CSV files to ensure data accessibility and flexibility. CSV files facilitate rapid offline analysis and backup, while database storage supports complex query operations and online processing.

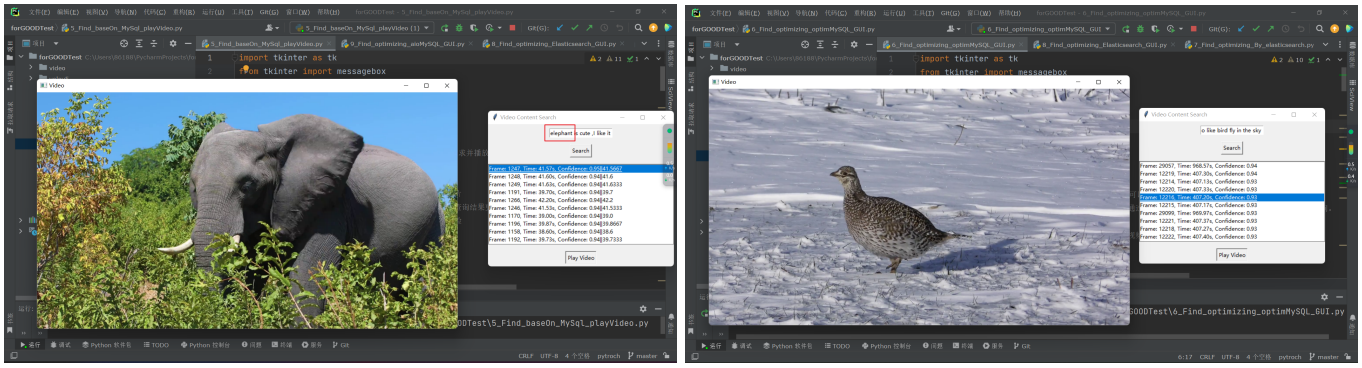
2) Function Planning:

- **Index Creation:** Through the `YOLOv5` [1] object detection algorithm, the system performs frame-by-frame analysis of the video stream, identifying key objects in the video and extracting corresponding `label`, `frame_index`, `time_stamp`, and `confidence` data. These data are first batch-stored in the MySQL database and backed up to CSV files, ensuring data integrity and system robustness.
- **Index Management:** The management functionality allows system users to update, query, and maintain stored index data. The system interacts with the database using standard SQL commands, and it also offers the capability to read data directly from CSV files, increasing operational flexibility.
- **Data Indexing and Search Functionality:** The system utilizes Python's standard GUI library (such as `Tkinter` [13]) to create an intuitive user interface, where users can enter search terms. The search functionality incorporates natural language processing technology (`spaCy` [4]) to parse complex text and extract keywords, thus precisely locating relevant video content based on user input.
- **Video Playback:** Based on the `time_stamp` selected by users through the search functionality, the system can quickly jump to and play the specified video frame. This feature is implemented using the `OpenCV` library [11], allowing direct video viewing within the user interface.

B. Implementation Phase

1) Index Creation and Data Import: During the implementation phase of index creation and data import, the system uses the `YOLOv5` object detection algorithm to perform frame-by-frame analysis of the video, with each frame being examined to identify and tag the objects that appear. The specific implementation steps are as follows:

- **Loading and Processing Video Frames:** Images are read frame-by-frame from the video file using `cv2.VideoCapture` [11]. Each frame image is first converted to RGB format, as this is the format required by the YOLO model, and then passed to the loaded `YOLOv5` model for object detection.
- **Object Detection:** For each frame, object detection is performed using the pre-trained `YOLOv5` model. Each detected object returned by the model includes the object's



(a) (b)

Fig. 1. Querying and playing video clips using MySQL: (a) finding an elephant; (b) finding a bird after optimization.

category, confidence level, and position in the frame. This information is transformed into labels (`label`), frame indexes (`frame_index`), timestamps (`time_stamp`, calculated from the frame rate and frame number), and confidence levels (`confidence`).

- **Data Storage:** The detection results are stored in two primary data storage mediums: MySQL database and CSV files. Using Python's `csv` module, the system formats the data into labels, frame indexes, timestamps, and confidence levels and writes it to CSV files. For database storage, the system uses the `mysql.connector` [14] module to perform batch insert operations into the database, significantly enhancing data insertion efficiency.

2) Search Implementation: The implementation of the search functionality focuses on providing fast and accurate video content retrieval. The specific implementation steps are detailed below:

- **Keyword Extraction:** Using the `spaCy` NLP library [4], the system analyzes the search text entered by the user to extract keywords. These keywords are used to match the label fields in the video index.
- **Video Content Retrieval:** Based on the extracted keywords, the system searches for matching video labels in the MySQL database or CSV files. Data is retrieved from the database using SQL queries, utilizing `LIKE` statements to match labels and ordering by confidence to ensure the most relevant results are displayed first.
- **User Interface Interaction:** A graphical user interface built with `Tkinter` [13] allows users to enter search terms, display search results, and select video segments to play. The search results list is dynamically updated in the GUI, displaying matching video's labels, frame indexes, timestamps, and confidence levels.

3) Video Playback Functionality: The video playback functionality allows users to play videos directly within the user interface after finding the video segments of interest. The implementation steps include:

- **Locating Video Frames:** Based on the timestamp of the video segment selected from the search results, the frame number to jump to is calculated. The current frame of the video stream is set using `OpenCV`'s `cv2.VideoCapture` [11] with the `CAP_PROP_POS_FRAMES` property.
- **Playing Video:** Video playback begins from the selected frame. The video frame is displayed in the GUI window using `OpenCV`'s `imshow` [11] function. Users can stop the playback by pressing a specific key (such as 'q').As shown in Figure 1a.

These implementation details ensure high system performance and a good user experience, meeting the needs of video analysis and retrieval. Through this approach, the system is not only capable of handling large-scale data but also provides real-time responses and accurate video content retrieval.

IV. Optimization Strategies and Implementation

A. Optimization of the Object Detection Algorithm

A major challenge in the implementation process was the high computational demand of the object detection algorithm when processing video data. Considering this, I prioritized improving the efficiency of the algorithm execution through GPU [15] [16] [17] acceleration.

By deeply analyzing the computational bottlenecks of the algorithm, I determined that utilizing the parallel computing capabilities of GPUs could significantly accelerate the processing. I opted for NVIDIA's CUDA [16] technology, which allows the `YOLOv5` model to execute directly on the GPU rather than on traditional CPUs, thus greatly reducing the processing time per frame. This method is particularly suitable for high-resolution videos as GPUs can effectively manage large-scale parallel threads.As shown in Figure 2.

B. Optimization of Database Operations

Addressing the inefficiency of the MySQL database in handling large-scale concurrent query operations, I utilized batch processing and precompiled statements [18]. The traditional method of inserting data one by one is highly inefficient when dealing with large amounts of data. Therefore, I implemented a strategy of batch processing and precompiling statements, which not only increased the speed of data insertion but also reduced the number of database accesses and network latency.

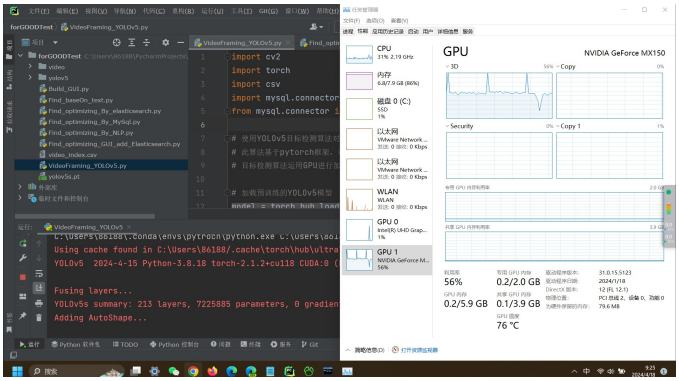


Fig. 2. Using GPU acceleration for object detection and image recognition to create index files and store them in MySQL.

I perform the insertion once enough data accumulates in the batch, significantly enhancing efficiency.

C. MySQL Database Query Optimization

To improve query efficiency and reduce response time, I adopted the following strategies:(As shown in Figure 1b)

1) **Index Creation:** Creating an index on the `label` field is based on the fact that indexes can significantly enhance query performance, especially in searches involving large datasets. This way, I could quickly find relevant video frames through labels. I used the SQL statement `CREATE INDEX idx_label ON objects (label);` to create an index on the `label` field.
2) **Results Caching Mechanism:** Considering that frequent execution of the same queries could lead to unnecessary database access, I introduced a results caching mechanism [19]. By caching the results of common queries, I can respond instantly to these repetitive queries, significantly reducing the load on the database.
3) **Asynchronous Operations and GUI Responsiveness Optimization:** In the user interface, I used asynchronous operations to handle database queries and display results, ensuring the responsiveness of the application. This avoids prolonged database operations from blocking the GUI thread, enhancing user experience.

D. Integration of Elasticsearch

Although optimizing MySQL database queries improved query efficiency well, stress testing showed that it still performs poorly under large-scale concurrent queries. To further enhance the capability of handling large-scale data, I chose Elasticsearch [7] [6] as the search engine. It is based on an inverted index and provides fast full-text search capabilities. Moreover, Elasticsearch supports pagination output [20], which helps in handling large volumes of results, reducing server load, and improving query performance and response speed. Additionally, I sorted the query results by confidence to prioritize the most relevant video segments, enhancing the accuracy of queries and user satisfaction. Through these measures, not only has the system's query performance improved, but it also ensures that the system remains stable and efficient in environments with large data volumes and frequent queries.As shown in Figure 3.

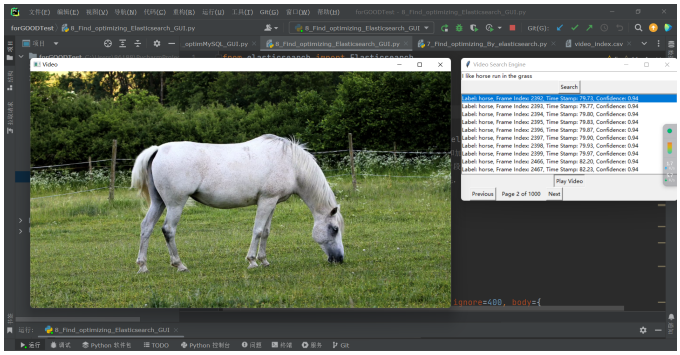


Fig. 3. Using Elasticsearch to accelerate the search of video clips and play video.

V. Feasibility Validation of the Solution

To ensure that this system meets design requirements and can operate stably and efficiently in real environments, as well as to explore higher-performance query methods, I have implemented a series of comprehensive tests and validations. These tests are designed to fully assess the functionality of the system, user interface friendliness, and performance under high load conditions.

A. Functional Testing

1) Data Index Verification: The core testing of the system's data indexing function was conducted first. I verified whether the system could correctly build data indexes to ensure that each video frame was correctly parsed and associated data accurately stored. By analyzing video frames frame-by-frame and using the object detection algorithm to identify object information (label, frame index, timestamp, and confidence), I verified whether the data was accurately reflected in the MySQL database and CSV files. This process included verifying each label and their precise timing positions in the video to ensure the completeness and accuracy of the indexes.

2) Search Functionality Testing: The testing of the search functionality focused on verifying whether the system could accurately find the corresponding video segments based on user-inputted text or labels. I designed a series of test cases, including queries on both common and uncommon labels, as well as based on complex text queries, to ensure the relevance and accuracy of the search results. Each test case had to return the correct video frame and correctly order them by confidence, validating the effectiveness of the search algorithm.

3) Pagination Functionality Testing: The testing of the pagination functionality aimed to ensure the system could manage and display a large number of search results. I tested whether the pagination mechanism could correctly navigate through multiple pages of results, including ensuring data consistency and accuracy when switching from one results page to another.

4) Video Playback Testing: The testing of the video playback functionality ensured that the selected video segments could play at the correct time points. Through practical operation, I verified that the system could precisely locate and play videos based on timestamps in the search results. Additionally, the test included verifying the smoothness of the playback, ensuring there were no synchronization issues or delays.

5) GUI Functionality Testing: I conducted thorough functional tests on all GUI components, including text input boxes, search buttons, result list boxes, and video playback controls. Each component had to respond to user actions and perform its function correctly. I also reviewed the layout and fonts of the user interface to ensure its friendliness and usability.

B. Robustness Testing of the System

To test the robustness of the system, I selected several wildlife educational videos of varying durations and content. These videos contained more than ten different categories of wild animals, providing an opportunity to challenge the accuracy and robustness of the system's object detection algorithm. Through frame-by-frame analysis and object recognition, I was able to assess the system's performance in real applications, recording and analyzing any false positives or misses.

C. System Stress Testing

Using Apache JMeter [12], a stress test involving massive concurrent queries was performed on a MySQL database. Initial results indicated that the query performance under the original configuration did not meet the requirements. To enhance performance, I introduced the Elasticsearch search engine and subjected it to similar stress testing. The results demonstrated that Elasticsearch significantly improved query efficiency, with the capacity to handle loads hundreds of times greater than MySQL. These test results not only confirmed the effectiveness of Elasticsearch in handling high concurrency and large-scale queries but also pointed the direction for future system optimizations.

The figure below visually represents the aggregate reports from stress testing both MySQL and Elasticsearch at different concurrency levels. Using Jupyter Notebook, the variations in throughput and average response time metrics were compiled into visual line graphs, facilitating a comparative visualization

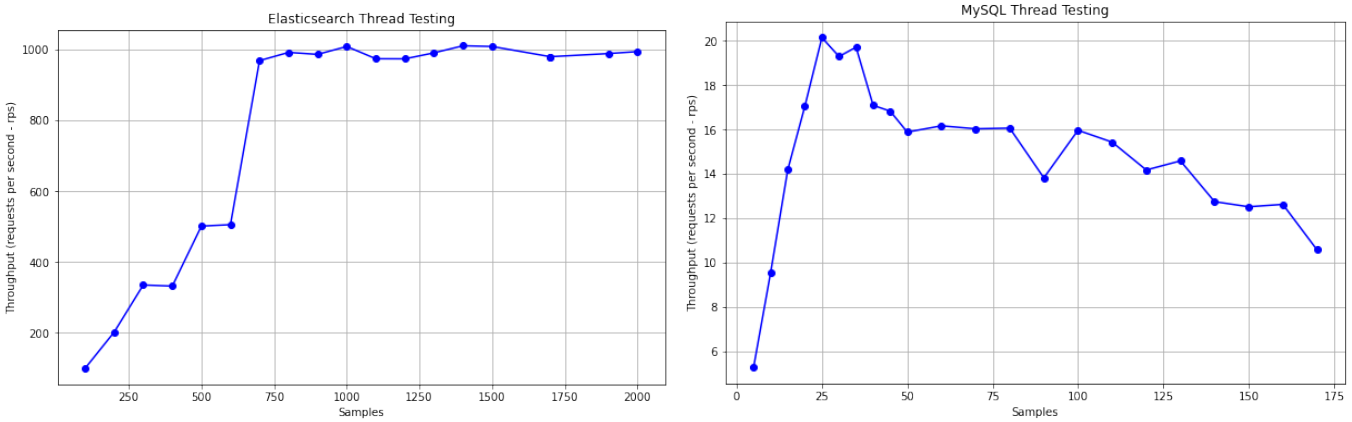


Fig. 4. Comparison of the changes in throughput for thread testing between the two: (a) Elasticsearch; (b) MySQL.

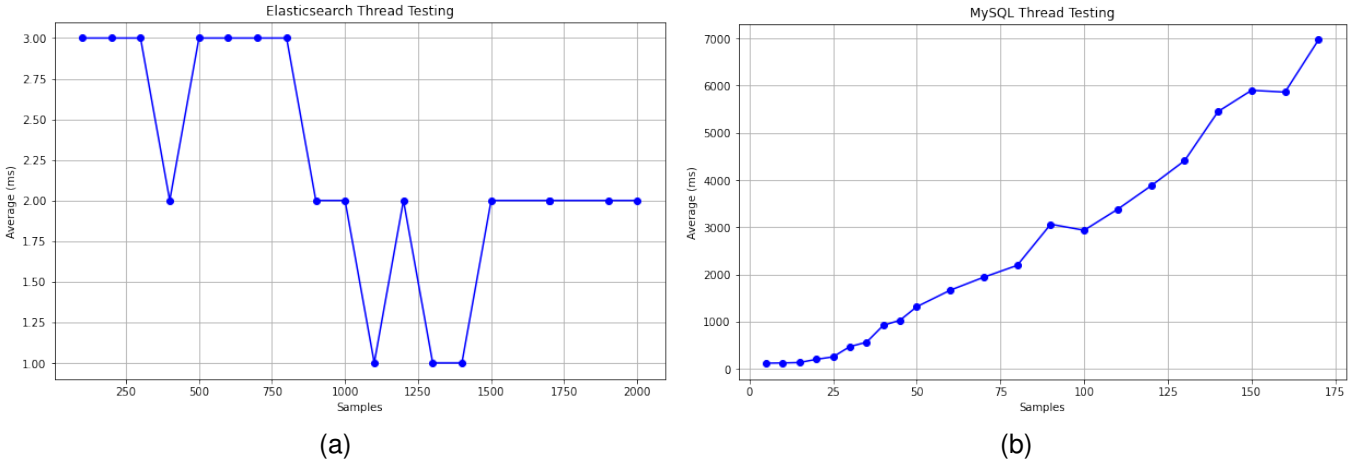


Fig. 5. Comparison of the changes in average response time for thread testing between the two: (a) Elasticsearch; (b) MySQL.

of how throughput and average response times changed with increasing thread counts.

As shown in Figure 4, the throughput for both systems increases with the number of concurrent threads. The results indicate that with Elasticsearch, the throughput stabilizes at 1000 requests per second after reaching 750 concurrent threads; in contrast, MySQL shows a performance decline after surpassing 25 concurrent threads.

Figure 5 illustrates a comparison of the average response times as concurrency levels change. In part (a), the results show that with Elasticsearch, the average response time remains almost constant at 1-3 milliseconds regardless of the number of threads, with fluctuations likely due to hardware limitations or interference from other processes. Conversely, part (b) shows that the response time for MySQL queries increases in direct correlation with the number of concurrent threads.

Overall, although there are some errors in the test results, it can generally be concluded that Elasticsearch performs significantly better than MySQL in handling high-concurrency queries.

These tests and validations have proven the system's effectiveness and stability in real-world applications. The results not only confirmed the rationality of the system design but also provided valuable data support for subsequent optimizations and expansions.

VI. Conclusion and Outlook

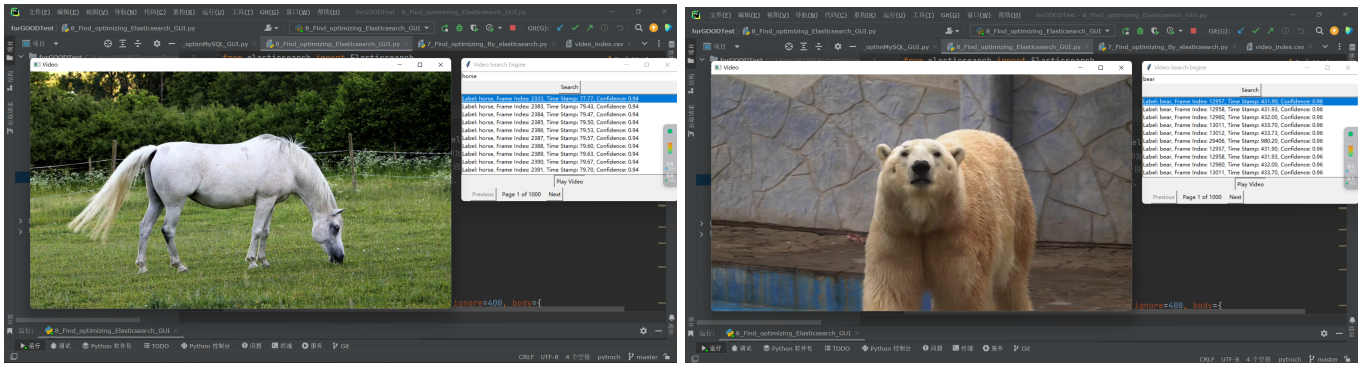
A. Project Outcomes

This project has successfully implemented a text-label-based video search tool, providing users with an intuitive and efficient platform to search for and play relevant video segments. By leveraging the advanced object detection capabilities of the `YOLOv5` algorithm, the video processing power of `OpenCV`, the fast search performance of `Elasticsearch`, the stable data management functions of the MySQL database, and the flexibility of the `PyTorch` framework, the system offers a robust video content retrieval solution. Additionally, with a graphical user interface provided by `Tkinter` and the natural language processing capabilities of `NLP` technology, the system not only enhances the user interaction experience but also improves the accuracy and response speed of searches. Here is a partial display of the system effects: As shown in Figure 6 and Figure 7.

B. Challenges Faced

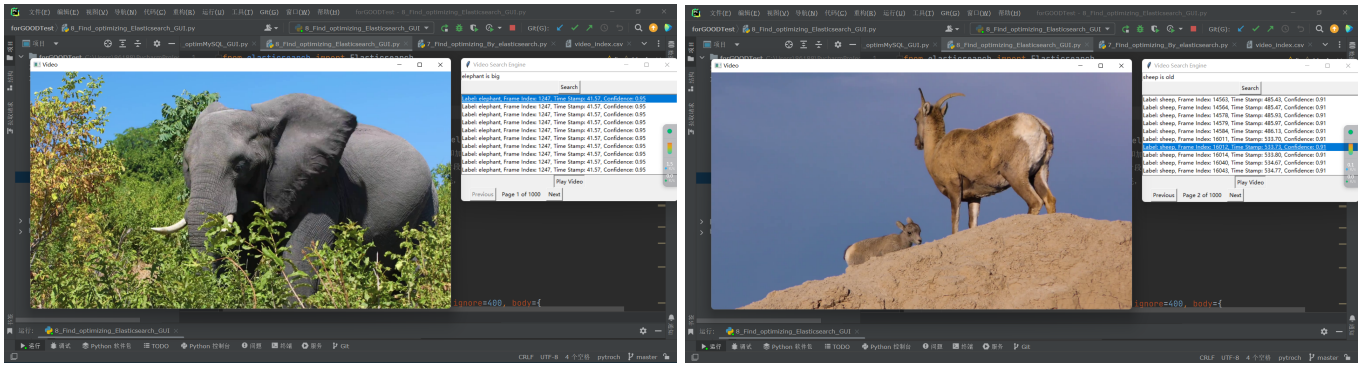
During the development and implementation of the project, I encountered several key challenges:

- **Performance Optimization:** As the data volume processed by the system continues to grow, maintaining the efficiency of search operations becomes a continuous challenge. Particularly in handling high-concurrency



(a) (b)

Fig. 6. Enter text in the window to quickly locate the video clip, then click the 'play video' button to start playback.(a) horse; (b) bear.



(a) (b)

Fig. 7. Enter text in the window to quickly locate the video clip, then click the 'play video' button to start playback.(a) elephant; (b) sheep.

requests, optimizing system architecture and database performance to reduce latency and increase throughput is a problem that needs to be addressed.

- **User Experience:** Although the current GUI already supports basic search and playback functions, the intuitiveness and responsiveness of the user interface need improvement. Designing a more user-friendly and dynamic interface to provide a more satisfactory user interaction experience is another challenge I face.

C. Future Work

In response to current challenges and to further enhance system performance and user experience, I believe there are several potential areas for development:

1) **Optimizing Search Query Performance:**
   - Asynchronous Database Operations: Adopt asynchronous techniques [21] [22] [23] in database operations to enhance the responsiveness and efficiency of the system, especially when handling large-scale data.
   - Adjusting `Elasticsearch` Configuration: Further optimize the analyzer and query settings of `Elasticsearch` to improve the capability and speed of handling complex queries.
   - Introducing Distributed Data Services: Consider using a distributed architecture [24] to enhance the processing capability of large-scale concurrent queries.
   - GPU-Accelerated Queries: Explore the use of GPUs to accelerate data processing and query operations [23] [25], especially in video data processing and complex search algorithms.
2) **User Experience:**
   - Improving User Interface: Add more interactive elements and visual feedback, such as progress bars and dynamic loading effects, to enhance the attractiveness and interactivity of the interface.
   - Expanding Features: Add video upload and automatic indexing features, allowing users to directly upload videos to the platform, where the system automatically recognizes content, extracts labels, and creates indexes.
   - Enhancing User Interaction: Improve the GUI design, add error handling, and user feedback mechanisms to ensure that users' issues during use are promptly addressed and resolved.

By perfecting these strategies in the future, I aim to elevate the system to new heights, not only achieving a leading position technologically but also significantly enhancing user satisfaction.

REFERENCES

[1] Ultralytics, "YOLOv5: Real-time object detection software," 2020, [Online; accessed 20-April-2024]. [Online]. Available: https://github.com/ultralytics/yolov5

[2] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, 2016, pp. 21–37.

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[4] E. AI, "spacy: Industrial-strength natural language processing," https://github.com/explosion/spaCy, 2023, accessed: 2024-04-21.

[5] H. Face, "Transformers: State-of-the-art natural language processing," https://github.com/huggingface/transformers, 2023, accessed: 2024-04-21.

[6] Elastic, "Elasticsearch: Restful, distributed search and analytics engine," https://github.com/elastic/elasticsearch, 2023, accessed: 2024-04-21.

[7] ——, "Elasticsearch documentation," https://www.elastic.co/guide/index.html, 2023, accessed: 2024-04-21.

[8] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[9] A. Paszke, S. Gross *et al.*, "Pytorch: An imperative style, high-performance deep learning library," https://github.com/pytorch/pytorch, 2019.

[10] PyTorch, "Pytorch: An open source machine learning framework that accelerates the path from research prototyping to production deployment," https://pytorch.org/, 2023, accessed: 2024-04-21.

[11] OpenCV Team, "Open source computer vision library," 2021, accessed: 2024-04-21. [Online]. Available: https://opencv.org/

[12] Apache Software Foundation, "Apache jmeter," https://jmeter.apache.org, 2023, accessed: 2024-04-21.

[13] Python Software Foundation, "Tkinter — python interface to tcl/tk," https://docs.python.org/3/library/tkinter.html, 2023, accessed: 2024-04-21.

[14] Oracle Corporation, "Mysql connector/python developer guide," https://dev.mysql.com/doc/connector-python/en/, 2023, accessed: 2024-04-21.

[15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[16] NVIDIA Corporation, "Cuda toolkit documentation," https://developer.nvidia.com/cuda-toolkit, 2023, accessed: 2024-04-21.

[17] PyTorch, "Pytorch documentation: Cuda semantics," https://pytorch.org/docs/stable/notes/cuda.html, 2023, accessed: 2024-04-20.

[18] B. Schwartz, P. Zaitsev, and V. Tkachenko, *High Performance MySQL*, 3rd ed. O'Reilly Media, Inc., 2012.

[19] A. Silberschatz, H. F. Korth, and S. Sudarshan, "Database system concepts," 2011.

[20] Elasticsearch, "Paginate search results," https://www.elastic.co/guide/en/elasticsearch/reference/current/paginate-search-results.html, 2023, accessed: 2024-04-20.

[21] B. Schwartz, P. Zaitsev, and V. Tkachenko, *High performance MySQL: optimization, backups, and replication*. " O'Reilly Media, Inc.", 2012.

[22] MongoDB Inc., "Mongodb documentation," https://docs.mongodb.com/, 2023, accessed: 2024-04-22.

[23] J.-H. LAI, Z.-C. XU, Y.-C. CHU, and G.-L. TAN, "Omegadb, a relational operator concurrent computing framework for heterogeneous architectures," *Journal of Computer Applications*, p. 0.

[24] Apache Hadoop, "Apache hadoop," https://hadoop.apache.org/, accessed: 2024-04-22.

[25] D. B. Kirk and W. H. Wen-Mei, *Programming massively parallel processors: a hands-on approach*. Morgan kaufmann, 2016.