INDUSTRIAL TRAINING REPORT

Internship training report on

Smart Vision for Visually Impaired

Submitted in partial fulfillment of the
Requirements for the award of

Degree of Bachelor of Engineering in Electronics and Communication

Carried out at

Einetcorp Pvt Ltd

Submitted by

A S V Dheeraj
01FE21BEC161

Under the guidance of
SUBMITTED TO

Prof. Sheela A. B.
College Guide
KLE Technological University

Mr. Manoj Bhat
Industry Guide
Einetcorp Pvt. Ltd.

SUBMITTED TO:
School of Electronics and Communication Engineering
KLE TECHNOLOGICAL UNIVERSITY
Hubballi

K.L.E SOCIETY'S
KLE Technological University,
HUBBALLI-580031
2024-2025

SCHOOL OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the internship project entitled **"Smart Vision for Visually Impaired"** is a bonafide work carried out by **"A S V Dheeraj"**, bearing University Seat No. **01FE21BEC161** in **Einetcorp Pvt. ltd.**, in partial fulfillment for the award for Bachelor of Engineering in Electronics and Communication in the School of Electronics and Communication Engineering of KLE Technological University, Hubballi for the academic year 2024-2025.

Prof. Sheela A. B.	**Dr. Suneeta V. Budihal**	**Dr. Basavaraj S. Anami**
Guide	**Head of School**	**Registrar**

External Viva:

Name of Examiners	**Signature with date**
1.	
2.	

EINETCORP

CIN
U72900KA2021PTC145676

Certificate of Internship

This is to certify that **A S V Dheeraj** bearing USN: 01FE21BEC161 has successfully completed an internship with EiNETCORP Pvt Ltd as a Machine Learning intern from the **15th of Jan 2025 to the 30th of June 2025.**

During his tenure of internship, **Mr. A S V Dheeraj** has worked on Implementing Optical Character Recognition (OCR) on Embedded Systems and Developing an Application Layer Pipeline and similar activities.

Besides exhibiting high comprehension capacity, he has demonstrated his skills with self-motivation to learn new ones. He has maintained an outstanding professional demeanor and showcased excellent moral character throughout the internship period.

We hereby certify that the candidate's overall work is good and satisfactory to the best of our knowledge.

Wishing the candidate all the best in his future endeavors.

For EiNETCORP PVT. LTD.

DIRECTOR

Manoj Bhat,
Founder Director
EiNETCORP Pvt Ltd

EINETCORP PVT. LTD. HUBLI.

//CERTIFIED TRUECOPY//

Phone +91 876 223 3319 | **Email** connect.einetcorp@gmail.com | KLE TECH PARK, BVB CAMPUS, Hubballi, 580031

DECLARATION

I hereby declare that the Industrial Training Report entitled **"Smart Vision for Visually Impaired"** is an authentic record of my own work as requirements of Industrial Project during the period from 15/01/2025 to 30/06/2025 for the award of degree of B.E. KLE Technological University, Hubballi under the guidance of **Prof. Sheela A. B.** and Industry guide **Manoj Bhat**

A S V Dheeraj
01FE21BEC161

Date :

ACKNOWLEDGMENT

The sense of contentment and elation that accompanies the successful completion of the Industrial project report and the report would be incomplete without mentioning the names of the people who helped in accomplishing this. I absolutely respect the inspiration, support and steering of all the people who have been instrumental in making this project fulfillment. I being the student of KLE Technological University, feel extremely grateful to Einetcorp Private Limited and the college for self assurance best owed in us and for entrusting our mission entitled **Einetcorp Private Limited Internship.**

It gives me immense pleasure to express my deepest sense of gratitude and sincere thanks to my highly respected and esteemed guides **Mr. Manoj Bhat** for their valuable guidance, encouragement and help for completing this work. Their useful suggestions for this whole work and co-operative behavior is sincerely acknowledged.

I also express my gratitude to **Prof. Sheela A. B.**, University guide, for her constant support and guidance. I also wish to express my gratitude to **Dr. Suneeta V. Budihal**, HOS of School of Electronics and Communication Engineering.

I would like to express my sincere thanks to **Dr. Ashok Shettar**, Vice Chancellor, KLE Technological University, Hubli for his kind-hearted support and for giving us this opportunity to undertake this project. Also, I like to thank **Mr. P.G.Tewari**, Principal for his whole-hearted support.

-A S V Dheeraj

ABSTRACT

This internship project focused on the development and deployment of a real-time object detection and text-to-speech (TTS) system on the Raspberry Pi CM5 platform, utilizing the Blaize Xplorer X600M M.2 AI accelerator. The project aimed to enable low-latency visual perception and audio feedback by integrating optimized YOLOv8/YOLOv11 object detection models with a TTS engine for voice-based output. The Blaize Picasso SDK and GStreamer pipeline framework were used to build efficient inference pipelines incorporating modules such as Kalman SORT tracking, bounding box rendering, and real-time label extraction. Detected objects were translated into natural speech using an embedded TTS model, creating a closed-loop assistive system suitable for edge AI applications. The solution demonstrated reliable performance under constrained hardware conditions, highlighting the potential of combining AI vision and audio technologies for real-world use cases such as assistive devices and smart monitoring systems. This project provided hands-on experience in hardware-software integration, edge AI optimization, and the deployment of multi-modal inference pipelines on embedded platforms.

Contents

Chapter 1

Introduction

The field of image processing has experienced tremendous growth in recent years, fueled by the progress in computer vision, artificial intelligence, and the increasing demand for efficient image analysis and understanding across various applications. Image processing algorithms play a critical role in numerous domains such as medical imaging, surveillance, autonomous vehicles, robotics, and many others. As these algorithms become more complex and computationally intensive, there is a growing need for powerful and dedicated hardware platforms that can execute them in real-time with high efficiency.

The Raspberry Pi Compute Module 5 (CM5), when integrated with the Blaize Xplorer X600M AI accelerator, creates a powerful and energy-efficient edge AI platform that can be used for a variety of real-time computer vision and machine learning activities. This combination combines the Blaize Graph Streaming Processor (GSP) architecture's specialized AI inferencing capabilities with the adaptability of the Raspberry Pi ecosystem, making it ideal for embedded vision applications like industrial inspection, smart surveillance, assistive technology, and more.

First, this platform's main strength is the combination of the Blaize accelerator's specialized AI compute fabric with the CM5's quad-core processing power. The Blaize X600M takes over computationally demanding AI tasks like object detection, text recognition, and tracking, while the Raspberry Pi manages system coordination, I/O, and user interfaces. With an average power consumption of only 3W, the accelerator has 16 GSP cores and can handle up to 6 TOPs of computation performance. Low-latency, real-time inferencing is made possible by this architecture, which would otherwise be impossible to achieve with just general-purpose CPUs. Time-sensitive edge applications benefit greatly from the GSP's streaming capability, which also enables continuous data flow processing with no overhead.

Second, this platform stands out due in large part to its form factor and energy efficiency. This configuration is perfect for embedded and wearable use cases because of the small size of the CM5 and the low-power design of the Blaize X600M. In contrast to high-end GPUs, which require massive enclosures and active cooling, this system may function well in thermally limited locations without compromising processing speed. Because of this, it is especially well-suited for deployments that are transportable, battery-operated, or on the rugged edge and have constrained power and thermal budgets.

Furthermore, the Raspberry Pi foundation and Blaize's software ecosystem speed up deployment and improve developer productivity. Using pre-built sample apps, NetDeploy model translation tools, and GStreamer plugins, the Blaize Picasso SDK provides a robust yet user-friendly interface for building AI pipelines. With only minor code modifications,

developers may apply preprocessing and postprocessing stages, incorporate well-known ONNX-based models, and perform optimal inference. The Raspberry Pi platform, on the other hand, guarantees wide Linux compatibility, GPIO access for hardware interface, and a wealth of community-supported libraries. When combined, these software features accelerate the development process, shorten integration times, and offer adaptability across various AI processes.

In conclusion, the Blaize Xplorer X600M M.2 accelerator and Raspberry Pi CM5 together provide an extremely powerful, low-power, and scalable AI computing platform for edge deployments. It is a promising option for next-generation embedded AI systems because of its special combination of modularity, performance, and software support, which allows developers to construct advanced vision applications in real-time.

1.1 Company Profile

Einetcorp is a Embedded AI company based in KLE Techpark Incubation Center Hubballi, specializing in Computer Vision that helps visually impaired people to become independent by developing and deploying Deep Learning applications on Edge devices. Founded in 2021, Einetcorp is a privately held company. Our team of industry experts is dedicated to driving innovation, shaping the future, and delivering top-notch AI solutions that helps visually impaired.

At Einetcorp, harnessing the power of Artificial Intelligence (AI), Machine Learning (ML), Deep Learning, Computer Vision, and Edge AI to provide cutting-edge solutions that drive enterprise-level transformations. Our comprehensive suite of technologies and expertise allows us to develop AI solutions tailored to the unique needs. The focus is on delivering results and equipping the clients with the tools and capabilities to leverage AI effectively.

Einetcorp Pvt Ltd at KLE Techpark Incubation Center emphasis on interdisciplinary collaborations bringing together specialists from many sectors such as engineering, AI, embedded systems, etc. Furthermore, the center provides opportunities for students to get practical experience through internships and research projects. The center prepares students for today's competitive landscape by giving access to cutting-edge devices.

Driven by a passion for leveraging deep technologies, continuing to push the boundaries of AI to unlock new possibilities. With a focus on innovation, reliability, Einetcorp is committed to empowering individuals with visual disabilities to harness the full potential of AI and enhance their quality of life.

In summary, Einetcorp is an Embedded AI company focused on delivering innovative computer vision solutions for the visually impaired. With expertise in Deep Learning, Computer Vision, and Edge AI, the company is committed to enabling independence and accessibility through cutting-edge AI applications deployed on edge devices.

Chapter 2

Technical Section

The implementation of image processing modules utilizing the Blaize AI accelerator platform is the main topic of this report's technical portion. This section explores the technical complexities of putting AI-based vision algorithms into practice utilizing the GStreamer-based pipeline architecture designed for effective edge inferencing, the Blaize Picasso SDK, and the NetDeploy model optimization procedure.

2.1 Blaize X600M M.2 Accelerator and Picasso SDK

In this section, the implementation of the Blaize Picasso SDK for deploying AI-powered vision modules is discussed, along with a technical exploration of its integration with NetDeploy for model conversion and its interaction with GStreamer-based plugins to enable efficient execution on edge AI hardware.

2.1.1 Blaize

The type of hardware known as an AI accelerator is specifically designed to maximize the performance of tasks related to artificial intelligence (AI) and machine learning (ML), especially those involving computer vision and deep learning inference. These accelerators are particularly well-suited for edge computing environments and real-time intelligent systems because they strive to deliver great computational throughput while maintaining power efficiency.

By providing hardware architectures specifically designed for matrix operations, tensor calculations, and parallel processing—essential elements of deep neural network workloads—AI accelerators set themselves apart from general-purpose processors. Accelerators such as NPUs (Neural Processing Units), DSPs (Digital Signal Processors), and GSPs (Graph Streaming Processors) offer a dedicated, low-latency pathway for inferencing, frequently with lower power consumption and smaller physical footprints, even though traditional CPUs and GPUs can perform AI tasks. This specialization allows developers to offload computationally intensive tasks from the host CPU, enabling more efficient system design and improved application responsiveness.

Support for software-defined programmability is a key feature for modern AI accelerators. Flexible integration into a variety of AI workflows, such as image segmentation, object detection, and natural language processing, is made possible by this approach. Graph compilers, optimization frameworks, and deployment tools are common components of accelerator providers' toolchains and SDKs, which enable smooth model transla-

tion and runtime execution. These software stacks give developers the ability to tightly regulate memory utilization, execution order, and parallelism, which enables them to customize inference pipelines to meet certain latency or performance requirements.

Furthermore, a range of model formats and development frameworks, including ONNX, TensorFlow, PyTorch, and Caffe, are supported by AI accelerators. In addition to enabling models learned in high-level settings to be tuned and run well on hardware backends, this guarantees compatibility with commonly used AI ecosystems. Additionally, several accelerators (such INT8 and BF16) include built-in support for mixed-precision execution, which enhances performance-per-watt ratios while maintaining model accuracy.

In conclusion, the implementation of effective, high-performance AI applications has advanced significantly thanks to AI accelerators. These accelerators reduce power consumption, system cost, and development complexity while enabling the real-time execution of complex inferencing tasks across domains ranging from consumer electronics to industrial automation. They do this by combining architectural specialization, software programmability, and support for contemporary AI frameworks.

2.1.2 Blaize Picasso SDK

A robust software development kit namely the Blaize Picasso SDK is intended for the deployment of AI inference applications on Blaize's edge AI accelerators. It offers a collection of GStreamer plugins, libraries, and tools that make it possible to create high-efficiency, low-latency pipelines for real-time computer vision jobs. The Blaize Graph Streaming Processor (GSP) architecture has been tailored for the SDK, enabling smooth integration of AI models with preprocessing, inference, postprocessing, and video input phases.

The Picasso SDK's modular design is a fundamental component that enables developers to build and modify pipelines with Blaize's proprietary plugins. Additionally, it comes with the NetDeploy tool, which supports preprocessing and postprocessing settings via straightforward YAML files and transforms standard ONNX models into formats compatible with GSP. By supporting hardware-level acceleration, effective memory management, and model optimization, the Picasso SDK makes it easier to deploy AI workloads on edge devices without compromising flexibility or performance.

2.1.3 Installation of Blaize Picasso SDK:

Use Prebuilt OS Image with Picasso SDK:

- The Picasso SDK and all required drivers and tools are included in the preconfigured OS image that Blaize provides.
- This approach guarantees optimal compatibility with compatible devices and streamlines the setup procedure.
- The system is prepared to boot and start development once the image has been flashed onto an SD card or USB disk using common imaging tools.

Manual Installation of OS and SDK (alternate):

- This includes downloading the SDK package from Blaize, flashing a compatible Linux system (like Ubuntu) onto the device, and following the installation instruc-

tions in the manual.

- System dependencies, kernel drivers, and environment variables are specified for SDK functionality during installation.

Kernel Driver Setup

Following installation, it's critical to confirm that the system recognizes the Blaize accelerator correctly.

- This involves verifying for PCIe connectivity and making sure the right kernel modules—which handle media processing and hardware interface—are loaded.
- These drivers enable the system to interact with the Blaize accelerator and make it easier to carry out AI tasks once they are activated.

Running SDK Sample Applications

- The Picasso SDK includes a collection of sample apps that show off practical use cases like tracking and object detection.
- The modular and extendable character of the SDK is demonstrated by these samples, which make use of GStreamer pipelines constructed with Blaize's proprietary plugins.
- These pipelines can be investigated, altered, and expanded by developers to meet the needs of particular applications.

Directory Structure Post-Installation

- The SDK files are arranged into a specified directory structure under the system's file hierarchy after installation.
- This include shared libraries, model conversion tools, executable tools, GStreamer plugins, configuration files, and sample apps.
- These components' obvious arrangement facilitates simple navigation and effective development processes.

2.2 NetDeploy

Net deployment is the process of preparing and integrating a trained neural network model into a target environment for inference. In industrial and embedded applications, the target environment typically consists of edge devices that have limited processing power, constrained memory, and lower power budgets compared to cloud servers or training workstations. The deployment process ensures that a model trained in a high-performance environment can be made compatible with the operational and performance constraints of a real-time system.

In deep learning, models are initially trained using 64-bit or 32-bit floating-point precision (FP64 or FP32). While this high precision is ideal for achieving training accuracy, it significantly increases model size, memory usage, and computational load — making it inefficient for real-time inference on edge devices. Precision reduction, which entails converting models to lower-bit representations like P16 (16-bit floating-point) for faster

computation with less accuracy loss and INT8 (8-bit integer) for extreme optimization in terms of size and speed, is a crucial component of net deployment in order to address this.

These lower-precision models are perfect for Blaze-class technology since they use less memory, power, and allow for faster inference. Deploying a neural network model in industrial automation and smart manufacturing systems requires modifying it to account for hardware-specific limitations, real-time responsiveness, and limited computational resources.

The net deployment process usually consists of the following steps: optimizing the model's size, speed, and power efficiency; converting the model into a portable format (such as ONNX). This procedure guarantees that models are not only accurate in prediction but also efficient, dependable, and compatible with the limitations of actual industrial systems. Resources like YAML files are used to configure the runtime environment, and specialized inference hardware (such as Blaze AI accelerators) is used to run the model.

2.2.1 ONNX

A uniform representation of neural networks is necessary to make this transformation and deployment process easier. Open Neural Network Exchange, or ONNX, is essential in this situation. An open-source model format called ONNX was created to represent deep learning models in a way that is independent of hardware and framework.

The framework independence of ONNX is one of its main benefits. Regardless of the initial training framework, models trained in PyTorch, TensorFlow, or PaddlePaddle may all be converted to ONNX format, allowing for a consistent deployment strategy. The intricacy and effort needed to modify models for various platforms are greatly decreased by this interoperability. Furthermore, ONNX functions as an intermediary format that is ideal for compilation and hardware-specific optimization, making it deployment-ready in a range of settings. Additionally, it facilitates performance scaling, which enables models to run effectively on a variety of hardware accelerators, such as GPUs, CPUs, and AI edge devices frequently seen in industrial automation. Additionally, by allowing teams to adhere to a standardized and repeatable procedure from model development to runtime execution, ONNX encourages homogeneity in the deployment pipeline. In addition to improving repeatability and maintainability, this methodical approach also fosters better cooperation between the development and deployment teams.

The fact that ONNX also forms the basis for a variety of post-processing and optimization tools was a significant topic covered during the course. In order to make the model lighter and faster during inference, these tools—which include ONNX Simplifier, ONNX Optimizer, and ONNX Runtime Graph Optimizer—offer features including graph trimming, constant folding, and redundant node elimination. When aiming for hardware accelerators like Blaze AI processors that have stringent latency or memory limitations, these optimizations are crucial. Additionally, ONNX enables a smooth workflow from high-precision training models to low-precision, deployment-ready formats while maintaining functionality and accuracy when paired with quantization tools.

2.2.2 YAML Files in Model Deployment

The YAML configuration file is another essential part of the deployment process. "YAML Ain't Markup Language," or YAML, is a human-readable, lightweight data serialization

format. YAML files are configuration descriptors used in AI deployment pipelines that specify how a model should be run, how input data should be preprocessed, and how outputs should be interpreted.

The model pipeline's input shape, tensor names, batch size, picture preprocessing methods (such as scaling, normalization, or channel reordering), and output post-processing logic may all be configured by developers and system integrators using the YAML files. These setups guarantee that the neural network appropriately prepares and interprets the raw data gathered by sensors or cameras in an embedded system. Additionally, YAML allows control device-specific runtime parameters, which are crucial for edge AI platforms like Blaze. These parameters include choosing precision levels (such INT8 or FP16), modifying thread counts, or turning on acceleration capabilities. YAML is a useful tool for changing deployment behavior without having to retrain the model or recompile the software because it is both machine-readable and simple for humans to update. More maintainability and reuse are ensured by the developers' ability to expand and modify models across many systems or product lines thanks to the separation of logic and configuration. Furthermore, YAML improves version control and reproducibility in deployment pipelines, enabling teams to precisely track, audit, and reproduce model configurations. Different devices may need slightly different inference behavior in industrial AI systems. Through this structured yet flexible approach, YAML contributes significantly to the robustness, adaptability, and clarity of the overall AI deployment workflow.

2.2.3 NetDeploy Result

The NetDeploy procedure produces a number of significant output files that are essential to the model deployment, optimization, validation, and debugging phases that follow. Files like vx-model.dot, pytorch-model.dot, and vx-app are among the main outputs, as are related resources like app.log, binary parameter files, and a checkpoints folder containing internal graph representations and files like checkpoints.npy. The model's computational graph is shown graphically in the.dot files, such as vx-model.dot and pytorch-model.dot, both before and after hardware-specific optimization. These are especially helpful for illustrating the model's structure, comprehending the connections between layers and operations, and spotting areas where operator fusion or simplification might be possible. The executable in charge of executing inference on the edge device is the vx-app binary, which is the compiled program that can be installed straight into compatible Blaze hardware platforms. The app.log file is crucial for debugging and verifying the accuracy of model conversion and compilation since it captures logs from the deployment process, including failures, warnings, and status messages. The optimized and quantized model weights, which are specific to the target hardware, are also included in the binary parameter file. For model verification, calibration, and performance benchmarking, a serialized record of intermediate activation outputs or tensor states is provided via the checkpoints.npy file located inside the checkpoints directory. To confirm the quality and consistency of inference after deployment, these checkpoints can also be compared to reference outputs from the original framework (such as PyTorch). When taken as a whole, these results not only verify that the deployment procedure was completed successfully, but they also offer the required resources for hardware-specific validation, runtime integration, visualization, and inspection. This guarantees that when the model is integrated into actual industrial or edge applications, it is not only deployment-ready but also optimized, interpretable, and dependable.

2.3 Application Development

For edge AI systems, application development involves planning and coordinating a processing pipeline that links several steps, such as data intake, image preprocessing, model inference, postprocessing, and output delivery. Usually modular, these pipelines have separate components that each perform specific tasks to guarantee reusability, scalability, and clarity. Real-time picture or video feeds are processed by the pipeline utilizing optimized deep learning models in vision-based applications, and the results are interpretable for use by decision-making systems or users.

Such pipelines' architecture usually consists of a series of steps whereby raw data from image sensors is first preprocessed to conform to the neural network model's input format. The inference engine, which usually runs on an AI accelerator, receives the preprocessed data and uses it to compute outcomes like text to speech generation or object detections. Relevant information, like item labels, bounding boxes, or confidence scores, is then extracted from these results using postprocessing. This information can then be utilized for activities further down the line, such as creating visual overlays, providing audio feedback, or initiating GPIO-based hardware responses.

2.3.1 Preprocessing and Postprocessing files

In order to guarantee that the input data is prepared appropriately for the model, preprocessing is essential. Input images for the majority of deep learning models must have a set size and be normalized using predetermined scale settings. Resizing, aspect-ratio preservation, padding, color format conversion (e.g., BGR to RGB), and pixel value normalization are examples of preprocessing chores. It is crucial to preserve spatial integrity during scaling in order to avoid feature distortion, which could impair model accuracy.

Region-of-interest (ROI) cropping, which reduces computing load and concentrates on pertinent areas of the image, is another preprocessing technique that may be used. In this technique, only particular parts of the frame are sent to the model for inference. To guarantee the best inference performance, these processes must be in line with the preprocessing procedures applied during model training. for example, in OCR region of interest is detected by detector where the text is present in the image shound be passed to next model recognizer.

The processes carried out following model inference to transform unstructured outputs into information that is comprehensible to humans is postprocessing. Decoding bounding box coordinates, assigning labels to anticipated class indices, using confidence thresholding, and executing non-maximum suppression (NMS) to get rid of duplicate detections are common postprocessing steps for object detection models. When it comes to text recognition, postprocessing could entail turning character prediction sequences into legible text strings.

The transformation of coordinates from the model's input resolution back to the initial input image size is likewise handled by postprocessing. Because the outputs need to be properly overlay or aligned with the original scene, this is particularly crucial when preprocessing entails resizing and padding. The result of postprocessing is typically a structured object or list containing detected entities, their positions, class names, and confidence scores.

2.3.2 Deploying on Accelerator

Deploying the model and pipeline to an AI accelerator is the last step once the pipeline has been validated on the host system. The first step in deployment is to use the model conversion tools offered by the accelerator's SDK to transform the trained neural network model in ONNX or comparable standard formats into a hardware-optimized format. These programs also use structured configuration files in YAML format to specify input-output tensor specifications and preprocessing/postprocessing configurations.

After loading the modified model onto the accelerator, the inference pipeline is built using the SDK's own graph-based APIs or software frameworks like GStreamer. The inference computations are handled by the accelerator, which relieves the host CPU of the taxing workloads and permits effective real-time execution.

Continuous input frames are fed into the pipeline during runtime, processed by the accelerator, and then returned to the host system for interpretation or action. This method greatly lowers latency, power usage, and reliance on cloud resources, which makes it perfect for embedded AI systems that must function offline or in locations with limited resources. When properly optimized and deployed, the entire application can operate in real-time with stable performance, high throughput, and low latency.

2.4 GStreamer

A popular open-source multimedia framework for creating and implementing complex media processing pipelines is GStreamer. For operations like video capture, decoding, conversion, processing, and output rendering, it makes it possible to integrate components in a modular fashion. Its plugin-based architecture, which enables the smooth integration of custom parts into streaming workflows, is the source of its flexibility. GStreamer is particularly useful in the context of artificial intelligence and machine learning, especially computer vision, because of its capacity to manage high-throughput data streams. In order to facilitate the real-time inference of video data using deep learning models, the Blaize Picasso SDK incorporates GStreamer into its development pipeline. This SDK is intended for the deployment of neural network models on the Blaize V1P hardware accelerator.—

GStreamer serves as a bridge connecting the AI inference engine and the multimedia content pipeline. It offers an organized and effective interface for handling continuous video frames from a variety of sources, including network streams, video files, and live camera feeds. Applications like YOLOv8 object detection include first capturing or ingesting the video information, after which it is sent via an inference engine that recognizes and labels things in every frame. Developers can create an end-to-end pipeline that automatically processes video input and renders inference results in real time by incorporating a bespoke GStreamer plugin made especially for the Blaize platform.Use cases where low-latency, edge-based AI processing is essential, such smart surveillance, autonomous driving, retail analytics, and others, benefit greatly from this integration.

This integration's modularity is its strongest point. While using GStreamer's built-in ecosystem for other features like decoding, synchronization, and rendering, developers can isolate the AI inference engine into a single GStreamer plugin element. Different pipeline components can be developed and optimized independently because to this division of responsibilities. Additionally, GStreamer facilitates pipelining and parallelism, which improves the scalability and performance of AI applications on hardware with

limited resources. These ideas guided the development of the Blaize SDK, which offers comprehensive assistance for building, assembling, and implementing unique GStreamer plugins that interface with its in-house hardware acceleration engine.

2.4.1 Constructing and Installing

There are several steps in the installation and building process for integrating GStreamer with the Blaize SDK, from setting up the environment to compiling and deploying the plugin. First, a suitable development environment, usually Ubuntu-based Linux systems with a compatible kernel, is required in order to extract and install the Blaize Picasso SDK. A set of tools, sample apps, model configuration files, and documentation that walk users through the setup process are included with the SDK. Additionally, it contains a variety of neural network models that are pre-packaged in compressed archive formats, including variants of YOLO implemented in TensorFlow, PyTorch, Keras, and ONNX.

Setting up the development environment comes next after installing the SDK. Installing dependencies like OpenCV, Python libraries, and hardware drivers needed to connect to the Blaize accelerator is part of this. After then, a model-specific configuration file must be created by the user. The model path, framework type, input dimensions, quantization parameters, and output layers are among the specifications included in this configuration file, which is frequently expressed in YAML format. The model inference graph, which specifies how data will move across the network during runtime on the Blaize device, is created using this setup.

To transform the YAML model description into executable artifacts, the SDK offers tools like NetDeploy and ModelTool. These tools generate a number of output files, such as application templates, parameter binaries, and C++ source files that depict the inference graph. After that, these files are added to the relevant directories in the source tree of the GStreamer plugin. The SDK contains a structure called gst-gsp that contains the C or C++ source code for the plugin, including the core file that specifies its functionality. Developers can create and support more versions by cloning the subdirectories that each supported model variant, such YOLOv5s, lives in.

Compiling the plugin is the next step after the required files have been uploaded and the plugin code has been modified as appropriate. A build system such as Buildroot, which creates a full, customized Linux operating system for embedded hardware, is usually used for this. The plugin and its dependencies are compiled as part of the system build and registered as packages under Buildroot. In order to facilitate simpler debugging and iterative development prior to deployment to the embedded device, the SDK additionally offers Makefiles and build scripts that enable users to compile and test the plugins on host PCs running Ubuntu.

Following a successful build, the produced plugin is either delivered to the target Blaize hardware along with a full system image or integrated into the SDK's runtime. This procedure guarantees that all model files, dependencies, and custom plugins are properly configured and co-located. By utilizing the full potential of the Blaize V1P accelerator, the final product is a self-contained, optimized AI inference system that can process video streams in real time via a GStreamer pipeline.

2.4.2 GStreamer Plugin Development

Creating a new GStreamer element that can interface with the Blaize runtime and run deep learning models is the first step in developing a GStreamer plugin specifically designed for AI inference on the Blaize platform. This plugin must implement particular initialization, data handling, and resource cleanup procedures in accordance with GStreamer's architectural principles. The plugin's main function is to serve as a bridge between the inference engine and the media pipeline, taking in video frames, processing them, running the model, and producing the output in a manner that can be used.

The plugin needs to effectively manage the input and output tensor buffers in order to support real-time performance. Preprocessing the incoming video frames is the initial stage in this pipeline. This usually entails converting the pixel data to a certain format, like bfloat16, and scaling them to meet the input dimensions required by the model. For this, the Blaize SDK offers specially designed OpenVX kernels that are extremely hardware-optimized. These kernels can be used as independent processing steps in the GStreamer pipeline or included straight into the plugin.

After preprocessing is finished, the inference graph created by the NetDeploy or ModelTool tool is loaded and run by the plugin. The neural network is represented by this graph in a format that the Blaize inference engine can execute. The plugin has to import the model parameters, call the graph execution functions, and set aside memory for the input and output tensors. High-efficiency execution is the goal, and batching and pipelining are supported when needed. In order to fully utilize the platform's acceleration capabilities, the Blaize runtime makes sure the graph is executed in hardware.

To extract useful information from the raw output tensors after inference, the plugin must post-process them. This comprises determining bounding boxes, confidence scores, and class labels for items that are recognized in object detection tasks. Post-processing is supported by the Blaize SDK via Python scripts that can be invoked from within the plugin. These scripts carry out tasks including visualization, coordinate scaling, and non-maximum suppression. After that, the output can either be transmitted downstream in the pipeline for additional processing or storage, or it can be rendered straight onto the video frames.

The plugin has an expandable and modular design. By making subdirectories for every model version, changing the plugin code to load the appropriate model files, and altering the input/output tensor configurations, developers can add support for additional model variants. Because just a few parameters and configuration files need to be altered and the majority of the plugin code is consistent across models, this method encourages reuse and standardization.

The plugin becomes an essential component of the GStreamer pipeline after it is finished. It is compatible with common GStreamer components like renderers, converters, and video decoders. With just a small amount of code outside the plugin itself, developers may now create fully functional multimedia applications that carry out AI inference in real time. Additionally, based on the requirements of the application, the plugin can be set up to support various operating modes, including streaming and single-frame processing.

Thus, modularity, efficiency, and hardware acceleration are all embodied in the Blaize GStreamer plugin development process. It enables developers to reliably and with low latency implement sophisticated AI models on embedded devices. These plugins open up new possibilities in a variety of sectors and use cases by directly integrating into the GStreamer framework, enabling the development of scalable, adaptable, and performance

real-time computer vision systems.

2.5 Assigned Tasks

The creation, refinement, and integration of a real-time assistive system that can translate environmental items and text into speech-based feedback for those with visual impairments. Implementing optical character recognition (OCR) using PaddleOCR, a reliable open-source program designed for embedded devices, was a crucial component of the project. Text detection and text recognition were the two phases of the OCR workflow. Because of its exceptional ability to identify both regular and irregular text sections, DBNet (Differentiable Binarization Network) was chosen for text identification. Preprocessing the input image while preserving the aspect ratio, utilizing a CNN backbone (ResNet-50) to extract hierarchical features, creating segmentation maps, and employing differentiable binarization to highlight text regions were all steps in the detection process. Bounding boxes were then generated from the processed binary maps to localize text areas accurately, even under challenging conditions such as curved fonts or variable orientations.

A Convolutional Recurrent Neural Network (CRNN) architecture was used for the recognition phase. After being cropped and scaled, the identified regions were run through a CNN in order to extract spatial information. After a Bidirectional Long Short-Term Memory (BiLSTM) network processed these features for contextual comprehension, a Connectionist Temporal Classification (CTC) decoder produced the final character sequence without requiring character-level alignment. This method guaranteed precise and effective text recognition in practical situations. Together, the OCR technology made it possible to reliably translate text from both handwritten and printed sources, which was essential to the system's overall aural feedback.

The system also needed an object detection pipeline that could locate and identify several objects in real time, in addition to OCR. The YOLOv8 model was chosen for this because it strikes a balance between computational economy and detection accuracy, making it appropriate for edge AI deployment. In order to facilitate interaction with the Blaize NetDeploy framework, the model was converted to the ONNX format. This allowed for additional optimization through quantization and preprocessing directives specified in YAML configuration files. A binary file appropriate for running on the Blaize X600M AI accelerator was created from the completed models. Low-latency inference appropriate for assistive scenarios needing real-time object identification was guaranteed by this preparation.

YOLOv8's architecture included a head for bounding box prediction, class confidence, and object localization with a backbone based on CSPDarknet for feature extraction. A feature pyramid network with a self-attention method improved detection performance for a range of object sizes and locations. A camera module attached to the Raspberry Pi CM5 was used to record real-time video frames. Blaize-optimized OpenVX kernels were used for preprocessing, scaling, padding, and picture format conversion on the frames. Region-of-interest cropping was used in several use cases to highlight particular areas of the frame. To achieve rapid inference, the preprocessed frames were run through a GStreamer pipeline that was integrated with Blaize plugins. In post-processing, redundant bounding boxes were eliminated by using non-maximum suppression and filtering predictions below a confidence level. Final results were mapped with class labels, resized to the original

frame size, and passed to the TTS module for audio feedback, offering real-time object awareness to the user.

The VITTS model was a transformer-based TTS engine that handled text-to-speech conversion. The model's modular design included a Transformer decoder to produce mel-spectrograms, a HiFi-GAN vocoder to synthesize the waveform, and a text front-end to transform input into phonemes. Both OCR outputs and identified item labels had to be transformed into speech that sounded natural via the TTS process. Audio synthesis with low latency was performed after text normalization, phoneme conversion, and mel-spectrogram production were completed in order. Via headphones or speakers that were attached, voice output was provided, providing clear and responsive audio responses.

In order to provide offline deployment, more work was done. Criteria like latency, model size, and voice quality were used to assess a number of neural TTS models, including Glow-TTS and Coqui TTS. Because of its capability for bespoke voice models, lightweight performance, and multilingual capabilities, Coqui TTS was selected. Because of its independence from the internet, it was especially well-suited for embedded assistive systems. Sample rates and synthesis parameters were changed to improve the clarity of the voice output. Fast and precise speech production for both text and object detection outputs was ensured by the smooth audio playback via the Raspberry Pi's included or attached audio hardware.

A GPIO-based control system was used to enable user interaction in the absence of a graphical user interface. The Raspberry Pi CM5 was interfaced with three physical buttons so that users could control the system or swap between modes. Every button was associated with a distinct function, such as triggering object detection, activating OCR, or powering the machine. Falling-edge signals from the GPIO pins were recorded by a control script, which then started the relevant processing modules. The script was created for non-blocking execution, and pull-up resistors were utilized to provide dependable input detection.

Techniques such as frame skipping and multithreading were used to improve system responsiveness. By processing only one out of every ten collected frames, the frame skipping technique conserved computational resources while preserving enough information for detection and recognition. Concurrent camera capture, model inference, and audio playback were guaranteed by the multithreaded design. Inter-thread communication, particularly between the OCR and TTS modules, was accomplished through queues. To improve speech relevance and interaction quality, a similarity filter was used to prevent repeating previously uttered topics.

A system service was developed for deployment in order to launch the main application automatically. The inference pipeline was launched and the GPIO configuration was initialized via a custom Bash script. The user was guaranteed a plug-and-play experience thanks to this configuration, which enabled the machine to boot into a ready state, play a welcome message, and wait for button presses. For non-technical users, the system was very dependable and accessible, requiring no manual interaction after booting.

Combining these objectives helped to develop a strong, embedded, real-time assistive system that can recognize text and objects in the environment and translate them into meaningful voice output. The integrated solution was appropriate for use in accessibility-focused embedded AI systems since it tackled real-world usability, performance optimization, and deployment issues.

Chapter 3

Conclusion

The key points summarizing the internship training on Implementing Image Processing Modules and Algorithms Utilizing Blaize Accelerator on Raspberry Pi CM5 Device:

1. **Understanding Blaize Architecture and SDK Tools:** The Blaize Graph Streaming Processor (GSP) architecture and the Picasso SDK were thoroughly examined before the training started. This gave fundamental insights into the architectural ideas underlying the modular creation of vision pipelines and stream-based AI execution.
2. **Installation and Configuration of Blaize SDK:** A Raspberry Pi CM5 host platform was used to install and setup the Picasso SDK. In order to facilitate communication between the host and the Blaize Xplorer M.2 accelerator, this procedure involved setting up the environment, installing drivers, and verifying PCIe connectivity.
3. **Model Conversion and Deployment using NetDeploy:** Using NetDeploy, deep learning models in ONNX format were transformed into GSP-compatible forms. This required using organized YAML configuration files to define quantization settings, input/output shapes, and preprocessing parameters.
4. **Designing the AI Inference Pipeline:** Blaize's optimized plugins were used to build GStreamer pipelines for processing picture streams. For efficient operation and scalability, components including data input, model inference, and result rendering were set up in a modular approach.
5. **Creating Custom Preprocessing Modules:** To make sure input photos complied with model specifications, special preprocessing procedures were put in place. To preserve spatial integrity and avoid distortion during inference, these techniques included padding, aspect-ratio preservation, and scaling.
6. **Postprocessing for Detection and OCR Results:** By removing bounding boxes, converting coordinates back to the original picture space, and linking the results to class labels and confidence scores, postprocessing logic was utilized to comprehend the raw inference outputs.
7. **Deploying Pipelines on Blaize Accelerator:** Workloads for inference were installed on the Blaize Xplorer M.2 accelerator. This made it possible to offload AI computation effectively, leading to low-latency processing that is appropriate for edge applications.
8. **Real-time Object Detection and Scene Understanding:** Real-time item of interest identification was achieved by integrating object detection models like YOLOv8. For processing later on, detected entities were either visually marked or transformed into various types of feedback.

9. **Text Detection and Recognition with OCR Models:** Compact, high-performance models were used to implement optical character recognition (OCR). The pipeline made it possible to identify and transcribe printed text in pictures and video feeds in real time.
10. **GPIO Button Interface for Mode Switching:** Toggling between application modes, like object detection and OCR, was made easier with the development of a GPIO-based control interface. This made it possible for embedded systems with constrained user interfaces to function contextually.
11. **Audio Feedback System using TTS:** In order to produce audio responses from visual data, a text-to-speech (TTS) engine was incorporated. This improved accessibility in situations involving assistive technology by enabling non-visual output modalities.
12. **Pipeline Optimization for Real-Time Performance:** Performance profiling was used to assess overall latency, memory utilization, and inference speed. To increase responsiveness and guarantee reliable real-time operation, optimizations were implemented.
13. **Integration of Custom GStreamer Plugins:** Additional GStreamer components were set up and expanded as needed to meet the needs of certain applications. The flexibility and functional depth of the pipeline were improved as a result.
14. **Multi-Task Switching and Application Modes:** The system's runtime configuration and control techniques allowed for smooth transitions between various visual tasks. This capacity to multitask enhanced the application's adaptability.
15. **End-to-End Testing and Real-World Evaluation:** The integrated system was tested in real time under a range of environmental circumstances. The system's readiness for real-world implementation was demonstrated by the evaluation, which concentrated on accuracy, latency, and usability.

In conclusion, this internship involved using Blaize hardware to construct an embedded AI application from start to finish. A modular, real-time AI vision solution appropriate for edge deployment in assistive situations was the outcome of the training, which encompassed system integration, model optimization, and pipeline execution.

Bibliography

1. Bala, Rounak Kumar, et al. "Pothole Detection using YOLO v8 Algorithm." 2023 9th International Conference on Smart Computing and Communications (ICSCC). IEEE, 2023.

2. Wang, Huibai, and Wenzhen Zang. "Research on object detection method in driving scenario based on improved YOLOv4." 2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC). Vol. 6. IEEE, 2022.

3. Karthi, M., et al. "Evolution of yolo-v5 algorithm for object detection: automated detection of library books and performace validation of dataset." 2021 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES). IEEE, 2021.

4. Liu, Rui, et al. "An improved faster-RCNN algorithm for object detection in remote sensing images." 2020 39th Chinese Control Conference (CCC). IEEE, 2020.

5. Qu, Xiao, et al. "Enhancing Multi-Scale Object Detection with YOLOv8 and Residual Feature Pyramid Networks for Real-Time Urban Scene Analysis." 2024 10th International Conference on Computer and Communications (ICCC). IEEE, 2024.

6. Y. Du, X. Pan, H. Xu, et al., "PP-OCR: A Practical Ultra Lightweight OCR System," arXiv preprint arXiv:2009.09941, 2020.

7.X. Zhou, C. Yao, H. Wen, et al., "EAST: An Efficient and Accurate Scene Text Detector," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 5551–5560.

8. F. Morales, "Keras-OCR," GitHub Repository, 2020..

9. J. Kim, S. Kim, J. Kong, and J. Yoon, "Conditional Variational Autoencoder with Adversarial Learning for End-to-End Text-to-Speech," in Proceedings of the 38th International Conference on Machine Learning (ICML), 2021. .

10. gTTS Developers, "gTTS: Google Text-to-Speech Python Library," PyPI, 2023..

11. MMOCR Contributors, "MMOCR: OpenMMLab Text Detection, Recognition and Understanding Toolbox," OpenMMLab, 2021.

rep1

ORIGINALITY REPORT

10%	8%	6%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

www.researchgate.net Internet Source	2%
www.huggingface.co Internet Source	1%
www.colab.ws Internet Source	1%
www.raspberrypi.com/documentation Internet Source	1%
github.com Internet Source	1%
bcmi.sjtu.edu.cn Internet Source	1%
vlib.itrc.as.ir Internet Source	1%