Cofster: Architectural Framework for an Intelligent Coffee System Using CV and NLP

Darie-Teofil Glanda
School of Automation and Computing
Politehnica University Timisoara
Timisoara, Romania
Email: darieglanda@outlook.com

Razvan-Virgil Bogdan
School of Automation and Computing
Politehnica University Timisoara
Timisoara, Romania
Email: razvan.bogdan@cs.upt.ro

Abstract—**Cofster, an innovative coffee-making system, aims to redefine the coffee creation experience by introducing a seamless and autonomous operation, eliminating the need for manual intervention. The primary focus of this endeavor involves the integration of cutting-edge technologies and architectures into the coffee-making process, resulting in a sophisticated system that enhances user control and overall satisfaction. A key feature of Cofster is its use of Computer Vision and Natural Language Processing. Using state-of-the-art detectors, with the models already pre-trained on the MS COCO dataset, and then further trained for our custom cup detection, it autonomously identifies custom cups and their sizes (small, medium, large), to start the coffee-making process. Additionally, a LangChain agent uses different versions of GPT and the RAG method, to combine stored data with the LLM, continuously updating the system based on user preferences, gathered through a response form for personalized drink choices. Cofster's operational efficiency is achieved with a mix of on-premise servers using microservice architecture and a serverless approach on AWS Cloud, enhancing scalability and resource utilization. The comprehensive repository for this project, containing all relevant code, documentation, and associated resources, is publicly accessible at the following URL: https://github.com/glandaDarie/Cofster.**

1. Introduction

THE rapid and comprehensive advancements in technology mark an epoch where the primary aim still lies in automating a wide range of human tasks. This paradigm, in hindsight, holds the potential to reposition individuals from the limitations of manual labor to a more prominent role as orchestrators, guiding the course of operations rather than solely executing them.

With the assistance of AI, this transformation becomes easier than ever to achieve. Tasks that were once deemed labor-intensive are now effortlessly accomplished (1). From smart homes to self-driving vehicles, the surge of automated systems utilizing AI revolutionizes how we perceive and interact with technology. Among these innovations, the creation of a smart coffee system would open doors to a new era for coffee stores, eliminating the necessity for additional personnel, removing labor work, enhancing automation (13). In the scheme of those things, the development of Cofster, which is an architectural framework integrating cutting-edge technologies, heralds a new era for coffee stores. This framework, enhances operational efficiency, but it also marks a new leap towards a technologically-driven future in the realm of coffee creation.

The architecture of Cofster is designed to embrace the microservices paradigm, showcasing a distributed and containerized infrastructure. We chose this approach, because it is integral to achieve flexibility, scalability, and maintainability in modern applications. We use docker-compose in order to run all the containers at the same time, more or less the service that handles the frame processing and the coffee creation one (the Raspberry Pi service).

2. Related Work

When integrating state-of-the-art detector models, like the ones from the YOLO family, in order to perform inference on cup detection, the conventional approach often relies on edge devices [(4), (5), (6), (7)]. However, our approach diverges by adopting a microservice architecture, effectively separating cup detection from the backend infrastructure. This architectural shift allowed us to capitalize on CUDA inference on our server, using GPU. It's worth noting that previous approaches opt to utilize older versions of YOLO [(8), (9), (10), (11)], due to their simpler architectures with fewer layers, which allowed for faster forward pass execution and higher FPS. In contrast, we leveraged the latest versions of YOLO available, YOLOv8 and YOLOv9, without such constraints. By decoupling the cup detection from the coffee creation process and utilizing GPU inference for these models, we significantly enhanced efficiency. This approach enabled us to use more complex models with additional layers without incurring any performance overhead.

When building a model to predict users' top k favorite drinks based on a set of questionnaire responses, where each question corresponds to a database column, previous implementations usually opt on using the Boosting strategy, more precisely XGBoost (12). However, our approach diverges from this norm by emphasizing the construction of a Multiple Layer Perceptron (MLP), offering a more unorthodox approach for such a task.

To view the camera's perspective, we devised a pipeline utilizing OpenCV at its core, integrated with a Kafka producer, featuring a single partition and a single replica, a Kafka consumer, WebSocket, and a React-based client. Additionally, these services are containerized. Kafka was employed to decouple the services and facilitate horizontal scaling, thus enabling the addition of multiple clients without any issues.

A similar approach (2), considered a more sophisticated pipeline, addressing the problem of social distancing during the COVID-19 pandemic. Their methodology involved OpenCV with a YOLOv3 model deployed on Kubernetes with the aid of KubeFlow. For data processing, Spark was utilized, and Kafka was employed for Backend-to-Backend communication. The data was subsequently passed to analytics using an API Gateway in the form of a WebSocket. Once the analytical report was generated, it was sent via email to the respective client.

Regarding the use of RAG techniques, a recent implementation makes use of retrieval-augmented models that generate a response based on multiple documents; however, they ignore the given topic and use only the local context of the conversation (3). In our pipeline, we make use of only one document, employing a Langchain agent integrating information from a Large Language Model (LLM) and our PostgreSQL data store.

3. Architecture

In the core of Cofster's software architecture (figure 1 and 2), lies a Flutter-based mobile application, serving as the intuitive user interface for coffee ordering. The Frontend utilizes the Provider state manager and ValueNotifier-ValueListenableBuilder, to ensure responsive interactions. The app contains the following pages:

Authentication Page: Users log in securely or create new accounts. Upon successful account creation, a confirmation email is sent, and users can personalize their profile with a photo, that is stored in S3 bucket. Additionally, they fill out a form to receive a coffee gift and a list of their favorite k drinks from a Neural Network.

Main Page: Displays vital information like favorite drinks, available drinks, active orders, and drink history. Includes a voice navigation feature for effortless access to preferred drinks.

Drinks Creation Page: Users customize beverage, with the option for autonomous drink creation based on preferences. Payment is processed using the Stripe Payment Service.

Map Page: Displays nearby coffee shops and their distances, leveraging the Google Maps API.

Order Page and Order Purchases: Users manage pending orders and view past drink purchases, powered by Firebase and DynamoDB.

Gift Page: Users redeem coffee gifts seamlessly, transitioning to the Stripe Payment Service for checkout, powered by DynamoDB.

The Drink Classifier Neural Network is designed to predict users' top k favorite drinks upon account creation, offering a personalized welcome gift of their favorite drink. The dataset, sourced from Google Forms, comprises both real responses and synthetic data generated by GPT-3.5. Data preprocessing is conducted using NumPy, Pandas, and Scikit-learn, incorporating an 80-20 train-test split, and normalizing the data using a standard scaler. Model training is done with the help of PyTorch, featuring a fully connected layer architecture. Training information includes Xavier weight initialization, batch size of 128, batch normalization, a learning rate of 0.1, dropout of 0.4 to 0.6 between each layer, ReLU activation, weight decay of 0.001 and a SGD as the optimizer. The model is deployed via a Flask API within a Docker environment, enabling real-time responses for users.

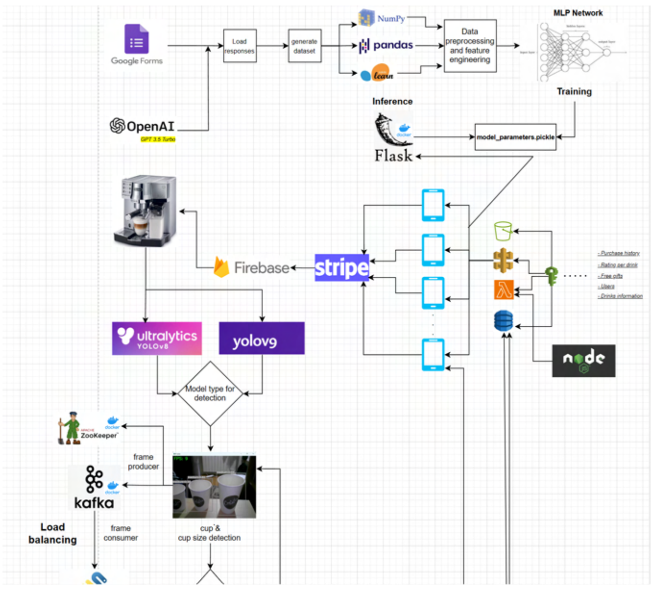


Figure 1: **Software System Architecture part 1.** Represents an abstract-level overview of the Cofster system, illustrating its overall structure and how its components interact. Continued on part 2.

Financial transactions are facilitated via the Stripe payment system, while user data is securely stored on AWS using the serveless stack in the form of: IAM, Lambda, API Gateway, S3, and DynamoDB. The other Backend, for coffee creation, cup detection and updating the coffee recipe, built with Flask, using microservices, enables horizontal scaling, preventing the cup detection and coffee update to be performed on the Raspberry Pi. This, on the cup detection side, opens doors for CUDA inference, using a camera connected to the server that runs OpenCV, in order to improve the FPS.

Regarding the updating of the coffee recipe, three microservices are used to dynamically update individualized coffee recipes, orchestrated by a Langchain Agent, utilizing the Retrieval Augmented Generation (RAG) method. Leveraging a Large Language Model (can choose any from GPT 3.5 to GPT 4 family), a text loader as a data store, and a PostgreSQL database for chat history, the agent dynamically adjusts users' coffee recipe based on their responses gathered through a simple survey presented on the Mobile Application Frontend, after they finish their coffee. A variation of the famous Bellman's updater equation is employed, in order to prioritize newer responses with higher probability, while still acknowledging the significance of older ones, ensuring a nuanced update to evolving user preferences. The Bellman updater implementation looks like this:

let $E = \{e_1, e_2, \ldots, e_n\}$ be the set of data elements, where e_i represents an element from the list of data elements.

let P_0 be the initial probability value.

let γ be the discount factor for the Bellman equation.

then, the trajectory T can be represented as a list of tuples:

$$T = \{(e_i, P_0 \times \gamma^{i-1})\}_{i=1}^{n}$$

each tuple in T contains a data element and its corresponding probability at each time step, where the probability is calculated using the Bellman equation:

$$P_i = P_0 \times \gamma^{i-1}$$

Meanwhile, the coffee creation process is facilitated by Firebase Realtime Database, ensuring seamless Frontend-Backend communication. Utilizing OpenCV with the YOLOv8 and YOLOv9 detectors, our system accurately identifies our custom cup. These detectors are pre-trained on the MS COCO dataset, and re-trained with our own data. Additionally, the system incorporates a mechanism to determine the size of the cups—whether small, medium, or large—by employing a threshold distance and fixing the camera position. Notably, the system verifies the presence of a white pipe atop the cup, ensuring correct positioning for the coffee creation process to commence. Implementation-wise, we leverage the region of interest (ROI) technique for the pipe, focusing on the top of the cup along the y-axis. This involves detecting a white rectangle of a certain area, established by a predefined threshold.

A pipeline has been established to facilitate the visualization of the cup perspective, for the clients. This pipeline is constructed utilizing Kafka, enabling the transmission of frames from Kafka producer to Kafka consumer. Subsequently, the frames obtained from the Kafka consumer are transmitted to a Websocket. The Websocket facilitates full duplex communication with clients, allowing each frame to be rendered on their screens. Integration of Kafka into this pipeline ensures scalability, accommodating potential expansion for multiple clients in the future.

On the hardware side of things, the architecture is rather a simple one, comprising of pumps for dispensing liquid and sugar, a Raspberry Pi to orchestrate machine operations, an 8x Serial Relay Board for signal redirection, a 220V DC to 12V AC convertor, and a Redragon GW800 1080P Camera for custom cup detection.

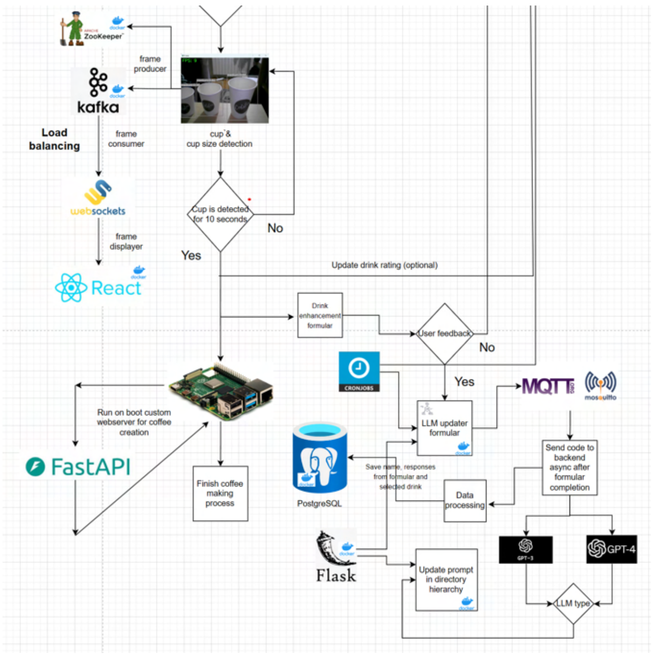

ZooKeeper
frame producer
kafka
Load balancing
frame consumer
cup & cup size detection
websockets
Cup is detected for 10 seconds
No
frame displayer
Yes
React
Update drink rating (optional)
Drink enhancement formular
User feedback
No
CRONJOBS
Yes
MQTT
mosquitto
Run on boot custom webserver for coffee creation
LLM updater formular
FastAPI
Finish coffee making process
PostgreSQL
Save name, responses from formular and selected drink
Data processing
Send code to backend async after formular completion
GPT-3
GPT-4
Flask
Update prompt in directory hierarchy
LLM type

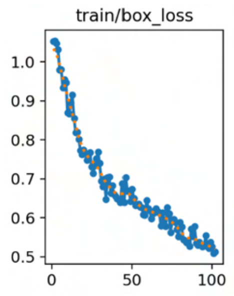
Figure 2: **Software System Architecture part 2.** Continuation of part 1, showing further details of the Cofster system architecture.

4. Results

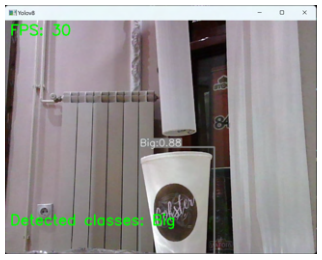
In this chapter, we present the results of our custom cup detection, employing the two latest versions of the YOLO model, YOLOv8 and YOLOv9. Since our models were fine-tuned for a specific type of cup, unique to our application, we could not directly compare our results with previous state-of-the-art methods. Consequently, we focused on comparing the performance of our models to evaluate their effectiveness in this custom cup detection.

The dataset comprises 540 images of cups. Both of the models are used with Transfer Learning, already trained on the MS COCO dataset. We froze the first 5 layers, and trained the parameters on the rest of them. The training was conducted on a RTX 3060 GPU for 400 epochs with a batch size of 16. To prevent overfitting, a dropout rate of 0.4 was applied between layers. Additionally, we implemented a weight decay of 0.01. The training employed the AdamW optimizer with a learning rate of 0.01, and utilized 16 workers for data loading. Each image in the dataset has a resolution of 640x640 pixels, with three color channels.

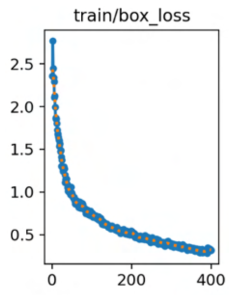
As seen in the figure 3, we achieve for YOLOv8 a training box loss of 0.50846, after approximately 100 epochs, because early stopping was enabled after 50, hence, stopping the training process. When doing inference, the model achieved approximately 30 FPS. For the YOLOv9 model, we managed to get a training box loss of 0.29018 as the best score, but, this drastically slowed the inference down compared to the YOLOv8, from 30, to approximately 12 FPS.



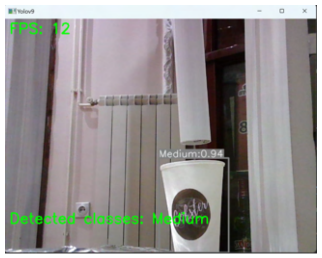
(a) Train box loss for YOLOv8.



(b) Detection YOLOv8 with FPS.



(c) Train box loss for YOLOv9.



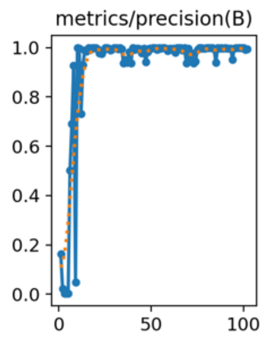
(d) Detection YOLOv9 with FPS.

Figure 3: Training box loss for the YOLOv8 and YOLOv9 models.

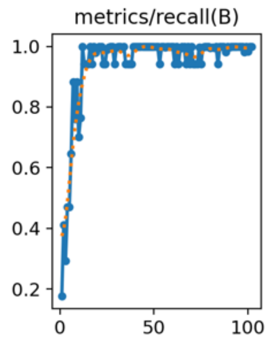
Figure 4 represent the training precision and recall for both the YOLOv8 and YOLOv9 models. The YOLOv8 model achieved an average training precision of 0.9217 and a training recall of 0.9419. In comparison, the YOLOv9 model achieved a higher training precision of 0.9496, although it had a slightly lower training recall of 0.9143.

5. Conclusion

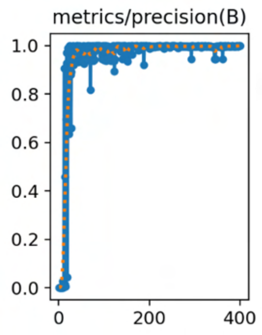
In conclusion, Cofster successfully demonstrates the potential of integrating advanced AI technologies into the coffee-making process, thereby enhancing user experience and operational efficiency. The adoption of a microservice architecture, utilizing both on-premise servers and cloud infrastructure, along with state-of-the-art models in the form of YOLOv8 and YOLOv9, highlight the innovative approach and technical sophistication of our implementation. The system's innovative features, such as autoregressive coffee creation using a LangChain agent with the RAG method and data streaming to clients via Kafka, redefine the coffee creation industry. Future work will focus on refining the personalization algorithms, for both the autoregressive coffee creation and the coffee recommender system, to further elevate the autonomous coffee-making process. This research opens new doors to more intelligent and efficient automation solutions across various domains.



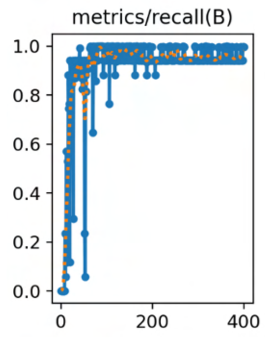
(a) Precision YOLOv8.



(b) Recall YOLOv8.



(c) Precision YOLOv9.



(d) Recall YOLOv9.

Figure 4: Precision and recall training For YOLOv8 and YOLOv9.

Acknowledgments

The authors would like to thank the University Politehnica of Timișoara, especially Dr. Eng. Cristina Stângăciu and Dr. Eng. Valentin Stângăciu, for their valuable ideas and support throughout the implementation process.

References

[1] L. D. Tyson and J. Zysman, "Automation, AI & work," *Daedalus*, vol. 151, no. 2, pp. 256–271, 2022.
[2] S. Melenli and A. Topkaya, "Real-time maintaining of social distance in covid-19 environment using image processing and big data," in *Trends in Data Engineering Methods for Intelligent Systems: Proceedings of the International Conference on Artificial Intelligence and Applied Mathematics in Engineering (ICAIAME 2020)*, 2021, pp. 578–589.
[3] Y. Ahn, S.-G. Lee, J. Shim, and J. Park, "Retrieval-augmented response generation for knowledge-grounded conversation in the wild," *IEEE Access*, vol. 10, pp. 131374–131385, 2022.
[4] D. Kim, S. Lee, N.-M. Sung, and C. Choe, "Real-time object detection using a domain-based transfer learning method for resource-constrained edge devices," in *2023 International Conference on Artificial Intelligence in*

Information and Communication (ICAIIC), 2023, pp. 457–462.

[5] S. Swain, A. Deepak, A. K. Pradhan, S. K. Urma, S. P. Jena, and S. Chakravarty, "Real-Time Dog Detection and Alert System using Tensorflow Lite Embedded on Edge Device," in *2022 1st IEEE International Conference on Industrial Electronics: Developments & Applications (ICIDeA)*, 2022, pp. 238–241.

[6] A. Parmar, R. Gaiiar, and N. Gajjar, "Drone based Potholes detection using Machine Learning on various Edge AI devices in Real-Time," in *2023 IEEE International Symposium on Smart Electronic Systems (iSES)*, 2023, pp. 22–26.

[7] C. Nigam, G. Kirubasri, S. Jayachitra, A. Aeron, and D. Suganthi, "Real-Time Object Detection on Edge Devices Using Mobile Neural Networks," in *2024 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE)*, 2024, pp. 1–4.

[8] V. A. M. Luis, M. V. T. Quiñones, and A. N. Yumang, "Classification of Defects in Robusta Green Coffee Beans Using YOLO," in *2022 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*, 2022, pp. 1–6.

[9] K. Che, Y. Liang, Y. Zeng, T. Li, X. Zhu, and W. Lv, "Revolutionizing Traditional Chinese Medicine Image Classification and Recognition with an Improved YOLOv5," in *2023 2nd International Conference on Health Big Data and Intelligent Healthcare (ICHIH)*, 2023, pp. 1–5.

[10] R. Gai, M. Li, and N. Chen, "Cherry detection algorithm based on improved YOLOv5s network," in *2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, 2021, pp. 2097–2103.

[11] P. Athira, T. P. M. Haridas, and M. H. Supriya, "Underwater object detection model based on YOLOv3 architecture using deep neural networks," in *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, vol. 1, 2021, pp. 40–45.

[12] K. Tseng, "Predicting victories in video games: using single XGBoost with feature engineering: IEEE BigData 2021 Cup-Predicting Victories in Video Games," in *2021 IEEE International Conference on Big Data (Big Data)*, 2021, pp. 5679–5682.

[13] R. Rathore, "The application of the robot for the coffee manufacturer," *Asian Journal of Research in Social Sciences and Humanities*, vol. 11, no. 12, pp. 148–152, 2021.