Computer Vision Based Smart Security System for Explainable Edge Computing

N. Nisbet and J. I. Olszewska
School of Computing and Engineering, University of the West of Scotland, U.K.

Keywords: Explainable Edge Computing, Security and Privacy, Computer Vision Applications, Explaining Object and Obstacle Detection, Trustworthy Intelligent Vision Systems, Transparent Decision-Support Systems, Explainable Artificial Intelligence.

Abstract: Smart cities aim to reach a high quality of life for their citizens using digital infrastructure which in turn need to be sustainable, secure, and explainable. For this purpose, this work is about the development of an explainable-by-design smart security application which involves an intelligent vision system capable of running on a small, low-powered edge computing device. Hence, the device provides facial recognition and motion detection functionality to any electronic motion picture input such as CCTV camera feed or video. It highlights the practicality and usability of the device through a support application. Our resulting explainable edge computing system has been successfully applied in context of smart security in smart cities.

1 INTRODUCTION

Modern smart cities deploy a growing number of edge computing technologies (Khan et al., 2020), which include among others internet of things (IoT) devices and artificial intelligence (AI) algorithms to process such collected data (Mishra et al., 2024), for diverse smart applications ranging from air quality monitoring (Zhu et al., 2023) to building operation (Slee et al., 2021), maintenance (Xiao et al., 2024) or surveillance (Sharma and Kanwal, 2024b). However, newest regulations and standards (Houghtaling et al., 2024), require these intelligent systems to be trustworthy. Therefore, using explainable artificial intelligence (XAI) (Li et al., 2024) to develop transparent and reliable edge computing systems is not an optional requirement but a cornerstone aspect of Society 5.0 (Chamola et al., 2023).

In particular, smart cities need trustworthy solutions for smart security (Sharma and Kanwal, 2024a) which integrates emerging technologies (such as IoT devices, cloud computing, edge computing) (Sodiya et al., 2024) and intelligent systems (like computer-vision systems) into traditional security measures to improve the safety and security of people, property, and data (Utimaco, 2024).

Indeed, smart cities tend to an increase in security, safety, and better living experiences for citizens, and these smart cities are made possible due to smart cameras and sensors. For example, the city of Barcelona uses c. one million different sensors around the city, monitoring everything from traffic to temperature and even waste management (Lea, 2020). Devices such as sensors and cameras vary in the amount of bandwidth required which is where edge computing is making a big impact, because, quicker and more cost effective than ever, the sheer amount of data produced by IoT devices is still time consuming to transport and to store. Moreover, the ever-growing popularity of IoT devices is creating more advanced versions producing much more data (Naveen and Kounte, 2019).

While the majority of smart devices across our cities are sensors that do not need to constantly send data every few seconds or even milliseconds, some of these sensors need quick decisions such as a smart road detecting an accident and wanting to close a lane. A single edge device could then control multiple sensors and enable these quick decisions to be made without having to send the data to a main hub. So cutting out this transfer time between multiple systems sitting in a queue waiting to be delt with can save valuable seconds. With millions of sensors dotted around a large area, there is also the possibility data could arrive at different times with the decision making delayed due to all the information not being processed in time. Edge devices are thus an efficient solution to build networks within networks in order to create fully automated systems with little or no impact on the wider grid.

As smart cities grow, expand, and develop over time, edge devices can work with cloud computing in easing the strain on sudden expansion. Since smart

Nisbet, N. and Olszewska, J.
Computer Vision Based Smart Security System for Explainable Edge Computing.
DOI: 10.5220/0013016300003886

In *Proceedings of the 1st International Conference on Explainable AI for Neural and Symbolic Methods (EXPLAINS 2024)*, pages 147-154
ISBN: 978-989-758-720-7

devices can come in all shapes and sizes, these small, low-cost edge devices can be fully customised to accommodate any device and slip into the smart city jigsaw with ease. As the technology develops, the upgrading process is much easier and cheaper with only the localised devices needing attention and not the full network (Khan et al., 2020). Having this amount of processing power available in small devices is enabling data processing closer to the source. Being able to make sense of the data locally allows quicker decisions as well as reducing transfer traffic and storage (Shi et al., 2016).

Furthermore, explainable edge computing can contribute to provide transparent applications closer to data sources such as IoT devices or local edge servers. This proximity to data at its source can deliver benefits, including not only faster insights but also explainable decisions along with improved response times and better bandwidth availability (IBM, 2024). Therefore, smart security often uses edge computing in conjunction with intelligent vision applications (Sharma and Kanwal, 2024b) in new-generation, real-time video surveillance systems (Visionplatform, 2024), in order to process and analyze visual data at the network's edge, close to where data is generated, whilst reducing latency and bandwidth usage.

However, most existing edge computing systems for smart surveillance embed deep learning algorithms, (Perwaiz et al., 2024), (Aminiyeganeh et al., 2024), (Sharma and Kanwal, 2024a) which are very popular but still not considered as explainable or transparent (Chamola et al., 2023).

So, in this study, we developed an explainable edge computing system, including two complementary applications - one (i.e. the *support application*) running on the main user's device and the other one (i.e. the *edge application*) integrated on the edge device, in order to detect, capture, and identify individuals from live videos in context of smart cities.

It is worth noting that our intelligent vision system is explainable by design and consists in the edge application embedded on the edge device with face and motion detection capabilities. Thus, the edge device is able to make trustworthy decisions on facial recognition by making the difference between known and unknown faces which it categorises appropriately.

On the other hand, a user can receive as much or as little information as they like with the support application designed to split data from the edge application to allow the user to receive all new information, known/unknown faces, or motion detection. This gives full flexibility on the users' end to select what they require, how, and when.

Thence, the main contributions of this work are the explainable design and user-friendly development as well as the real-world deployment of the explainable edge computing system for smart security.

The paper is structured as follows. In Section 2, we describe the developed explainable edge computing system. Then, we present some experiments using our system in real-world environment, as reported in Section 3. Conclusions are drawn up in Section 4.

2 DEVELOPED SYSTEM

2.1 System Design

2.1.1 Communication

The main aim for the smart security application is to run remotely on a small edge device, and this implies the method of communication needs to be simple but robust. To make the most of the connectivity features available in current small edge suitable devices such as single board devices, the most common of these communication technologies had to be explored. Hence, Bluetooth is a very cheap add-on for any device. However, it was not considered in this work due to its poor connection range. On the other hand, Wi-Fi was considered and used for some testing, but this could not be the only means of connection due to not all devices having this capability. Therefore, to cater for the most widely available option, Ethernet has been chosen as the main source for connectivity, which allows for the best transfer speeds, while Wi-Fi, which is almost as quick as Ethernet, has been used when the range and needs for extra equipment were considered as acceptable.

Besides, to provide the best flexibility, one of the most common web services which is called a Representational State Transfer Application Programming Interface (REST API) was selected (Au-Yeung, 2020). Indeed, a REST API enables applications to communicate with a server across a network. Data and commands are transferred as JavaScript Object Notation (Json), meaning they are not language specific. A REST API works by handling GET, POST, PUT and DELETE requests through URLs, controlling the direction and type of data being sent or received (RedHat, 2020). The REST API is handled by the micro web framework Flask enabling the application to send and receive Hypertext Transfer Protocol (HTTP) requests or responses.

2.1.2 Execution

To enable any camera to be smart implies external processing and the possible use of a third-party li-

brary. Facilitating this for a low-powered, limited processing edge device requires careful considerations. Indeed, the process of taking an image and transforming it into usable data involves advanced computer vision techniques (Sharma and Kanwal, 2024a), and there are different algorithms for face recognition to choose from (Rostum and Vasarhelyi, 2024), (Chen and Krzyzak, 2024), (Wood and Olszewska, 2012).

In particular, this system needs to integrate image/video processing methods for detecting, capturing and identifying an individual from live videos. Detecting an individual involves locating a human face in a live video or a still image. Capturing is the process of taking that information and storing it in a way that can be used, and identifying it is knowing who the detected person is (Thales, 2020). Moreover, this system aims to be an explainable edge computing system. Therefore, the Histogram of Oriented Gradients (HOG) algorithm which is based on an explainable machine learning approach (Chamola et al., 2023), has been selected. It uses a black and white or grey scale version of an image, where each pixel is compared to the ones round about. From this, the change in gradient is noted, creating a detailed pattern of the image that can be matched with a face database for face recognition and identification purposes. Another benefit of these patterns is that it is also very powerful at detecting shapes or moving objects which is why it is the best option for this work.

2.1.3 Storage

One of the main advantages of a small edge device is the ability to make quick decisions, reducing thus the amount of data transfer, but limiting the scope for storage types that can be considered - with cloud or remote varieties not suitable. Additionally, not all edge devices are capable of running an operating system (OS) such as MS Windows. Therefore, cross compatibility is required (Wheeler and Olszewska, 2022).

Despite MySQL being a highly popular management system and available for all OS's, its recommended memory requirements of 4-8GB make it not suitable for this type of device. Another popular database called PostgreSQL is also unsuitable due to its high storage requirements of 512 MB.

Indeed, for a device with limited storage, processing power and possible transfer rates, the database needs to be lightweight. With these requirements, the Python-based TinyDB fits the bill. Storing data in Json format fits also well with the REST API side of the application. In addition, the database takes up minimum storage and is user friendly. A TinyDB can be setup like any other with a single database containing different tables. Due to the miniscule space each one takes up, it is easier, and more manageable, to have a database for each section, as shown in Fig. 1.

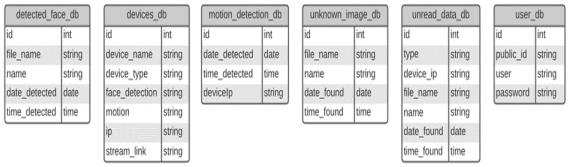


Figure 1: Edge application storage diagram.

Figure 2 illustrates how the data flows in our system and how the different databases are managed. Hence, the users make requests to the edge device from the application on their computer to access the data through the API call. The API can then control the retrieval or insertion of the data to the correct location. From the edge software, the detected data is programmatically directed to the correct location to be stored until required.

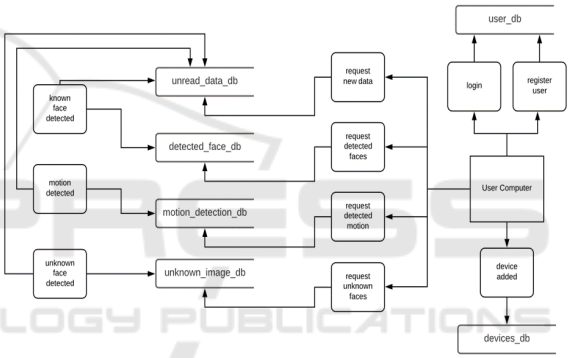


Figure 2: System database management diagram.

2.2 System Applications

Our system consists of two different applications, namely, the main *edge application* and a *support application*, as illustrated in Fig. 3. The edge application is performing the main data and video processing, while the support application demonstrates how the edge device can be integrated into other systems.

2.2.1 Edge Application

The edge application has been written mainly using Python 3 version and has been structured in five classes (see Fig. 4), with the main class `api_server` handling the communication between the application and external network. The video processing has been split into a class each for facial recognition and motion detection with another for processing both. Keeping these separate enable them to be run individually or together with these processes being managed by the `start_process` class.

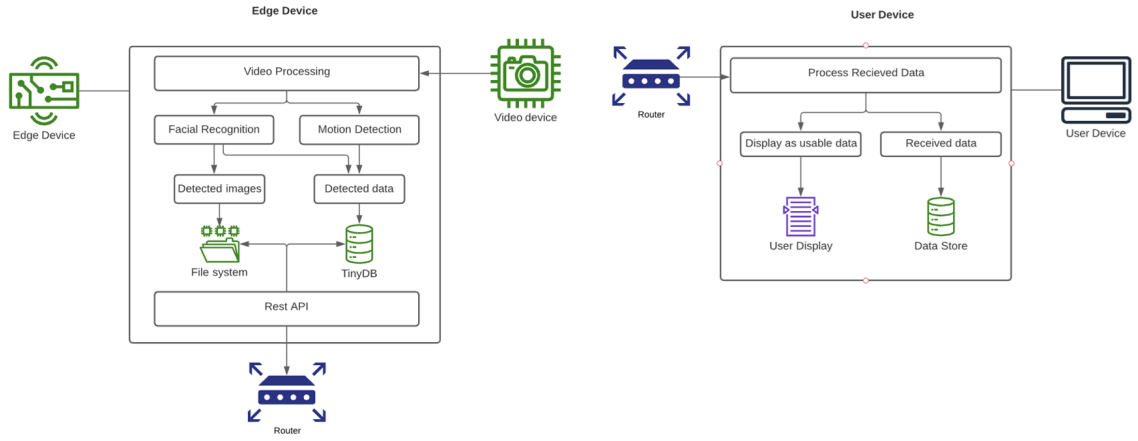


Figure 3: Our system architecture.

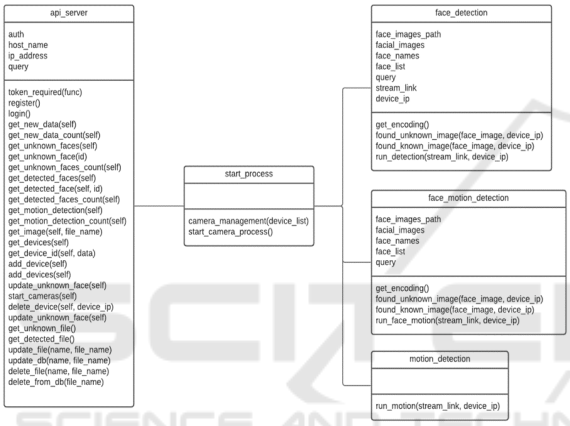


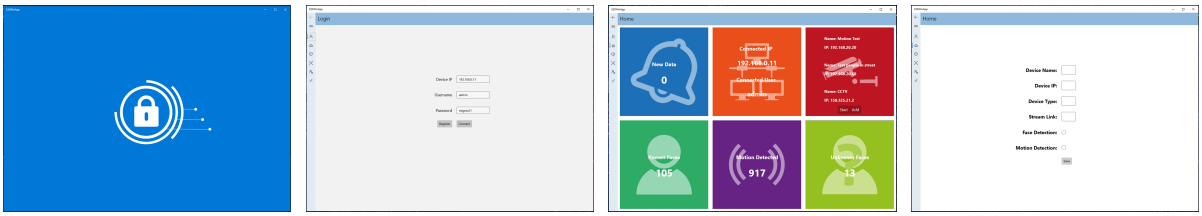
Figure 4: Class diagram for the Edge Application.

To allow for multiple camera feeds to be processed simultaneously, the edge application runs the `concurrent.futures` library. Available since Python 3.2, it has two subclasses, namely, `ThreadPoolExecutor` and `ProcessPoolExecutor`, using multi-threading and multi-processing, respectively. What makes this better than other multi-processors is that tasks are added to a pool which allocates them to available resources. This pooling increases speed and reduces computing power by creating a repository of all the active tasks and only releases them when the resources are available (Forbes, 2017).

On the other hand, the edge application uses the Open Source Computer Vision Library (OpenCV) with the Dlib library to capture, store and retrieve data from images and video. The Dlib library is a toolkit originally written in C++ containing machine learning algorithms. A number of these tools have been made available for use with Python by means of an API. OpenCV handles a number of these pretrained algorithms which for this application is mainly the facial landmark detector for HOG encoding. Therefore, using OpenCV alongside the facial recognition library (Geitgey, 2020) enables the edge application to inspect live video footage for face identification. In particular, as OpenCV stores colour in the BGR colour space, this needs to be converted in the more usable RGB. Using the video capture class to gain the footage as an array of images, the application can then scan each frame using the facial recognition face locations class. This uses the pretrained Dlib HOG SVM to find any facial patterns and the popular library NumPy to return the patterns as array. These patterns can then be compared to the pre-saved images of known faces through the face recognition compare faces class and the similarity can be confirmed using the face distance class. The found faces are shown using the OpenCV rectangle class which enables a box to be displayed using the outer most co-ordinates of the detected pattern. If the face is known, i.e. matches a stored image, the image filename, date, and time are saved to the tiny database table `detected_face` using the `insert` method from the table class (Siemens, 2023). If the detected face is not known, i.e. does not match any of the stored images, the image with the newly detected face is saved to a local folder using the OpenCV function `imwrite`. The details are then saved to the `unknown_image` table in the TinyDB, all the details are also saved to the `unread_data` table.

The motion detection also utilises the video capture class to gain the live video footage as a sequence of images. This allows the first and second frame to be captured using the function `read` which returns decoded images that can then be compared to each other using the function `absdiff` which calculates the absolute difference for each element in the two arrays.

When motion or a difference in between the frames has been detected, the new image containing the difference is simplified to a gray-scale, Gaussian-



(a) (b) (c) (d)

Figure 5: Support Application screens: (a) splash screen; (b) login/register screen, (c) home screen; (d) add-device screen.

blurred image by applying OpenCV functions This image is then filtered to reduce noise by removing pixels with too high and low values. These values are then dilated to over-emphasise the edges and enable the contours to be used to box off the different area or detected motion. As with the face detection, the date and time of the detected motion is inserted into the TinyDb along with the details of the device IP from which the detection was found.

2.2.2 Support Application

To highlight the versatility of the edge application and RESTful API, the support application (see Fig. 5 (a)) has been built on the .Net Core platform or Universal Windows Platform desktop application. The layout for the page views are created using the Extensible Application Markup Language (XAML) using grids to control the position of visual tools, e.g. TextBlock.

The main and most important screen is the login/register page, as displays in Fig. 5 (b), as this is how the user will connect to the actual edge device. From here, there is a home screen (see Fig. 5 (c)) which provides the user with an overview of the available statistics together with the functionality to start and add devices. Using the add-device screen (see Fig. 5 (d)), all the details can be added for a camera with options for the detection setting with face and/or motion selectable.

The data provided on the data screen (see Fig. 5 (c)) gives a breakdown of all the data that has been detected since last viewing. When the API from this page has been called, the data is wiped meaning no new repeat data will be sent providing the user with a data stream that can be stored locally.

3 EXPERIMENTS

3.1 Test Setup

To fully test both applications, the developed edge software was installed on a Raspberry Pi device (Raspberry Pi 4 Model B, Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC 1.5GHz, 4 GB RAM, Ubuntu 20.10 OS). The support application was installed on a laptop (Dell Latitude E5530, Intel Core i7-3610QM CPU 2.30GHz, 8GB RAM, Windows 10 Professional (20H2) OS) which was communicating with the edge device, as shown in Fig. 6. The support application was then used to receive, collect and display data from the edge device.

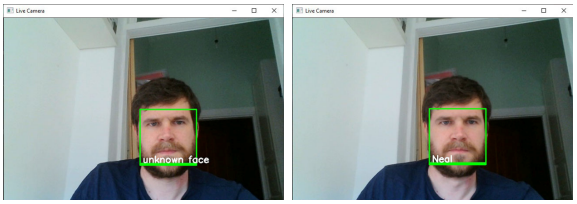
Figure 6: Overview of the system hardware setup deployed in real-world environment.

3.2 Face Detection

The initial testing of our software using the laptop's built-in webcam device involved five different individuals and was performed to test the functionality for the face detection where the face was detected as 'unknown' after comparing the image against the saved images in the database, as exemplified in Fig. 7(a). This unknown detected image was then saved through the support application as a 'known' person who was then detected when carrying out the same test again, as illustrated in Fig. 7(b).

Both devices detected the same number of faces in each of the 35 carried out tests in this series, and the results on both the laptop and the Raspberry Pi devices gave an accuracy of 97.14%.

A further series of 200 tests was carried out to test the full system using a CCTV test video from Panasonic and containing nine different persons who were evolving in that video over time. Both devices were set to 24fps with the same test video and subject images. Each test was repeated twice - at first,



(a) (b)

Figure 7: Example of face detection: (a) a detected 'unknown face'; (b) added to the database and then updated to 'known face'.

all the faces were 'unknown' and secondly, all the faces were 'known'. For these latter series of tests on the CCTV video, the cumulative computational time were measured on both devices. For the series of tests which were implemented with no known faces, the laptop completed the full batch of tests in 12 hours 8 minutes, while the Raspberry Pi took only 6 hours 17 minutes. The other series of tests using all the known faces was completed much quicker, with the laptop taking 9 hours 20 minutes, and the Raspberry Pi only 4 hours 44 minutes. These results were representational of the real-world situation, where a much higher processing time is needed for detecting an unknown face rather than a known face when performing a one-to-many search in the database, leading to the computational processing time of the overall system being not only dependent of the image processing approaches but also of the one-to-many search algorithm complexity. In terms of performance, the first series of tests showed a precision of over 90%, while the second series of tests reached a precision of 100% where no person was missed in any of the tests, and everyone was detected correctly. Furthermore, even with the lighting reflections and background clutter, there were no false positives detected. Therefore, these tests highlight that the system performance is related to the training dataset which quality needs to be carefully considered when deploying such edge computing system.

Besides, during testing on the laptop, it was noted through its Task Manager that the application used 20-25% CPU and peaked at almost 2GB memory but steadied out at 250-300MB with the power usage very high. Having the application running in the background did not appear to cause any performance issues to other applications being used. In comparison, the Raspberry Pi monitored via its activity manager has used more processing power, between 25-35% but the memory usage was 250-400MB with a peak of almost 600MB. When using video for testing on the Raspberry Pi, it was noted that if the device ran hot, with the temperature symbol flashing in the top corner, its CPU usage rose to 75%. So, additional caution should be exercised with handling the Raspberry Pi device as it should be kept in a cool place out of the sun, with an optional fan fitted.

3.3 Motion Detection

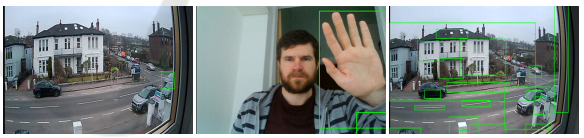
For the motion detection, the system sensitivity can be adjusted, depending on the device used and subjects needed. This is achieved by increasing or decreasing the `contourArea` which is the area used to compute the object motion.

Hence, tests were carried out for two different scenarios, as shown in Fig. 8, with a close webcam and distant CCTV camera which required very different values.

On one hand, Fig. 8 (a) illustrates a typical environment this type of device is used for and performs well - only detecting the motion of people and cars. It should be noted this test was carried out on quite a still day. However before settling on, the final value motion was detected on the trees.

On the other hand, Fig. 8 (b) demonstrates the other end of the motion scale, with a webcam used to test close quarters' motion. For this test, the `contourArea` setting was much higher, to allow for the closer and therefore larger objects' motion detection.

Besides, caution should be taken when using the motion detection feature of the edge computing system that the device is fixed and so it does not move, in order to avoid false positive detections due to the camera shake, as one can observed in Fig. 8 (c).



(a) (b) (c)

Figure 8: Examples of motion detection: (a) on high sensitivity; (b) on low sensitivity; (c) caused by camera shake.

3.4 Discussion

The developed applications, namely, edge application and support application, were both successfully tested using standard and advanced software testing approaches (Black et al., 2022), with some series of tests described in Sections 4.2-4.3, in order to deliver a reliable edge computing system.

Furthermore, this system's development process also included user tests involving the participation of five individuals with different technical knowledge and backgrounds, to provide feedback helping the developer to achieve a user-friendly system.

Besides, during the development of this explainable edge system, ethical considerations were analysed as well, e.g. to ensure that data and sensitive information were stored and handled in an ethical way. It is worth noting that the use of CCTV with smart technology such as face recognition is a relatively new concept with the laws and e guidelines still being created at national and international levels (e.g. the General Data Protection Regulation (GDPR), EU AI Act 2024, etc.). Although this application does not venture into this area of detail or intrusiveness, it highlights many possible issues the public had and still has with this area. As CCTV cameras become more powerful and smaller with high quality lenses and image processing, it is important they are used appropriately (Van Noorden, 2020). Albeit for this application, consent would not be available due to the device being aimed at the security sector, full consideration into the handling, storing, and removal of data has been considered. Indeed, full control over the naming and storing has been programmed in to make it easy to add or remove images and details. Moreover, all companies using this edge computing system are expected to adhere to government guidelines.

It is worth noting that many of the issues relating to ethics can be improved by the security of the device (Quincozes et al., 2024) and careful consideration over storage (Ahmadi, 2024). Thence, this application is running on an Ubuntu device which is password protected. The communication is also secured with a password, while the main source of storage of the Raspberry Pi, which is the memory card, can be encrypted should it need to be. For a user to access the edge device, they need to register their details by sending them using the `HttpClient` which encodes them and sends them through a Basic `AuthenticationHeaderValue` (Microsoft, 2024). The API server on the edge side stores the submitted details in the TinyDb `users_db` table and encrypts the password using the Python library Werkzeug function `generate_password_hash`. The encryption is created using SHA256 hash algorithm with salt, which is random extra data for added security in the format `method$salt$hash`. The added salt encryption helps disguise the same password used by more than one user. SHA265 is a variant of the highly popular SHA-2 which sets the size to 256 bits, meaning all hashed codes are the same size. To check the credentials on login, the function `check_password_hash` compares the received password against any stored password and returns true or false. For added security, a random twelve-digit public ID is generated during registration which will be used for creating a token on login. When a registered user logs into the edge device, their details are checked against the contents of `user_db` for a match. If all the details are confirmed, a token is generated using the library PyJWT or Json web token which uses the HS256 algorithm. HS265 is a Hash-based Message Authentication Code (HMAC) combining, in this case, a predefined secret key to encode the generated user public ID using H265 (Otemuyiwa, 2017). For extra security, an expiry date can be added to the token to force users to have to login and to stop old tokens being active longer than they need to be. On the Windows support application, the token is returned on login through the `HttpResponseMessage` and stored locally in memory. Any future requests made to the edge device for data are thus sent using this token as Digest authorisation. These received requests are then decoded using the secret key to gain the user public key which is check against the results in the `user_db`.

4 CONCLUSIONS

In this work, we developed an explainable edge computing prototype for the next-generation smart environments which require privacy, security and trust. Hence, we built a system running two applications, i.e. a user-friendly application to support the main edge application integrating an intrinsically explainable computer vision technologies together with a small, low-powered edge computing device, in order to provide an accurate whilst responsible AI-based solution for real-world smart security.

REFERENCES

Ahmadi, S. (2024). Security Iimplications of edge computing in cloud networks. *Journal of Computer and Communications*, 12:26–46.

Aminiyeganeh, K., Coutinho, R. W. L., and Boukerche, A. (2024). IoT video analytics for surveillance-based systems in smart cities. *Computer Communications*, 224:95–105.

Au-Yeung, J. (2020). Best practices for REST API design. Available online at: https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/.

Black, R., Davenport, J. H., Olszewska, J. I., Roessler, J., Smith, A. L., and Wright, J. (2022). *Artificial Intelligence and Software Testing*. BCS Press.

Chamola, V., Hassija, V., Sulthana, A. R., Ghosh, D., Dhingra, D., and Sikdar, B. (2023). A review of trustworthy and explainable artificial intelligence (XAI). *IEEE Access*, 11:78994–79015.

Chen, G. and Krzyzak, A. F. (2024). Face recognition via selective denoising, filter faces and HOG fea-

tures. *Signal, Image and Video Processing. Springer*, 18:369–378.

Forbes, E. (2017). Executors and pools. In *Learning Concurrency in Python*, pages 159–188. Packt.

Geitgey, A. (2020). `face_recognition` package. Available online at: https://face-recognition.readthedocs.io/en/latest/face_recognition.html.

Houghtaling, M. A., Fiorini, S. R., Fabiano, N., Goncalves, P. J. S., Ulgen, O., Haidegger, T., Carbonera, J. L., Olszewska, J. I., Page, B., Murahwi, Z., and Prestes, E. (2024). Standardizing an ontology for ethically aligned robotic and autonomous systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 54(3):1791–1804.

IBM (2024). What is edge computing? Available online at: http://www.ibm.com/topics/edge-computing .

Khan, L. U., Yaqoob, I., Tran, N. H., Kazmi, S. M. A., Dang, T. N., and Hong, C. S. (2020). Edge-computing-enabled smart cities: A comprehensive survey. *IEEE Internet of Things Journal*, 7(10):10200–10232.

Lea, P. (2020). *IoT and Edge Computing for Architects*. 2nd ed. Packt.

Li, Y., Xiao, Y., Gong, Y., Zhang, R., Huo, Y., and Wu, Y. (2024). Explainable AI: A way to achieve trustworthy AI. In *Proceedings of the IEEE International Conference on Big Data Security on Cloud*, pages 150–155.

Microsoft (2024). .NET documentation. Available online at: https://docs.microsoft.com/en-gb/dotnet/.

Mishra, A. K., Reddy, R. R., Tyagi, A. K., and Arowolo, M. O. (2024). Artificial intelligence-enabled edge computing: Necessity of next generation future computing system. In *IoT Edge Intelligence*, page 67–109. Springer.

Naveen, S. and Kounte, M. R. (2019). Key technologies and challenges in IoT edge computing. In *Proceedings of the IEEE International Conference on IoT in Social, Mobile, Analytics and Cloud*, pages 61–65.

Otemuyiwa, P. (2017). Brute Forcing HS256 is Possible: The Importance of Using Strong Keys in Signing JWTs. Available online at: https://auth0.com/blog/brute-forcing-hs256-is-possible-the-importance-of-using-strong-keys-to-sign-jwts/.

Perwaiz, N., Fraz, M. M., and Shahzad, M. (2024). Smart surveillance with simultaneous person detection and re-identification. *Multimedia Tools and Applications. Springer*, 83:15461–15482.

Quincozes, V. E., Quincozes, S. E., Kazienko, J. F., Gama, S., Cheikhrouhou, O., and Koubaa, A. (2024). A survey on IoT application layer protocols, security challenges, and the role of explainable AI in IoT (XAIoT). *International Journal of Information Security*, 23:1975–2002.

RedHat (2020). What is a REST API? Available online at: https://www.redhat.com/en/topics/api/what-is-a-rest-api.

Rostum, H. M. and Vasarhelyi, J. (2024). Comparing the effectiveness and performance of image processing algorithms in face recognition. In *Proceedings of the IEEE International Carpathian Control Conference*, pages 1–5.

Sharma, H. and Kanwal, N. (2024a). Smart surveillance using IoT: A review. *Radioelectronic and Computer Systems*, 109(1):116–126.

Sharma, H. and Kanwal, N. (2024b). Video surveillance in smart cities: Current status, challenges and future directions. *Multimedia Tools and Applications. Springer*, pages 1–46.

Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646.

Siemens (2023). TinyDB. Available online at: https://github.com/msiemens/tinydb.

Slee, D., Cain, S., Vichare, P., and Olszewska, J. I. (2021). Smart lifts: An ontological perspective. In *Proceedings of the International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (KEOD)*, pages 210–219.

Sodiya, E. O., Umoga, U. J., Obaigbena, A., Jacks, B. S., Ugwuanyi, E. D., Daraojimba, A. I., and Lottu, O. A. (2024). Current state and prospects of edge computing within the Internet of Things (IoT) ecosystem. *International Journal of Science and Research Archive*, 11(1):1863–1873.

Thales (2020). Facial recognition: top 7 trends. Available online at: https://www.thalesgroup.com/en/markets/digital-identity-and-security/government/biometrics/facial-recognition .

Utimaco (2024). What is Smart Security?. Available online at: https://utimaco.com/service/knowledge-base/emergency-communications-and-public-warnings/what-smart-security.

Van Noorden, R. (2020). The ethical questions that haunt facial-recognition research. *Nature*, 587:354–358.

Visionplatform (2024). What is edge computing for computer vision and what are the benefits?. Available online at: https://visionplatform.ai/what-is-edge-computing-for-computer-vision-and-what-are-the-benefits/ .

Wheeler, D. and Olszewska, J. I. (2022). Cross-platform mobile application development for smart services. In *Proceedings of the IEEE International Symposium on Computational Intelligence*, pages 203–208.

Wood, R. and Olszewska, J. I. (2012). Lighting-variable AdaBoost based-on system for robust face detection. In *Proceedings of the International Conference on Bio-Inspired Systems and Signal Processing*, pages 494–497.

Xiao, Y., Huo, Y., Cai, J., Gong, Y., Liang, W., and Kolodziej, J. (2024). ERF-XGB: An Edge-IoT-based explainable model for predictive maintenance. *IEEE Transactions on Consumer Electronics*, 70(1):4016–4025.

Zhu, Y., Al-Ahmed, S. A., Shakir, M. Z., and Olszewska, J. I. (2023). LSTM-based IoT-enabled CO2 steady-state forecasting for indoor air quality monitoring. *Electronics*, 12(1):1–12.