الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria

وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research

جامعة سعد دحلب البليدة
SAAD DAHLAB University - BLIDA

كلية التكنولوجيا
Faculty of Technology

قسم الإلكترونيك
Department of Electronics

Final Thesis

Domain: Science and Technology
Field: Electronics
Specialization: Embedded Systems

Presented by

Laid Abderrahmane & Ouffa Wissal

Thesis Title

Design and Implementation of a custom PCB-Based Edge AI System Using STM32 for Real-Time State Detection and Condition Monitoring

---

Supervisor: Mme D.Naceur

Academic year 2024-2025

Acknowledgment

First and foremost, we are profoundly grateful to Allah for His countless blessings and guidance throughout our journey. Without His wisdom and grace, this work would not have been possible. We would like to express our heartfelt gratitude to our supervisor Madame Naceur, whose continuous support, dedication, and enthusiasm have been a true source of motivation throughout this journey. Over the past two years, she has not only guided us academically but also created a positive and encouraging environment that made learning genuinely enjoyable. Her trust, patience, and insightful feedback have pushed us to grow, both personally and professionally.

We are also deeply thankful to our families for their constant support, understanding, and love, which have been fundamental to our progress and perseverance.

Finally, we would like to thank the jury members for their time, valuable feedback, and constructive remarks which have contributed to the enrichment of this work.

Dedication :

Abderrahmane Laid

All praise and thanks to Allah, whose guidance and mercy made this possible.

To my parents, thank you for your endless support, love, and prayers. I wouldn't be here without you.

To my beloved fiancée, your unwavering belief in me, your patience, and your constant encouragement have been a source of strength throughout this journey. Your love and support mean the world to me, and I am endlessly grateful to have you by my side.

To my late grandfather, your memory lives in every step I take. Your wisdom, kindness, and the bond we shared continue to guide and inspire me. This achievement is dedicated to you,I hope I've made you proud.

To Madame Naceur, thank you for your dedicated teaching, your generous guidance, and the invaluable wisdom you've shared. Your encouragement and commitment have left a lasting impact on my path, and I am truly thankful for all you've done.

Finally, I would like to sincerely thank the jury members for their time, valuable feedback, and constructive remarks which have contributed to the enrichment of this work.

Ouffa Wissal

All praise and thanks to Allah for granting me the strength, patience, and guidance throughout this work.

First and foremost, I would like to express my deepest gratitude to my dear mother, who has been my rock throughout this journey. I am forever grateful for the way she has always surrounded me with love and encouragement, offering unwavering support that made every step of this path lighter.

To my father, thank you for your strength, wisdom, and constant encouragement.

A heartfelt thank you as well to my beloved sister Majida and brother Athman your love, humor, and support have brought light to difficult moments, and I'm truly grateful to have you both by my side.

I am also deeply thankful to my precious fiancé for his continuous support, patience, and motivation during this entire journey. your understanding and belief in me have made even the most challenging moments easier to overcome, and your presence has been a constant source of comfort and inspiration.

I would like to sincerely thank my supervisor, Madame Naceur, for her valuable guidance, expertise, and continuous support. Her insightful advice and encouragement have greatly contributed to my academic and personal growth, and have made this journey both enriching and enjoyable.

Finally, I extend my sincere thanks to the jury members for taking the time to evaluate my work, and for their valuable feedback and contributions.

.

ملخص:

تقدّم هذه الرسالة نظام حوسبة طرفية لمراقبة الآلات الصناعية في الزمن الحقيقي، حيث يتم دمج المتحكم الدقيق إس تي إم 32 إف 103 آر بي مع نانو إدج إيه آي ستوديو لمراقبة حالة الآلة على الجهاز نفسه دون الاعتماد على السحابة. يتم التقاط بيانات الاهتزاز من المستشعر إم بي يو 6050 عبر دي إم إيه، وتحليلها باستخدام الذكاء الاصطناعي المدمج، ثم عرضها محلياً على شاشة أو إل إي دي، بالإضافة إلى إرسالها إلى لوحة تحكم ويب عبر ناقل كان ووحدة إي إس بي 32. تضمن لوحة إلكترونية مخصصة ذات 6 طبقات (بي سي بي) وهيكل برمجي معياري معالجة منخفضة التأخير وقابلية للتوسّع. توضح هذه الحلول كيف يمكن للذكاء الاصطناعي المدمج والخفيف أن يمكّن من الصيانة التنبؤية بتكلفة منخفضة.

الكلمات المفتاحية:

الحوسبة الطرفية، إس تي إم 32، نانو إدج، دي إم إيه، بي سي بي

Résumé :

Ce mémoire présente un système d'edge computing pour la surveillance industrielle en temps réel, combinant le microcontrôleur STM32F103RB et l'outil NanoEdge AI Studio pour analyser l'état des machines en local, sans dépendance du cloud. Les données vibratoires d'un capteur MPU6050 sont acquises via DMA, analysées par une IA embarquée, puis affichées sur un écran OLED tout en étant transmises à un tableau de bord web (via le bus CAN et un ESP32). Une carte PCB personnalisée à 6 couches et une architecture firmware modulaire garantissent un traitement à faible latence ainsi qu'une évolutivité optimale. Cette solution illustre comment l'IA embarquée permet de mettre en œuvre une maintenance prédictive économique.

Mots-clés: Edge computing, STM32, NanoEdge, DMA, PCB

Abstract:

This thesis presents an edge-computing system for real-time industrial machine monitoring, combining an STM32F103RB microcontroller with NanoEdge AI Studio to Monitor Machine state on-device without cloud dependency. Vibration data from an MPU6050 sensor is captured via DMA, analyzed by embedded AI, and displayed locally on an OLED while being relayed to a web dashboard (via CAN bus and ESP32). A custom **6**-layer PCB and modular firmware architecture ensure low-latency processing and scalability. The solution demonstrates how lightweight embedded AI can enable cost-effective predictive maintenance.

Keywords: Edge-computing, STM32, NanoEdge, DMA, PCB

List of acronyms and abbreviations

AI	**Artificial Intelligence**
CAN	**Control Area Network**
DMA	**Direct Memory Access**
EAI	**Embedded Artificial intelligence**
FFT	**Fast Fourier Transform**
FPS	**Frame Per Second**
GPIO	**General-Purpose Input/Output**
HAL	**Hardware Abstraction Layer**
HSE	**High-Speed External**
I2C	**Inter Integrated Circuit**
ICM	**Isolation Covariance Matrix**
IPCA	**Incremental Principal Component Analysis**
MCU	**Microcontroller Unit**
MML	**Miniaturized Machine Learning**
MPU6050	**Motion Processing Unit 6050**
OLED	**Organic Light Emitting**
PCB	**Printed Circuit Board**
PLL	**Phase-Locked Loop**
RGB LED	**Red-Green-Blue Light Emitting Diode**
USB	**Universal Serial Bus**
ZSM	**Zero-Shot Model**

Table of Contents

General introduction ……..……1

Chapter 1 ……..3

1.1. Introduction ……4

1.2. Edge Computing and Embedded AI in Industrial Systems……4

1.1.1. Edge Computing definition……4

1.1.2. Cloud and Edge Computing Paradigms ……5

1.1.3. Embedded Artificial Intelligence ……7

1.1.4. Tiny ML……8

1.3. Condition Monitoring……11

1.3.1. Predictive Maintenance and Condition Monitoring……11

1.3.2. Benefits of condition monitoring……12

1.3.3. Types of Condition Monitoring……12

1.4. NanoEdge AI Overview……13

1.4.1. Definition ……13

1.4.2. NanoEdge AI Library ……14

1.4.3. NanoEdge AI Studio capabilities……15

1.4.4. NanoEdge AI Studio limitations……15

1.5. NanoEdge AI data format and Use Cases……16

1.5.1. Time series versus cross sectional……16

1.5.2. Defining important concepts……16

1.5.3. Designing a relevant sampling methodology……17

1.6. STM32 Microcontroller Technology……18

1.6.1. STM32 Overview ……18

1.6.2. STM32F103RB……20

1.7. Motion Sensors and Vibration Monitoring ……23

1.7.1. Motion sensors ……23

1.7.2. Vibration monitoring ……24

1.7.3. Focus on Accelerometers for Vibration Monitoring……25

1.8. Human-Machine Interfaces in Embedded Systems……25
1.8.1. The functionality of Human-Machine Interfaces (HMIs)……26
1.8.2. The Link Between HMI and Internet of Things (IoT)……26
1.9. Conclusion ……27

Chapter 2 ……29

2.1. Introduction ……30
2.2. System overview……30
2.3. Hardware Design and Prototyping Process……32
2.3.1. Initial Microcontroller Testing……33
2.3.2. Breadboard Functional Prototype ……33
2.3.3. Transition to a custom PCB ……35
2.4. Sensor Integration : MPU6050……35
2.4.1. Physical Connection via I^2C……35
2.4.2. I^2C Communication and DMA Optimization……37
2.4.3. Register Configuration and Initialization……38
2.4.4. Sampling Strategy with Interrupts……39
2.4.5. Datasheet-Guided Design & Lessons Learned from Sensor Integration……39
2.5. Microcontroller Logic: STM32F103RB……41
2.5.1. Clock Configuration and Oscillator Selection……41
2.5.2. Microcontroller Pin Configuration and Peripheral Mapping……43
2.5.3. Peripheral Coordination and Simplifications……46
2.6. NanoEdge AI Integration ……47
2.6.1. Sensor data collection and preparation……47
2.6.2. Time-Series Buffer Design for Edge AI……47
2.6.3. Other Algorithm Types Explored……52
2.6.4. Model Deployment on STM32……53
2.7. Display Layer : SSD1306 OLED ……54
2.7.1. Display Connection and Communication……54

2.7.2. Display Modes: Eco vs Fast Refresh ……55
2.7.3. Visual Layout Design……55
2.8. Auxiliary Components in the System Schematic ……57
2.8.1. USB Interface and 3.3V LDO Regulator……57
2.8.2. A-RGB LED ……57
2.8.3. Tactile Switch (Boot/Reset)……58
2.8.4. I/O Connectors and Expansion Headers……59
2.8.5. Can BUS……60
2.9. Full System Schematic ……61
2.10. Output Interface Extension: ESP32 + CAN Bus Architecture……62
2.10.1. Why a Remote Output Module Was Needed……63
2.10.2. Schematic Breakdown of the ESP32-CAN Module……63
2.10.3. Full Schematic of the ESP32-CAN Monitoring Board……66
2.10.4. System Role and Data Flow……67
2.11. Conclusion ……68
Chapter 3……69
3.1. Introduction ……70
3.2. Development Environment ……70
3.2.1. STM32CubeIDE – Firmware Development & Debugging……71
3.2.2. KiCad: Hardware Design from Schematic to Final Board……72
3.2.3. NanoEdge AI Studio: Embedded AI Training & Deployment……73
3.2.4. PCB Manufacturing and Assembly……74
3.2.5. VS Code and PlatformIO Extension ……75
3.2.6. SNC ALMITECH: Manufacturing the second PCB……76
3.3. PCB Fabrication and Assembly……77
3.3.1. Design Workflow Overview……78
3.3.2. Layer-by-Layer PCB Breakdown……85
3.3.3. Final PCB Visualization & Assembly Overview……93
3.3.4. Capabilities of a 6-Layer Rigid PCB (via JLCPCB)……96
3.3.5. Final Bill of Materials (BOM)……97

3.3.6. Lessons Learned and Manufacturing Tips……..97
3.4. Wireless Result Transmission Module (ESP32 + CAN Interface) ……..97
3.4.1. System Architecture Flow……..98
3.4.2. PCB Overview and 3D Rendering……..98
3.5. Firmware Implementation and System Logic……..104
3.5.1. System Initialization Flow……..105
3.5.2. Peripheral Configuration……..106
3.5.3. Task Flow in Main Loop ……106
3.5.4. Sensor Interface with DMA: MPU6050 Data Handling……109
3.5.5. Embedded AI Integration: NanoEdge Inference Logic……110
3.5.6. Display Output: OLED Feedback……111
3.5.7. CAN Transmission & ESP32 Web interface……112
3.6. NanoEdge AI Integration: Training and Evaluation ……115
3.6.1. Signal import summary ……115
3.6.2. Benchmarking & Model Selection……118
3.6.3. Model Validation and Library Selection……120
3.7. Results and System Evaluation……125
3.7.1. State Visualization and OLED Feedback……125
3.7.2. Inference Behavior and Response Time……129
3.7.3. Score Consistency and Model Accuracy……129
3.7.4. Result Transmission and Web Visualization……129
3.8. Conclusion ……132
General Conclusion ……133
Bibliographies ……135
Appendix ……137

List of figures

Figure 1.1: Conventional cloud computing structure……5

Figure 1.2: Two-way computing streams in edge computing……6

Figure 1.3: NanoEdge AI Studio ……13

Figure 1.4: NanoEdge AI data format……16

Figure1.5: STM32F103RB MCU ……20

Figure 1.6: Overview of STM32f103rb Internal Bus System and DMA Integration ……22

Figure 1.7: MPU6050 Module……25

Figure 2.1: System Data Flow Overview: From sensing to AI-powered feedback……32

Figure 2.2: LQFP64 support board for STM32F103RB……33

Figure 2.3: Breadboard prototype of full system……34

Figure 2.4: Extracted Schematic: MPU6050 to STM32 Wiring……36

Figure 2.5: MPU6050 Typical Operating Circuit……37

Figure 2.6: Efficient Data Path Using DMA and Interrupts……38

Figure 2.7: Raw Signal Path from MPU6050 to AI Model……39

Figure 2.8: MPU6050 Internal Architecture ……40

Figure 2.9: Clock configuration with external HSE source and PLL for 72 MHz system clock……41

Figure 2.10: High-speed external clock source AC timing diagram ……42

Figure 2.11: STM32F103xx performance line LQFP64 pinout ……43

Figure 2.12: STM32F103RB Extracted Schematic: Peripheral and Communication Mapping……45

Figure 2.13: Pin loading conditions ……46

Figure 2.14: sample vs buffer……48

Figure 2.15: Difference between independent sample vs temporal buffer……49

Figure 2.16: Basic format for (AD, NCC, 1CC)……51

Figure 2.17: Basic format for Extrapolation……52

Figure 2.18: Extracted view of the OLED to STM32 I^2C wiring from the full system Schematic…..54

Figure 2.19: Display Logic Flow……56

figure 2.20: USB and Regulator Extracted Schematic……57

figure 2.21: A-RGB LED Extracted Schematic……58

figure 2.22: Tactile Switch Extracted Schematic……59

figure 2.23: Connector Extracted Schematic……60

figure 2.24: CAN BUS Extracted Schematic……60
Figure 2.25: Full Schematic of the STM32 Main Board……62
figure 2.26: CAN bus transceivers Communication module ……64
figure 2.27: The 3 CAN bus transceivers Extracted Schematic……64
figure 2.28: ESP32-C3……65
Figure 2.29: Dual 8-pin connectors Extracted Schematic……66
Figure 2.30: ESP32-CAN Monitoring Board Schematic……66
Figure 2.31: ESP32-CAN Monitoring Board Data Flow……67
Figure 3.1: STM32Cube IDE Workspace……72
Figure 3.2: KiCad PCB Design Workspace……73
Figure 3.3: NanoEdge AI Studio interface showing the project……74
Figure 3. 4: JLCPCB Homepage ……75
Figure 3.5: Platform IO extension in VS code……76
Figure 3.6: ALMITECH Homepage……77
Figure 3.7: an example of the MCU footprint ……79
Figure 3.8: PCB Layers……80
Figure 3.9: checking Errors……81
Figure 3.10: Exporting PCB Gerber File……81
Figure 3.11: Exporting Pick Place File……82
Figure 3.12: PCB Manufacturing Order on JLCPCB……83
Figure 3.13: PCB Production and Process in JLCPCB Factory ……84
Figure 3.14: PCB Design Workflow from Schematic to Fabrication……85
Figure 3.15: Top Layer Primary Component and Signal Routing……86
Figure 3.16: Inner Layer 1 Solid Ground Plane (GND)……87
Figure 3.17: Inner Layer 2 Mixed Signal and Ground Layer……88
Figure 3.18: Inner Layer 3 Secondary Signal and 5V Power Routing……89
Figure 3.19: Inner Layer 4: Dedicated 3.3V Power Plane……90
Figure 3.20: Bottom Layer: Secondary Ground Plane and Auxiliary Routing……91
Figure 3.21: Full 6-Layer Stackup of the Custom PCB……92
Figure 3.22: 2D Visualization of Bare PCB Structure (No Components) from top and bottom……93
Figure 3.23: Assembled 3D view for Diagno Disk (Top View)……94
Figure 3.24: Assembled 3D view for Diagno Disk (Bottom View)……95

Figure 3.25: Assembled 3D view for Diagno Disk (different angles)……95
Figure 3.26: Final Fabricated Board (Real Image)……..96
Figure 3.27: data flow of the system……98
Figure 3.28: Top Copper Layer ……99
Figure 3.29: Bottom Copper Layer……100
Figure 3.30: Top Silkscreen ……101
Figure 3.31 All Layers Routing of the Diagno Disk Hub PCB……102
Figure 3.32: 3D Rendered View of the Diagno Disk Hub PCB from the side ……103
Figure 3.33: 3D Rendered View of the Diagno Disk Hub PCB (ESP32 + CAN Interface……104
Figure 3.34: Firmware Initialization Flow……105
Figure 3.35: STM32F103RB pin configuration in Cube IDE……106
Figure 3.36: Main loop Flowchart……107
Figure 3.37: MPU6050 Initialization and Data Acquisition Workflow……109
Figure 3.38: Interruption workflow……110
Figure 3.39: Button Control Workflow……111
Figure 3.40: Display Output Workflow……112
Figure 3.41: The similarity request ……113
Figure 3.42: CAN Transmission Workflow……113
Figure 3.43: ESP32 Continuous Data Acquisition and WebSocket Process……114
Figure 3.44: Initial regular Signal Import Interface in NanoEdge AI Studio……115
Figure 3.45: Initial abnormal Signal Import Interface in NanoEdge AI Studio……116
Figure 3.46: Regular signal overview in NanoEdge AI Studio……117
Figure 3.47: Abnormal signal overview in NanoEdge AI Studio……118
Figure 3.48: Benchmark Progression overview in NanoEdge AI Studio……119
Figure 3.49: Search Space overview in NanoEdge AI Studio……120
Figure 3.50: Validation overview in NanoEdge AI Studio……121
Figure 3.51: Failed Real-Time Classification by ZSM Model……122
Figure 3.52: ICM Model Validation Showing Accurate State Separation……123
Figure 3.53: Final Library Report (ICM) Generated by NanoEdge AI Studio……124
Figure 3.54: OLED Display Frames Under Various Conditions: idle……126
Figure 3.55: OLED Display Frames Under Various Conditions: Transient anomaly……127

Figure 3.56: OLED Display Frames Under Various Conditions: Severe anomaly……...128

Figure 3.57: Web Dashboard Displaying Live Anomaly Score via ESP32 Module……130

List of tables

Table 1.1: Comparison of STM32 Microcontroller Families ……19

Table 1.2: peripheral of STM32F103 ……21

Table 1.3: Comparison table of motion sensors……23

Table 2.1: I²C Connections……36

Table 2.2: MPU6050 Register Initialization Summary ……38

Table 2.3: I²C Timing Specification Table ……14

Table 2.4: I²C Connections for OLED……54

Table 2.5: pin configuration of the LED……58

Table 2.6: pin configuration for the Tactile Switch ……59

Table 3.1: Summary Comparison of ICM, IPCA, ZSM, MML……119

Table 1: Core PCB Manufacturing Specifications ……138

Table 2: Drill Capabilities and Hole Parameters ……139

Table 3: Trace and Spacing Constraints ……140

Table 4: Solder Mask Capabilities ……141

Table 5: Summary Bill of Materials for the 6-Layer Diagno disc PCB ……142

General introduction

Machines are no longer just mechanical systems driven by simple commands they're becoming intelligent, adaptive, and connected. In today's industrial and research environments, the ability to sense, interpret, and act upon real-world data in real time is at the heart of *condition monitoring*, predictive maintenance, automation, and human-machine collaboration. As the pace of digitization accelerates, embedded systems are being tasked with more complex responsibilities, including running AI algorithms directly at the edge.

This thesis presents the complete development of a real-time, AI-powered embedded system designed for condition monitoring through detecting subtle changes in machine motion and reporting them through both local displays and remote dashboards. The goal was to create a small, self-contained unit that could sense motion, recognize machine behavior states, and communicate results, all using low-power hardware and no external processing.

We chose the STM32F103RB microcontroller as the core processor, known for its balance of performance and affordability, and built a 6-layer custom PCB that integrates key peripherals such as a motion sensor (MPU6050), OLED display, push button, CAN transceiver, and RGB indicator. The system was trained using NanoEdge AI Studio, a tool that enabled the generation of lightweight AI models capable of detecting machine behavior states in real time. To expand visibility beyond the local screen, we developed a secondary board based on ESP32, which received state results over CAN bus and hosted a live dashboard accessible via Wi-Fi.

This journey was not only technical, but also conceptual: how do you bring meaningful intelligence to microcontrollers? And how do you make that intelligence visible and practical in real-world use?

To make this project both technically rich and clear to understand, the thesis is organized into three chapters:

- **Chapter 1** lays the groundwork. It explores the evolution of embedded AI and edge computing, the challenges of machine state monitoring, and why tools like NanoEdge AI are essential for enabling intelligence on small devices.
- **Chapter 2** describes the design process, from early breadboard prototyping to hardware architecture and software conception. Here, each component is introduced with its role in sensing, processing, displaying, and transmitting machine behavior.
- **Chapter 3** brings everything to life. It covers the firmware implementation, PCB fabrication,

model evaluation using NanoEdge AI Studio, and final results. This chapter includes testing scenarios, visualization strategies, and the full deployment setup both on-screen and wirelessly.

By integrating embedded intelligence, compact hardware, and dual visualization layers, this thesis demonstrates how real-time state detection can be achieved on microcontrollers and how these systems can be made robust, scalable, and ready for real-world deployment.

Chapter 1

General Concepts

1.1. Introduction

Machines are becoming smarter, faster, and more independent. As technology moves forward, the ability to monitor what machines are doing and how they're behaving what's commonly called *condition monitoring*—has become more important than ever. Thanks to edge computing and embedded AI, it's now possible to bring intelligence directly into small devices, allowing real-time decisions without always relying on the cloud.

This first chapter sets the stage for everything that follows. It looks at the big ideas behind embedded monitoring systems and condition monitoring, starting with what edge computing and embedded AI really mean, and how movements like TinyML are shaping this new generation of technology.

Because of how central it is to the project, special attention is given to NanoEdge AI Studio, the tool we used to create machine learning models that work directly on our microcontroller.

Finally, we introduce the STM32F103RB microcontroller, the heart of the system, and explain why it was chosen to power the real-time monitoring, decision-making, and condition monitoring capabilities we needed.

By building a solid understanding of these technologies and concepts, we can better appreciate the design and choices that come together in the system described in the next chapters.

1.2. Edge Computing and Embedded AI in Industrial Systems

1.2.1. Edge Computing definition

Edge computing refers to processing, analyzing, and storing data closer to where it is generated to enable rapid, near real-time analysis and response. In recent years, some companies have consolidated operations by centralizing data storage and computing in the cloud. But the demands of new use cases enabled by billions of distributed devices from advanced warehouse and inventory management solutions to vision-enhanced robotic manufacturing lines to advanced smart cities traffic control systems have made this model unsustainable.

Additionally, the increased use of edge devices from Internet of Things (IoT) devices, such as smart cameras, mobile point-of-sale kiosks, medical sensors, and industrial PCs to gateways and computing infrastructure for faster, near real-time actionable insights at the data source is driving exponential growth in the amount of data generated and collected. Edge computing provides a

path to reap the benefits of data collected from devices through high-performance processing, low-latency connectivity, and secure platforms. [1]

1.2.2. Cloud and Edge Computing Paradigms

Massive amounts of data are generated every second by power transmission networks. In 2023, the world generated approximately 123 zettabytes of data, according to International Data Corporation. Advanced data analytic algorithms are used to transform such data into information and knowledge, which can be then used for network operations and/or parent energy services. Such data analytics rely upon information and communication technologies (ICTs): they have a critical role in data collection, transfer and processing. Computing is a critical function of ICTs: it determines how data analytics typical of transmission networks are performed, and thus becomes the foundation for transmission network operations and services.

Cloud Computing, geo-distributed devices and equipment are connected to cloud data centers, supporting centralized decisions and issuing control orders. The cloud is an abstraction which separates the actions of storing, retrieving and computing on data from the physical constraints of doing so.[2]

The schematics below **(figure 1.1 [2])** illustrate the conventional cloud computing structure.

Figure 1.1: Conventional cloud computing structure.

- Data producers generate raw data and transfer it to a cloud.
- Data consumers send requests for consuming data to this cloud.

However, the cloud is no longer just a storage shed. Its users demand that it contributes a lot more to computing tasks and more quickly than ever before. This is the case of Internet of Things IoT solutions, which will empower Transmission System Operators (TSOs). The quantity of raw data produced by TSOs will be huge: it will make conventional cloud computing not efficient enough to

handle the computing demand. This means that most of the data produced by IoT will be consumed at the "edge of the telecommunication networks".

The schematics below **(figure 1.2 [2])** illustrates the two-way computing streams in edge computing: the "things" are not only data consumers but also play as data producers. At the edge, the things can not only request service and content from the cloud but also perform the computing tasks from the cloud. Edge performs computing offloading, data storage, caching, and processing. It also distributes request and delivery service from cloud to users.

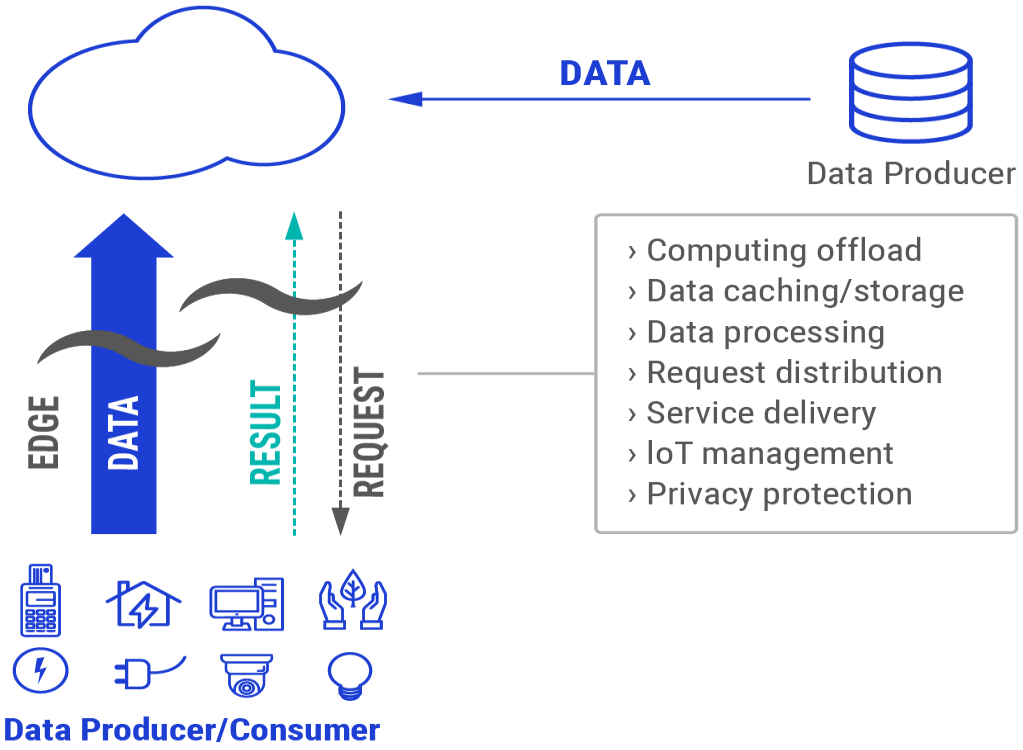


Figure 1.2: Two-way computing streams in edge computing.

Three types of architecture are described for practical **Cloud Computing (CC)** and **Edge Computing (EC)** implementations in power transmission networks:

- **"Thing" tier:** This layer is widely addressed in the context of the IoT. The thing tier covers most of the electrical equipment and the communication access in the so-called Smart Grids (SG). This tier oversees the operations of SG and realizing operation/control orders. Communication connections need to be established to transfer data to high layers.

- **"Edge" tier**: The edge tier contains the intermediate storing; communication "edge" is a relative concept. For instance, "smart metering" belongs:
    - to the thing tier when it performs sensing and transferring data
    - to the edge layer when it becomes a platform for home energy analytics (computation).
- **"Cloud" tier**: This tier consists of control, storing and computing centers. Compared with the edge tier, the clouds are designated with high- performance storage and computing elements. These powerful resources are deployed to perform complex analyses with a long-term time scale and a grid-wide geographical scope. [2]

A) Challenges in exploiting the full potential of CC and EC

- Limited bandwidth resources
- Heterogeneous working environments
- Naming - The naming scheme for EC needs to handle the mobility of things, a highly dynamic network topology, and privacy and security protection, as well as the scalability targeting the extremely large amount of unreliability
- Data Abstraction
- Service Management
- Privacy and Security concerns
- Optimization Metrics – In EC there are multiple layers with different computation capabilities. Workload allocation becomes an issue i.e. which layer to handle the workload or how many tasks to assign at each part
- Dynamic workload allocation.[2]

1.2.3. Embedded Artificial Intelligence

Embedded AI, or EAI, refers to integrating artificial intelligence into embedded systems, allowing devices to process data and make decisions autonomously. Unlike traditional AI models that rely on cloud-based infrastructure, embedded AI operates within edge devices, processing sensor data in real time. This leads to faster and more efficient operations without depending on constant connectivity.

At its core, embedded AI takes AI capabilities – such as neural networks and convolutional models – and embeds them into low-power, high-performance devices.

Whether in microcontrollers for industrial automation or AI-powered smartphones, embedded AI reduces the need for external servers, optimizes local data processing, and empowers devices to make independent, real-time decisions.[3]

Why Embedded AI Matters:

- **Real-Time Intelligence:** Devices can analyze and act on data instantly, such as detecting abnormal vibrations in machinery before a fault occurs.
- **Enhanced Data Privacy**: Sensitive information, like biometric or health data, can be processed locally, reducing security risks.
- **Reliability in Harsh Environments**: Edge devices remain functional even in remote or disconnected areas (e.g., offshore oil platforms or isolated manufacturing sites).

Industries including healthcare, industrial IoT, smart homes, autonomous vehicles, and energy infrastructure are increasingly adopting embedded AI to enable smarter, faster, and more secure decision-making at the edge.[3]

1.2.4. Tiny ML

Tiny Machine Learning (TinyML) is a breakthrough in artificial intelligence (AI) that enables machine learning (ML) models to run on ultra-low-power microcontrollers instead of traditional cloud-based data centers. This advancement significantly reduces energy consumption, enhances data security, and allows real-time AI processing. With applications spanning across smart cities, precision agriculture, industrial automation, healthcare, and wearable AI, TinyML is trans- forming the way AI is deployed in constrained environments. Understanding TinyML Unlike traditional ML, which relies on large computing infrastructure with ample processing power, memory, and cloud storage, TinyML integrates three key components to function efficiently on constrained devices Hardware, Software, Optimized ML Algorithms. [4]

A) Application of TinyML

TinyML allows real-time edge machine learning using inference on low-power microcontrollers. This has created a wide range of applications in healthcare, agriculture, home automation, industry, and environmental monitoring. In healthcare, TinyML-based wearable devices can track vital signs like heart rate or oxygen saturation and detect anomalies locally without requiring cloud connectivity, ensuring privacy and having immediate alerts. Likewise, in farming, sensors based on TinyML can monitor soil composition, identify pests, and forecast plant disease—even in locations with sparse internet access.

Smart home solutions are enhanced by TinyML through always-on, low-power, edge devices capable of voice recognition, environmental audio detection, or control of lights and appliances as a function of occupancy—all of which are user-privacy-enabled through on-device processing of the data. In industrial settings, TinyML is used for predictive maintenance, where it detects equipment anomalies using real-time sensor data, avoiding failures and reducing downtime. Environmental applications include tracking air or water quality using battery-powered sensors that process and act on data locally, which is ideal for mass deployment in urban or rural environments. From wildlife conservation to smart traffic systems, TinyML is destined to be a cornerstone for smart edge computing, driving smarter, more efficient, and greener applications in daily environments.[5]

B) Key components of TinyML System

- **Development Platforms and Tools**

TensorFlow Lite Micro is perhaps the most popular alternative to run ML models on microcontrollers. It offers a highly optimized runtime to execute TensorFlow models on devices with at least 16 KB of RAM.It includes integer-only models and is specially built to execute without an operating system.

Edge Impulse is an extremely popular platform for training, developing, and deploying TinyML models. It provides a simple-to-use interface for model development, data collection, and deployment on embedded hardware. It has an impressive number of supported development boards and sensors, which makes it a good choice for quick prototyping.

Arduino IDE and PlatformIO are used extensively in programming microcontrollers. Through libraries specific to TinyML like Arduino_TensorFlowLite, programmers are able to integrate machine learning models directly into Arduino-based applications. [5]

- **Frameworks for Optimization and Inference**

The ARM CMSIS-NN library contains very optimized kernels for neural networks aimed at boosting performance on the Cortex-M microcontrollers. The inference performance on embedded systems is smooth due to the extremely limited memory usage and high performance it provides.

uTensor is an inference engine specifically designed for embedded systems, which makes it incredibly light. Because it is developed to fit seamlessly into TensorFlow models, deployment of ML on Cortex-M boards, which are considerably resource-limited, can be done without losing too much performance or accuracy. [5]

- **Hardware Platforms**

ARM Cortex-M series microcontrollers such as the M3, M4, and M7 are some of the most commonly used chips in TinyML solutions. These processors balance low power and sufficient computing to be able to execute elementary ML inference.

Arduino Nano 33 BLE Sense is a miniature board featuring an ARM Cortex-M4 processor and a set of onboard sensors (microphone, accelerometer, temperature) that allows it to be a good platform for TinyML exploration and teaching. ESP32's dual core processor and embedded Wi-Fi/Bluetooth offer additional processing and memory functions compared to the usual microcontrollers, which allow developers to run somewhat more advanced models at the edge.

STM32 series microcontrollers by STMicroelectronics are widely utilized in industrial TinyML applications. The boards support various sizes of memory and peripherial options and are compatible with a lot of ML frameworks as well as toolchains [5]

C) Advantages of TinyML

There are a number of advantages with running tinyML machine learning models on embedded devices.

- **Low Latency:** Since the machine learning model runs on the edge, the data doesn't have to be sent to a server to run inference. This reduces the latency of the output.
- **Low Power Consumption:** Microcontrollers are ultra-low power which means that they consume very little power. This enables them to run without being charged for a really long time.
- **Low Bandwidth:** As the data doesn't have to be sent to the server constantly, less internet bandwidth is used.
- **Privacy:** Since the machine learning model is running on the edge, the data is not stored in the cloud. [6]

1.3. Condition Monitoring

Condition monitoring is a systematic process that involves the continuous assessment of the health and performance of machines, equipment, and systems in an industrial setting. The primary objective of condition monitoring is to detect any deviations from normal operation and identify potential issues before they result in downtime or failures. This is a proactive method of maintenance that allows organizations to make educated decisions regarding repairs and replacements, optimizing various maintenance efforts and reducing overall operational costs.
At the core of any effective condition monitoring system are specialized condition monitoring solutions and condition monitoring equipment. These tools collect and analyze data from various sensors and devices strategically placed throughout the factory or production facility.[7]

1.3.1. Predictive Maintenance and Condition Monitoring

Predictive maintenance is a maintenance strategy that aims to predict when equipment is likely to fail so that maintenance can be performed in due time to prevent the failure. It contrasts with traditional reactive maintenance, which involves fixing equipment after it has already failed, and preventive maintenance, which follows a fixed schedule regardless of the actual condition of the equipment. Both these methods result in high costs, either due to failure-derived downtime, or unnecessary replacement of costly components and equipment.

Condition monitoring plays a vital role in predictive maintenance. By continuously monitoring the health and performance of equipment, condition monitoring systems identify early warning signs of potential issues. Such signs may be abnormal vibrations, temperature fluctuations, clamping force,

or changes in fluid flow patterns. These can be indicative of looming problems. The ability to detect these precautionary signs in real-time allows maintenance professionals to act on them before actual issues occur.[7]

1.3.2. Benefits of condition monitoring

Condition monitoring offers several other advantages over traditional maintenance approaches, including:

Improved maintenance planning: Condition monitoring provides real-time data on the performance of a system or component, which can be used to optimize maintenance planning and scheduling. This data helps to reduce the frequency of maintenance activities, while ensuring that they are executed only when needed, based on actual system performance.

Increased equipment lifespan: By detecting and addressing problems preemptively, condition monitoring helps to extend the lifespan of equipment and components, reducing the need for costly replacements or repairs and maximizing the ROI of assets.

Improved operational efficiency: Condition monitoring can help to identify inefficiencies in a system or component, such as excessive energy consumption or unnecessary wear and tear. By addressing these issues, operational efficiency can be improved, leading to reduced costs and improved productivity.

Enhanced safety: Condition monitoring can help to identify potential safety hazards, such as worn or damaged components, before they cause harm to personnel or equipment. This approach helps to improve overall safety and reduce the risk of accidents and injuries.[8]

1.3.3. Types of Condition Monitoring

Condition monitoring encompasses various techniques and parameters to access the health and performance of equipment. Here are some of the most commonly used types of condition monitoring:

- **Vibration Monitoring:** Vibration analysis is a widely employed technique for monitoring rotating machinery. By analyzing the vibration patterns, frequencies, and amplitudes, maintenance professionals can detect imbalances, misalignments, bearing defects, and other potential issues.
- **Temperature Monitoring:** Temperature is a crucial parameter to monitor in any application. Abnormal temperature variations can indicate problems such as overheating, cooling system failures, or insulations issues.
- **Pressure Analysis:** Pressure variations can be indicative of issues in systems such as hydraulic or pneumatic equipment. Continous pressure monitoring helps identify leaks, blockages, valve malfunctions, or excessive load conditions.[9]

1.4. NanoEdge AI Overview

1.4.1. Definition

NanoEdge AI Studio is a free software provided by ST integrate AI into to any embedded machine learning project running on **any Arm® Cortex®-M MCU**. It acts like a search engine, finding the best AI libraries for given project. This software needs input datasets to figure out the optimal mix of data processing, model structure, and settings.

Once it finds the best setup for the given data, it creates AI libraries. These libraries make it easy to use the data processing and model into existing C based embedded systems. [10]

Figure 1.3 : NanoEdge AI Studio

1.4.2. NanoEdge AI Library

NanoEdge AI Libraries are the output of NanoEdge AI Studio. They are a static libraries for embedded C software on Arm® Cortex-M® microcontrollers (MCUs). Packaged as a precompiled. a file, it provides the building blocks to integrate smart features into C code without requiring expertise in mathematics, machine learning, or data science.

When embedded on microcontrollers, the NanoEdge AI Library enables them to automatically "understand" sensor patterns. Each library contains an AI model with implementable types of algorithms for tasks like:

- **learning signal patterns:** The system can observe normal behavior and build its own internal knowledge without needing pre-labeled data.
- **detecting anomalies:** It can recognize when incoming signals behave differently from the learned normal patterns, indicating possible faults or unexpected behavior.
- **classifying signals:** The library can identify which predefined state or condition the machine is currently in, based on live sensor data.
- **extrapolating data:** It can predict future sensor values or trends based on past and current observations, useful for forecasting behavior or early warnings.[10]

Each kind of project in NanoEdge has its own kind of AI library with their functions but they share the same characteristics:

- **Highly optimized:** Designed for MCUs (any Arm® Cortex®-M).
- **Memory efficient:** Requires only 1-20 Kbytes of RAM/flash memory.
- **Fast inference:** Executes in 1-20 ms on Cortex®-M4 at 80 MHz.
- **Cloud independent:** Runs directly within the microcontroller.
- **Easy integration:** Can be embedded into existing code/hardware.
- **Energy efficient:** Minimal power consumption.
- **Static allocation:** Preserves the stack.
- **Data privacy:** No data transmission or saving.
- **User-friendly:** No machine learning expertise required for deployment.

All NanoEdge AI Libraries are created using NanoEdge AI Studio.[10]

1.4.3. NanoEdge AI Studio capabilities

NanoEdge AI Studio can:

- Search for optimal AI libraries (preprocessing + model) given user data.
- Simplify the development of machine learning features for embedded developers.
- Minimize the need for extensive machine learning and data science knowledge.
- Utilize minimal input data compared to traditional approaches.

Using project parameters (MCU type, RAM, sensor type) and signal examples, the Studio outputs the most relevant NanoEdge AI Library. This library can be either untrained (learning post embedding) or pretrained. The library performs inference directly on the microcontroller, involving:

- Signal preprocessing algorithms (for example, FFT, PCA, normalization).
- Machine learning models (for example, kNN, SVM, neural networks such as MLP, proprietary algorithms).
- Optimal hyperparameter settings.

The process is iterative: import signals, run a benchmark, test the library, adjust data, and repeat to improve results. [10]

1.4.4. NanoEdge AI Studio limitations

NanoEdge AI Studio:

- Requires user-provided input data (sensor signals) for satisfactory results.
- Libraries' performances are heavily correlated to the quality of the data imported.
- Does not offer ready-to-use C code for final implementation. Users must write, compile, and integrate the code with the AI library.

In summary, NanoEdge AI Studio outputs a static library (.a file) based on user data, which must be linked and compiled with user-written C code for the target microcontroller.[10]

1.5. NanoEdge AI data format and Use Cases

NanoEdge AI Studio and its generated NanoEdge AI Libraries are designed to be sensor-agnostic, offering compatibility with all sensor types including vibration, current, sound, and other measurement modalities. The platform automatically extracts meaningful patterns from input data regardless of the sensing technology.[10]

1.5.1. Time series versus cross sectional

Time series refers to data that contains information of multiple points in time.

Cross sectional refers to data that contains information at a single point in time.

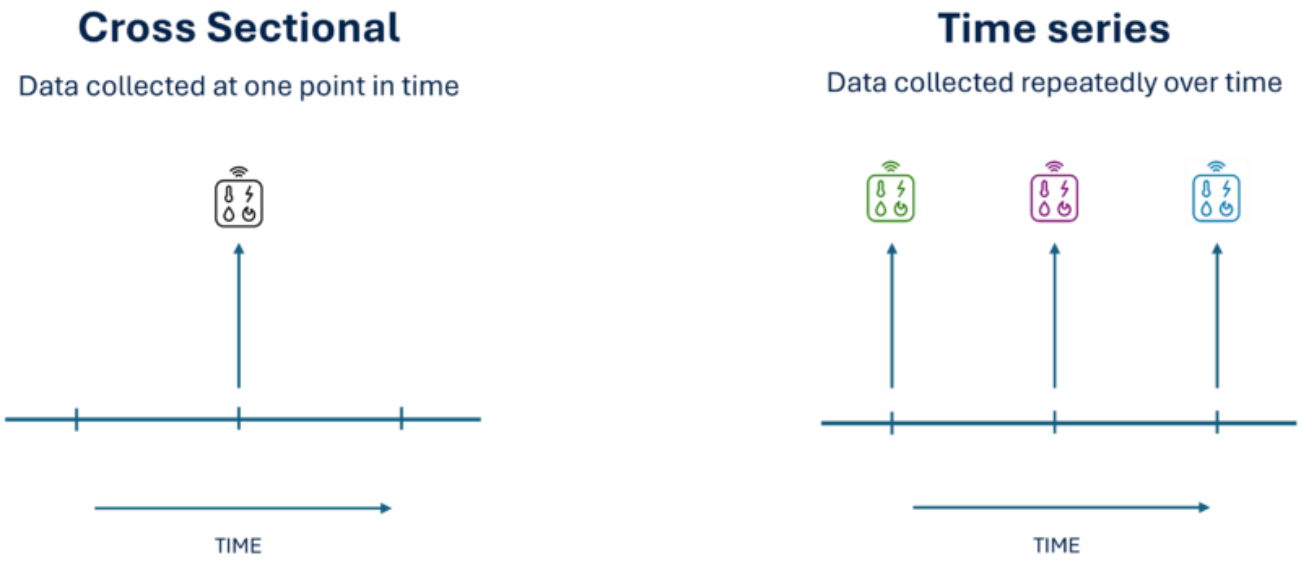


Figure 1.4: NanoEdge AI data format

In an embedded environment, in general, working with time series data is prioritized for analysis. Information about a motor at a single point in time is, most of the time, useless. Most of the useful information is contained in the evolution in time of the data, not in a single observation. in applications like real estate price prediction—where input features (e.g., square footage, room count) are static—temporal evolution is unnecessary.[10]

1.5.2. Defining important concepts

Here are some clarifications regarding the important terms that are used in NanoEdge:

- **Axis/Axes**: In NanoEdge AI Studio, the axis/axes are the total number of variables outputted by the sensor used for a project. For example, a 3-axis accelerometer outputs a 3-variables
- **sample (x,y,z)** corresponding to the instantaneous acceleration measured in the 3 spatial directions. In case of using multiple sensors, the number of axes is the total number of axes of all sensors. For example, if using a 3-axis accelerometer and a 3-axis gyroscope, the number of axes is 6.
- **Signal**: A signal is the continuous representation in time of a physical phenomenon. We are sampling a signal with a sensor to collect samples (discrete values). Example: vibration, current, sound.
- **Sample**: This refers to the instantaneous output of a sensor, and contains as many numerical values as the sensor has axes. For example, a 3-axis accelerometer outputs 3 numerical values per sample, while a current sensor (1-axis) outputs only 1 numerical value per sample.
- **Data rate**: The data rate is the frequency at which we capture signal values (samples). The data rate must be chosen according to the phenomenon studied. Also pay attention to having a consistent data rate and number of samples to have buffers that represent a meaningful time frame.
- **Buffer**: A buffer is a concatenation of consecutive samples collected by a sensor. A dataset contains multiples buffers. The term **Line** is also used when talking about buffers of a dataset. Buffers are the input for NanoEdge, being in the Studio or in the microcontroller.[10]

1.5.3. Designing a relevant sampling methodology

Compared to traditional machine learning approaches that might need hundreds of thousands of signal examples, NanoEdge AI Studio requires significantly less data—typically around 100-1000 buffers per class, depending on the use case.

However, the data must be **qualified**, containing relevant information about the physical phenomena to be monitored. Designing a proper sampling methodology is crucial to ensure that all desired characteristics of the physical phenomena are accurately captured and translated into meaningful data.

Key considerations:

- **Study the right physical phenomenon:** Ensure that the data captures the correct physical events.
- **Define the environment:** Match the data collection environment to the real deployment environment.
- **Work with buffers, not samples:** Buffers provide more context and patterns than individual samples.
- **Work with raw data:** Use unprocessed data for better accuracy.
- **Choose a relevant sampling rate and buffer size:** Ensure these settings capture the necessary details.
- **Use a temporal window that makes sense:** Select a time frame that covers the relevant events.
- **Start small and iterate:** Begin with a small dataset and refine as needed. [10]

1.6. STM32 Microcontroller Technology

1.6.1. STM32 Overview

STMicroelectronics is manufacturing the STM32 MCUs which are based on the Arm® Cortex®-M processor. They are 32-bit RISC Arm processor cores, optimized for cost and power sensitive MCUs. **RISC Arm (Reduced Instruction Set Computer)** is an instruction set that allows to divide the commands into several, simple and not complicated instructions that are meant to realize small objectives. The STM32 family offers a large range of devices combining [11]:

- Full integration
- Ease of development thanks to the large ecosystem and professional development tools including free ones
- Very high performance thanks to Arm® Cortex®-M core and ST ART Accelerator™ for some families
- Real-time capabilities
- Digital signal processing
- Low-power and low-voltage operation

Several applications rely on STM32:

- Industrial control systems
- Human interface devices
- Smart metering
- Motor control
- Medical instruments
- Buildings and security (alarms, access control, power meters…)
- Consumer products (PC peripherals, GPS, gaming…)
- Internet of Things
- Connectivity

A) Varieties of STM32 MCUs

There are already 16 Cortex-M based microcontroller families (or series).

The STM32 families are split into 4 groups divided as follows:

	Family	Core	Max Frequency	Flash
High Performance	STM32H7	Cortex-M7 - Cortex-M4	480 MHz - 240 MHz	1 to 2 Mbytes
	STM32F7	Cortex-M7	216 MHz	256 Kbytes to 2 Mbytes
	STM32F4	Cortex-M4	180 MHz	64 Kbytes to 2 Mbytes
	STM32F2	Cortex-M3	120 MHz	128 Kbytes to 1 Mbyte
Mainstream	STM32G4	Cortex-M4	170 MHz	32 to 512 Kbytes
	STM32F3	Cortex-M4	72 MHz	16 to 512 Kbytes
	STM32F1	Cortex-M3	72 MHz	16 Kbytes to 1 Mbyte
	STM32G0	Cortex-M0+	64 MHz	16 to 512 Kbytes
	STM32F0	Cortex-M0	48 MHz	16 to 256 Kbytes
Ultra-low-power	STM32U5	Cortex-M33	160 MHz	1024 to 2048 Kbytes
	STM32L5	Cortex-M33	110 MHz	256 to 512 Kbytes
	STM32L4+	Cortex-M4	120 MHz	512 Kbytes to 2 Mbytes
	STM32L4	Cortex-M4	80 MHz	64 Kbytes to 1 Mbyte
	STM32L1	Cortex-M3	32 MHz	32 to 512 Kbytes
	STM32L0	Cortex-M0+	32 MHz	8 to 192 Kbytes
Wireless	STM32WB	Cortex-M4 - Cortex-M0+	64 MHz - 32 MHz	256 Kbytes to 1 Mbyte
	STM32WL	Cortex-M4	48 MHz	64 Kbytes to 256 Kbytes

Table 1.1: Comparison of STM32 Microcontroller Families [11]

The STM32 portfolio continues to be extended with more integration and performance and more complex peripherals. All ones with generic or dedicated application field.

The previous table helps to select the family of MCUs suitable depending on application needs, but it's required to review deeply the product characteristics and components to fine-tune the choice.[11]

1.6.2. STM32F103RB

The STM32F103RB from STMicroelectronics belongs to the medium-density performance line of microcontrollers, featuring a 32-bit Arm® Cortex®-M3 RISC core. This microcontroller combines real-time control capabilities with advanced communication interfaces and low power consumption, making it ideal for high-performance embedded applications.

Selection of the STM32F103RB followed comprehensive benchmarking. Its architectural strengths - particularly in real-time processing and energy efficiency - perfectly match the project's core requirements for edge AI implementation.[12]

Figure1.5: STM32F103RB MCU

The STM32F103RB microcontroller was selected for its balanced performance and peripheral integration. Table below summarizes its key specifications and their relevance to this project's edge computing requirements

Peripheral		STM32F103RB
Memory	**Flash - Kbytes**	**128**
	SRAM - Kbytes	**20**
Timers	**General-purpose**	**3**
	Advanced-control	**1**
Communication	**SPI**	**2**
	I2C	**2**
	USART	**3**
	USB	**1**
	CAN	**1**
GPIOs		**51**
12-bit synchronized ADC Number of channels		**2** 16 channels(1)
CPU frequency		**72 MHz**
Operating voltage		**2.0 to 3.6 V**
Operating temperatures		Ambient temperatures: -40 to +85 °C / -40 to +105 °C Junction temperature: -40 to + 125 °C

Table 1.2: peripheral of STM32F103 [12]

A) Selection Criteria and Merits of STM32F103RB

- Offers a good balance between performance and low power consumption, making it a solid choice for real-time, sensor-based systems — even in tough machine environments.
- Handles real-time state monitoring while helping to extend battery life thanks to its efficient power features.
- Comes with a wide range of communication options (I^2C, SPI, UART, CAN), which makes it easy to connect with sensors and other components.
- Budget-friendly roughly 22% cheaper than similar ARM-based microcontrollers — and well-supported through clear documentation, a large online community, and ST's own development tools like STM32CubeIDE.

B) Direct Memory Access (DMA)

One of the key features of the STM32F103RB microcontroller is the presence of a Direct Memory Access (DMA) controller. The Direct memory access (DMA) embedded in the STM32F103RB microcontrollers is used to provide high-speed data transfers between peripherals and memory and between memory and memory. Data can be quickly moved by the DMA without any CPU action. This keeps CPU resources free for other operations The DMA channels can access any memory-mapped location, including: • AHB peripherals, for instance the CRC generator, • AHB memories, for instance the SRAM, • APB peripherals, for instance the USART peripheral [13]

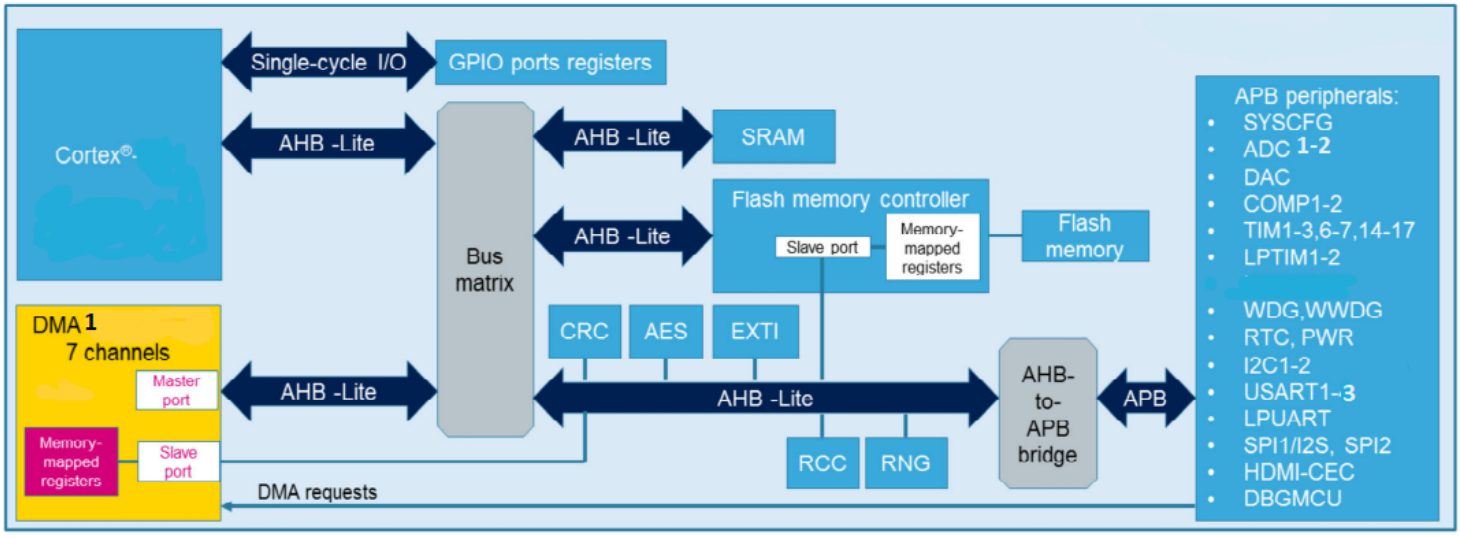


Figure 1.6: Overview of STM32f103rb Internal Bus System and DMA Integration [13]

1.7. Motion Sensors and Vibration Monitoring

1.7.1. Motion sensors

Motion sensors play an important role in a variety of industries and applications.

Active motion sensors, such as vibration sensors, tilt sensors, gesture sensors, accelerometers, ultrasonic sensors, microwave sensors and optoelectronic sensors, detect and analyses motion through different mechanisms.

Passive motion sensors, such as passive infrared sensors (PIR), detect motion by sensing energy changes.

Motion sensors enable precise motion detection and improve safety, efficiency, and automation in different areas.

The table below **(table1.1) [14]** will present the comparison between some motion sensors:

Motion Sensor	Types	Working Principle	Applications	Detecting Acts
Vibration sensors	Active	Convert vibration signals into electrical signals.	Mechanical equipment monitoring, Earthquake monitoring, Motion control.	Vibration or shaking.
Tilt Sensors		Convert tilt signals into electrical signals.	Structural monitoring, Mechanical equipment monitoring, Motion control.	Tilt angle.
Gesture sensors		Convert gesture signals into electrical signals.	Game control, Smart home, Human-machine interaction.	Gesture
Accelerometer		Convert acceleration signals into electrical signals.	Motion control, Smart watches, Safety airbags.	Acceleration
Ultrasonic Sensors		Emit ultrasound and receive echoes, calculate the distance of an object based on the echo time.	Distance measurement, Motion control, Smart home.	Distance and Motion
Microwave Sensors		Emit microwaves and receive echoes, calculate the distance and velocity of an object based on the echo time.	Motion control, Smart home, Security monitoring.	Distance and Motion
Photoelectric sensors		Emit rays of light and detect the reflection or obstruction of the light.	Motion control, Smart home, Human detection	Distance and Motion
Passive Infrared Sensors	Passive	Detect the infrared radiation emitted by an object.	Security monitoring, Smart home, Human detection	Object Motion

Table 1.3: Comparison table of motion sensors

1.7.2. Vibration monitoring

When managing and operating an industrial system, condition monitoring is critical. Proper condition monitoring, which maximizes the operative capacity of equipment while minimizing exposure risk, prolongs the life of assets, which, in turn, leads to less equipment downtime, fewer interruptions to operations, improved efficiency, and, ultimately, reduced asset life-cycle costs. For industrial equipment, vibration is one of the major variables for which proactive condition monitoring is essential.[15]

A) **Root cause analysis**

Vibration monitoring and analysis can help with root cause analysis; if there is damage to a piece of equipment, tracking down the root source of vibration can lead to an understanding of the cause of that damage and insight into how to avoid re-occurrence.[15]

B) Predictive Maintenance

While root cause analysis is an important benefit, vibration monitoring is perhaps most valuable when used in predictive maintenance – as a means of preventing the failure from occurring in the first place. Using a vibration sensor, vibration data can be tracked in real time. Early detection of increased vibration within a piece of equipment allows for maintenance actions to be cost-effectively planned ahead of time, preventing damage from occurring or developing.[15]

C) Primary Types of Vibration Sensors

There are three primary types of vibration sensors, each with their own advantages and disadvantages:

- **Accelerometers:** Measure absolute vibration acceleration directly
- **Velocity Sensors:** Measure absolute vibration velocity directly
- **Displacement Sensors (Proximity Probes or Eddy Current Probes) -** Measure relative vibration in displacement directly (DC – static displacement / AC – variable displacement) [15]

1.7.3. Focus on Accelerometers for Vibration Monitoring

Among the different types of motion sensors, **accelerometers stand out** for their ability to capture even the smallest or fastest changes in movement. They can detect acceleration along one, two, or three axes, which makes them especially useful for monitoring **vibration in machines that rotate or move**.

In this project, we use a **3-axis accelerometer the MPU6050** to measure how a mechanical system moves and shakes in real time. This allows us to track:

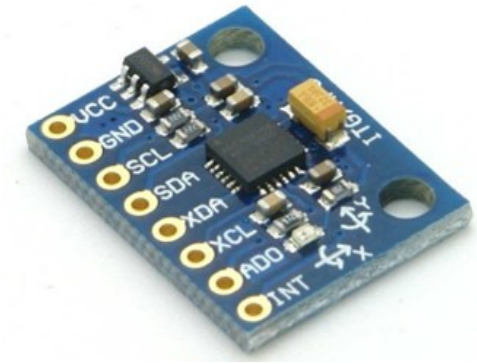


Figure 1.7 : MPU6050 Module

- Vibration patterns
- Directional changes in motion
- Sudden impacts or irregularities

Using accelerometers for vibration analysis is a **proven method in predictive maintenance**. It helps spot early signs of issues like imbalance, looseness, or misalignment each of which creates its own vibration "signature" that the system can learn to recognize.[16]

1.8. Human-Machine Interfaces in Embedded Systems

A Human Machine Interface (HMI) refers to the point of interaction between humans and machines. It encompasses the hardware and software components that enable users to control, monitor, and communicate with machines, systems, or processes. HMIs facilitate information exchange, providing users with a visual representation of the system's status and enabling them to

input commands or receive feedback. In essence, HMIs act as a bridge, translating complex technical data into a format that is easily understandable and usable by humans.

1.8.1. The functionality of Human-Machine Interfaces (HMIs)

To understand how an HMI works, it is essential to familiarize ourselves with the basic elements of an HMI system:

D) **Input Devices:** These devices allow users to input commands or information into the system. Examples include buttons, keyboards, touchscreens, voice recognition systems, and motion sensors.

E) **Output Devices:** These devices present information or feedback to the user. Common output devices include displays, lights, alarms, speakers, and haptic feedback mechanisms.

F) **Control System:** The control system processes user input and generates corresponding output signals to control the machine or system. It acts as the intermediary between the user and the underlying technology, ensuring the correct execution of commands and providing feedback.

G) **Software:** HMI software plays a crucial role in creating an intuitive and user-friendly interface. It enables the customization of visual elements, data representation, and interaction capabilities, allowing developers to design interfaces that meet specific user requirements.

As embedded systems continue to evolve, HMIs are increasingly integrated with networked technologies, forming a key component of Internet of Things (IoT) ecosystems.

1.8.2. The Link Between HMI and Internet of Things (IoT)

The Internet of Things (IoT) is revolutionizing the way devices and systems interact and communicate. HMIs play a crucial role in connecting IoT devices to humans, providing interfaces to monitor and control IoT-enabled systems. HMIs enable users to access IoT data, visualize sensor readings, and perform actions remotely. For example, in a smart home setting, an HMI could allow users to adjust temperature settings, control lighting, or manage security systems through a mobile app or voice-activated interface. The integration of HMIs and IoT expands the capabilities of both

technologies, creating a more interconnected and intelligent ecosystem. [16]

1.9. Conclusion

In this first chapter, we explored the key ideas and technologies behind real-time embedded monitoring systems. We saw how edge computing and embedded AI are changing the way devices think and act, pushing intelligence closer to the source of data.

We also introduced the **STM32F103RB microcontroller**, showing how it plays a critical role in running local analysis directly inside the system. A big part of this work involved using **NanoEdge AI Studio**, which made it possible to bring machine learning to a low-power, small-memory device without needing to write complicated AI algorithms ourselves.

Along the way, we discussed why detecting multiple machine states not just anomalies matters when it comes to keeping systems healthy and reliable. We also touched on the importance of sensing motion and giving users simple ways to see what's happening, through local human-machine interfaces.

With all of this groundwork in place, we are now ready to dive into the next chapter, where the actual system design, hardware choices, and software integration will be explained in detai

Chapter 2

Conceptual Design and Architecture

2.1. Introduction

Building a reliable embedded AI system goes far beyond just selecting a microcontroller and writing some code, it requires deliberate architectural planning across both hardware and software layers. In this chapter, we walk through the conceptual and technical design decisions that shaped our system, starting from low-level sensor integration to the overall system structure.

We began with the essentials: sensing motion using an MPU6050 accelerometer, managing high-frequency data streams via DMA, and processing this data in real time using an STM32F103RB microcontroller. From there, we incrementally introduced intelligent logic using NanoEdge AI Studio, optimized the display layer through dynamic OLED control, and extended our system's capabilities by integrating a secondary ESP32-based PCB for remote monitoring over Wi-Fi.

Rather than treating each piece of hardware or software in isolation, we focused on how each layer communicates, adapts, and contributes to the whole, ensuring that the system remains robust, scalable, and responsive under real-world conditions.

2.2. System overview

Before diving into the technical details, it’s important to understand how the entire system works as a whole. Our solution is designed around a compact, layered architecture that allows for real-time monitoring of motion-based behavior, intelligent analysis directly on the device, and clear user feedback, all without relying on cloud connectivity.
We broke the system down into four main layers to make it easier to design, explain, and build:

A) Sensing Layer

At the core of the sensing layer is the **MPU6050**, a 3-axis accelerometer and gyroscope sensor. It continuously captures motion and vibration data from the machine in real time. This module serves as the “sensory system” of the project, detecting subtle changes in behavior that could indicate early mechanical faults or abnormal machine states. The sensor operates at a high sampling frequency, delivering timely, high-resolution motion data essential for reliable state detection.

B) Processing Layer

Next, the **STM32** microcontroller that serves as the brain of the system, receives this raw motion data using the I²C protocol and **Direct Memory Access (DMA**) to optimize data transfer and reduce CPU load. During the training phase, motion data is streamed to a PC through COM PORT USART serial communication and saved as .csv files for use in NanoEdge AI Studio**.** After training, the microcontroller autonomously handles data acquisition, runs embedded AI inference in real time, and interprets machine behavior to determine whether the system is in a normal or abnormal state, all within the constraints of an embedded system.

C) AI Intelligence Layer

This layer turns motion data into meaningful insights. Using **NanoEdge AI Studio**, we trained an embedded AI model by importing real sensor data, extracting features, and selecting the best algorithm. The tool then generated a lightweight C library, integrated into the STM32. This enables fast, local classification of machine behavior without cloud access or external computing.

D) Display Layer

This layer combines local and remote feedback.the **SSD1306 OLED** shows essential AI results in real time such as system state, confidence score, and update frequency. At the same time, an **ESP32 module** hosts a dynamic web page that displays detailed monitoring data over Wi-Fi, accessible from any device. For broader communication, a **CAN bus** interface is also integrated, enabling display updates on external dashboards or industrial systems.

The overall process is summarized in the flowchart below, illustrating how motion signals travel from the sensor to the microcontroller, then through the AI analysis pipeline, and finally to the OLED display and web page for real-time feedback.

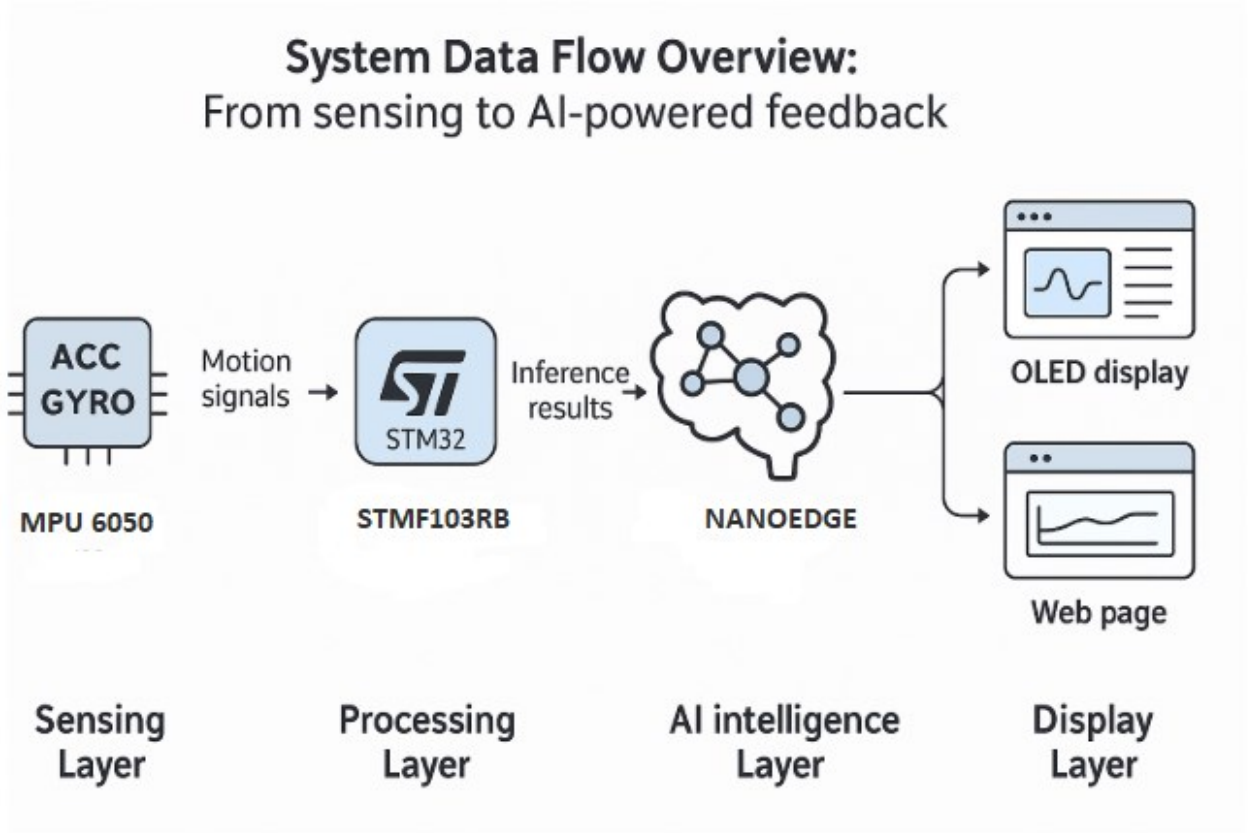


Figure 2.1: System Data Flow Overview: From sensing to AI-powered feedback

To implement this layered architecture in practice, two custom hardware boards were developed:

- A **main STM32-based PCB**, which handles real-time data acquisition, embedded AI inference, and local feedback display
- A **secondary ESP32-based module**, which receives processed results via CAN and transmits them wirelessly to a web interface

These two boards work together, **one collects and processes data at the machine**, while the other **sends the results wirelessly to a web page**. Their full design, how they were built, and how they work together will be explained in Chapter 3.

2.3. Hardware Design and Prototyping Process

Designing the hardware wasn't a one-step job it was a process of trial, validation, and refinement. We didn't just jump straight into building a PCB. Instead, we followed a hands-on path: start with a working prototype, verify each component, then consolidate everything into a compact, reliable board design. This section walks through that evolution and highlights the decisions that led us to a final, functional embedded system.

2.3.1. Initial Microcontroller Testing

Before integrating the microcontroller into any system, we first tested the STM32F103RB on a **LQFP64 support development board**. This initial step helped us confirm the microcontroller was functional and programmable. It also gave us a clear map of the **actual pinout behavior**, so we could later route components with confidence.

This minimal environment included only the necessary peripherals: power regulation, reset circuit, SWD programming header, and boot configuration jumpers.

This approach helped us confirm:

- The microcontroller could be flashed and debugged properly
- Basic I/O such as I²C worked reliably
- Clock configuration and USB behavior were stable with an 8 MHz external crystal

Figure 2.2: LQFP64 support board for STM32F103RB

2.3.2. Breadboard Functional Prototype

With the microcontroller validated, we built a working prototype using individual breakout modules on a breadboard:

- MPU6050 for motion sensing
- SSD1306 OLED for display
- Tactile Switch: Mode toggling/reset/AI learning
- Jumper wires for I²C connections
- Basic power supply (regulated 3.3V)

This stage helped us validate the full functionality of our system. We confirmed:

- The sensor readings matched expected values
- The AI model could run and respond properly on STM32
- The display updated in real-time

It also exposed issues like **slow screen refreshes** (solved later with DMA and I²C fast mode) and **breadboard instability**, which justified the move to a PCB.

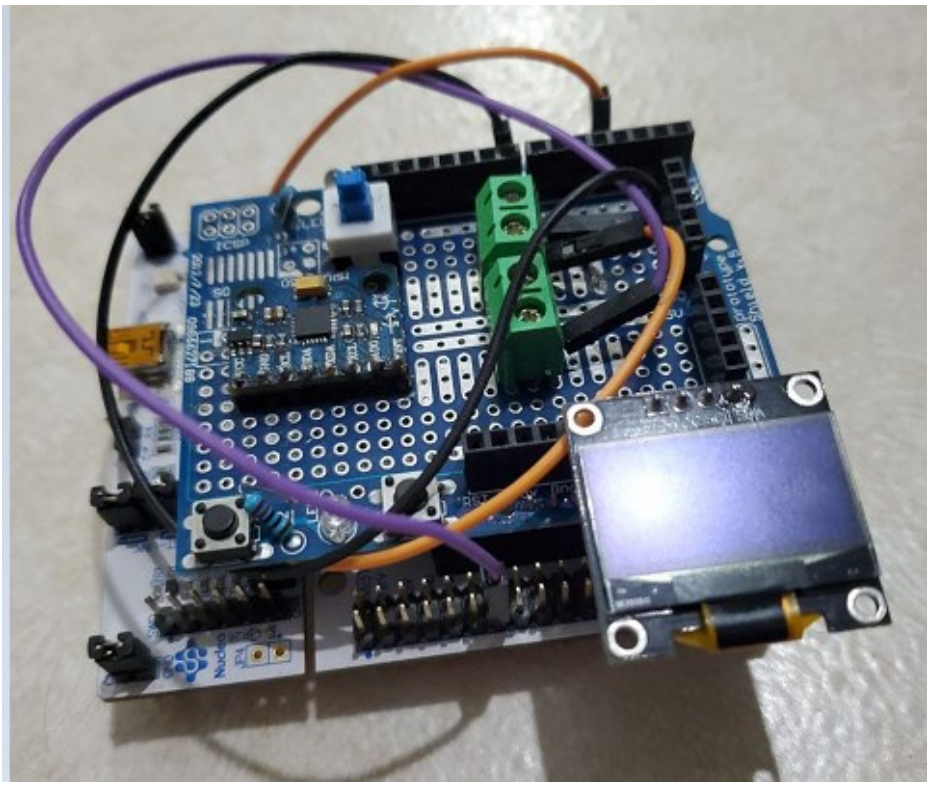


Figure 2.3: Breadboard prototype of full system

2.3.3. Transition to a Custom PCB

Breadboards are great for prototyping, but they're far from ideal for a reliable or portable system they are unsuitable for long-term, vibration-prone environments. To address these concerns, we began designing a dedicated PCB **(main pcb)**, integrating all validated components into a unified, vibration-resistant, and compact platform. This step would allow:

- **Mechanical durability** during motion/vibration
- **Signal integrity** through dedicated GND and power planes
- **Compact, professional layout** (40 mm × 40 mm)
- **Aesthetic & product-grade appearance** for possible future commercialization

For this transition we chose a **rigid PCB** unlike temporary setups, this type of board ensures mechanical durability, stable signal transmission, and clean integration of all critical components. As the design evolved, we also factored in support for external communication and wireless monitoring, which led to the inclusion of **ESP32 headers** and a **dedicated CAN bus interface** in the final concept. To support this communication path without interfering with the main processing board, we also developed a **second, simpler 2-layer PCB** specifically for the ESP32 and CAN modules. Together, these additions made the architecture more modular, scalable, and capable of delivering real-time results over a Wi-Fi dashboard.

- ***Details of both PCB designs, layout, and final fabrication are presented in Chapter 3.***

2.4. Sensor Integration : MPU6050

To monitor motion and detect mechanical states in real time, this project relied on the MPU6050, a popular 3-axis accelerometer and 3-axis gyroscope module. The sensor was chosen for its accuracy, availability, and compatibility with embedded systems like STM32.

2.4.1. Physical Connection via I^2C

The MPU6050 was connected to the STM32F103RB via the I^2C protocol, this was chosen based on datasheet and ease of use within our software environment. The following table summarizes the connection:

MPU6050 Pin	STM32F103RB pin	Description
VCC	3.3V	Powe Supply
GND	GND	Ground
SDA 2	PB11	I²C Data Line
SCL 2	PB10	I²C Clock Line

Table 2.1: I²C Connections

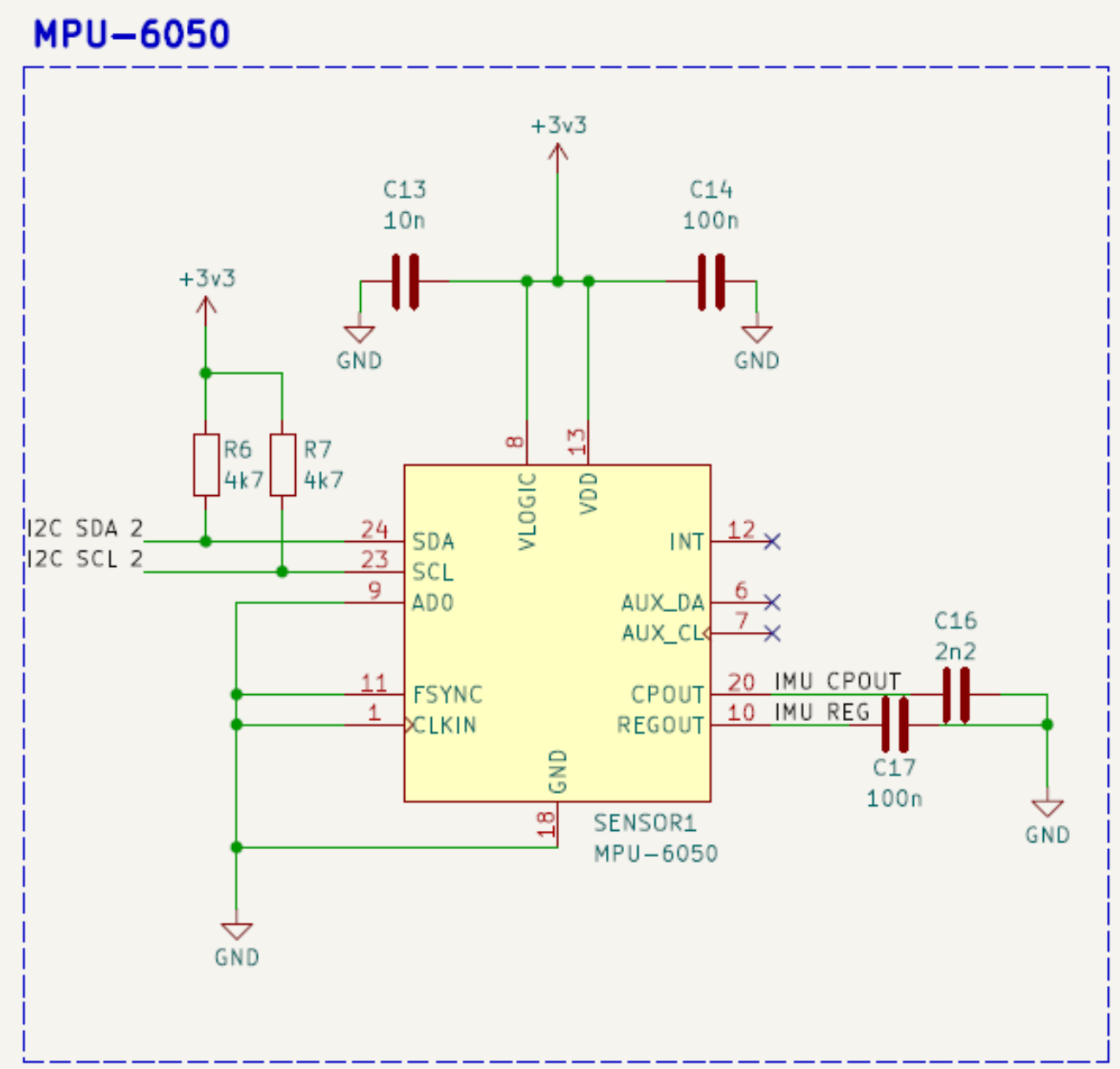


Figure 2.4: Extracted Schematic: MPU6050 to STM32 Wiring

To ensure robust communication, **4.7kΩ pull-up resistors** were added to both SDA and SCL lines, as recommended in the official datasheet. Additionally, **100 nF ceramic decoupling capacitors** were placed between VCC and GND to filter voltage spikes and stabilize power delivery. These hardware choices follow the Typical Operating **Circuit** provided by InvenSense (see **Figure 2.**5[18]), helping to prevent I²C instability and power noise.

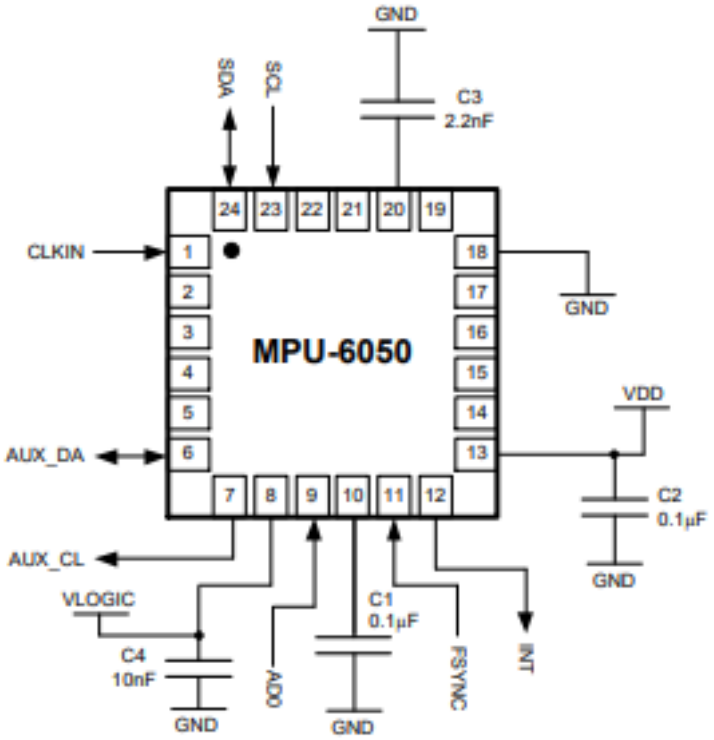


Figure 2.5: MPU6050 Typical Operating Circuit

2.4.2. I²C Communication and Direct Memory Access Optimization

Initially, sensor data was read using **polling**, but this method caused CPU blocking during read cycles. To resolve this, the system was upgraded to use **DMA** in combination with interrupts. When the MPU6050 signals that new data is available (via its **INT pin**), the STM32 triggers a DMA transfer that stores six bytes (X, Y, Z accelerometer axes) into a memory buffer, freeing the CPU for other tasks like AI inference and display updates.

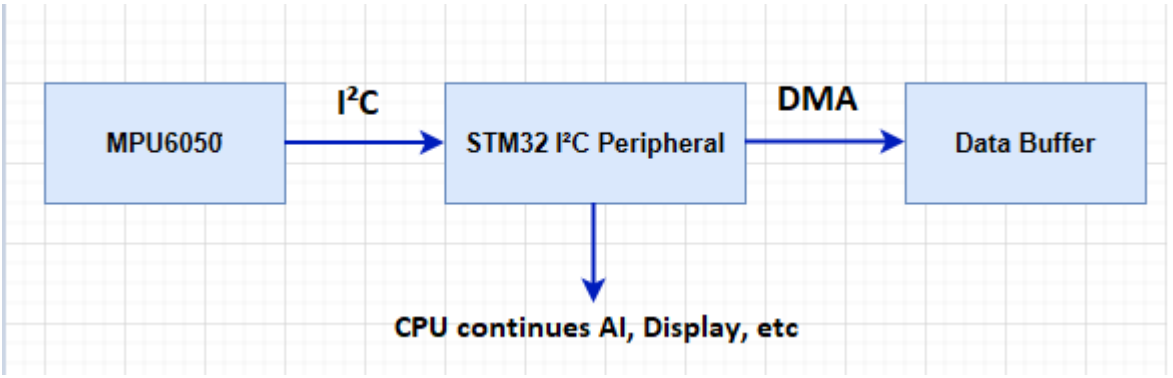


Figure 2.6: Efficient Data Path Using DMA and Interrupts.

The STM32 uses DMA to offload MPU6050 readings directly into memory, enabling the CPU to remain free for AI inference and display updates.

2.4.3. Register Configuration and Initialization

To configure the MPU6050, a custom library was developed that directly interacts with the device's internal registers. The initialization procedure involved writing specific values to key configuration registers as defined in the **official datasheet**. The following table summarizes the most relevant register settings used:

Register Name	Address	Purpose	Configured value
Power Management 1 : ***PWR_MGMT_1***	**0x6B**	Wake up the sensor, select clock source	**0x00**
Sample Rate Divider ***SMPLRT_DIV***	**0x19**	Sets sample rate divider = 1 kHz / (1 + 1) = 1000 Hz	**0x01**
Accelerometer Config ***ACCEL_CONFIG***	**0X1C**	Set accelerometer range (±2g/4g/8g/16g)	**0X00**(±2g)
Interrupt Enable ***INT_ENABLE***	**0x38**	Enable interrupt to notify new data ready	**0x01**

Table 2.2: MPU6050 Register Initialization Summary [18]

These values were written via I²C commands at startup, using a lightweight library that directly accessed the sensor's internal registers.

2.4.4. Sampling Strategy with Interrupts

The MPU6050's data-ready interrupt was configured to trigger every **1ms**, achieving a steady **1000 Hz sampling rate**. This frequency was chosen as a balance between capturing detailed motion and maintaining low memory usage.

Notably, we found that data normalization was unnecessary in this application. Since the NanoEdge AI Studio handled internal preprocessing including **Fast Fourier Transform (FFT)** and dynamic scaling the raw accelerometer values (signed 16-bit integers) were fed directly into the model without transformation.

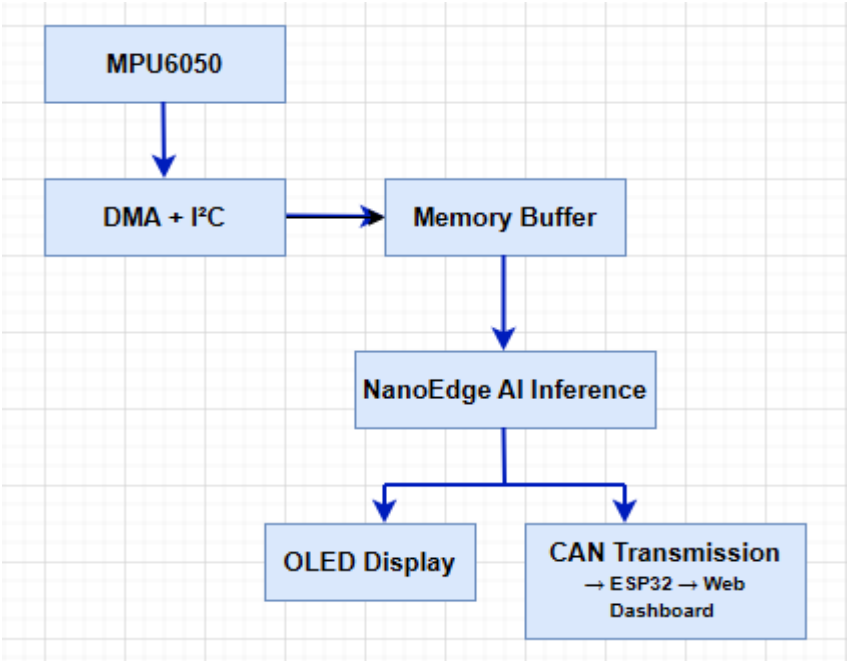


Figure 2.7: Raw Signal Path from MPU6050 to AI Model

2.4.5. Datasheet-Guided Design & Lessons Learned from Sensor Integration

The MPU6050 integration journey was deeply shaped by both experimentation and a strong reliance on its official datasheet. During early development, we assumed that using both the

accelerometer and gyroscope with sensor fusion algorithms (**Kalman filtering**) would yield the most accurate results. However, after prototyping and reviewing literature on vibration analysis, we found that accelerometer data alone was **sufficient** for detecting the machine states relevant to our project. This decision reduced system complexity while preserving performance.

To optimize both hardware and firmware, the MPU6050 datasheet served as a technical backbone. **Key aspects it guided include:**

- **Component selection:** Pull-up resistors for I²C lines and decoupling capacitors near VCC and GND pins
- **Power-up sequence and clock source selection:** (PWR_MGMT_1 – Register 107)
- **Sampling configuration**: using SMPLRT_DIV (Register 25) for a 1000 Hz data rate
- **Sensor output data formatting**: referencing Registers 59–64 for raw accelerometer values
- **I²C protocol constraints**: including hold times, bus speed, and timing margins

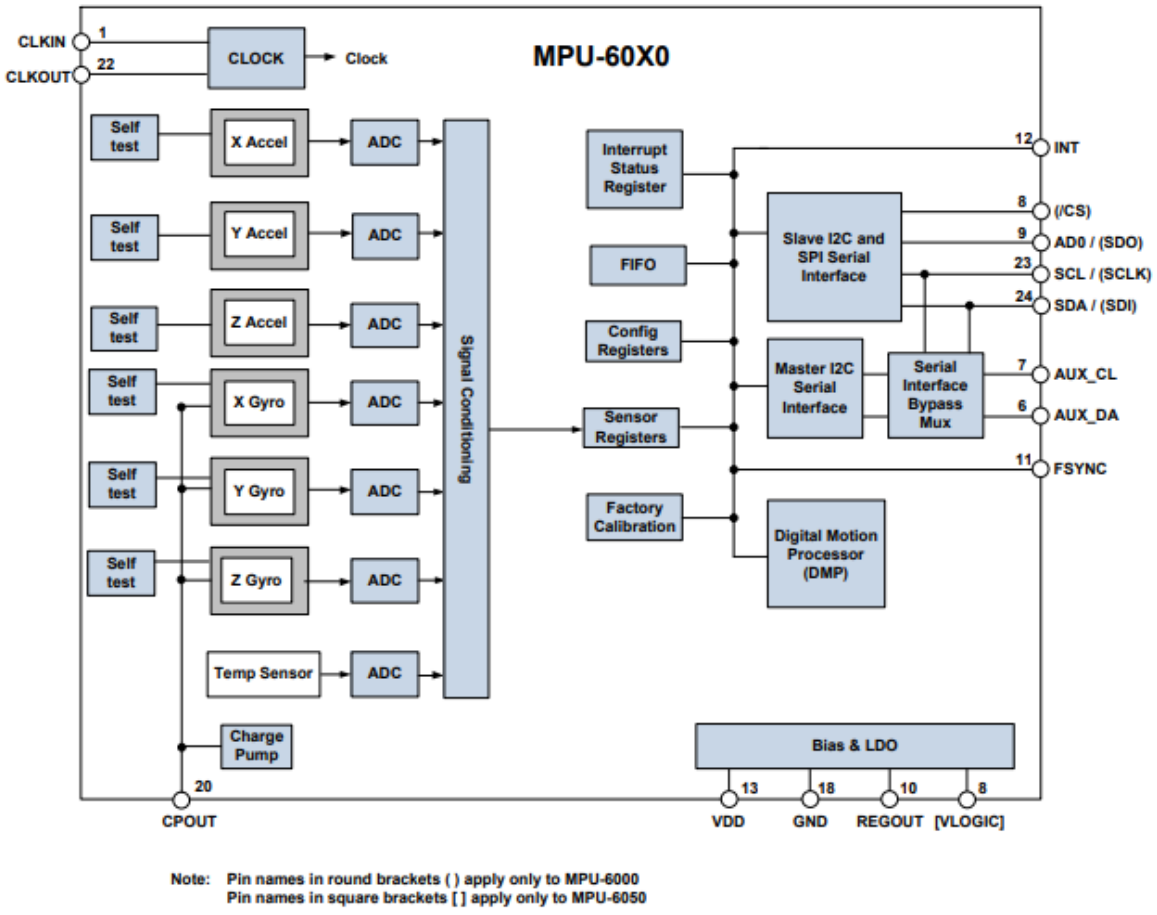


Figure 2.8: MPU6050 Internal Architecture (from official datasheet) [18]

- A simplified block diagram showing the sensor's core structure, including signal conditioning, ADCs, FIFO, and interrupt logic.

Parameters	Conditions	Min	Typical	Max	Units	Notes
I²C TIMING	**I²C FAST-MODE**					
f_{SCL}, SCL Clock Frequency				400	kHz	
$t_{HD.STA}$, (Repeated) START Condition Hold Time		0.6			µs	
t_{LOW}, SCL Low Period		1.3			µs	
t_{HIGH}, SCL High Period		0.6			µs	
$t_{SU.STA}$, Repeated START Condition Setup Time		0.6			µs	
$t_{HD.DAT}$, SDA Data Hold Time		0			µs	
$t_{SU.DAT}$, SDA Data Setup Time		100			ns	
t_r, SDA and SCL Rise Time	C_b bus cap. from 10 to 400pF	20+0.1C_b		300	ns	
t_f, SDA and SCL Fall Time	C_b bus cap. from 10 to 400pF	20+0.1C_b		300	ns	
$t_{SU.STO}$, STOP Condition Setup Time		0.6			µs	
t_{BUF}, Bus Free Time Between STOP and START Condition		1.3			µs	
C_b, Capacitive Load for each Bus Line			< 400		pF	
$t_{VD.DAT}$, Data Valid Time				0.9	µs	
$t_{VD.ACK}$, Data Valid Acknowledge Time				0.9	µs	

Table 2.3: I²C Timing Specification Table [18]

- Timing characteristics used to validate our communication reliability under 500 Hz operation.

These visual references supported not only the electrical stability of our integration but also helped us troubleshoot initial misconfigurations and guide our I²C-DMA optimization strategy.

2.5. Microcontroller Logic: STM32F103RB

At the heart of our embedded monitoring system is the STM32F103RBT6 microcontroller serves as the central processing unit of our system, selected for its balance between processing power, peripheral flexibility, and proven reliability in low-power real-time applications. This section highlights the configuration strategy used to adapt the microcontroller for vibration sensing, AI inference, and peripheral coordination.

2.5.1. Clock Configuration and Oscillator Selection

At the start of the project, we considered using the STM32's **internal** RC oscillator for simplicity. However, we encountered a critical limitation: **USB communication failed to initialize**. This was because the internal oscillator lacks the precision needed to generate the exact 48 MHz clock required by USB protocols.

To resolve this, we adopted an **external 8 MHz crystal oscillator**, enabling:

- Stable USB peripheral support
- PLL (Phase-Locked Loop) scaling of the system clock up to **72 MHz**
- Reliable and high-speed processing for AI inference and sensor coordination

This configuration was achieved through STM32CubeMX, where the external crystal was set as the HSE (High-Speed External) source, and the PLL was enabled to derive the 72 MHz system clock. The tool provided a clear visualization of how the frequencies were routed internally across the microcontroller's subsystems.

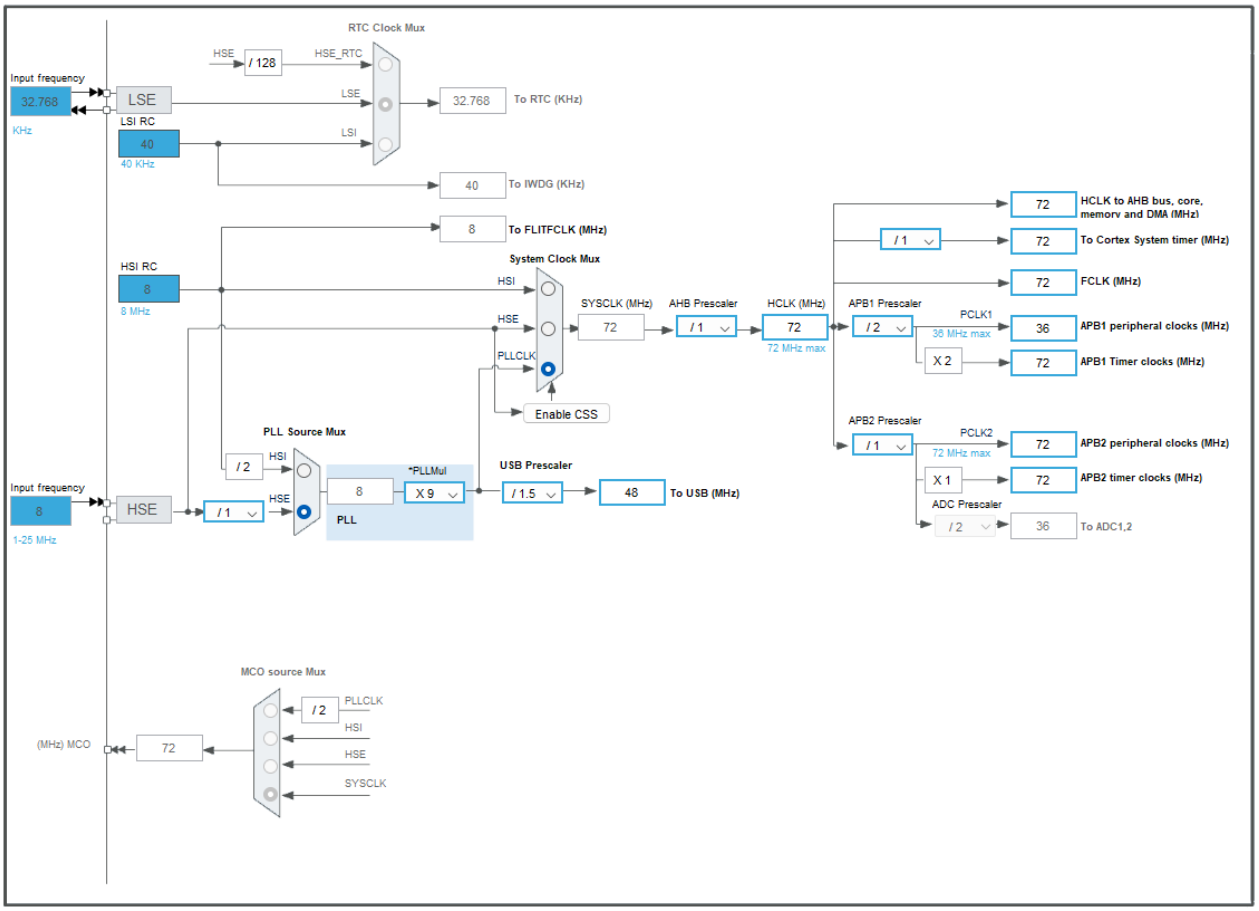


Figure 2.9: Clock configuration with external 8 MHz HSE source and PLL for 72 MHz system clock

A) High-speed external clock generated from a crystal/ceramic resonator

The high-speed external (HSE) clock can be supplied with **a 4 to 16 MHz** crystal/ceramic resonator oscillator. All these information given in this paragraph are based on characterization results obtained with typical external components. In the application, the **r**esonator and the load capacitors have to be placed as close as possible to the oscillator pins in order to minimize output

distortion and startup stabilization time. Refer to the crystal resonator manufacturer for more details on the resonator characteristics (frequency, package, accuracy.

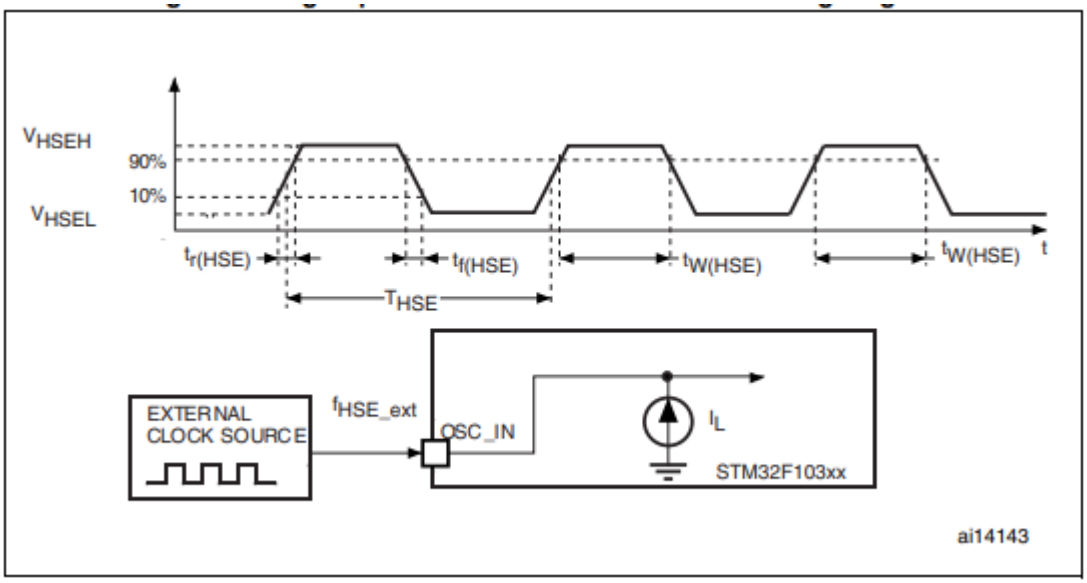


Figure 2.10: High-speed external clock source AC timing diagram [12]

2.5.2. Microcontroller Pin Configuration and Peripheral Mapping

the STM32F103RB microcontroller was configured with a carefully planned peripheral mapping. Given the limited number of communication interfaces, special attention was given to distributing tasks across I²C, USART, and GPIO resources without overlap.

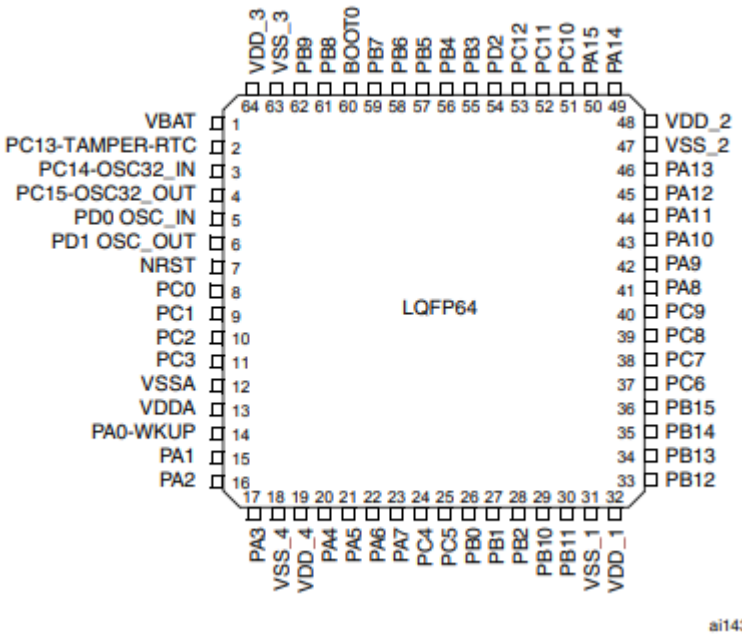


Figure 2.11: STM32F103xx performance line LQFP64 pinout [12]

Key configurations include:

- **I²C1 (PB6 – SCL2, PB7 – SDA1):** Connected to the MPU6050 motion sensor for continuous, high-speed vibration sampling.
- **I²C2 (PB10 – SCL1, PB11 – SDA1):** Dedicated to the SSD1306 OLED display to visualize the AI inference results in real time.
- **USART1 (PA9 - TX, PA10 - RX):** Redirected to interface with the CAN transceiver module, enabling transmission of AI results to a secondary board via the CAN bus.
- **USB D+ / D– (PA11, PA12):** Connected to a micro-USB port for power delivery and optional serial communication.
- **Boot configuration (BOOT0) and reset (NRST):** Managed using external pull-up/down circuits and a tactile push button.
- **PA1 (Push button input):** Assigned to trigger key states such as learning mode or system reset.
- **Power input (3.3V LDO regulated):** Clean power is supplied to the MCU and peripherals, stabilized by a bulk capacitor and six local decoupling capacitors.
- **8 MHz External Crystal Oscillator:** Used with PLL to generate a precise 72 MHz system clock and the required 48 MHz USB peripheral clock. This replaced the default internal RC oscillator, which proved insufficient for USB operation.

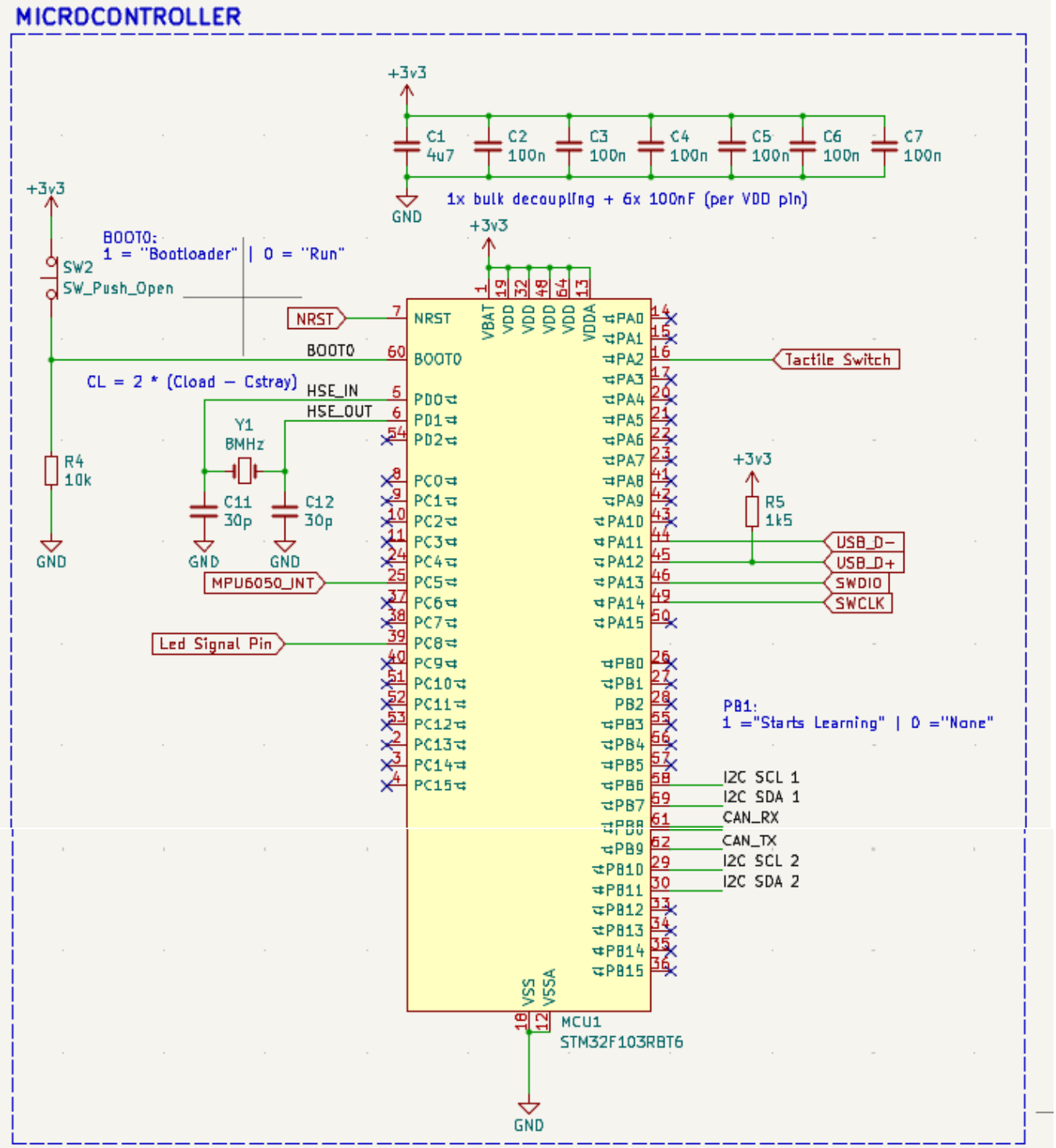


Figure 2.12: STM32F103RB Extracted Schematic: Peripheral and Communication Mapping

In addition to communication lines and logic pins, passive components around the STM32F103RB play a critical role in ensuring reliable operation, especially in vibration-prone or noise-sensitive environments.

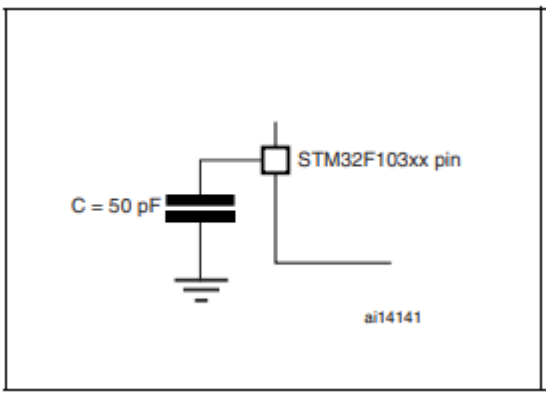


Figure 2.13 : Pin Loading conditions [12]

A) Decoupling Capacitors

To stabilize voltage and suppress transient spikes:

- **One bulk capacitor (4.7 µF)** is placed at the main VDD input to handle sudden current demands.
- **Six 100 nF ceramic capacitors** are placed as close as possible to the MCU's power pins. This proximity is essential to minimize inductive noise and provide local charge during high-speed switching events (as recommended by STM32 design guidelines).

These components help maintain clean 3.3V operation, reducing susceptibility to glitches during sensor acquisition or CAN transmission.

B) Boot Control and User Interaction

A **tactile push-button** (labeled SW_Push_Open) and **BOOT0 pull-down resistor (10kΩ)** form the bootloader control interface:

- When held high, the MCU enters **bootloader mode**—useful for flashing firmware without SWD tools.
- In normal operation, BOOT0 remains grounded via the resistor, booting into flash memory automatically.

A **second tactile push button** (labeled tactile switch) Connected to the NRST pin for full system restarts

2.5.3. Peripheral Coordination and Simplifications

Initially, we anticipated needing preprocessing like data normalization. However, NanoEdge AI Studio was able to handle raw accelerometer input directly. This allowed us to:

- Simplify the firmware logic

- Avoid extra RAM/CPU overhead
- Maintain precision by feeding unaltered sensor data into the model

2.6. NanoEdge AI Integration

As part of bringing machine intelligence into our embedded system, this section details how we used **NanoEdge AI Studio** to design, train, and deploy an AI model directly onto our microcontroller. The strength of this approach lies in its ability to run smart decision-making algorithms locally without needing cloud connectivity or heavy processing hardware.

2.6.1. Sensor data collection and preparation

Before anything else, the system needed real data to learn from. To collect this, we created a small custom program on the STM32 that continuously read values from the MPU6050 accelerometer. Since this data had to be sent to a PC for analysis and training, we also wrote a custom function to send the sensor readings through USART and COM PORT, so they could be displayed and sent via the **serial monitor** on the computer.

This allowed us to monitor how the values changed over time while the machine was in operation, and to export full sets of readings as **.**csv files. These files contained 3-axis motion data (X, Y, Z), sampled at **1000 Hz**, and were organized into fixed-length **buffers** of **256 samples**. Each buffer captured a small snapshot of motion just enough to represent short patterns that NanoEdge could learn from.[10]

We collected multiple datasets under different conditions:

- Normal operation (used as training input for anomaly detection)
- Artificially disturbed conditions (for testing or classification tasks)

These buffers were then imported into NanoEdge AI Studio via **serial monitor** for training.

2.6.2. Time-Series Buffer Design for Edge AI

To effectively teach the AI how our system behaves over time, we chose the time-series project format in NanoEdge AI Studio. This approach allowed us to work with **motion data** captured across small time windows**,** rather than isolated sensor readings. By structuring the accelerometer data

into fixed-length **temporal buffers**, the model could learn not just individual values, but how they evolve which is essential for detecting vibration-based anomalies in real-world conditions.

When selecting time series in the first step of a project in NanoEdge AI Studio, there is multiple sensors as options:

- **Accelerometer:** 1, 2, or 3 axes
- **Current:** 1 axis
- **Hall effect:** 1, 2, or 3 axes
- **Microphone:** 1 axis
- **Time-of-Flight:** 4x4 or 8x8 matrix
- **Generic:** As much axis as you are using (example: current and accelerometer 3 axes = 4 axes)

Because time series involves multiple samples in time, the data imported are expected to contain multiple samples of the number of axes selected.[10]

A) Work with temporal buffers

NanoEdge AI Studio expects buffers and not just samples of data.

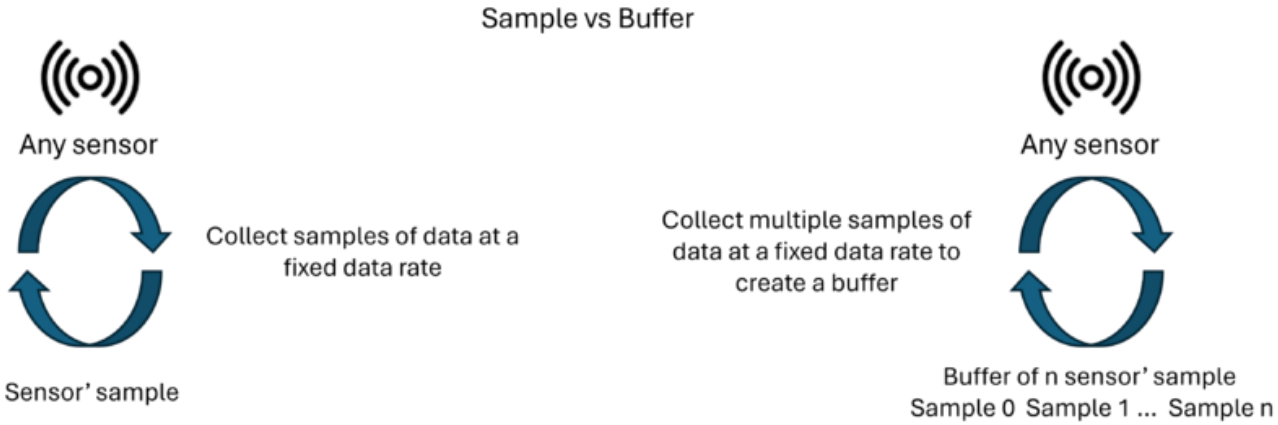


Figure 2.14: sample vs buffer

*Buffer = sensor' number of axes * number of samples taken at a fixed data rate (Hz)*

It is crucial to maintain a consistent sampling frequency and buffer size

In machine learning, working with buffers is advantageous because they contain information on the

evolution of the phenomenon studied. Samples are just a single point in time representing the phenomenon. It is easier to distinguish patterns in buffers than in simple values.

For example, consider two signals with identical data but reversed temporal order (t_o to t_n vs. t_n to t_o). These signals can be treated as two distinct classes. The task then becomes classifying a new signal by determining whether it matches one of these predefined sequences.

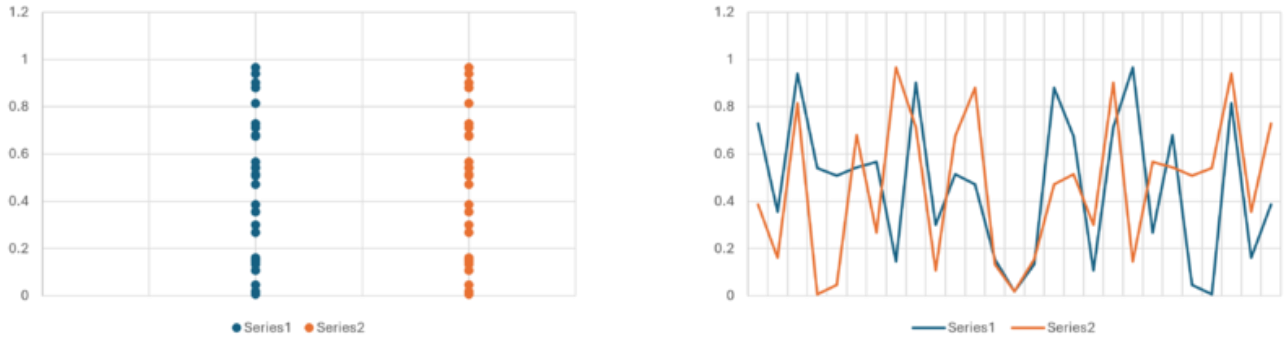


Figure 2.15: Difference between independent sample vs temporal buffer

- **Left plot:** Samples independently
- **Right plot:** Samples as temporal buffers

A new signal known to belong to one of two classes needs classification:

- Looking at individual samples (left approach) makes classification impossible since all values exist in both classes. A single sample lacks sufficient information for classification.
- Using the temporal buffer immediately reveals which class the signal matches through direct comparison.[10]

B) General rules

The Studio requires:

- Each line in a dataset must represent a single, independent signal example composed of multiple samples.
- The buffer size to be a power of two and remain constant throughout the project.
- The sampling frequency to remain constant throughout the project.

- All signal examples for a specific "class" to be grouped in the same input file (for example, all "nominal" regimes in one file and all "abnormal" regimes in another for anomaly detection).

General considerations for input file format:

- .txt / .csv files
- Numerical values only, no headers
- Uniform separators (single space, tab, comma, or semicolon)
- Decimal values formatted with a period (.)
- Fewer than 16,384 values per line
- Consistent number of numerical values on each line
- Minimum of 20 lines per sensor axis
- Fewer than ~100,000 total lines
- File size less than ~1 Gbit

Specific formatting rules:

- Anomaly detection, 1-class classification, and n-class classification projects share general rules.
- Extrapolation projects require target values for extrapolation [10]

C) Basic format

Following the rules from the previous part, the format for **anomaly detection (AD), N-class classification (NCC)** or **1-class classification (1CC)** is the same and is described in the example below:
the system uses a 3-axis accelerometer (MPU6050) to capture motion patterns. Each buffer stores a fixed number of readings for the X, Y, and Z axes. This structure enables NanoEdge AI to analyze short bursts of motion and learn the characteristic behavior of the monitored equipment.
The format used for **anomaly detection, classification, and outlier detection** libraries in NanoEdge AI Studio is:

- **Number of axes**: 3 (X, Y, Z)

- **Number of samples per buffer**: *n*
- **Buffer size**: 3 × *n*
- **Sampling frequency**: depends on the application's precision needs (e.g., 500 Hz)
- **Temporal length**: *n* samples / data rate (in Hz) [10]

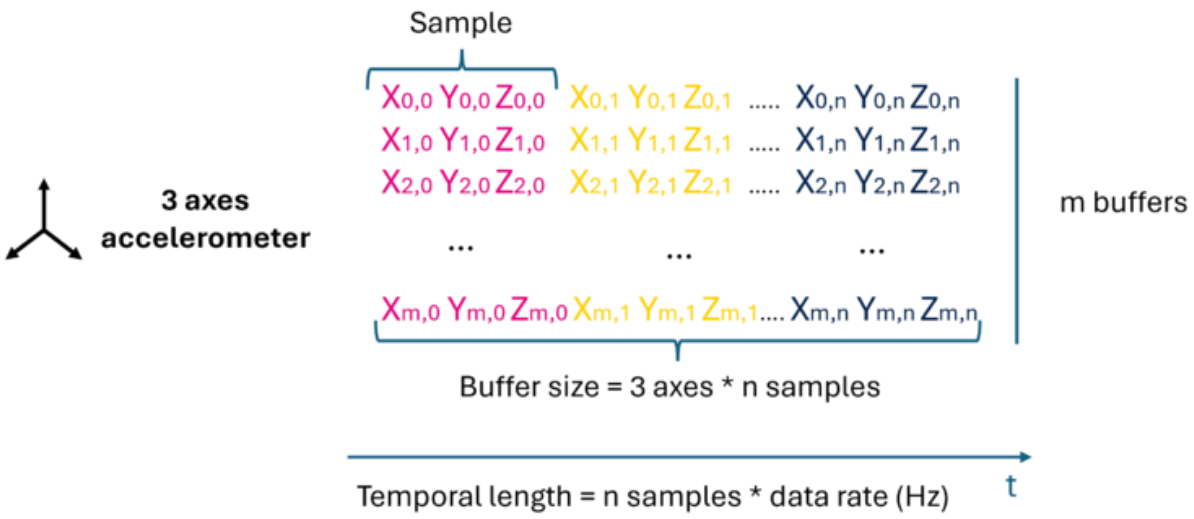


Figure 2.16: Basic format for (AD, NCC, 1CC)

For instance, if the sampling frequency is **500 Hz** and the buffer contains **200 samples**, the temporal length of the signal segment is:

Temporal length=200 samples500 Hz=0.4 seconds

This means the model analyzes 0.4 seconds of motion data at each inference.

D) Variant format: Extrapolation projects

In extrapolation projects, NanoEdge AI requires a modified buffer format. In addition to sensor data, each buffer includes a target value the output the AI should learn to predict.

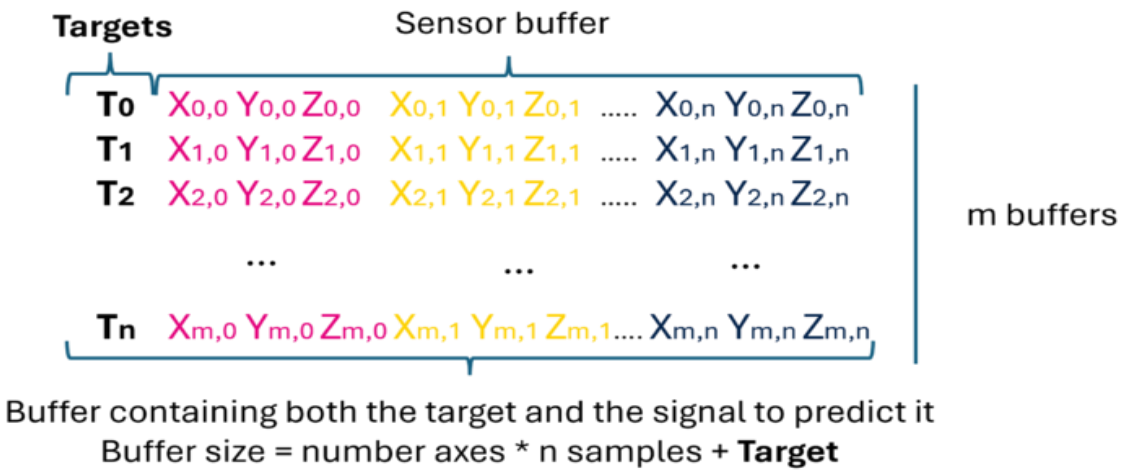


Figure 2.17: Basic format for Extrapolation

For example, to predict the magnitude of vibration or another numerical value based on accelerometer readings, each buffer contains:

- A leading column for the target value
- Followed by the same 3-axis × *n* samples format

Buffer size=3×n+1 (target)

This target could represent vibration amplitude, tilt angle, or any measurable value correlated with motion. During inference, only the sensor buffer is provided, and the AI outputs the predicted target. [10]

2.6.3. Other Algorithm Types Explored

To take full advantage of NanoEdge AI Studio's versatility, we explored its core algorithm types like classification (N-Class), anomaly detection, outlier detection, and extrapolation. All of these modes are supported by our system and can be implemented with the same underlying architecture. In fact, NanoEdge enables four fundamental AI capabilities for embedded devices:

- **Learning signal patterns**: the system observes normal machine behavior and builds internal models without the need for labeled data.

- Detecting anomalies: it identifies when incoming signals differ from the learned baseline, flagging potential faults or unusual events.

- Classifying signals: it recognizes which of several predefined states the system is currently in

(e.g., idle, vibration, movement).

- Extrapolating data: it can predict future signal trends based on observed patterns useful for forecasting or preventive maintenance.

While all of these capabilities were technically viable on our hardware, **anomaly detection** was selected as the primary mode for this thesis due to its simplicity, clarity of output, and strong alignment with real-world industrial diagnostics. Unlike classification or extrapolation, it required no labeled datasets or future prediction modeling just a clear sense of **"normal"** behavior and the ability **to detect deviation**.

This made it easier to validate, explain, and visualize in both laboratory testing and real-time embedded operation. Nevertheless, **as shown in Chapter 3**, we successfully implemented and **tested N-Class classification with the same system**, confirming that our platform remains flexible and ready for expansion into more advanced AI functionalities in future iterations

2.6.4. Model Deployment on STM32

Once the training phase was done, NanoEdge AI Studio gave us a ready-made C library containing the trained model. We brought this library into our project using STM32CubeIDE, where it was compiled alongside the rest of our firmware.

After flashing the program to the STM32's internal memory, the model became a built-in part of the device no need for any external server or cloud connection. Everything runs directly on the microcontroller.

During operation, the STM32 collects data from the sensor into a **buffer of 256 samples**. When the buffer is full, it sends that data to the NanoEdge AI function. The function then quickly analyzes the signal and gives back an **anomaly score** that tells us how "normal" or "suspicious" the current behavior is.

What makes this setup great is how lightweight and efficient NanoEdge AI is, it fits perfectly on the STM32, runs fast, and uses very little memory or power. That's exactly what we needed for our edge device.

2.7. Display Layer: SSD1306 OLED

Displaying real-time AI inference was not just useful for debugging but also crucial for providing intuitive, immediate feedback to the user. To achieve this, we integrated a compact SSD1306-based OLED display (128×64 pixels), driven over I²C communication, into the system. The display offers a minimal yet effective visual interface for monitoring machine behavior.

2.7.1. Display Connection and Communication

The SSD1306 OLED module was interfaced with the STM32F103RB using the **I²C protocol**, connected as follows:

SSD1306 Pin	STM32F103RB pin	Description
VCC	**3.3V**	**Powe Supply**
GND	**GND**	**Ground**
SDA 1	**PB7**	**I²C Data Line**
SCL 1	**PB6**	**I²C Clock Line**

Table 2.4: I²C Connection for OLED

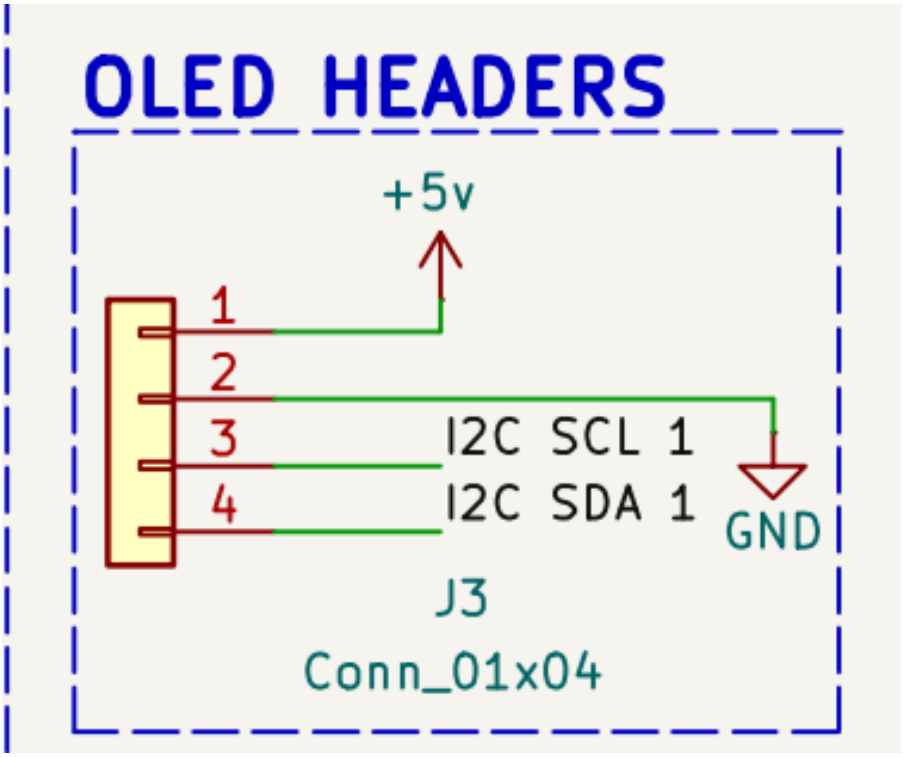


Figure 2.18: **Extracted view of the OLED to STM32 I²C wiring from the full system Schematic.**

Despite I²C being slower than SPI, the system achieved a solid frame rate after Fast mode (discussed earlier). Before that, the refresh rate was ~**4–5 FPS**. After implementing the optimization display code, we reached up to **24 FPS**, making it visually responsive in real time.

In addition to OLED feedback, a CAN-based web interface was also added to expand monitoring options (detailed in Chapter 3).

2.7.2. Display Modes: Eco vs Fast Refresh

To balance power consumption and performance, we created two display modes:

- **Fast Update Mode (Unstable State):**
  If the AI detects an unstable or unknown condition, the display refreshes quickly (~30 FPS), giving real-time feedback and alerting the user immediately.
- **Eco Mode (Stable State):**
  If the system is in a known, stable condition, the screen refresh rate is lowered to ~1 FPS. This reduces CPU usage and saves power, especially useful in embedded or battery-powered deployments.

Mode switching is handled dynamically based on AI output, using a lightweight condition check in the main loop**.**

2.7.3. Visual Layout Design

Rather than crowding the display, the screen was intentionally kept clean and glanceable. Two main elements were used to communicate machine health:

A) Score

The OLED screen shows a real-time numerical score (0–100) generated by the NanoEdge AI model, representing the system's operational health:

- **100%** = completely normal operation
- **0%** = highly anomalous behavior detected

This Score gives immediate visual feedback about the system's health without needing to interpret numerical scores.

B) Motion Line Graph

Alongside the score bar, we added a simple waveform-style line graph that represents the real-time behavior of the score:

- When no anomaly is detected, **the line remains flat** indicating stable operation.
- When irregularities are present, **the line fluctuates** offering a more visual cue of instability.

This subtle movement makes it easier to **spot transient faults** or brief disturbances.

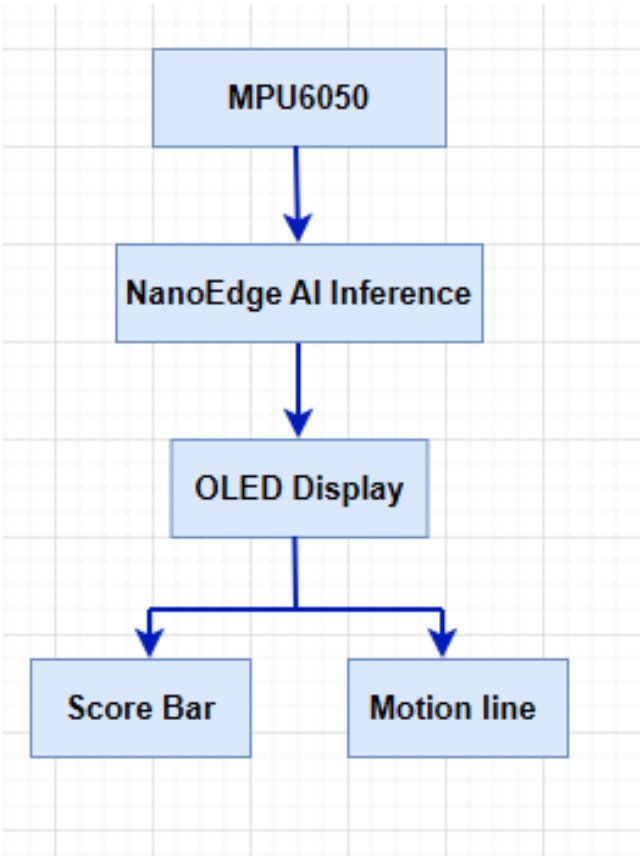


Figure 2.19: Display Logic Flow

- in chapter 3 we will visualize real testing results captured from the system and capture the score bar and motion line graph

2.8. Auxiliary Components in the System Schematic

While the core of the system revolves around sensing, inference, and display, several additional components were added to ensure interfacing, usability, and power stability. These supporting elements though often overlooked, played a critical role in enabling real-world deployment and robust operation.

2.8.1. USB Interface and 3.3V LDO Regulator

A micro-USB port was integrated into the main PCB for both power input and serial communication. To regulate the 5V USB input down to the required 3.3V level, an LDO voltage regulator was included. This ensured stable power for the STM32 and all peripherals, with decoupling capacitors placed close to the regulator and microcontroller to reduce ripple and transient noise.

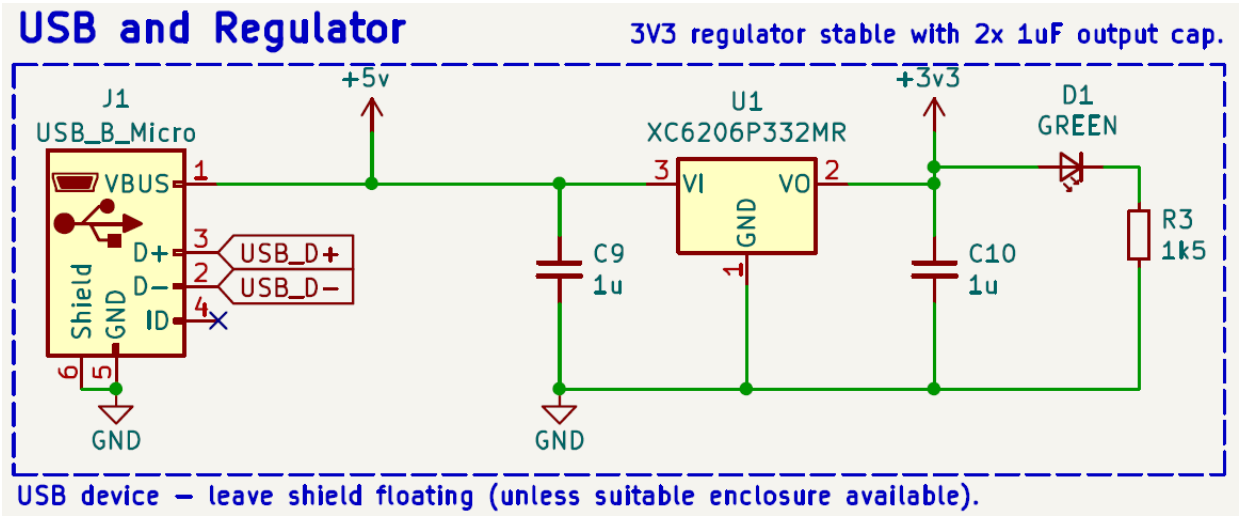


Figure 2.20: USB and Regulator Extracted Schematic

2.8.2. Addressable-RGB LED

To provide instant visual feedback on the system's health status, an A_RGB LED was integrated into the main PCB design. This component allows for a dynamic and color-based representation of the **anomaly score output** from the NanoEdge AI inference.

Color States :

- **Green**: Stable condition → the system detects no irregularities
- **Red**: Critical anomaly → the behavior is significantly abnormal

- **Blue**: Idle state → the device is powered but no sensor data is currently being processed

This **color gradient** from green to red offers a more **intuitive and scalable alerting mechanism**, especially in environments where the OLED screen may not be visible or accessible. It also allows technicians or operators to observe machine behavior at a glance, even from a distance.

The LED is controlled directly via a **GPIO pin on the STM32**, using a simple PWM-based color mapping logic derived from the AI's output score. In future versions, this could be extended to include blinking patterns for fault type or data logging indicators.

A-RGB LED Pin	STM32F103RB pin	Description
VDD	**3.3V**	**Powe Supply**
GND	**GND**	**Ground**
DIN	**PC8**	**Control Data Signal input**
DOUT	**/**	**Control Data Signal output**

Table 2.5: pin configuration of the LED

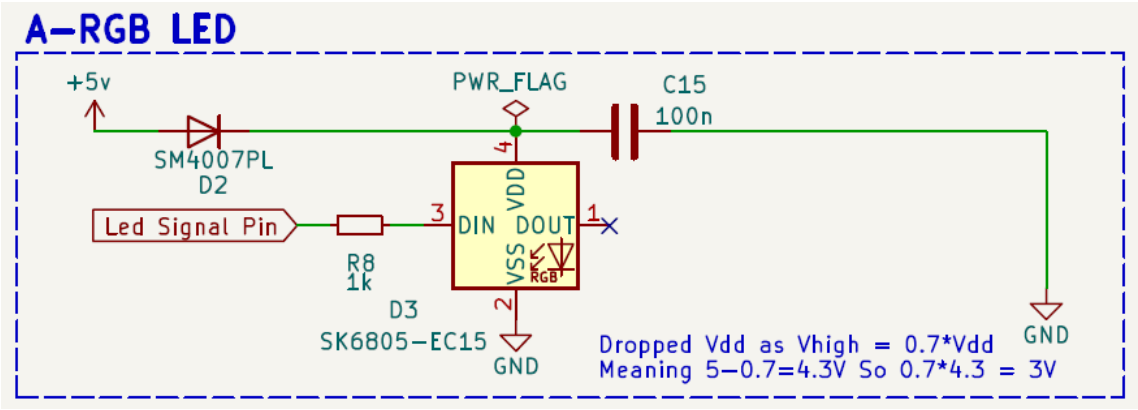


Figure 2.21: A-RGB LED Extracted Schematic

2.8.3. Tactile Switch (Boot/Reset)

A momentary push-button switch was added and routed to the BOOT or RESET pin **PA2** of the STM32F103RB. This allowed for:

- Manual entry into bootloader mode
- Triggering resets during debugging or firmware flashing

This addition streamlined development and avoided the need for external ST-Link-based resetting in some stages.

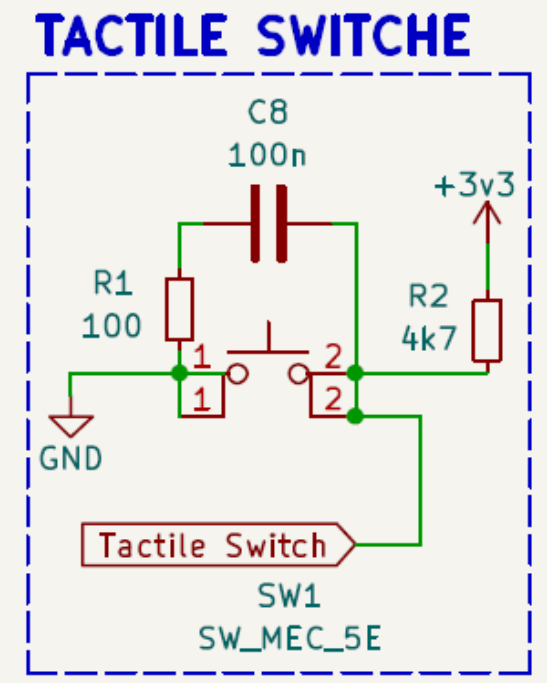


Figure 2.22: Tactile Switch Extracted Schematic

2.8.4. I/O Connectors and Expansion Headers

These connectors allow for easy interfacing with expansion boards and modular testing setups. All were placed considering mechanical accessibility and signal integrity.

Connector	STM32F103RB pin	Description
TP1	**3.3V**	**Powe Supply**
TP2	**PA14**	**SWCLK (Serial Wire Clock)**
TP3	**GND**	**Ground**
TP4	**PA13**	**SWDIO (Serial Wire Data I/O)**
TP5	**NRST**	**Reset**
TP6	**5V**	**Powe Supply**

Table 2.6: pin configuration for the Tactile Switch

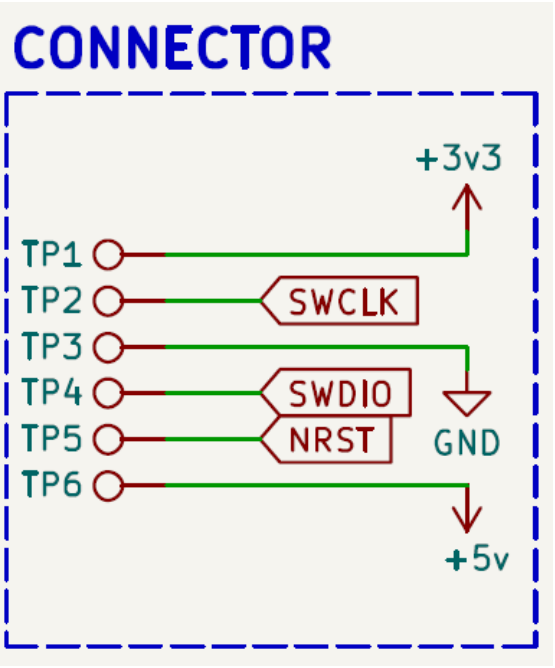


Figure 2.23: Connector Extracted Schematic

2.8.5. CAN BUS

To enable robust communication between the STM32F103RB and external modules—particularly for transmitting AI inference results a CAN bus interface was integrated into the main PCB.

The CAN lines (CAN_TX and CAN_RX) are routed to a dedicated header connected to a separate board that includes a CAN transceiver and an ESP32 module **(detailed in the next section).**

- **Reliable long-distance communication** using differential signaling
- **Noise resistance** suitable for industrial environments
- **Scalability,** allowing multiple nodes to be added to the bus

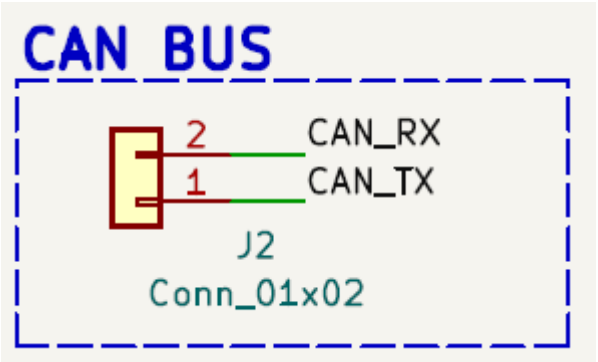


Figure 2.24: CAN BUS Extracted Schematic

2.9. Full System Schematic

After detailing each subsystem (sensor, display, power regulation, communication, and microcontroller core), we now present the complete schematic of the STM32-based main board as a unified view.

This schematic consolidates all the functional blocks that were individually explored—ranging from the MPU6050 interface and SSD1306 OLED connections, to power regulation circuits, boot/reset logic, and the CAN communication header.

Each major region of the schematic corresponds to one of the following functional blocks:

- **Core Logic Unit: STM32F103RB** microcontroller and related capacitors, crystal oscillator, SWD/debug header.
- **Sensor Interface: MPU6050** with I^2C connection, decoupling capacitor, and optional INT line.
- **Display Output: SSD1306 OLED** wiring with dedicated I^2C lines and optional power enable pin.
- **Power Management:** 3.3V LDO regulator, USB input connector, filtering capacitors**.**
- **Status Indicators & I/O:** A-RGB LED, tactile boot/reset switch.
- **CAN Bus Output Header:** Connected to TX/RX for STM32-to-ESP32 data transfer.

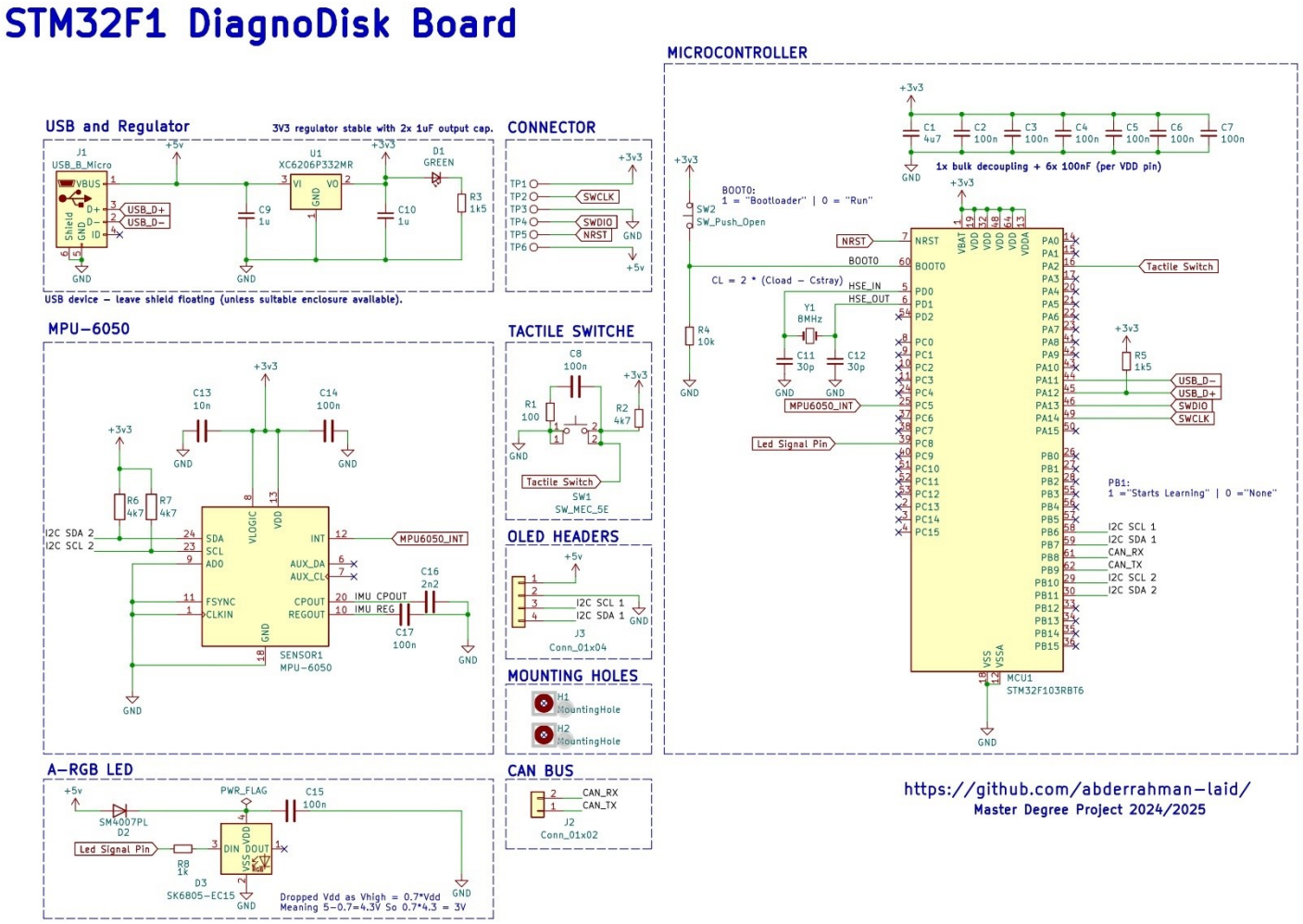


Figure 2.25: Full Schematic of the STM32 Main Board

This schematic guided the final PCB layout process, ensuring electrical correctness and manufacturability while preserving modularity for future upgrades or diagnostics.

2.10. Output Interface Extension: ESP32 + CAN Bus Architecture

As the system matured, a local OLED display alone proved insufficient for scalable monitoring, especially in real-world industrial environments where physical access to the device might be limited. To address this, a **dedicated remote monitoring module** was designed and integrated into the architecture, allowing the AI-generated results to be transmitted wirelessly over Wi-Fi and visualized on a web dashboard.

2.10.1. Why a Remote Output Module Was Needed

While the SSD1306 OLED is ideal for real-time debugging or small-scale diagnostics, it is limited in visibility, display area, and physical range. The goal of this new module was to:

- **Extend the system's usability** by providing wireless access to results.
- **Offload communication tasks** from the STM32, preserving compute power for AI inference.
- **Enable remote logging and visualization** of vibration patterns and anomaly scores.

2.10.2. Schematic Breakdown of the ESP32-CAN Module

The secondary PCB was kept simple and modular, consisting of:

- Three CAN bus transceivers
- Dual 8-pin female headers to mount the ESP32 board directly onto the PCB
- Minimal passive components and power distribution circuitry

Below is a detailed explanation of each section in the schematic.

A) CAN BUS Transceivers

The board includes **three CAN transceivers**, each playing a distinct role:

- **Master CAN Transceiver (CAN1):**
  This module is responsible for receiving AI inference results from the STM32 (main board) over the CAN bus. It handles the decoded data and passes it to the ESP32 for further processing.
- **Slave CAN Transceivers (CAN2 and CAN3):**
  These two transceivers are **connected to an independent CAN bus**, allowing the module to forward or route data as needed. This setup supports future system expansions, such as:
  Monitoring multiple STM32 boards

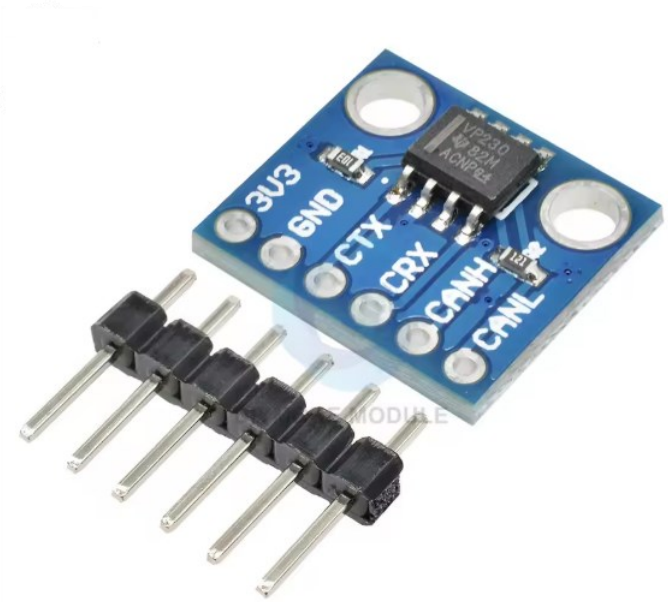


Figure 2.26: CAN bus transceivers Communication module

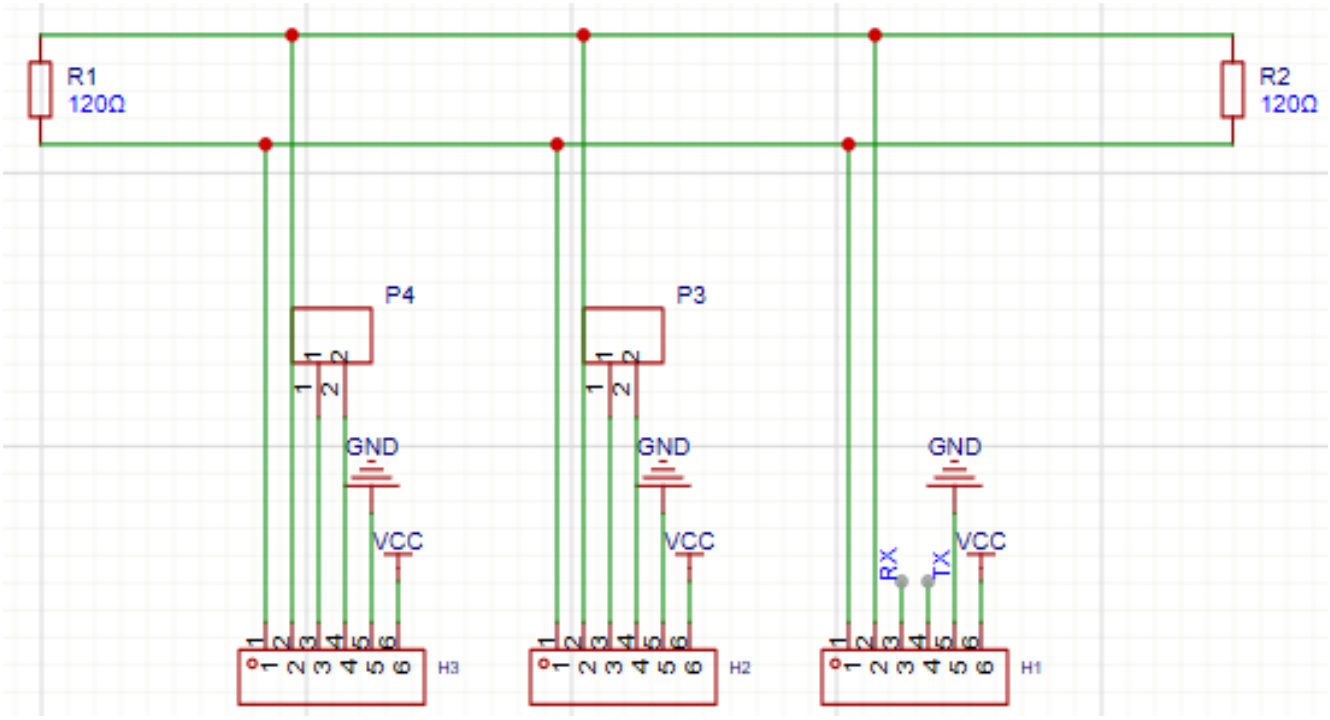


Figure 2.27: The 3 CAN bus transceivers Extracted Schematic

B) ESP32 Integration via Stacked Headers

The ESP32-C3 Mini is interfaced using dual 8-pin female headers, allowing:

- Easy module insertion/removal
- Seamless firmware flashing or replacement
- Clear signal routing without soldered joints

The headers are wired for:

- **UART** (to receive CAN output)
- **Power (3.3V) and GND**
- Optional GPIOs for LEDs or diagnostics

This modular approach makes the board adaptable and easy to maintain.

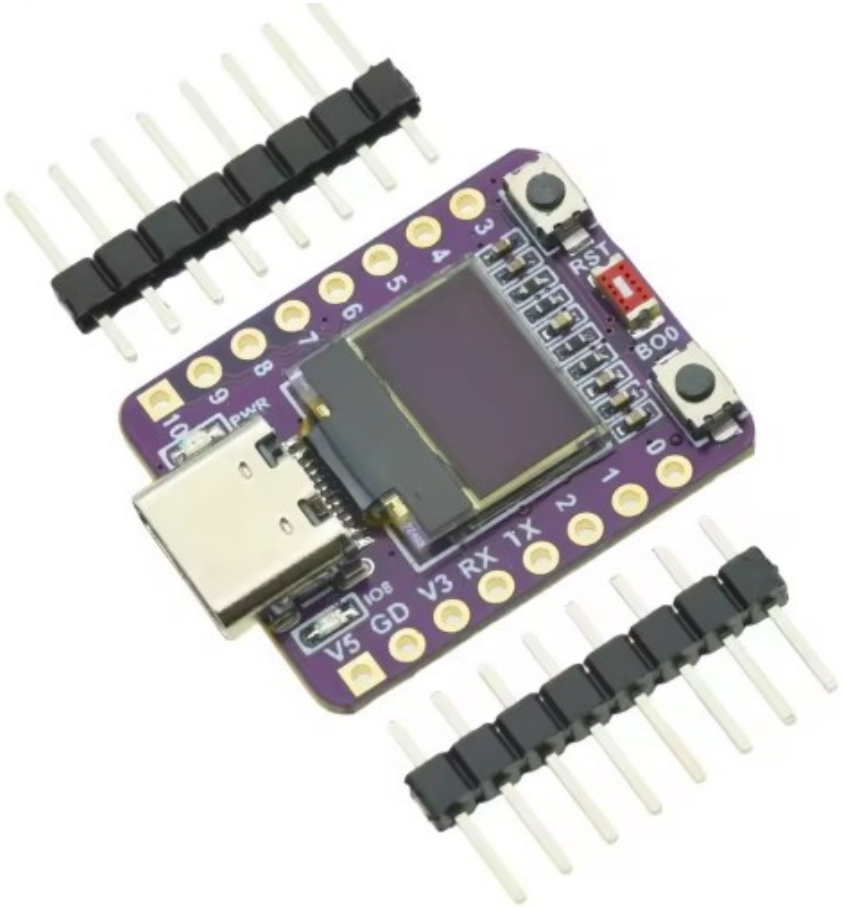


Figure 2.28: ESP32-C3

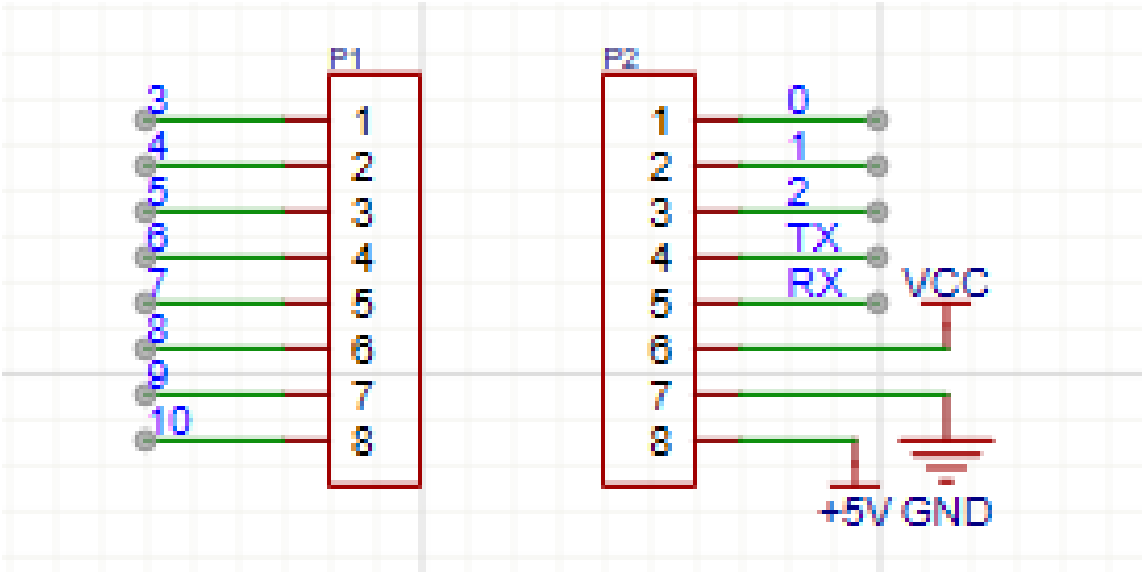


Figure 2.29: Dual 8-pin connectors Extracted Schematic

2.10.3. Full Schematic of the ESP32-CAN Monitoring Board

A complete schematic showing all three CAN transceivers, the ESP32 headers, and key supporting circuitry, including power and signal routing.

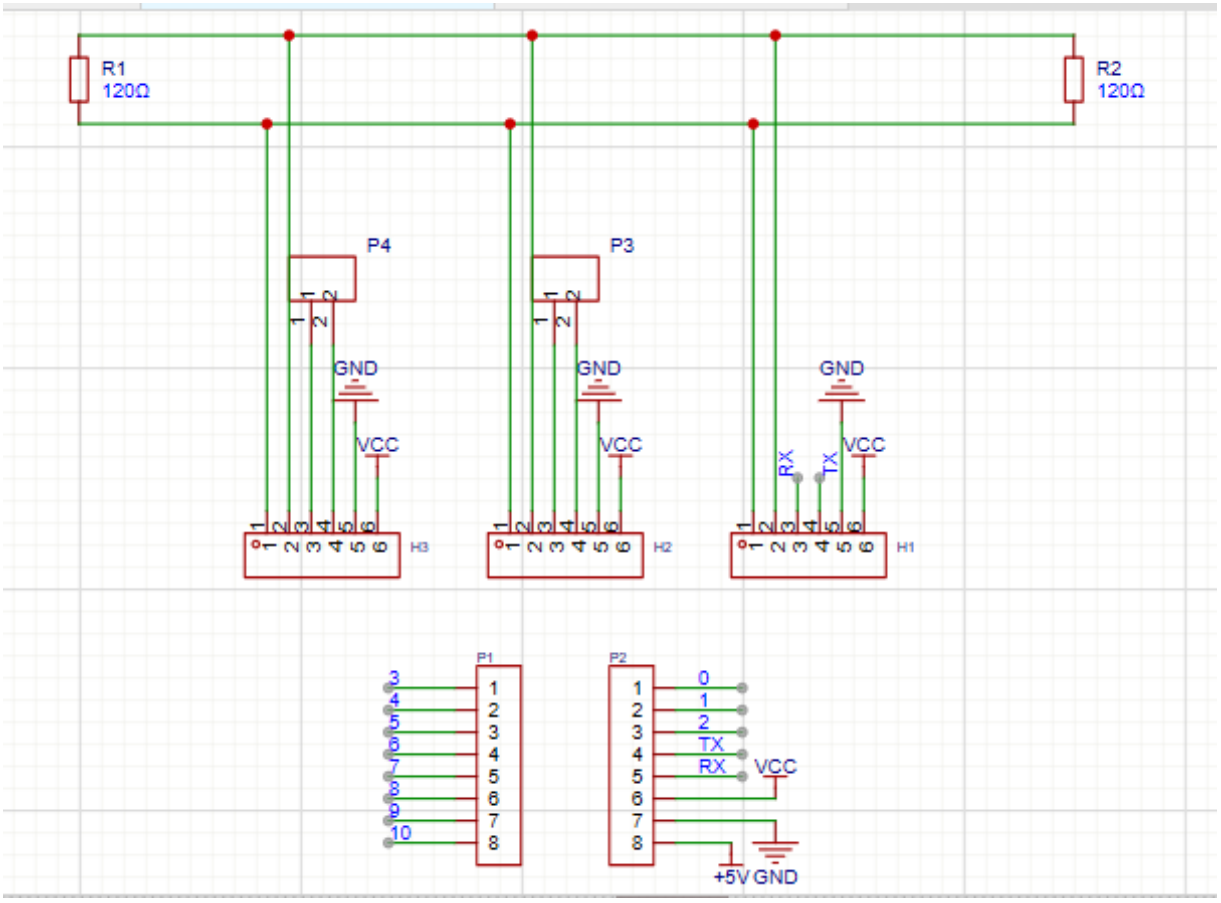


Figure 2.30: ESP32-CAN Monitoring Board Schematic

2.10.4. System Role and Data Flow

The secondary PCB acts as a wireless transmission node, receiving AI-generated results from the STM32 board and publishing them through a web interface.

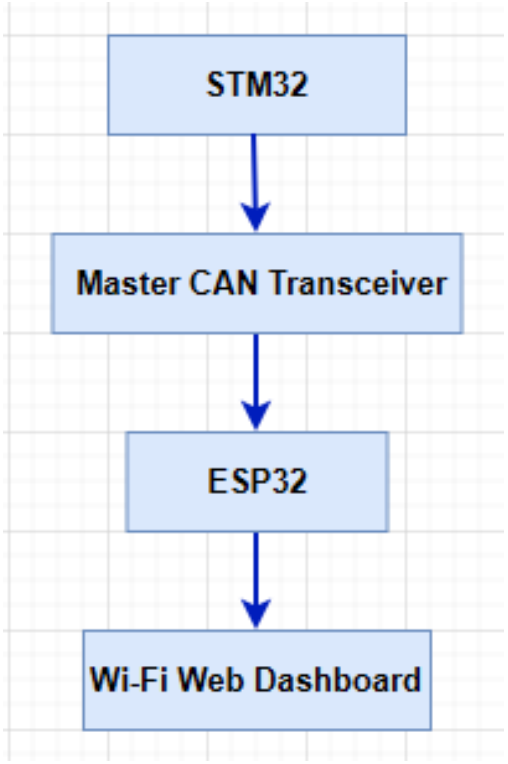


Figure 2.31: ESP32-CAN Monitoring Board Data Flow

Meanwhile, **Slave CAN Transceivers** remain connected to an external CAN bus, opening pathways for:

- Additional machine state inputs
- Relay of CAN messages across distributed modules
- Diagnostic or mirrored outputs to other monitoring systems

2.11. Conclusion

This chapter laid the foundation for the embedded monitoring solution presented in this work, we defined the architecture that brings our embedded AI system to life. Every decision from sensor sampling strategies and power-efficient microcontroller configuration to the modular design of a secondary CAN+ESP32 board was guided by performance needs, real-time constraints, and practical deployment considerations.

By combining classic embedded design principles with modern AI integration at the edge, the result is a system that can not only process and classify machine behavior locally, but also communicate those results wirelessly and intelligently.

Now that the blueprint is in place, the next chapter will show how each of these design elements was brought into the real world through circuit fabrication, firmware implementation, model testing, and full system validation.

Chapter 3

Implementation, Testing & Results

3.1. introduction

After planning, choosing the right tools, and designing each part of the system, it was time to bring everything to life.

This chapter focuses on the real-world realization of the project turning schematics into a physical PCB, turning code into working firmware, and turning raw sensor data into intelligent decisions through AI models. It walks through how the system was built, tested, and proven to function under real conditions.

We begin with the tools that made it all possible: STM32CubeIDE for writing and debugging firmware, KiCad for designing a multi-layer PCB, and NanoEdge AI Studio for generating AI models. From there, we show how the system was gradually assembled starting with prototypes and ending in a compact, professional-grade 6-layer PCB.

Next comes the firmware: how we collected sensor data using DMA, ran AI inference with NanoEdge libraries, and gave visual feedback through an OLED screen. You'll also see how we tested the system in various machine states — such as stable, vibrating, or disturbed — and measured how well the AI responded.
To make the system even more usable in practical settings, a second, independent PCB was created: a communication bridge based on CAN bus and ESP32**,** which receives AI scores from the main board and displays them wirelessly on a web interface. This addition showcases the system's ability to operate not only locally, but also as part of a connected industrial ecosystem.

Finally, the chapter presents testing scenarios and results: real machine-state simulations, OLED and web-based output comparisons, AI model benchmarking**,** and **system performance metrics** such as latency and power usage. Together, these demonstrate the reliability, intelligence, and real-world readiness of the full solution.

3.2. Development Environment

The development environment for this project brought together several tools, each playing a key role in turning ideas into a working system. From writing embedded code to designing the custom PCB and building the AI model, the workflow was built using accessible, open-source, or

manufacturer-supported tools.

3.2.1. STM32CubeIDE Firmware Development & Debugging

To bring our embedded system to life, we relied on STM32CubeIDE a powerful, all-in-one development environment offered by STMicroelectronics. It provided all the essential tools for firmware development, from configuration to debugging

STM32CubeIDE is built on the Eclipse®/CDT™ framework, using the GCC toolchain for compilation and **GDB** for debugging. It integrates STM32CubeMX, which allowed us to visually configure the STM32F103RB's peripherals (I^2C, USART, DMA, etc.) and generate clean initialization code. The IDE supports switching back to the peripheral configuration at any point, which was helpful when we needed to tweak DMA or clock settings without touching our core application code.

Key Features Utilized in This Project

- **Graphical pin configuration** for I^2C, USART, and GPIOs
- **System clock tuning** to scale from 8 MHz (HSE) up to 72 MHz
- **Memory and stack usage analyzer** to optimize AI integration
- **Live debugging tools**, including register views and fault analysis

This IDE was not just a convenience, it saved time, minimized mistakes, and gave us insight into both low-level hardware and high-level firmware.[19]

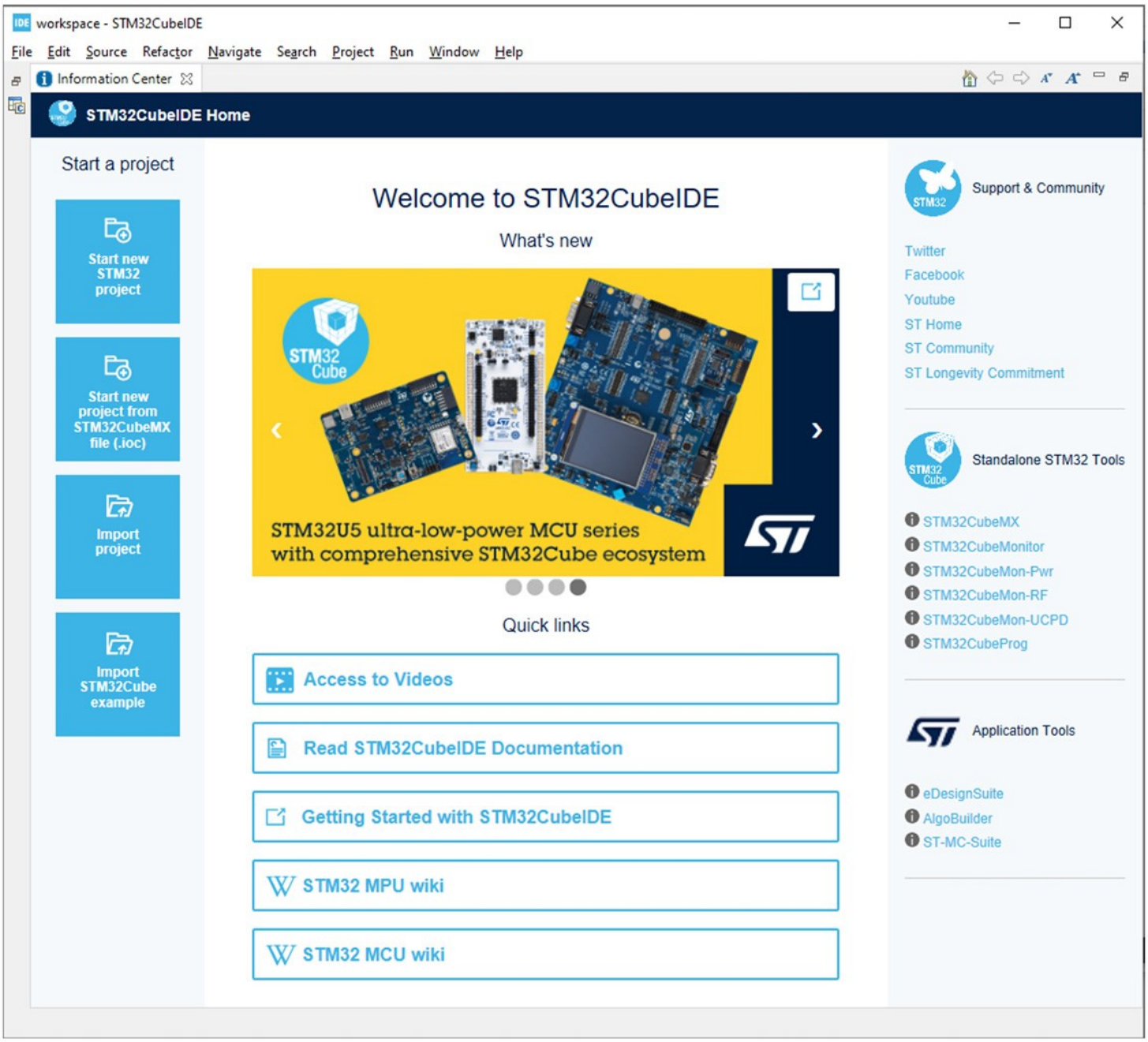


Figure 3.1: STM32Cube IDE Workspace

3.2.2. KiCad: Hardware Design from Schematic to Final Board

To design and fabricate a compact, reliable, and customized PCB for our project, we used **KiCad**, a powerful and open-source electronic design automation (EDA) tool suite. KiCad enabled us to perform schematic capture, PCB layout, 3D visualization, and Gerber file generation, all essential

steps in transitioning from prototype to production-ready hardware.

KiCad is a free and open-source electronic design automation (EDA) suite. It features schematic capture, integrated circuit simulation, printed circuit board (PCB) layout, 3D rendering, and plotting/data export to numerous formats. KiCad also includes a high-quality component library featuring thousands of symbols, footprints, and 3D models. KiCad has minimal system requirements and runs on Linux, Windows, and macOS**.[20]**

Key features utilized:

-Schematic Editor for circuit design, including I²C communication paths, power rails, and connectors

-PCB Layout Editor for trace routing, layer configuration, and compact board sizing

-D**esign Rule Check (DRC)** to verify spacing and electrical constraints

-3D Viewer for visual inspection and clearance checks before fabrication

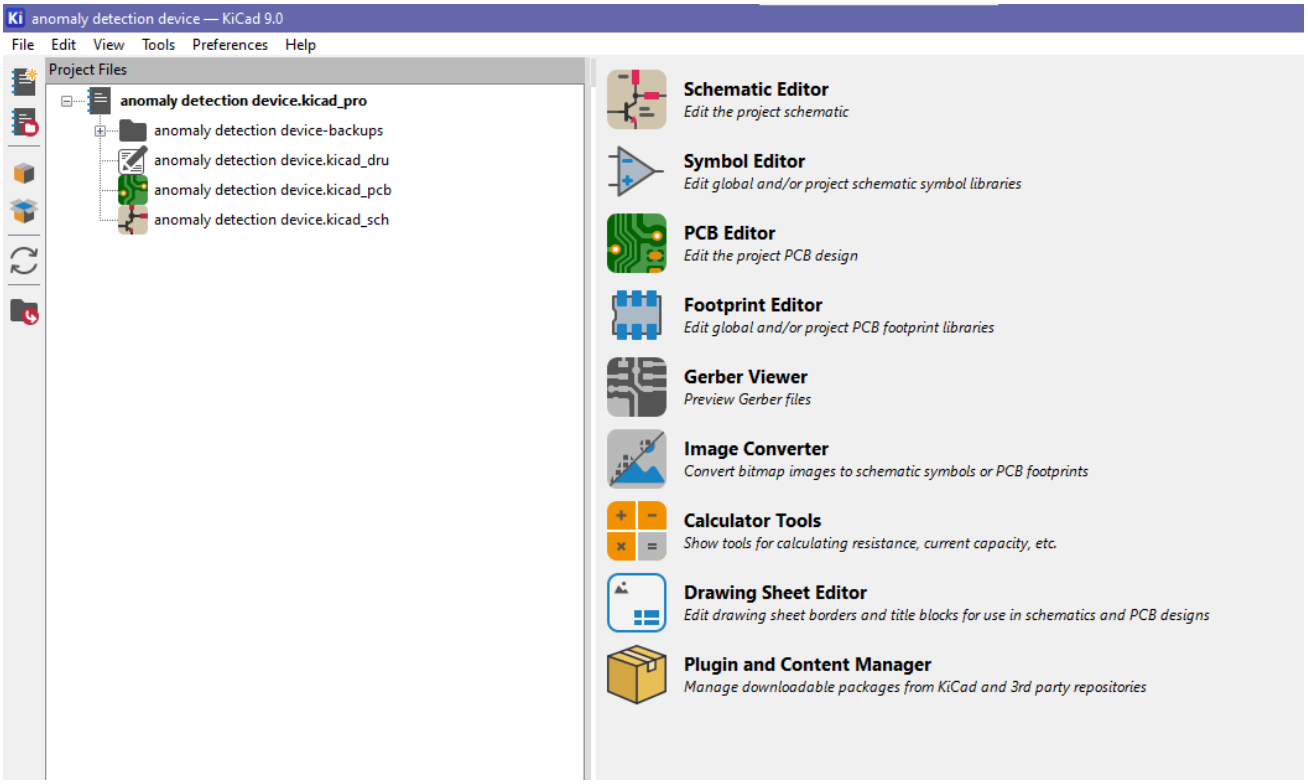


Figure 3.2: KiCad PCB Design Workspace

3.2.3. NanoEdge AI Studio: Embedded AI Training & Deployment

To bring intelligence into our system, we used **NanoEdge AI Studio**, an intuitive development tool by STMicroelectronics for creating machine learning models optimized for STM32

microcontrollers. NanoEdge AI Studio allowed us to collect motion data, train models, test various AI algorithms (e.g., anomaly detection, classification), and export a lightweight inference library.

The tool required no prior knowledge of AI or data science, making it ideal for embedded system developers. It supported buffer-based input, FFT preprocessing, and real-time benchmarking to choose the best-performing model.

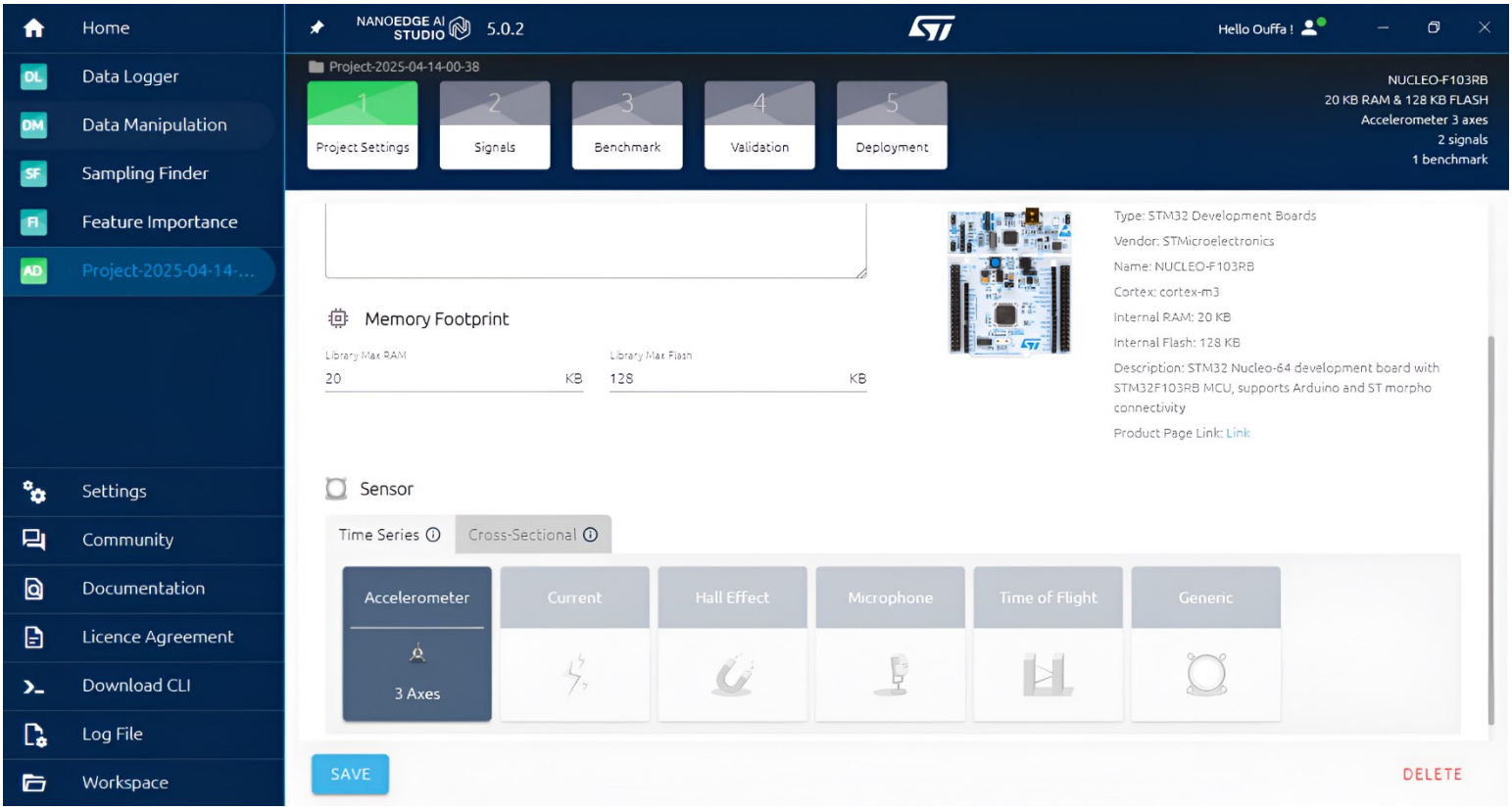


Figure 3.3: NanoEdge AI Studio interface showing the project

3.2.4. PCB Manufacturing and Assembly

After finalizing our PCB design in KiCad, we used **JLCPCB**, a leading low-cost online PCB manufacturing service, to fabricate the board. JLCPCB was selected due to its support for KiCad-generated Gerber files, fast turnaround times, and the ability to produce multilayer boards with high precision.

We exported our Gerber files directly from KiCad and uploaded them to the JLCPCB platform, which validated the design and provided instant feedback. Within days, we received a professionally fabricated board ready for assembly.

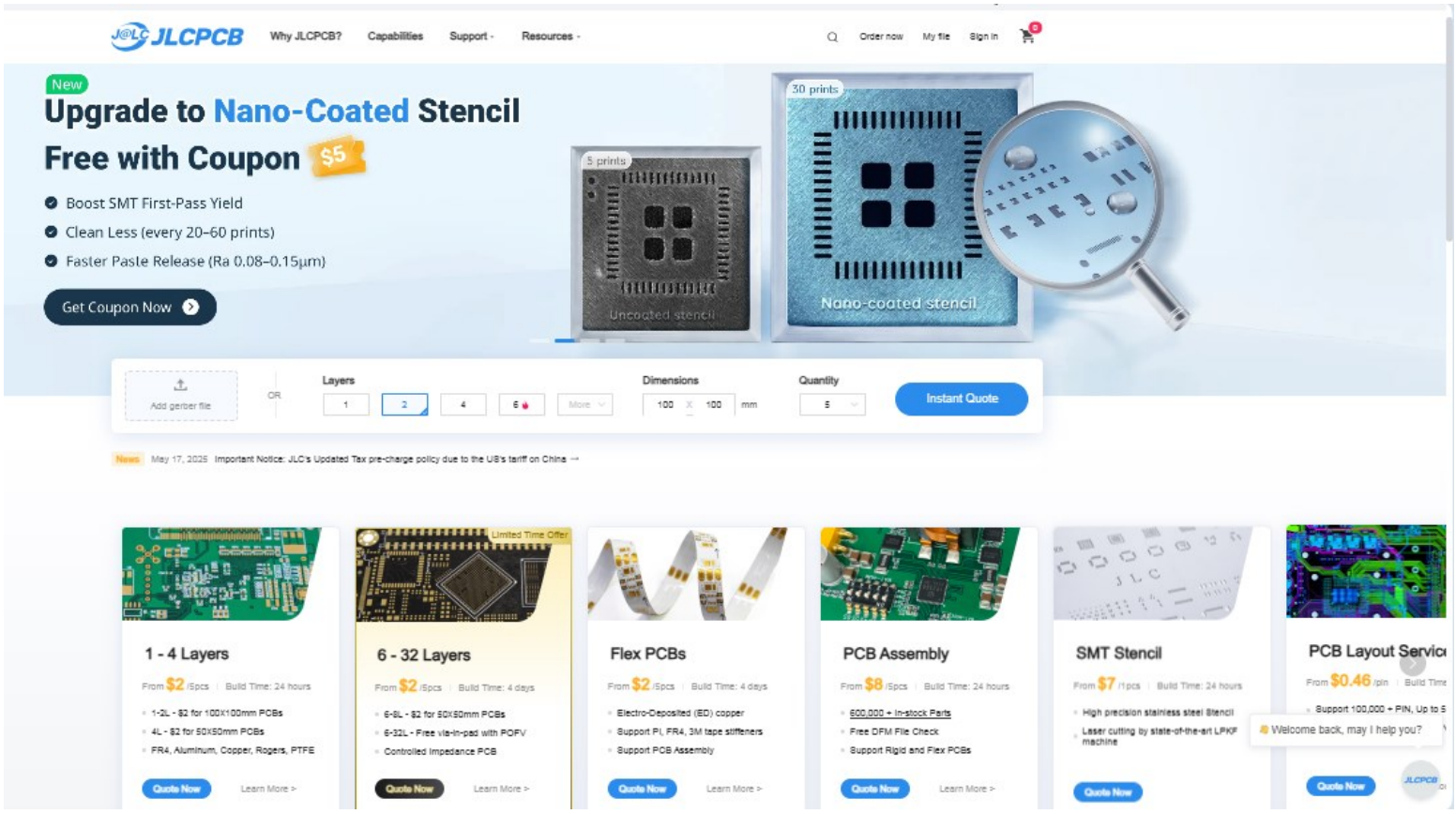


Figure 3. 4: JLCPCB Homepage [21]

3.2.5. VS Code and PlatformIO Extension : Secondary Development Environment

Although the primary firmware development was done using STM32CubeIDE, **Visual Studio Code (VS Code)** with the **PlatformIO extension** was also utilized for auxiliary tasks and quick prototyping especially for ESP32-related firmware development. PlatformIO offers cross-platform compatibility, integrated library management, and an intuitive interface for managing multiple embedded projects.

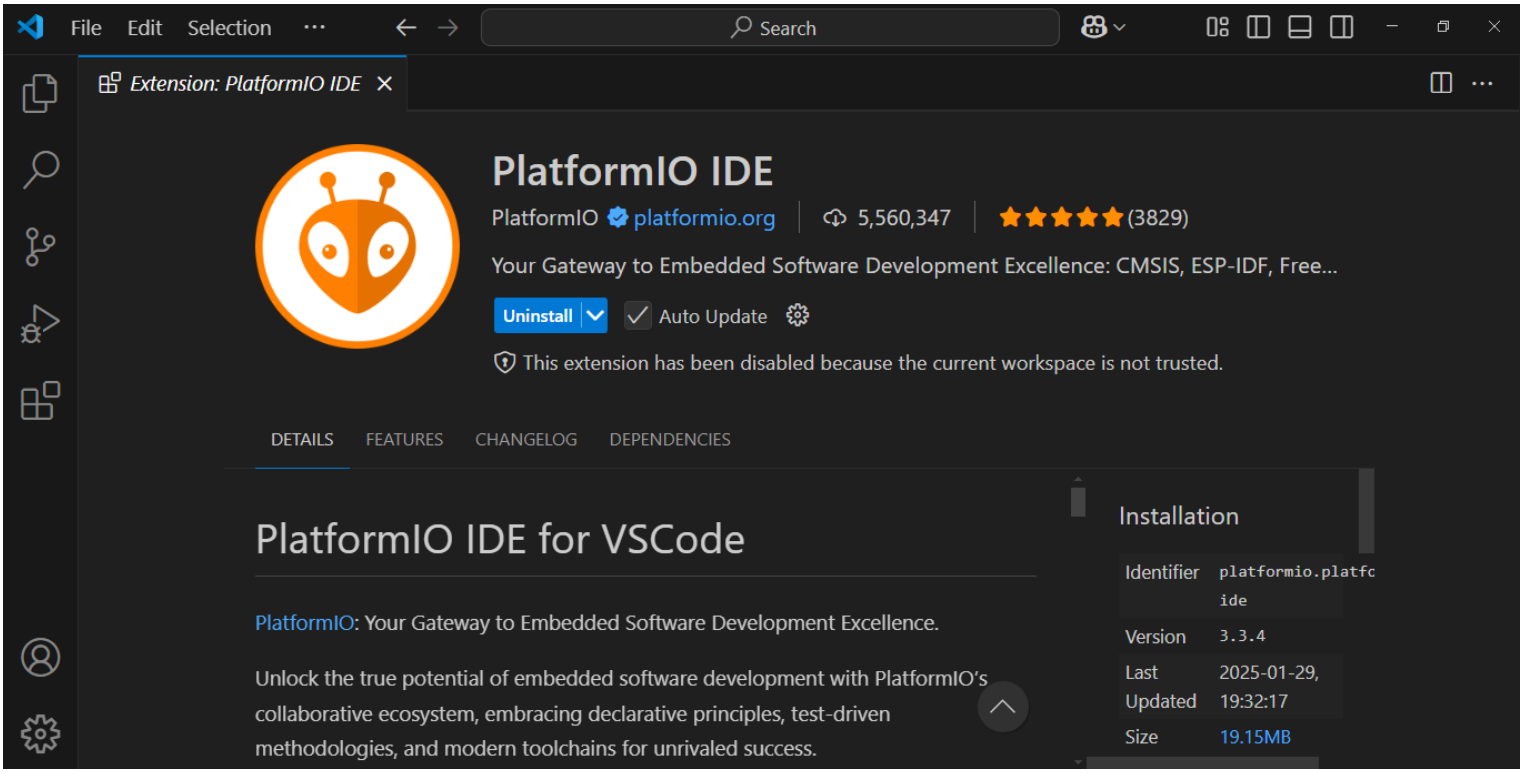


Figure 3.5: Platform IO extension in VS code

3.2.6. SNC ALMITECH: Manufacturing the second PCB

SNC ALMITech is an Algerian company with extensive experience since 1991. Based in Algiers, it has built a strong reputation in professional electronic manufacturing. With a skilled team of academics, a modern industrial setup, and a well-defined product/service offering tailored to our market, the company is also ready for export as all our products comply with international standards. [22]

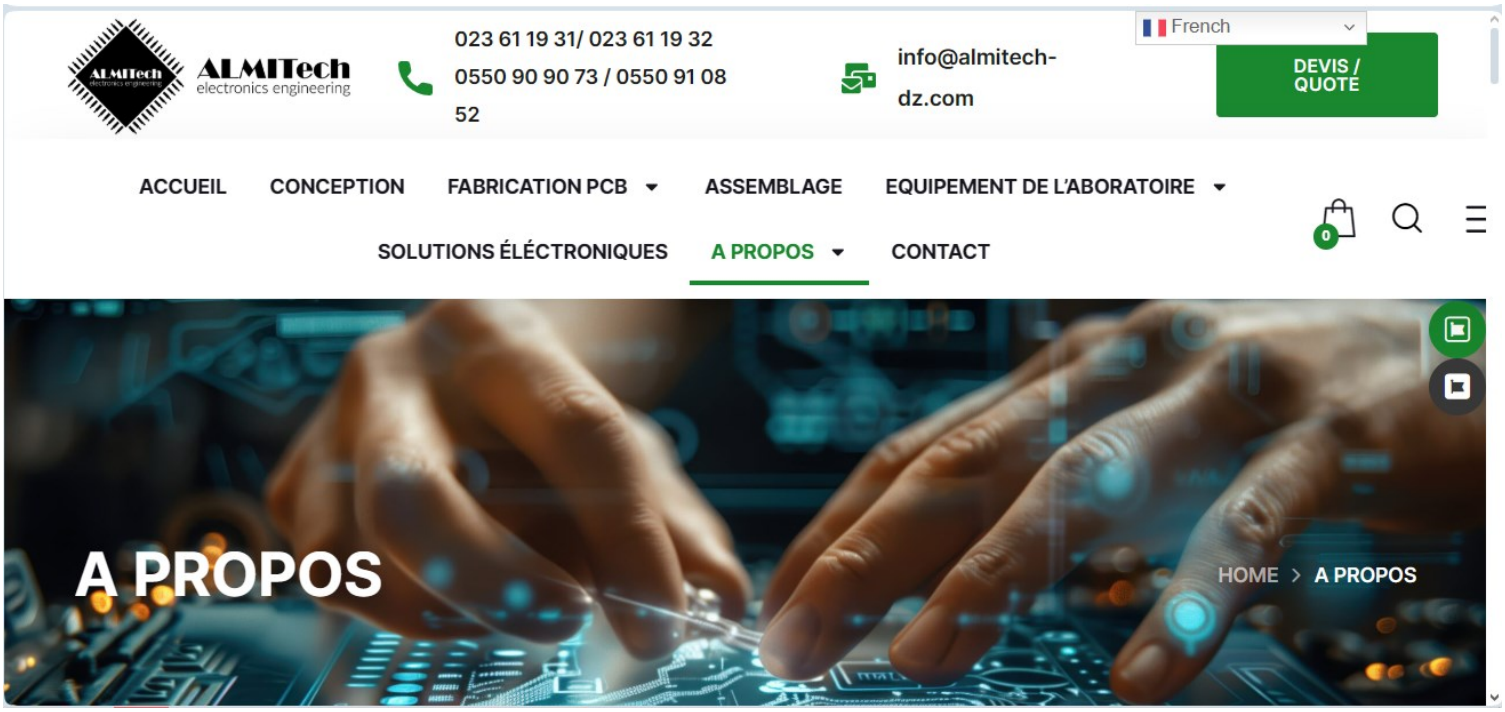


Figure 3.6: ALMITECH Homepage

3.3. PCB Fabrication and Assembly

Why a Custom PCB Was Necessary?

While the initial phases of this project were developed on a breadboard using modular components, the transition to a custom-designed PCB became a necessary step both functionally and strategically. The shift from prototyping to a rigid, professional board brought significant improvements in performance, reliability, integration, and deployment readiness.

Limitations of the Breadboard Setup:

- **Unstable Wiring:** Jumper cables often disconnected during vibration testing.
- **I²C Noise:** Long wires degraded communication between MPU6050 and OLED.
- **Fragility:** Breadboards were not suited for environments with mechanical stress.
- **Power Fluctuations:** Shared rails introduced instability during simultaneous operations.

Solutions provided by Rigid PCB:

The custom PCB resolved these issues by offering:

- **Vibration resistance** through soldered, low-impedance connections.
- **Optimized layout** for shared I²C communication and power integrity.
- **Scalability**, enabling the same unit to monitor multiple motors or subsystems simultaneously.
- **Professional appearance**, making the design suitable for future productization.

Having established the necessity of transitioning from a fragile breadboard to a robust, professional-grade PCB, the next step was translating our design vision into a manufacturable layout. This required careful schematic capture, layered routing, and rule-driven validation, all of which we carried out using the KiCad software suite.

3.3.1. Design Workflow Overview

Once the hardware design was finalized and validated schematically (see Chapter 2), the next step was to bring the system to life as a professional printed circuit board. This section details the practical workflow we followed from layout and rule checks to manufacturing using KiCad for design and JLCPCB for production.

Unlike earlier stages that focused on component selection and interconnection logic, this workflow emphasized electrical integrity**,** manufacturability, and compact integration**.**

- **Step 1: Preparing the Schematic for Layout:** The completed schematic from KiCad's Schematic editor was imported into the PCB editor. All hierarchical sheets were linked, and **Electrical Rule Checks (ERC)** ensured there were no open nets, duplicates, or configuration errors.
- **Step 2: Assigning Component Footprints and Net Rules:** Every component was matched with an appropriate footprint from KiCad’s libraries.

- **Through-hole vs SMD** was considered for soldering and routing.
- **Net classes** were defined for critical signals like I²C, CAN, and power lines, each with custom trace widths and clearance rules to enforce layout constraints.

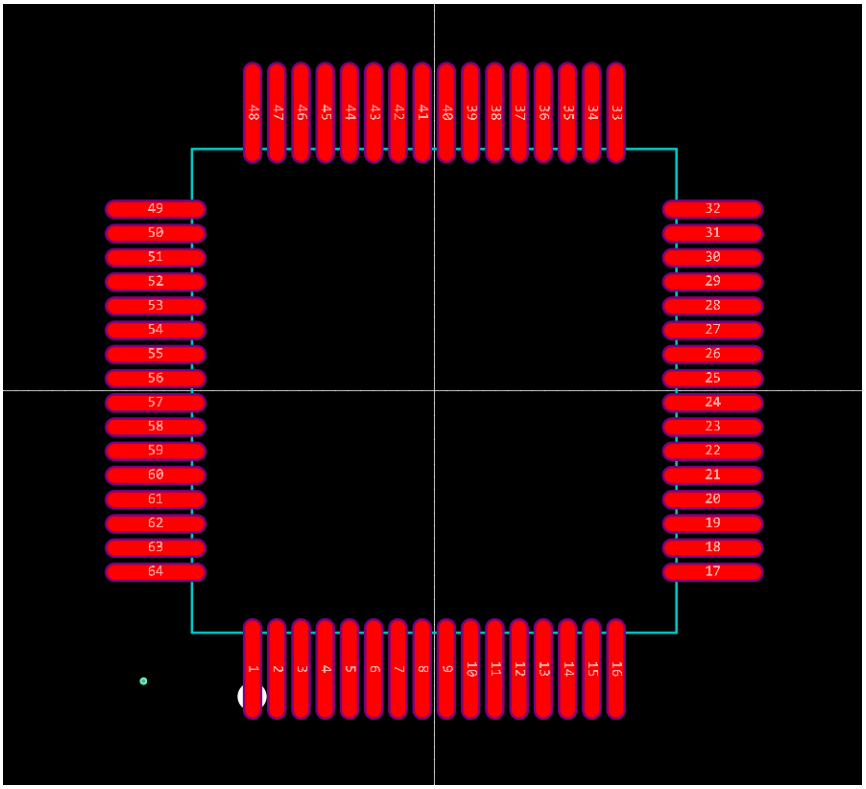


Figure 3.7: an example of the MCU footprint

- **Step 3: Layer Planning and PCB Routing:** A **6-layer stackup** was selected to separate signals and power, improve EMI immunity, and allow for clean routing.

  Routing priorities were:

- Top Layer: all signal lines, component connections
- Inner Layers: Dedicated ground and power planes
- Bottom Layer: Auxiliary signals and headers plus a dedicated ground

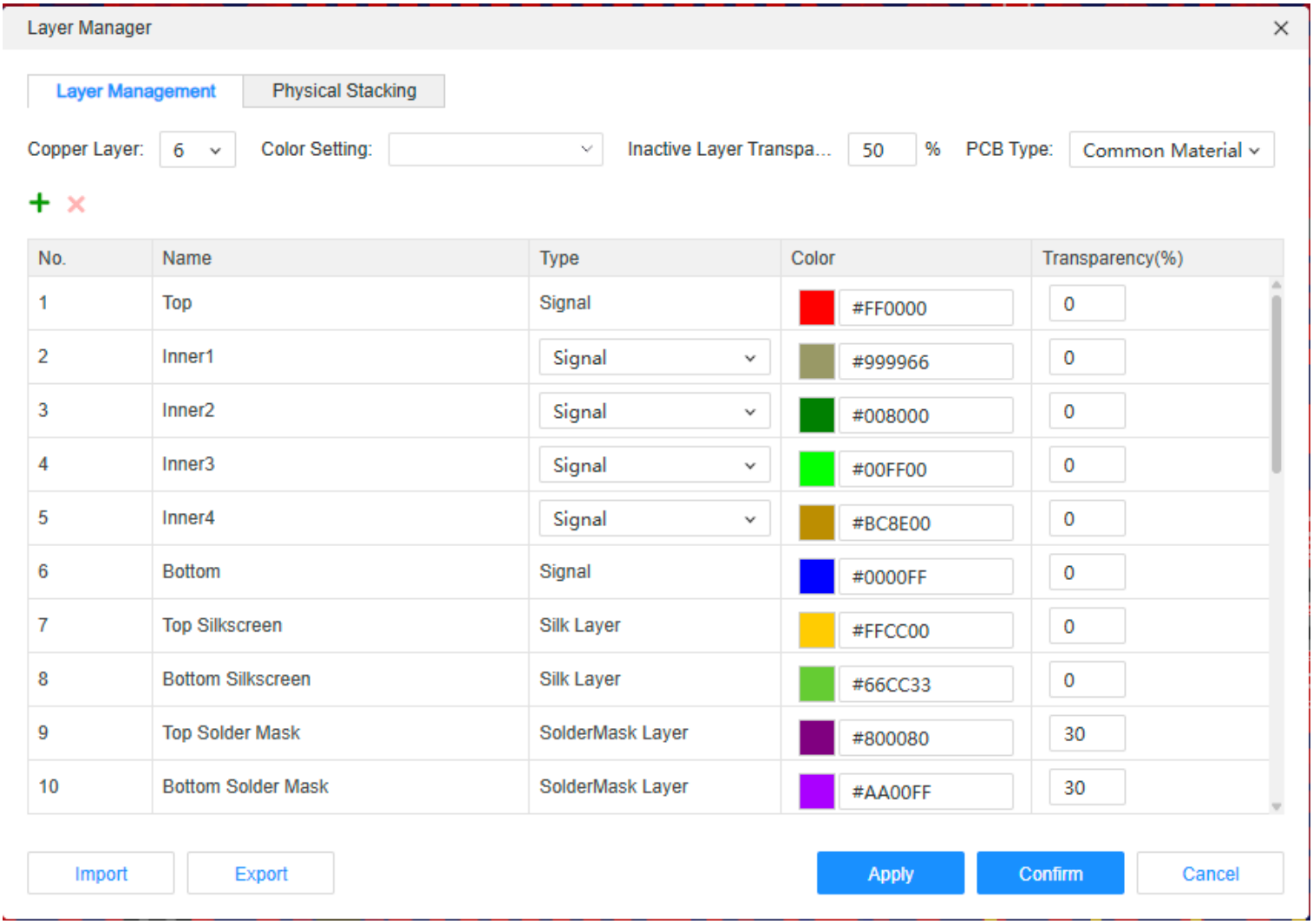


Figure 3.8: PCB Layers

- **Step 4: Rule Checking and Optimization:** After routing, we ran KiCad's **Design Rule Check (DRC)** to detect spacing violations, unconnected pads, or improperly routed vias. Silkscreen placement and mechanical outlines were reviewed to prevent overlaps and ensure readability.

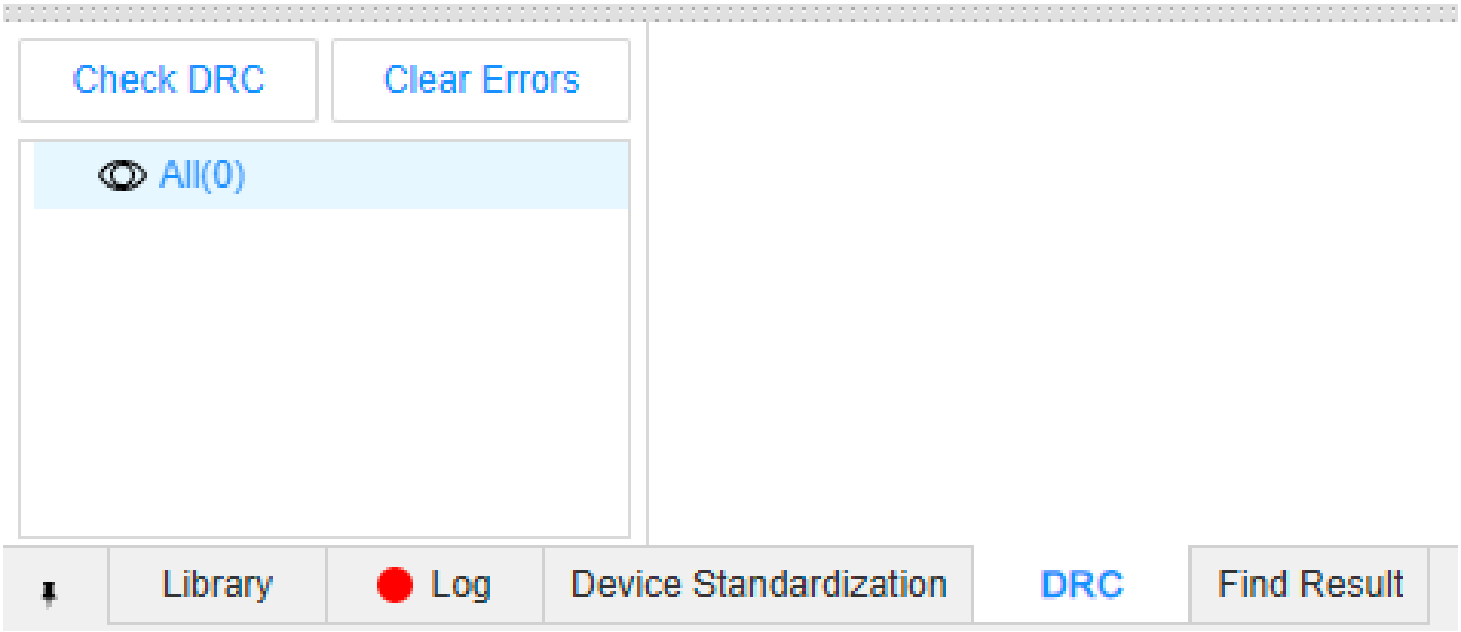


Figure 3.9: checking Errors

- **Step 5: Generating Production Files:** We generated standard **Gerber files** and **netlists**, ready for fabrication. These included:

- Top/bottom copper layers
- Solder mask and paste mask layers
- Silkscreen layers
- Board outline and drill hits

All files were verified with a Gerber viewer to prevent manufacturing errors.

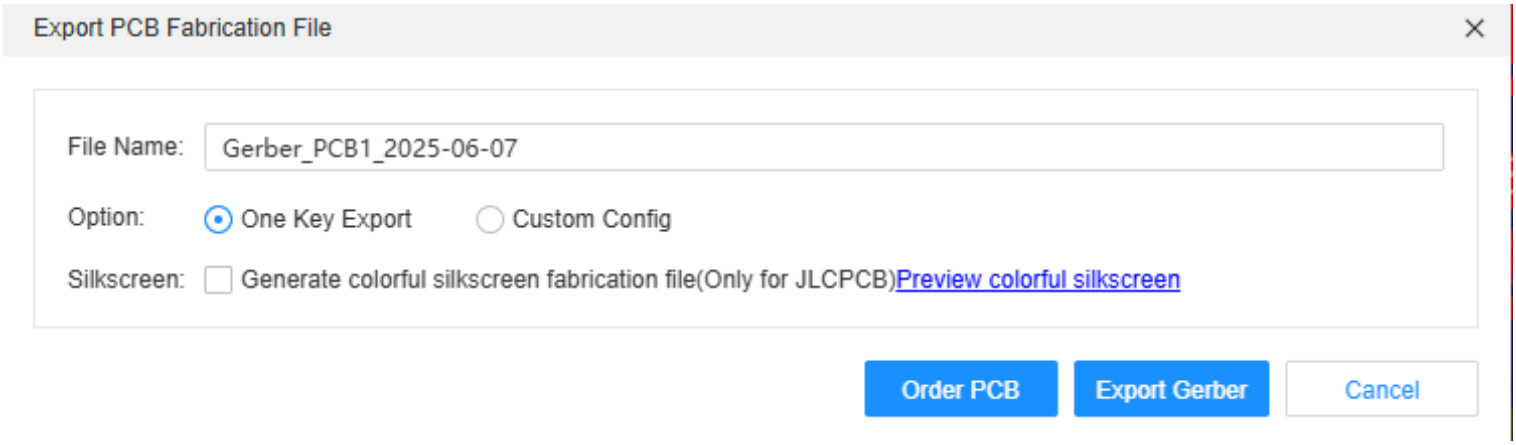


Figure 3.10: Exporting PCB Gerber File

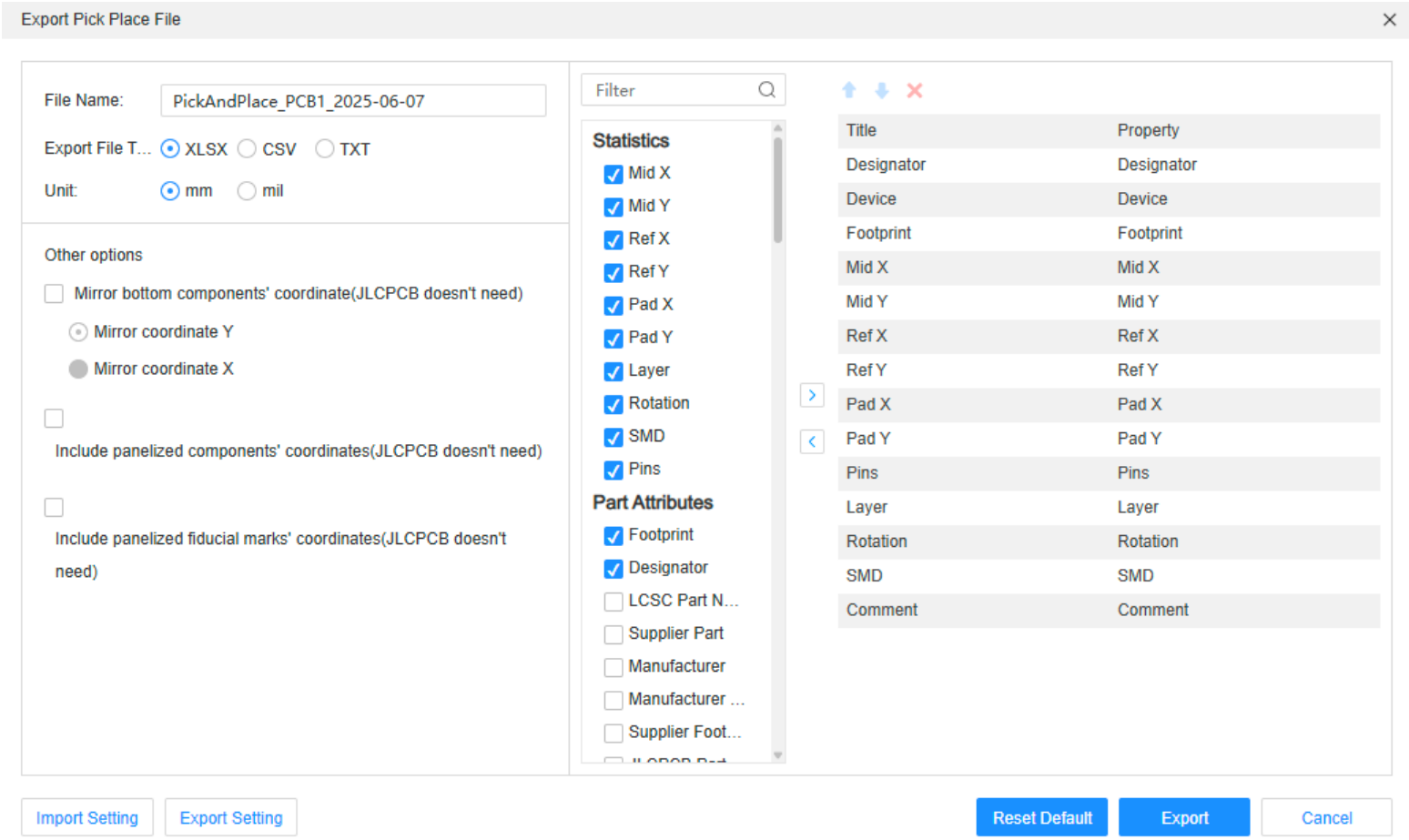


Figure 3.11: Exporting Pick Place File

- **Step 6: Fabrication at JLCPCB:** The finalized Gerbers was uploaded to **JLCPCB**, a manufacturer supporting high-precision multilayer boards. Key specifications selected:

- **6-layer board** with defined stackup
- 1.6 mm board thickness
- HASL finish, green solder mask
- 1 oz copper per layer

Figure 3.12: PCB Manufacturing Order on JLCPCB

During fabrication, JLCPCB provided live updates on production stages, including drilling, plating, etching, and final QC ensuring full visibility into the board's birth from digital to physical.

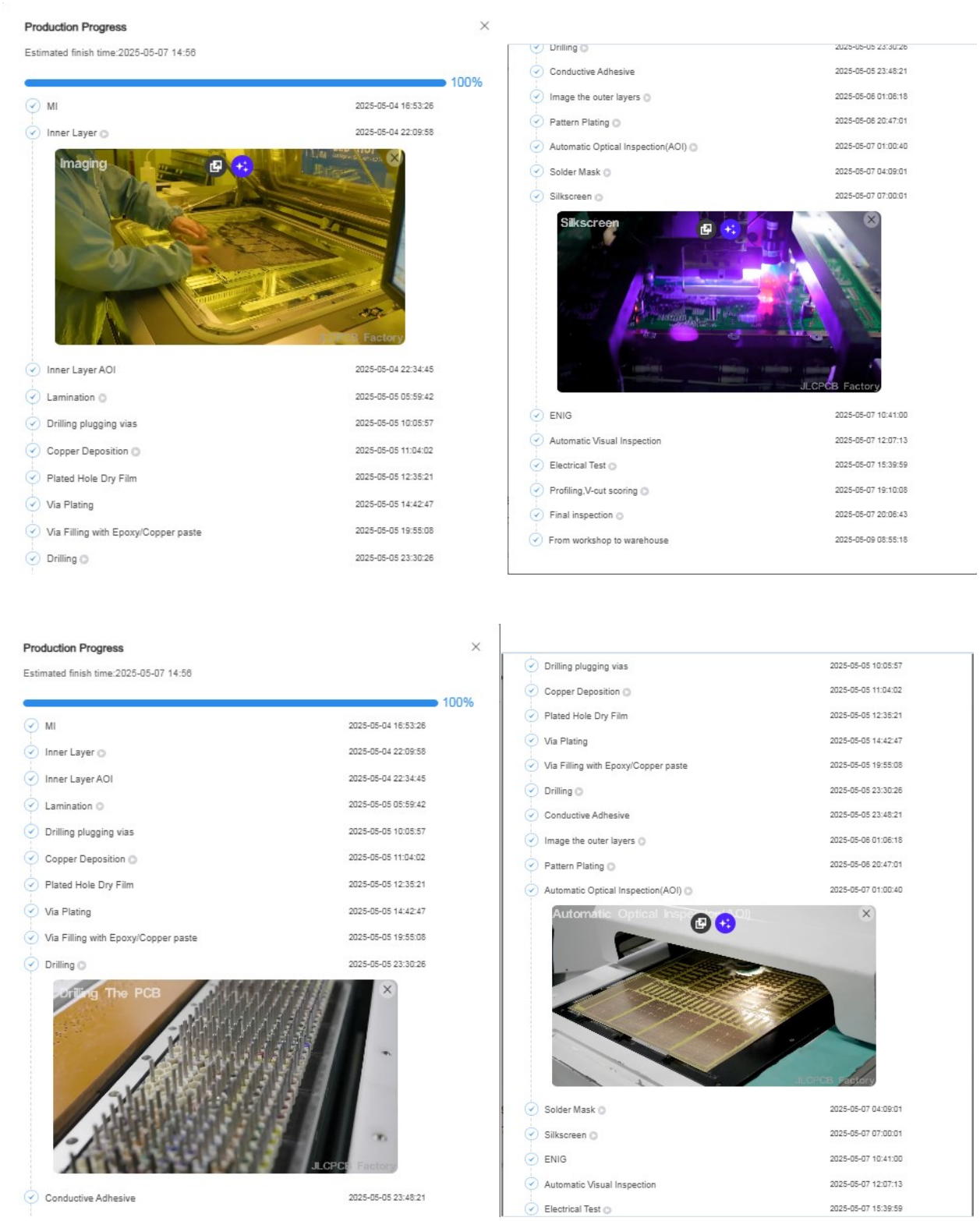


Figure 3.13: PCB Production and Process in JLCPCB Factory

The (***Figure 3.14***) Below Summarize the PCB design workflow:

Figure 3.14: PCB Design Workflow from Schematic to Fabrication

Once the PCB design process was complete in KiCad, including schematic capture and layout validation, we proceeded to analyze and document the role of each individual PCB layer. Given the 6-layer structure, each layer was designed with specific electrical and mechanical responsibilities in mind, as outlined below.

3.3.2. Layer-by-Layer PCB Breakdown

To meet the performance, reliability, and noise-resilience demands of our embedded AI system, we designed a 6-layer rigid PCB**.** This multilayer configuration was chosen to optimize **signal integrity**, **power delivery**, and **electromagnetic compatibility (EMC)** particularly important in systems involving real-time AI inference and sensitive motion data acquisition.

The stack-up was carefully arranged to separate noisy and sensitive signals, provide strong return paths, and minimize electromagnetic interference (EMI). Below is a detailed breakdown of each layer:

A) Top Layer: Primary Signal and Component Routing

The top layer is the main routing and component placement layer. It contains virtually all the active and passive components, including:

- The **STM32F103RB microcontroller**
- **MPU6050** accelerometer
- **SSD1306 OLED** header
- **Tactile push buttons** (for reset and boot selection)
- **USB interface and 3.3V LDO regulator**
- **RGB LED**
- I^2C communication lines
- GPIO lines and external interrupts

To preserve signal quality, **sharp corners were avoided**, and longer traces were routed with impedance control in mind. This layer is tightly coupled with the ground plane below it to support **low-inductance return paths** and reduce EMI.

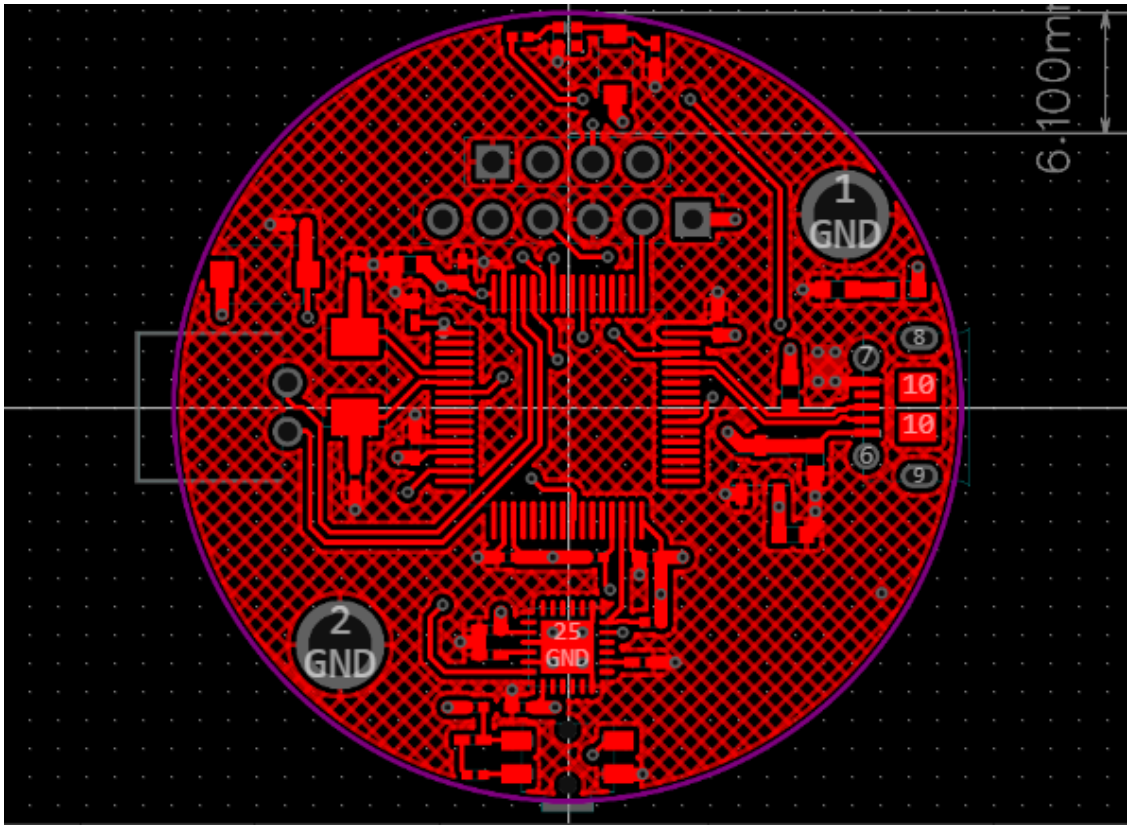


Figure 3.15: Top Layer Primary Component and Signal Routing

B) Inner Layer 1: Solid Ground Plane (GND)

The first internal layer is a dedicated ground plane**,** providing:

- Low-impedance return paths for all top-layer signals
- Effective shielding for high-speed or sensitive lines (e.g., I²C)
- Reduced **ground bounce** in digital circuits (e.g., from DMA-triggered I²C readouts)

Placing the ground plane directly beneath the top signal layer supports strong signal-GND pairing, essential for minimizing **crosstalk** and achieving stable high-frequency performance.

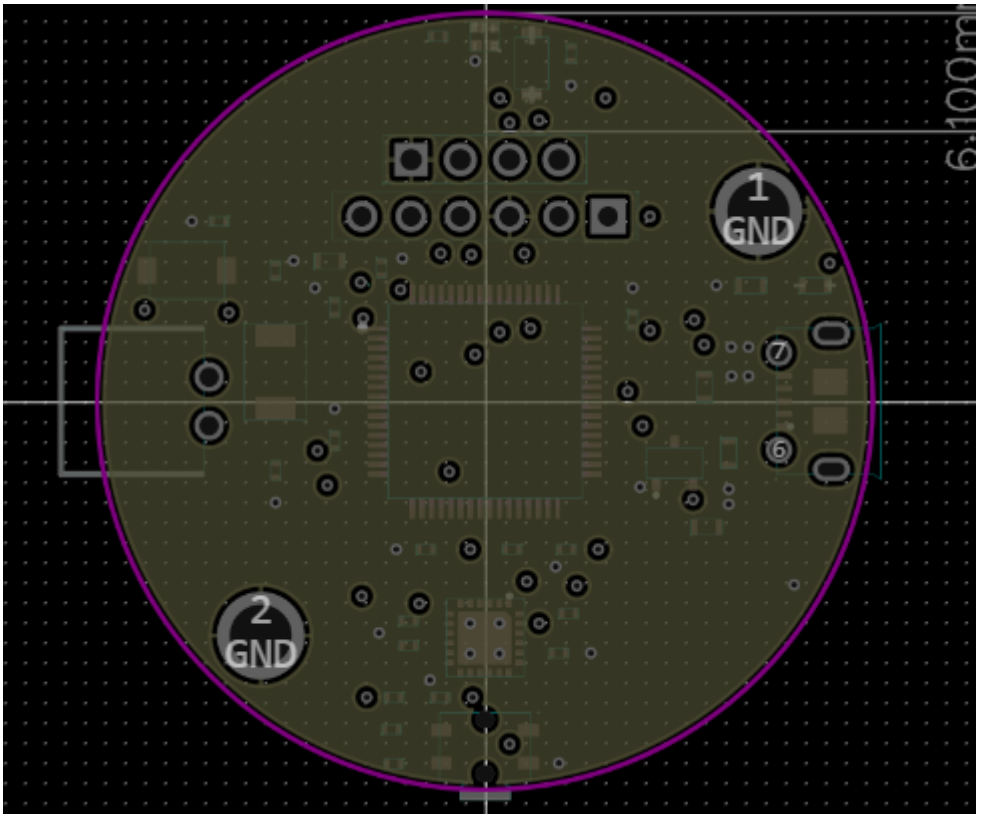


Figure 3.16: *Inner Layer 1 Solid Ground Plane (GND)*

C) Inner Layer 2: Mixed Signal + GND Plane

This layer supports :

- Control and auxiliary signals like:
  - RGB LED signal line
  - Tactile push button traces (reset/boot)

- I²C SDA1
- Selective GND plane sections to support signal integrity

The design separated **power and signal traces** to reduce cross-talk, with **adequate spacing** between control lines and GND copper pours to maintain clear paths for return currents.

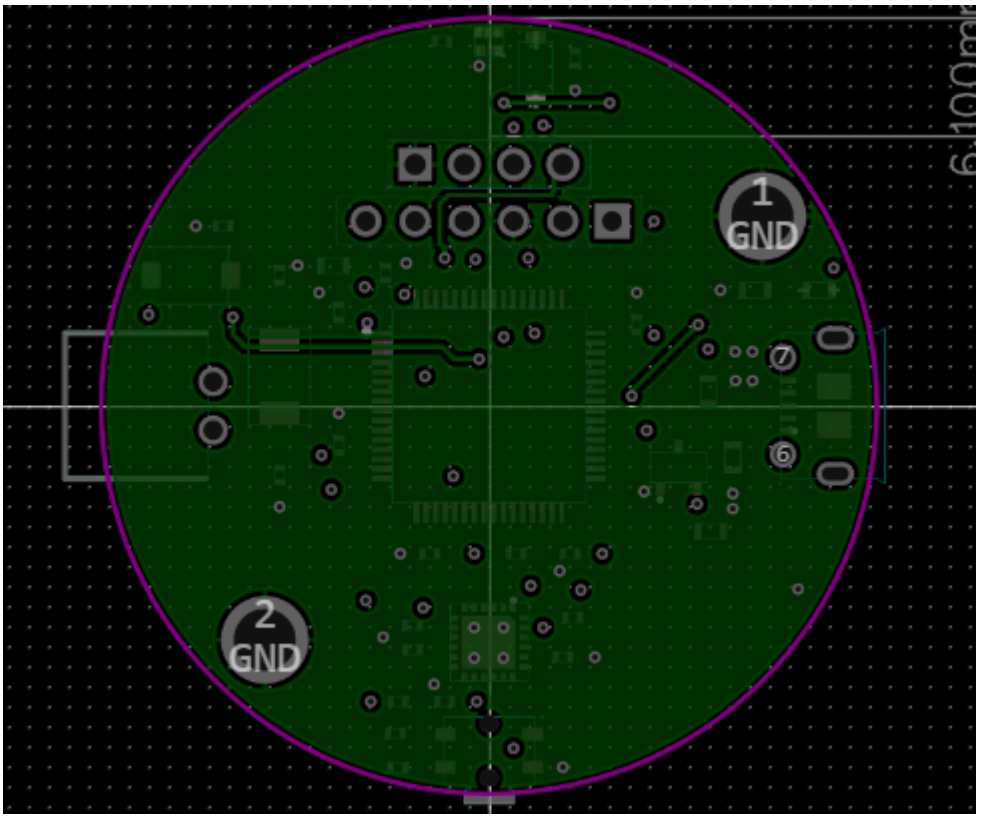


Figure 3.17: Inner Layer 2 Mixed Signal and Ground Layer

D) Inner Layer 3: Secondary Signal Routing + 5V Power

This layer serves two purposes:

- Secondary routing for:
  - Tactile switch
  - MPU6050 interrupt line
  - Debug header lines (e.g., SWDIO, PAD6 to MCU)
- A **5V power line**, routed from **Pad 2** on the connector to a via that reaches the bottom layer

The 5V trace was routed with special care, **isolated from other signals**, maintaining clearance to avoid interference, and supported by adjacent **GND pours** to absorb noise and stabilize power.

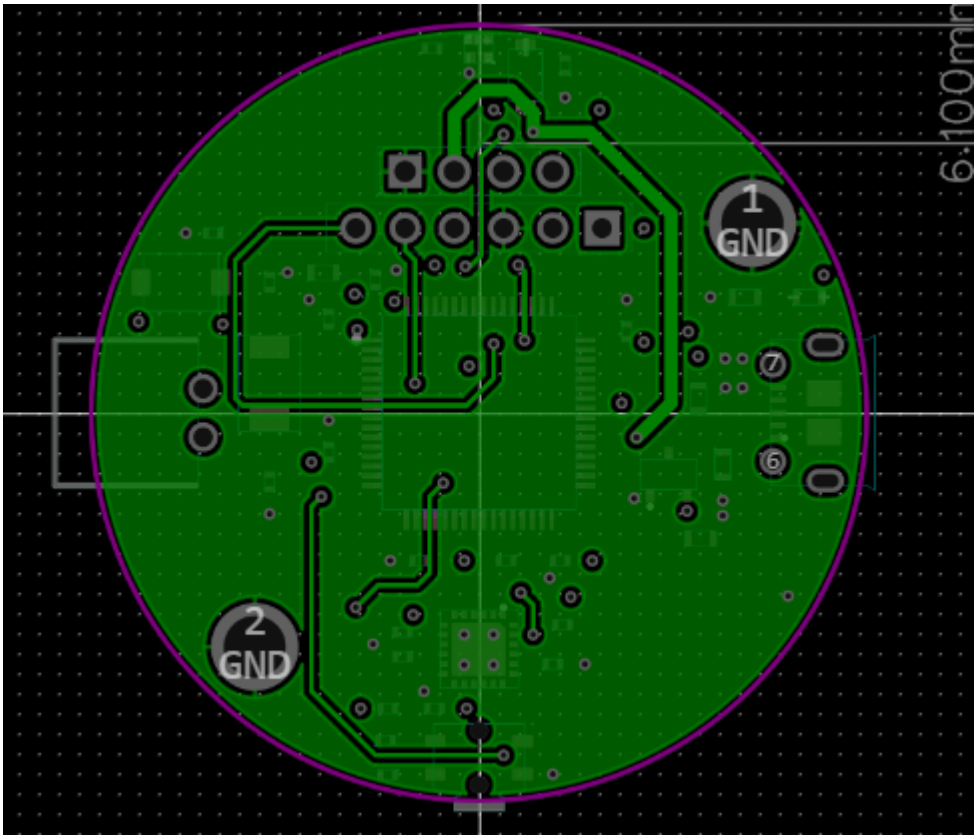


Figure 3.18: Inner Layer 3 Secondary Signal and 5V Power Routing

E) Inner Layer 4: Dedicated 3.3V Power Plane (VCC)

This layer is entirely dedicated to VCC (3.3V**)** distribution, ensuring:

- Uniform and stable power delivery to all low-voltage components
- Reduced voltage drop due to copper plane continuity
- Isolated, low-noise power plane for sensors and microcontroller

This separation helps maintain reliable sensor readings and inference consistency during high-load states.

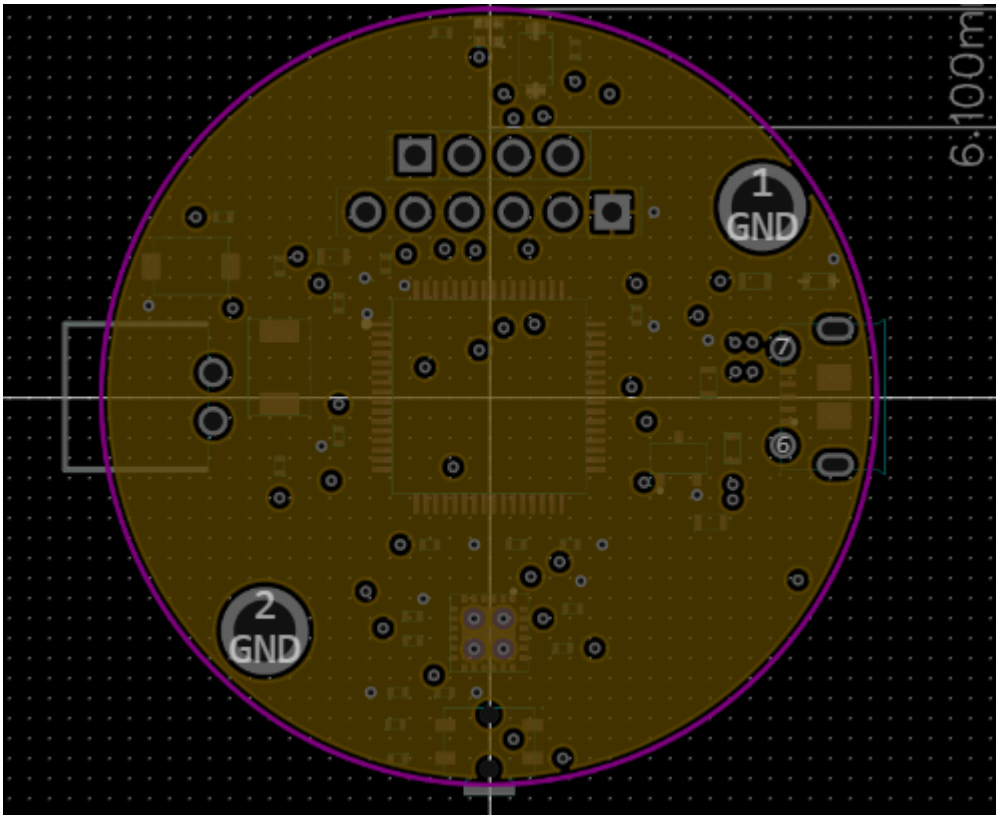


Figure 3.19: Inner Layer 4: Dedicated 3.3V Power Plane

F) Bottom Layer: Secondary Ground Plane

The bottom copper layer acts as a secondary ground plane, enhancing signal return paths and reducing EMI. It also routes low-priority signals like SWD debugging lines and reset switches. It ensures:

- Reinforces grounding and shielding for the entire board
- Hosts non-critical routing to avoid congestion on upper layers
- Includes via clearance rings to isolate from unintended GND contact

This dual role improves both electrical stability and layout flexibility.

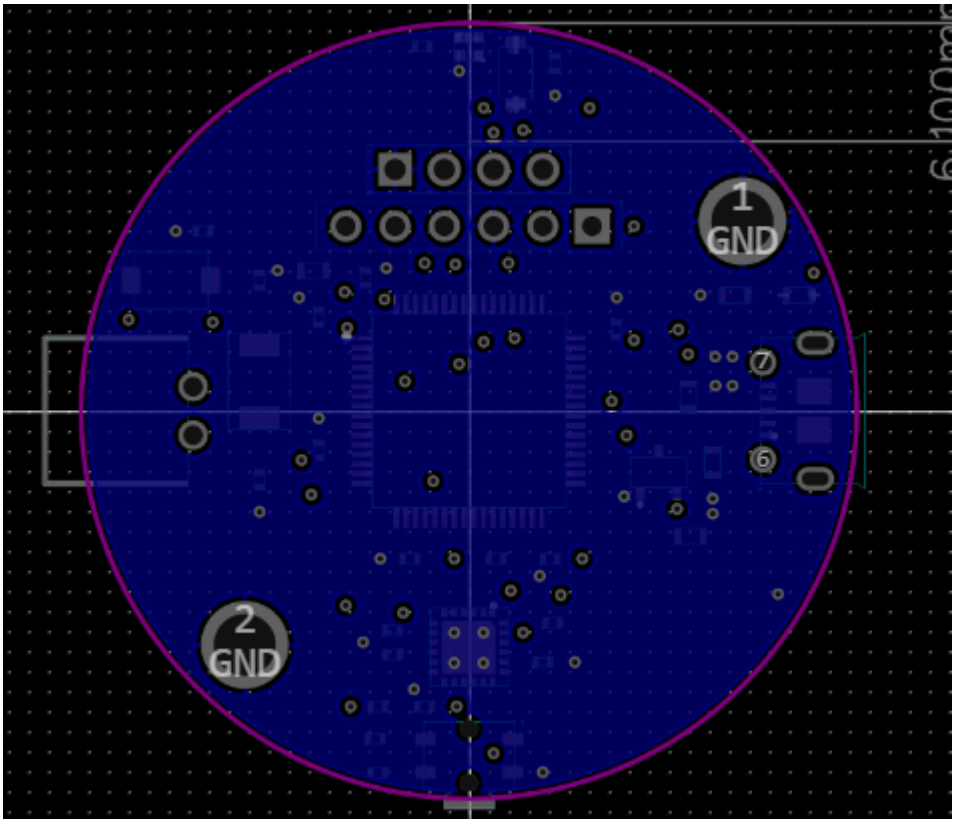


Figure 3.20: Bottom Layer: Secondary Ground Plane and Auxiliary Routing

G) Layer Stack

To better illustrate the internal structure and organization of our 6-layer PCB, the figure below shows a complete stack-up view of all layers used in the board's fabrication. Each layer serves a specific electrical or mechanical purpose as shown in previous sections, contributing to signal integrity, power distribution, EMI shielding, and component routing.

The layers are arranged from top to bottom as follows:

1. **Top Layer** – Main component placement and signal routing
2. **Inner Layer 1** – Solid Ground Plane (GND)
3. **Inner Layer 2** – Mixed Signals + GND fill
4. **Inner Layer 3** – 5V & auxiliary signals + GND isolation
5. **Inner Layer 4** – 3.3V Power Plane
6. **Bottom Layer** – Secondary Ground + Debug/low-priority routing

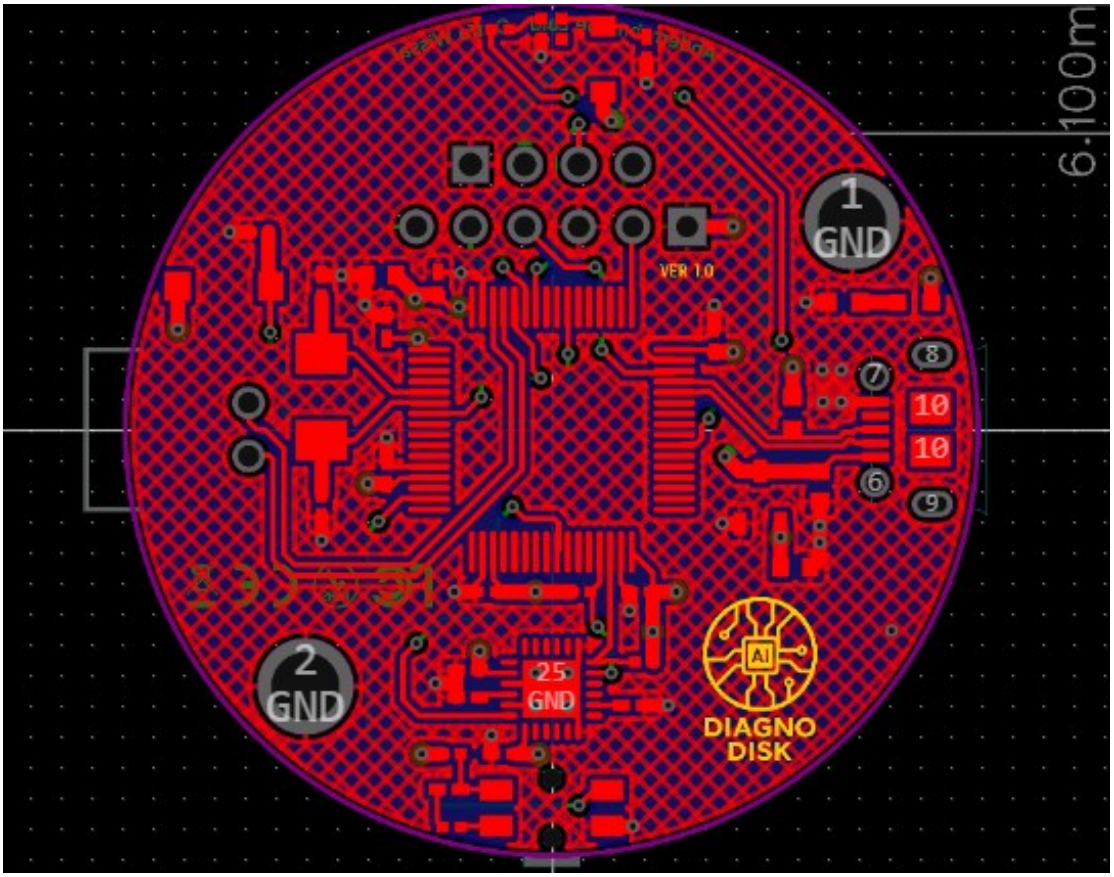


Figure 3.21: Full 6-Layer Stackup of the Custom PCB

This structured stack-up was critical for maintaining robust electrical performance under real-time vibration monitoring and inference workloads, while also allowing compact, EMI-conscious layout decisions.

H) 2D View of the Bare PCB (Without Components)

Before soldering the components, the PCB reveals its raw structure, highlighting how the 6-layer stack was routed, organized, and manufactured. This view gives insight into the complexity of the layout and the space optimization achieved during design.

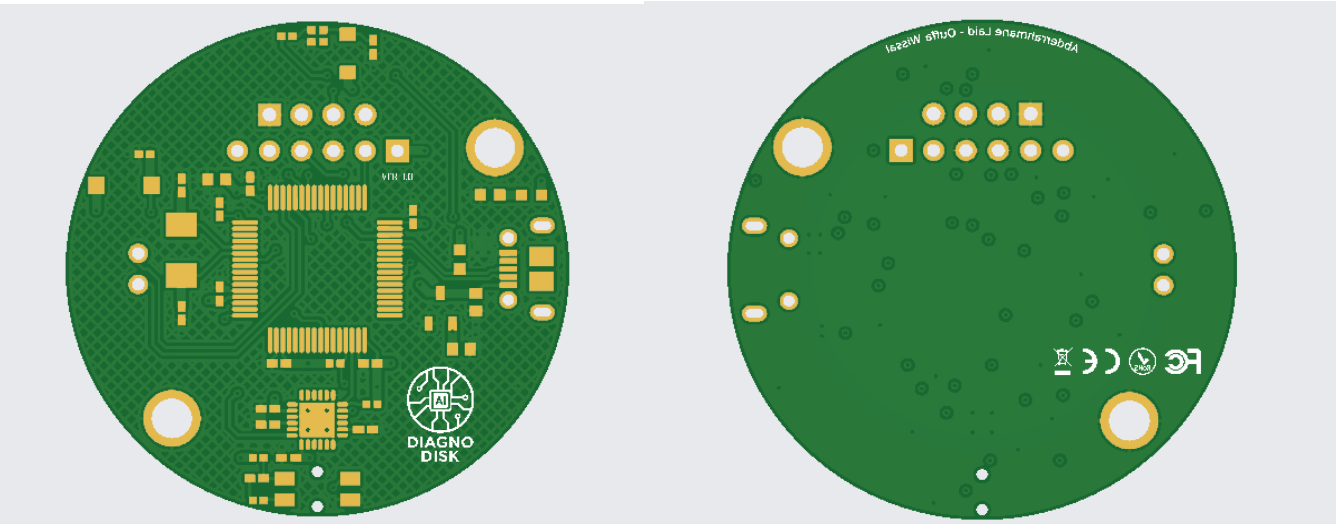


Figure 3.22: 2D Visualization of Bare PCB Structure (No Components) from top and bottom

This render shows the board's physical layout, via placement, and trace density before any parts were assembled.

3.3.3. Final PCB Visualization & Assembly Overview

After completing the design, validation, and fabrication process, the final PCB was assembled and evaluated both mechanically and electrically. The board was manufactured as a 6-layer rigid PCB using FR4 (Standard Tg 135–140°C) material, with a green solder mask**,** white silkscreen, and overall dimensions of 40 mm × 40 mm**.** The board thickness was 1.6 mm, suitable for balancing durability with compact form.The layout was engineered for compactness, vibration resistance, and scalability for deployment in real-world industrial environments. factor.

Key Visible Components:

- **Center** – **STM32F103RBT6 microcontroller**, the core processing unit responsible for handling sensor data, AI inference, and peripheral communication.
- **Right side** – **Micro USB port and 3.3V voltage regulator**, which provide stable power supply to the entire board.
- **Left side** – **Tactile push-button and CAN Bus**, used for reset or user-triggered events (e.g., switching modes).
- **Top center** – **OLED header and 6-pin connectors**, designed for I^2C communication with the SSD1306 display and expansion to external modules.

- **Top edge** – **Addressable RGB LED (A-RGB)** for visual feedback (e.g., system status or anomaly alert).
- **Bottom** – **MPU6050 motion sensor**, responsible for capturing vibration and motion data across three axes, and another push button for the AI learning

To better illustrate the structural and spatial layout of our board, we generated 3D renders of the assembled PCB using KiCad's integrated viewer. These images highlight the compact integration, component accessibility, and clean routing essential for embedded AI applications in real-world environments.

The renders provide a comprehensive view of the design from different angles:

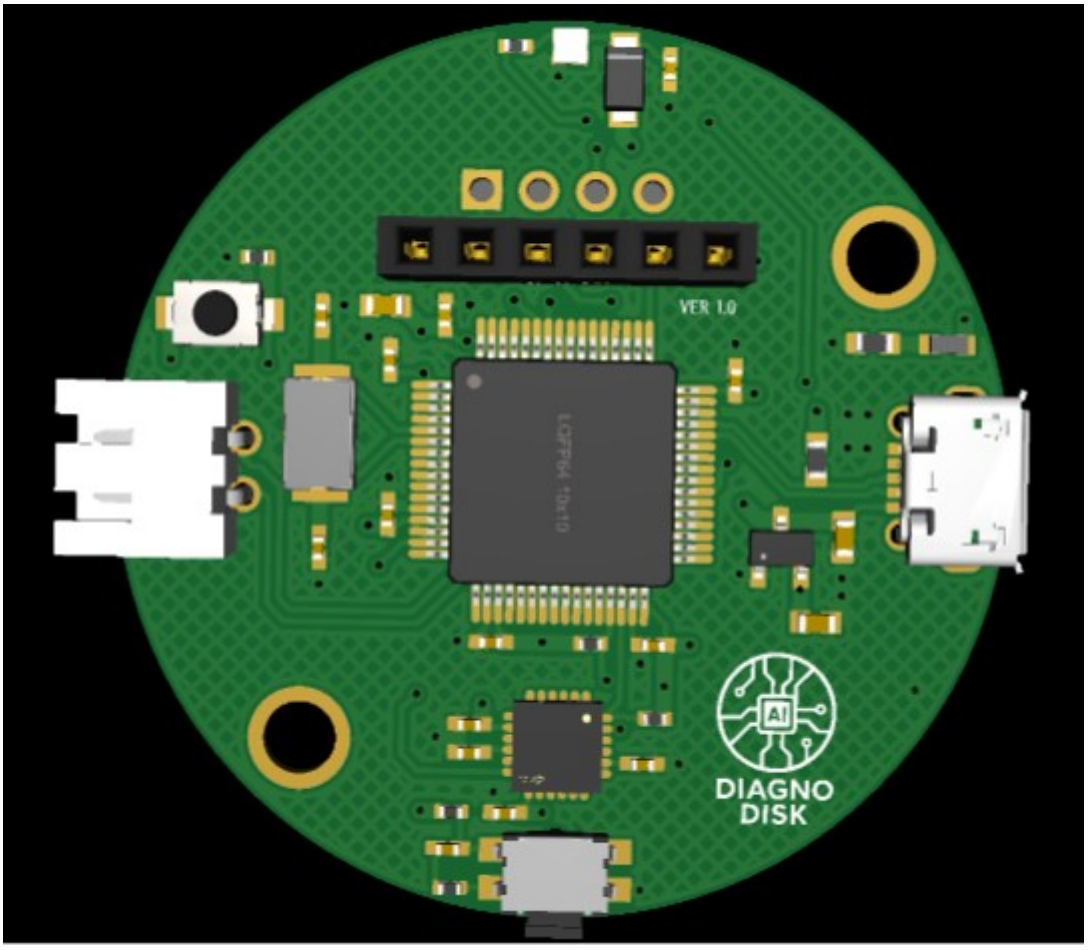


Figure 3.23: Assembled 3D view for Diagno Disk (Top View)

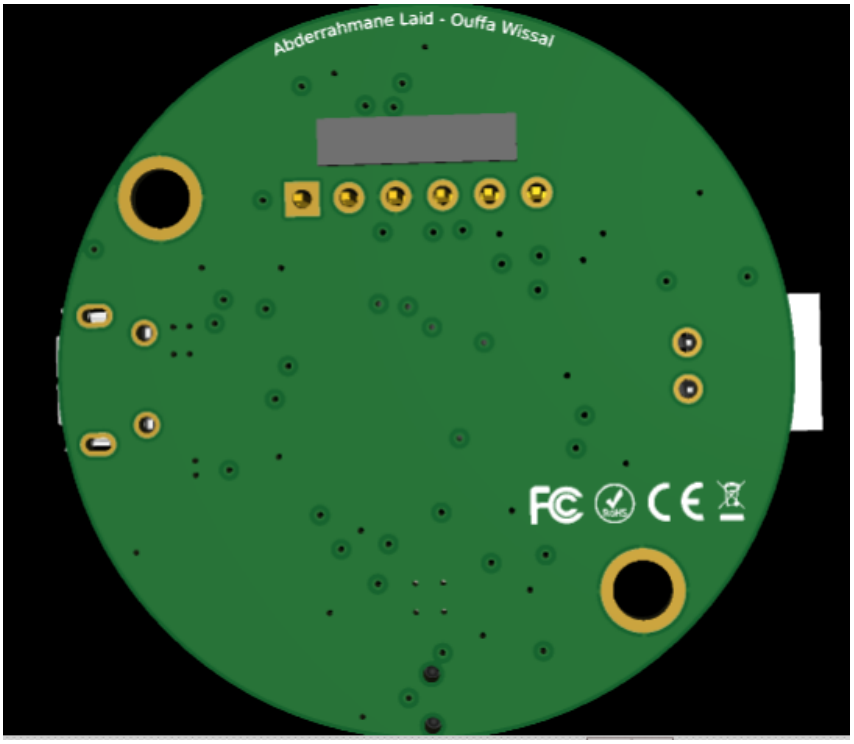


Figure 3.24: Assembled 3D view for Diagno Disk (Bottom View)

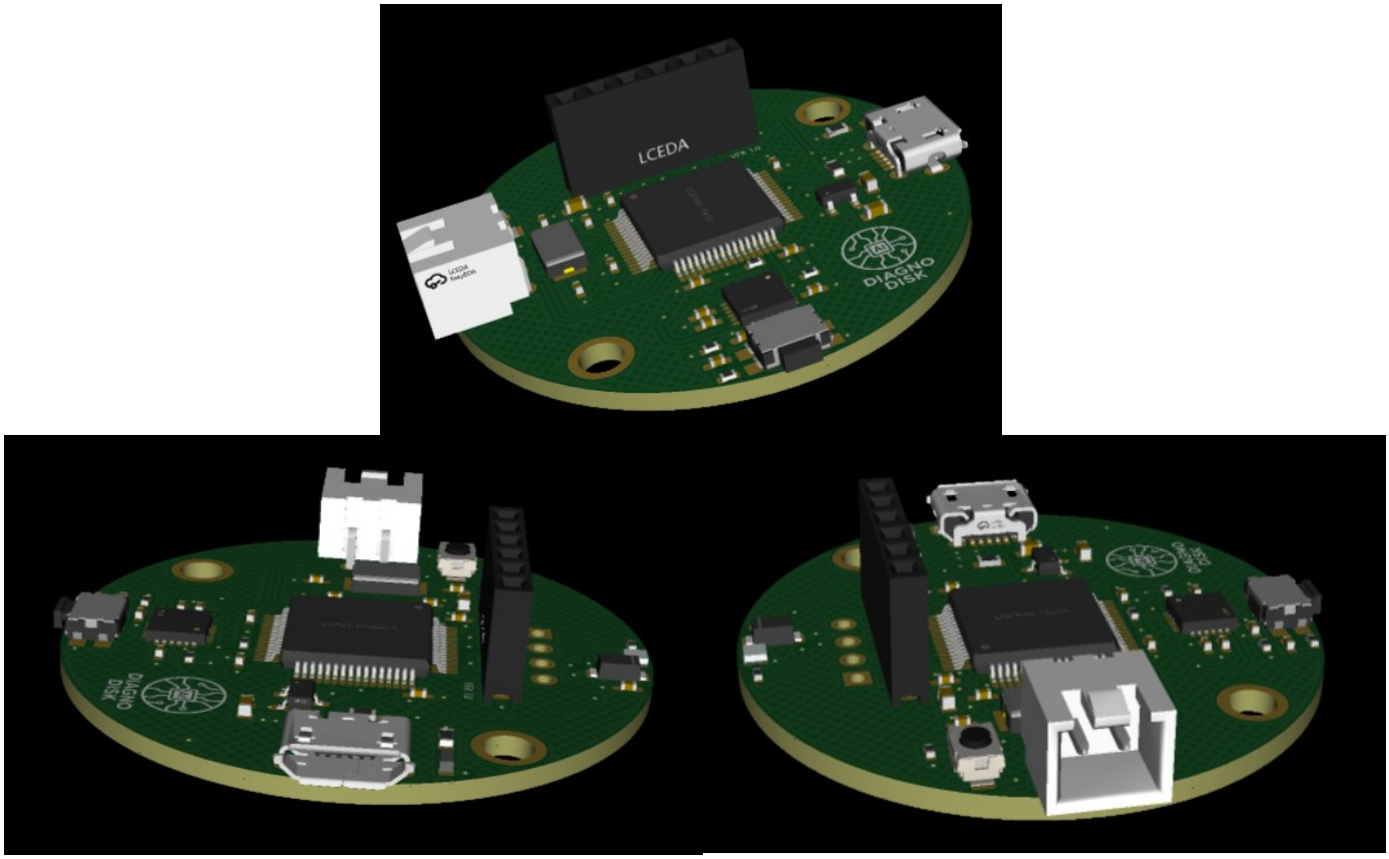


Figure 3.25: Assembled 3D view for Diagno Disk (different angles)

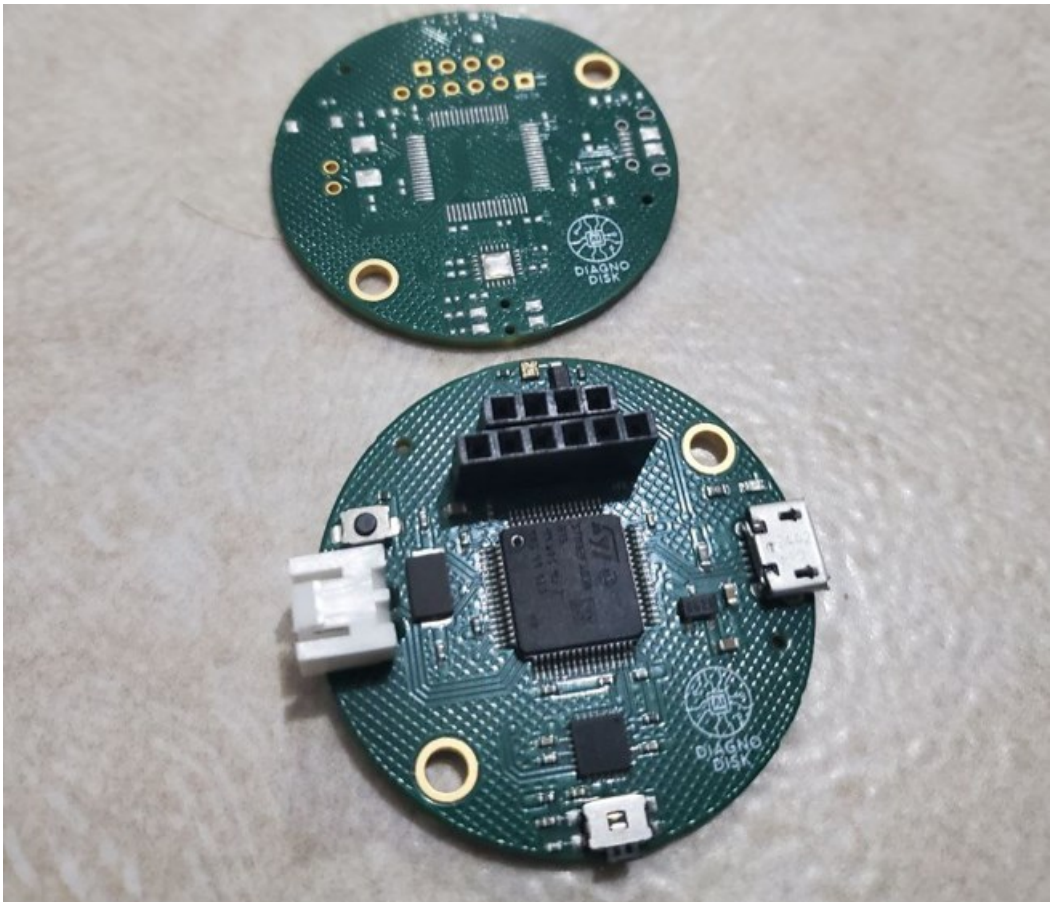


Figure 3.26: Final Fabricated Board (Real Image)

The selected 6-layer stackup not only met our functional requirements but also leveraged the advanced manufacturing capabilities offered by JLCPCB. This next section highlights the specific performance and reliability advantages we gained by choosing a rigid, multi-layer configuration.

3.3.4. Capabilities of a 6-Layer Rigid PCB (via JLCPCB)

As previously discussed, the final board was fabricated through JLCPCB, which offers a wide range of manufacturing capabilities supporting our 6-layer rigid PCB design. The complete list of fabrication specifications, including core PCB parameters, drilling capabilities, trace and spacing constraints, and solder mask properties, is provided in **Appendix (Tables .1 to 4)**.

3.3.5. Final Bill of Materials (BOM)

The Bill of Materials summarizes the key components used in the design and fabrication of the 6-layer PCB. Each component was selected to meet electrical, mechanical, and functional constraints while ensuring compatibility with JLCPCB's assembly service.

The full Bill of Materials is provided in **Appendix (Table .5).**

3.3.6. Lessons Learned and Manufacturing Tips

As we transitioned from schematic design and simulation into real-world manufacturing, several important lessons emerged each offering valuable guidance for future embedded hardware development. This section summarizes the important takeaways gained during our experience working with JLCPCB and designing a compact, high-performance 6-layer PCB.

-Design for Manufacturability Early: Small details such as pad clearance, drill sizes, and component spacing, if overlooked it can result in DRC violations or difficult soldering. We learned to consult JLCPCB's capabilities before finalizing the layout, especially regarding minimum trace widths and via sizes.

-Be Cautious with Multi-Layer Stackups: While multi-layer boards offer routing flexibility and signal integrity, they require careful planning. Ensuring proper power/ground pairing and placing sensitive signals between shielding planes proved essential for reducing EMI in our vibration-sensitive application.

-Clear Silkscreen Matters: Clean labeling greatly made the PCB visually professional and user-friendly, especially when handed off to collaborators.

-DRC is Your Friend: KiCad's Design Rule Check (DRC) system prevented multiple issues before fabrication. We configured rules according to JLCPCB's limits, which ensured our design met manufacturability standards from the start.

3.4. Wireless Result Transmission Module (ESP32 + CAN Interface)

To extend the usability of the embedded diagno disk system and make results accessible beyond the local display, a secondary PCB was designed and implemented. This board integrates an ESP32 microcontroller with a CAN transceiver, enabling wireless transmission of real-time inference results to a web-based dashboard via Wi-Fi.

While the main STM32-based PCB is responsible for sensing, inference, and local feedback through the OLED screen, this secondary module serves a communication role allowing operators or monitoring systems to observe machine status remotely without physical access to the device.

3.4.1. System Architecture Flow

The data flow within the system is structured as follows:

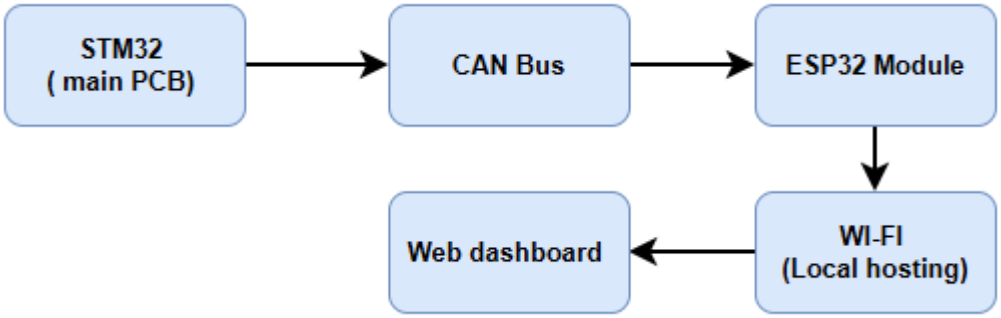


Figure 3.27: data flow of the system

Once the STM32 completes an inference cycle using the NanoEdge AI library, the resulting score or state (e.g., *"80%" for anomaly*) is transmitted via CAN bus to the ESP32. Unlike cloud-based setups, the **ESP32 acts as a local web server**, directly hosting the results webpage accessible over the same Wi-Fi network. This self-contained approach eliminates the need for external servers and ensures that results are viewable in real time from any device connected to the network.

3.4.2. PCB Overview and 3D Rendering

To provide a comprehensive view of the wireless transmission module, this section showcases both the 3D rendering and the assembled second PCB from multiple angles. Designed as a standalone 2-layer rectangular board, it handles wireless result visualization by receiving data via the CAN bus and transmitting it to a web dashboard using the onboard ESP32-C3. The routing view from KiCad featuring red and blue traces for the top and bottom copper layers respectively further illustrates the signal paths and physical layout, highlighting both the mechanical structure and final form factor of the module.

A) Top Copper Layer (Red Traces)

The top layer handles:

- **ESP32-C3 interface connections**
- **CAN_H and CAN_L routing** to and from the central CAN transceiver (Master CAN Module)
- Signal routing for key digital lines between modules

Care was taken to minimize trace length and avoid sharp angles, ensuring signal integrity and reducing EMI.

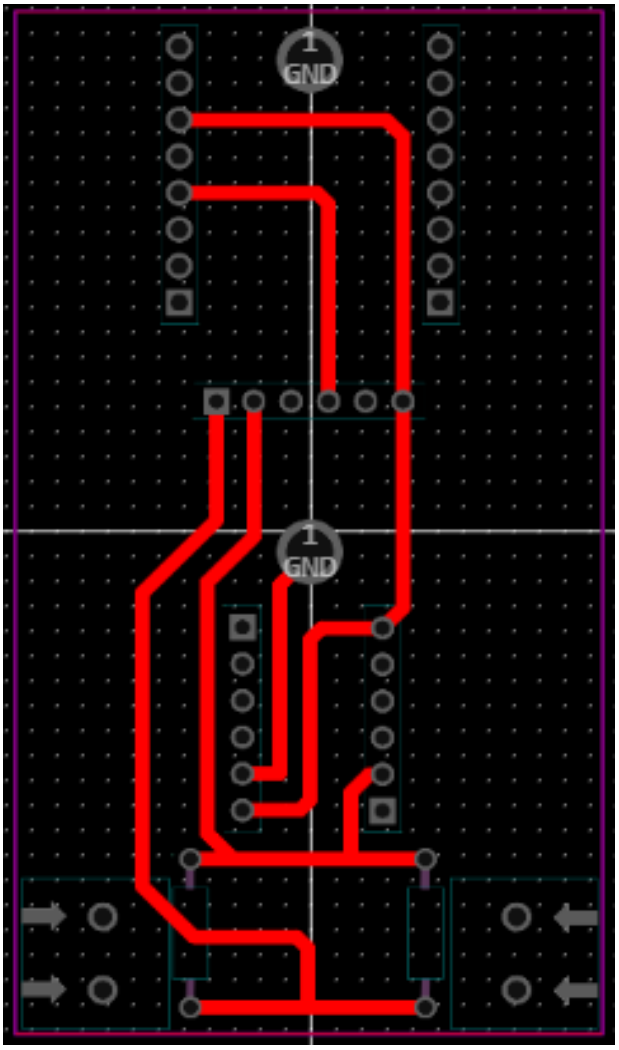


Figure 3.28: Top Copper Layer

B) Bottom Copper Layer (Blue Traces)

The bottom layer is mainly used for:

- **Ground routing**
- **Power distribution lines** (e.g., 5V and GND to slave CAN modules)
- Supporting vias for inter-layer connectivity

This separation of digital, power, and ground paths improves overall electrical stability and modularity.

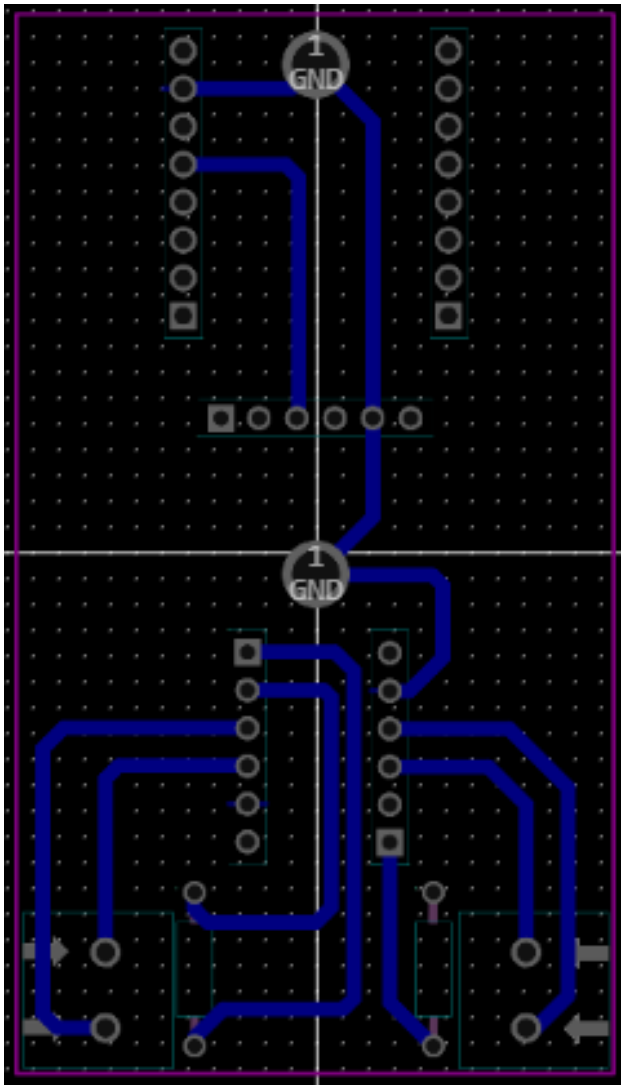


Figure 3.29: Bottom Copper Layer

C) Top Silkscreen Annotations

To enhance usability and modularity, the top silkscreen layer was labeled with identifiers for all major functional blocks. These include:

- **ESP32-C3 MINI** – The onboard microcontroller acting as a web server.
- **Master CAN Module** – Receives AI results from the STM32 via the CAN bus.
- **Slave CAN Module 1 & 2** – Expansion slots for additional CAN-based nodes.
- **CAN HIGH / CAN LOW** – Clearly marked differential transmission lines.
- **Branding & Version Marking** – Including the DiagnoDisk Hub logo and version reference

This labeling improves readability during assembly, debugging, and potential field deployment.

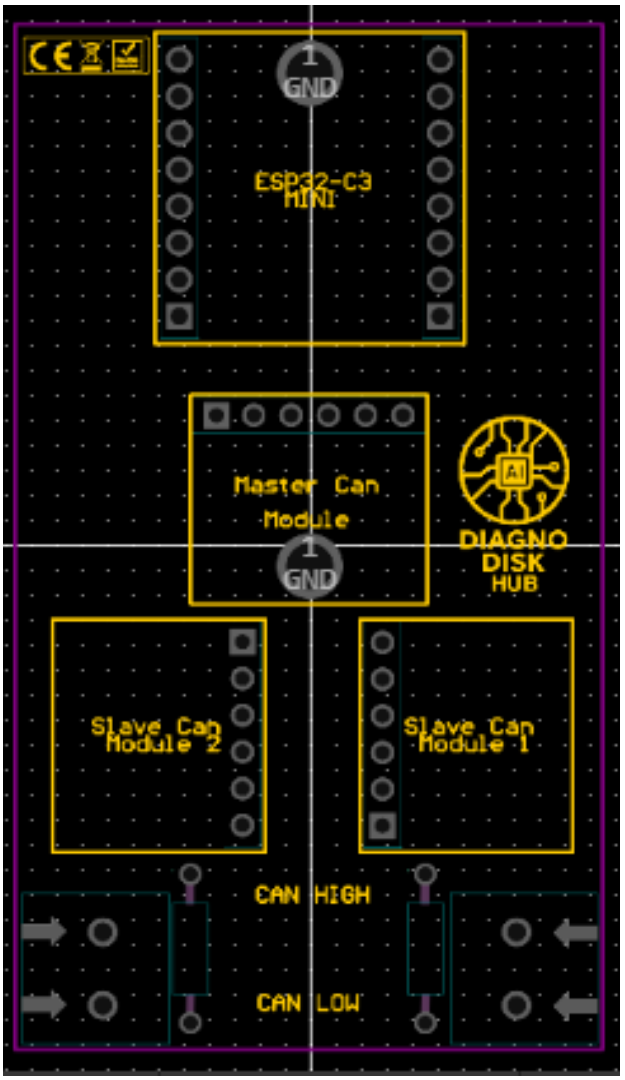


Figure 3.30: Top Silkscreen

D) Stack Layers

This rendering shows the finalized board, including silkscreen markings for all key modules. The clean component layout reflects the modular design philosophy—ESP32 at the top, central CAN master interface, and slave expansion channels below—facilitating plug-and-play diagnostic expansion.

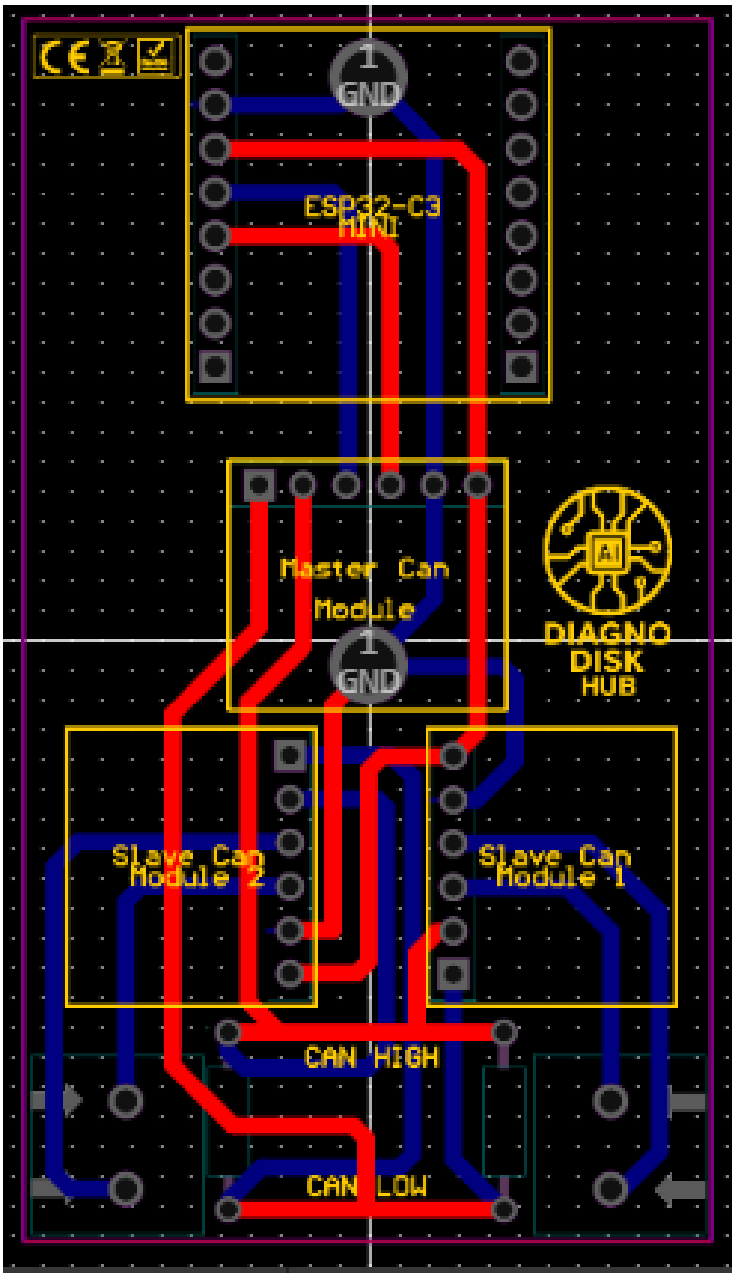


Figure 3.31 All Layers Routing of the Diagno Disk Hub PCB

E) 3D View of the Diagno Disk Hub PCB

After completing the design, validation, and fabrication stages, the secondary PCB—responsible for wireless result transmission—was assembled and tested both mechanically and electrically. It was manufactured as a 2-layer rigid PCB using FR4 substrate, with a blue solder mask and white silkscreen**,** providing a clean, professional appearance. The compact layout was specifically engineered for modularity and ease of integration into broader industrial systems. Despite its simplicity, the board's design ensures reliable CAN communication **and** Wi-Fi connectivity, making it suitable for real-world diagnostic deployment.

Visible Components & Roles :

- **Center:** ESP32-C3 Mini that acts as a Wi-Fi web server and CAN receiver.
- **Top:** CAN bus master module receives data from STM32.
- **Bottom corners:** Two slave CAN modules – for distributed diagnostics or expansion.
- **Silkscreen labels:** CANH, CANL, ESP, and module roles clearly marked for debugging and modular use.

To showcase this compact design, annotated 3D renders and top-layer silkscreen views were generated, illustrating the system layout and modular structure.

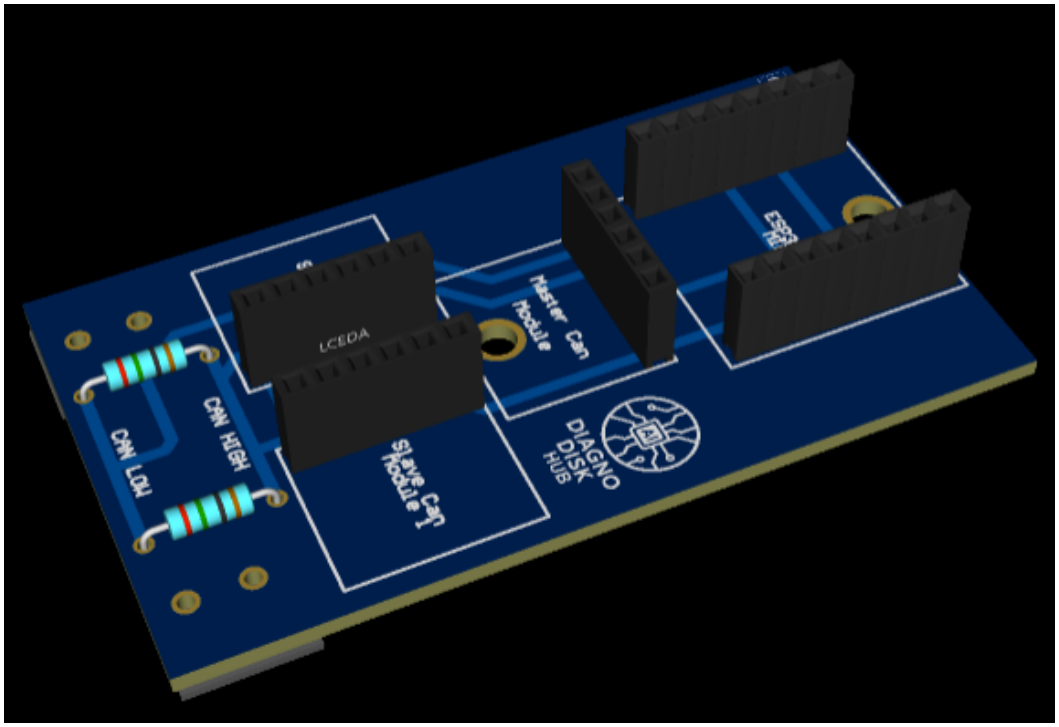


Figure 3.32: 3D Rendered View of the Diagno Disk Hub PCB from the side

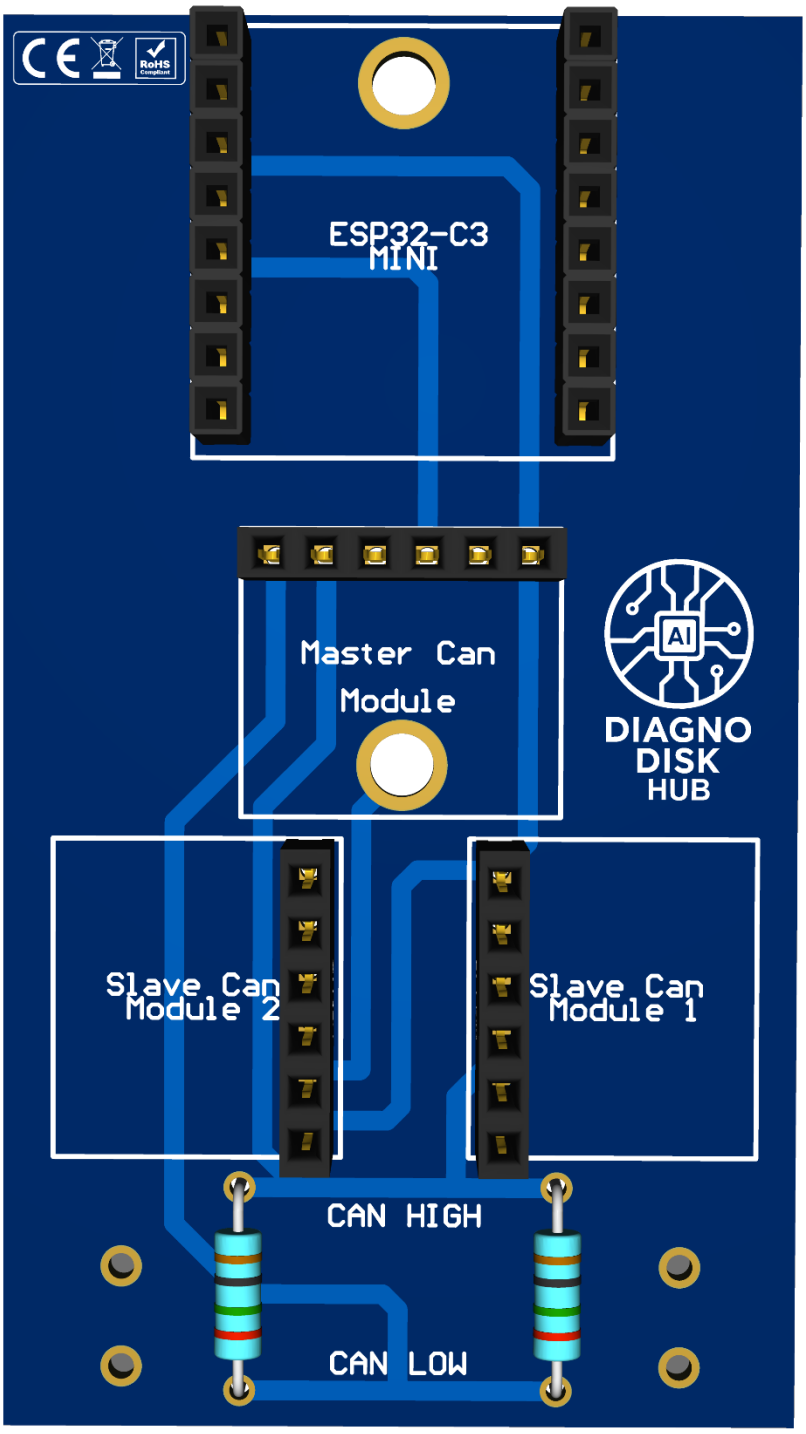


Figure 3.33: 3D Rendered View of the Diagno Disk Hub PCB (ESP32 + CAN Interface

3.5. Firmware Implementation and System Logic

This section outlines the embedded logic that powers the entire system from low-level peripheral configuration to real-time It highlights the sequence and structure of operations, from sensor initialization and data acquisition to AI-based inference, display rendering, and CAN-based wireless

transmission.

The firmware was written in C using STM32CubeIDE and relies on the HAL (Hardware Abstraction Layer) for peripheral interfacing. The workflow integrates I2C communication, DMA for efficient data transfer, external interrupts, NanoEdge AI inference, and multiple output modalities (OLED display and CAN transmission) and remote data transmission via CAN bus. This section outlines the logical flow of the firmware, initialization steps, task sequencing, and code structure, with diagrams to visualize core processes.

3.5.1. System Initialization Flow

The firmware began as a prototype on a Nucleo board and later migrated to our custom 6-layer STM32F103RB PCB. The system was programmed and debugged using **ST-Link** and **Serial Wire Viewer (SWV)**, The firmware is configured to use an external 8 MHz high-speed crystal oscillator (HSE), scaled internally via PLL to 72 MHz, enabling USB virtual COM operation and maximizing performance.

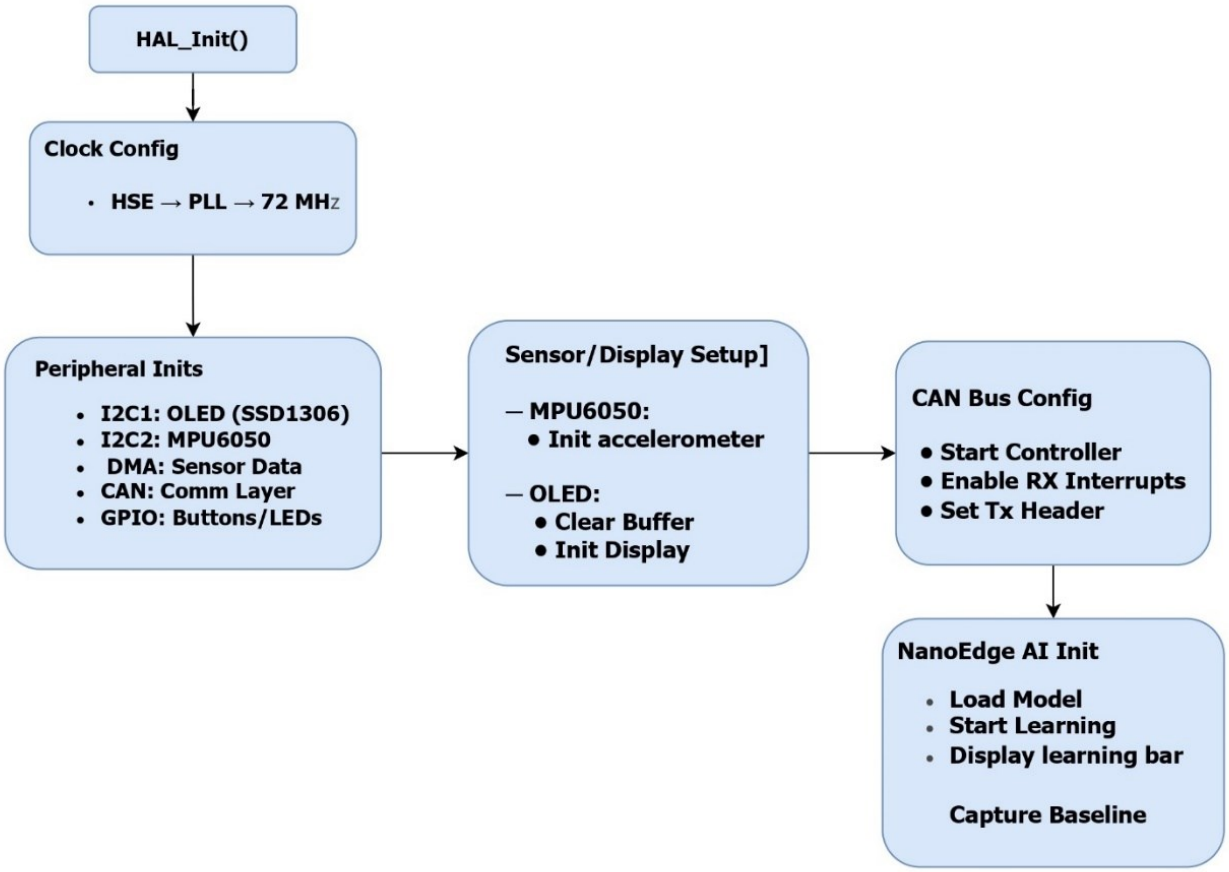


Figure 3.34: Firmware Initialization Flow

A visual overview of how the MCU boot sequence proceeds from HAL_Init() through peripheral setup and library initialization.

3.5.2. Peripheral Configuration

All peripherals were configured using STM32CubeIDE. The I2C2 bus handles the MPU6050 sensor, I2C1 is reserved for the SSD1306 OLED. CAN is configured in normal mode, using standard ID messages with receive FIFO interrupts enabled. GPIOs are assigned for the tactile switch (used to toggle operating modes) and interrupt detection on the MPU6050's INT pin **Figure 3.34** below shows the peripheral on the STM32F103RB MCU.

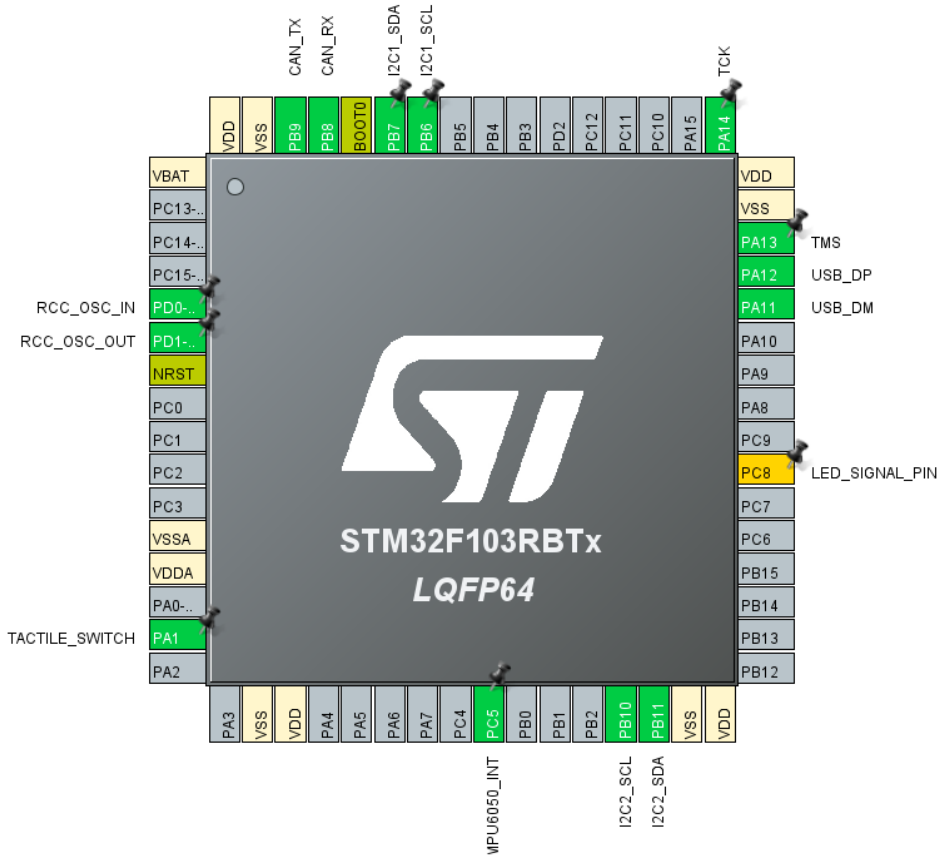


Figure 3.35: STM32F103RB pin configuration in Cube IDE

3.5.3. Task Flow in Main Loop

The system operation follows a structured workflow as illustrated in (**Figure 3.35) t**he firmware operates in two modes controlled by **device_job:**

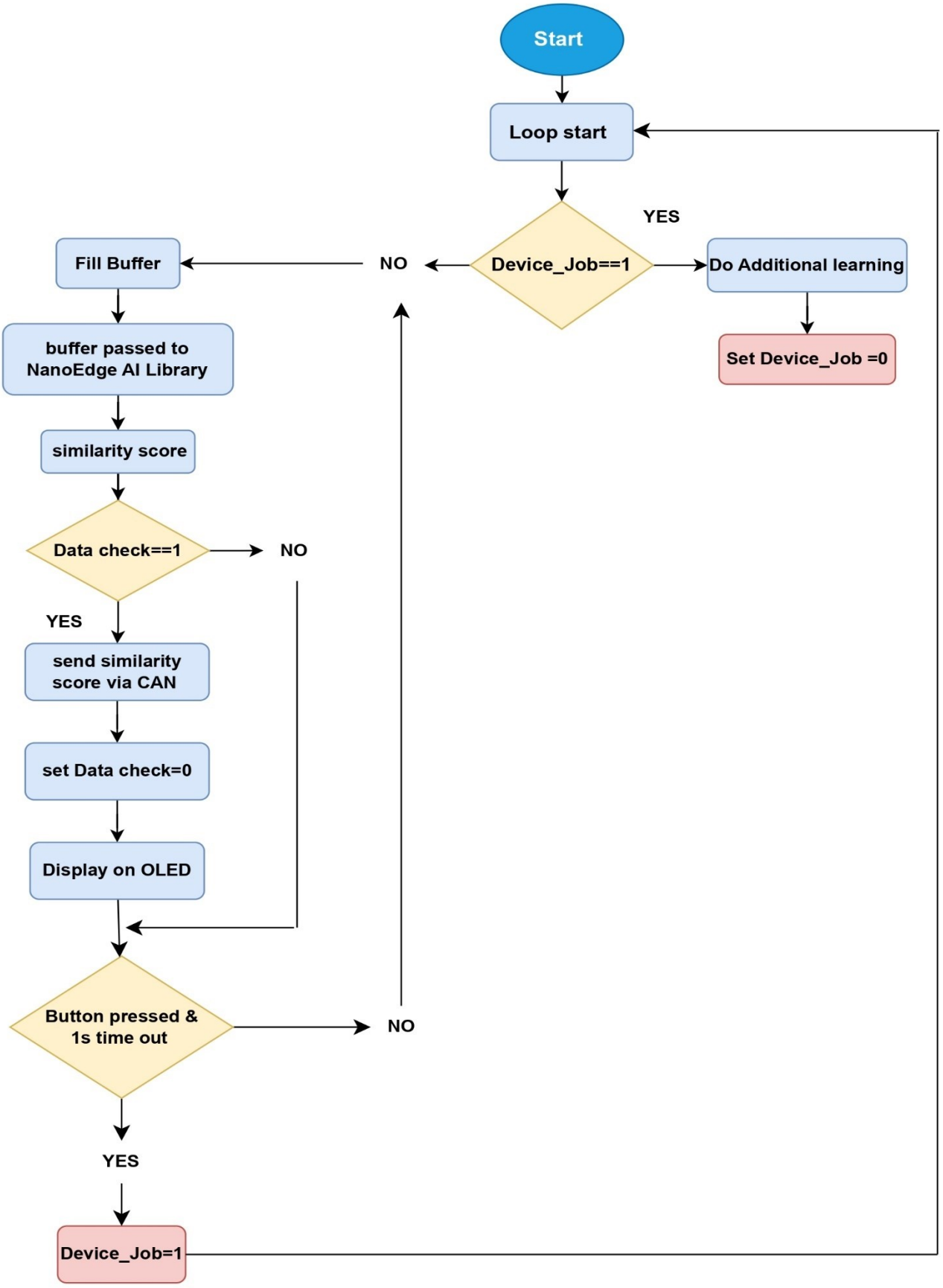


Figure 3.36: Main loop Flowchart

A) Monitoring Mode (device_job = 0):

in this default mode, the device performs continuous real-time monitoring:

- Sensor data is acquired and buffered.
- The buffered data is passed to the NanoEdge AI library for anomaly detection, which returns a similarity score.
- If the data is validated for transmission (refer to **Figure 3.41**), the similarity score is sent via the CAN bus to external systems.
- The similarity score is also displayed on the OLED screen in real-time.
- The system monitors the push button to check for user input (see **Figure 3.39: Push Button Flowchart**). If the button is pressed and held for more than 1 second, the system prepares to switch to learning mode.

B) Learning Mode (device_job = 1):

In this mode:

- The device enters the NanoEdge AI learning phase, during which it observes clean sensor data to establish or update the baseline behavior of the system.
- Once learning is complete, the system automatically switches back to Monitoring Mode for continued operation.

3.5.4. Sensor Interface with DMA: MPU6050 Data Handling

The diagram **Figure 3.36** below illustrates the two-phase operation of the MPU6050 motion sensor:

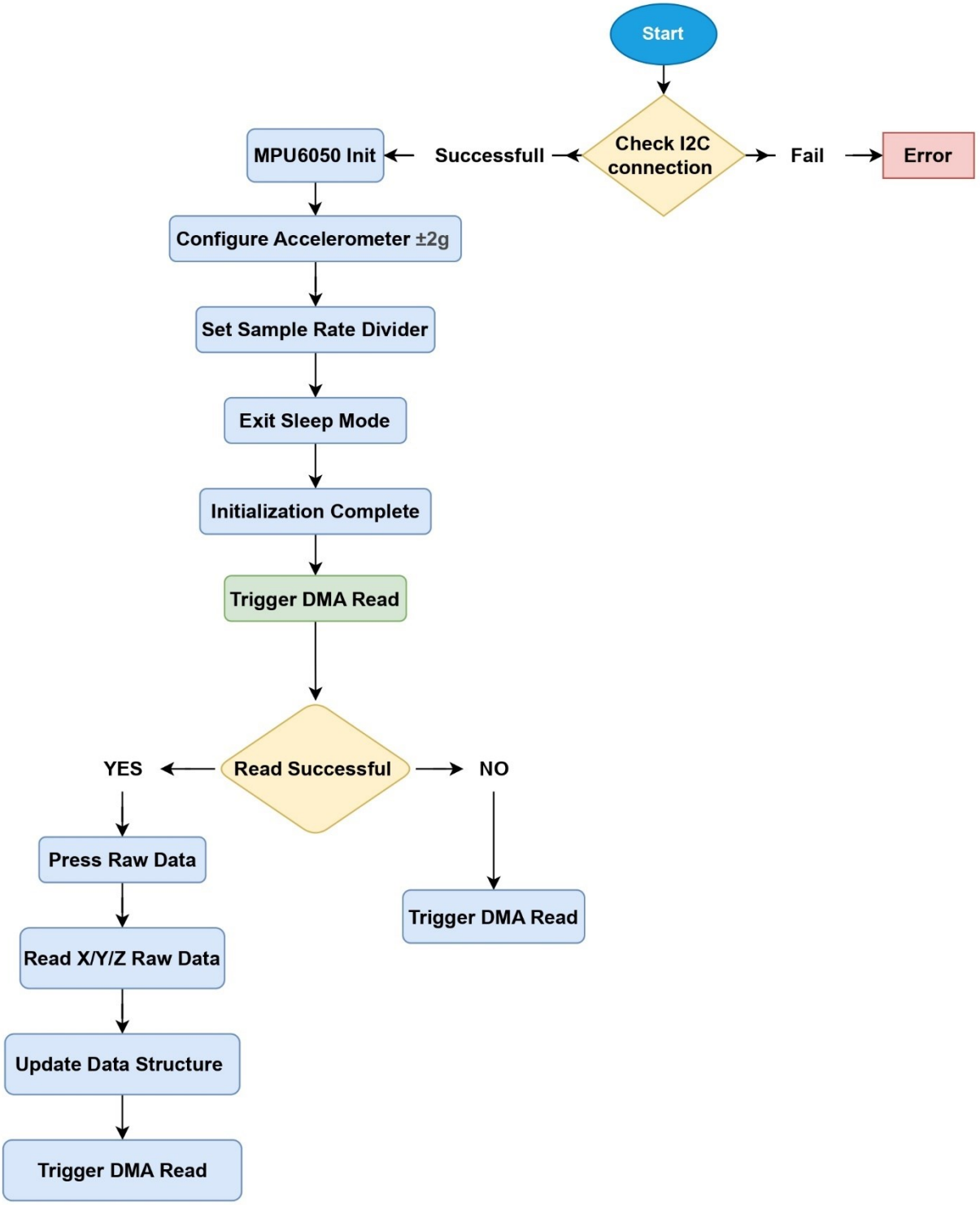


Figure 3.37: MPU6050 Initialization and Data Acquisition Workflow

To avoid CPU blocking during high-rate sampling, the MPU6050 was interfaced using DMA. The interrupt line was activated on every data-ready event (INT pin), which triggered a DMA transfer from sensor to memory.

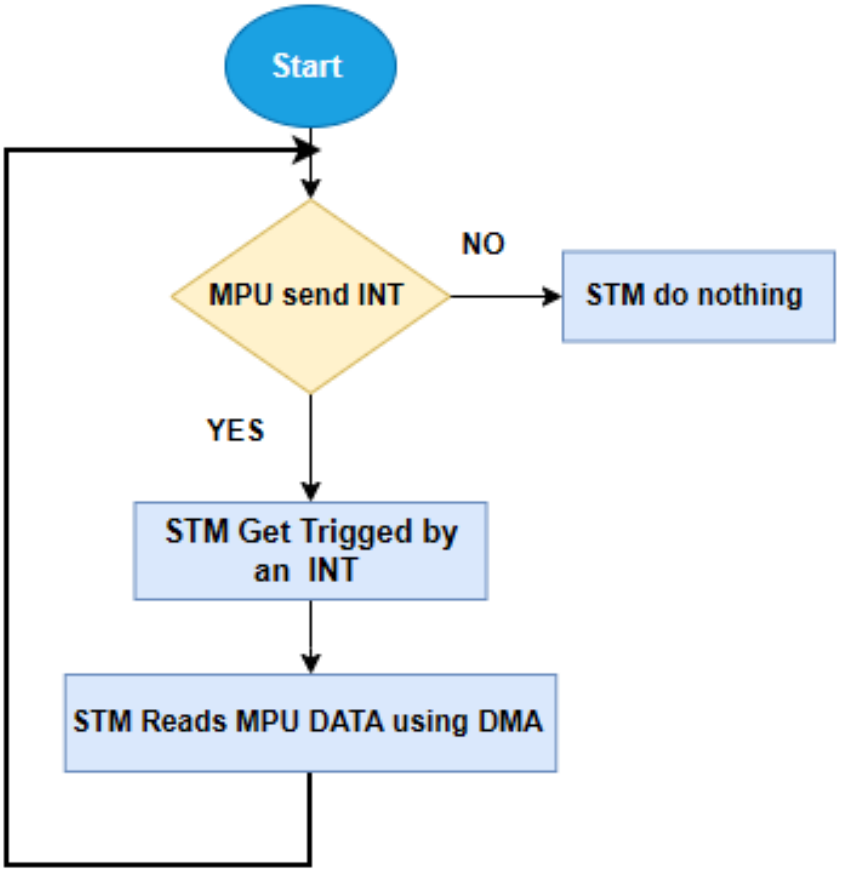


Figure 3.38: Interruption workflow

3.5.5. Embedded AI Integration: NanoEdge Inference Logic

After training in NanoEdge AI Studio, the best library (ICM) was exported as a lightweight .c/.h pair and integrated directly into the STM32 project.

The AI function receives 3-axis acceleration data as input, processes it internally (FFT, feature extraction), and returns a **similarity score** (0–100%). This score is used for OLED display, CAN transmission, and USB logging. All the details are in the NanoEdge AI ***section 3.6***

This flowchart (Figure 3.38) outlines the interrupt-driven button handling that triggers mode transitions in the system (Main loop):

Non-Blocking Detection:

- A hardware interrupt fires on button press (no polling delays).
- Sets a flag (**button_pressed = 1)**

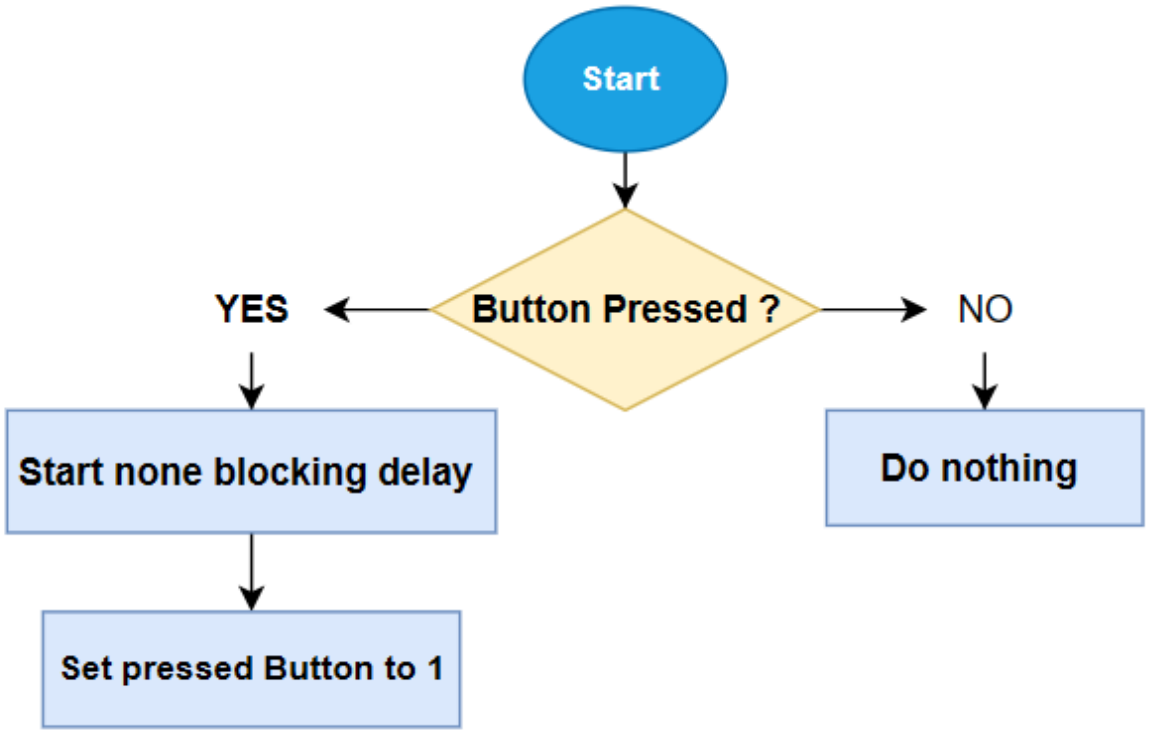


Figure 3.39: Button Control Workflow

After the button is pressed it will set Device job=1 and that will activate the NanoEdge AI learning Function.

3.5.6. Display Output : OLED Feedback

The SSD1306 OLED module was used to display:

- Similarity score bar
- Real-time waveform graph

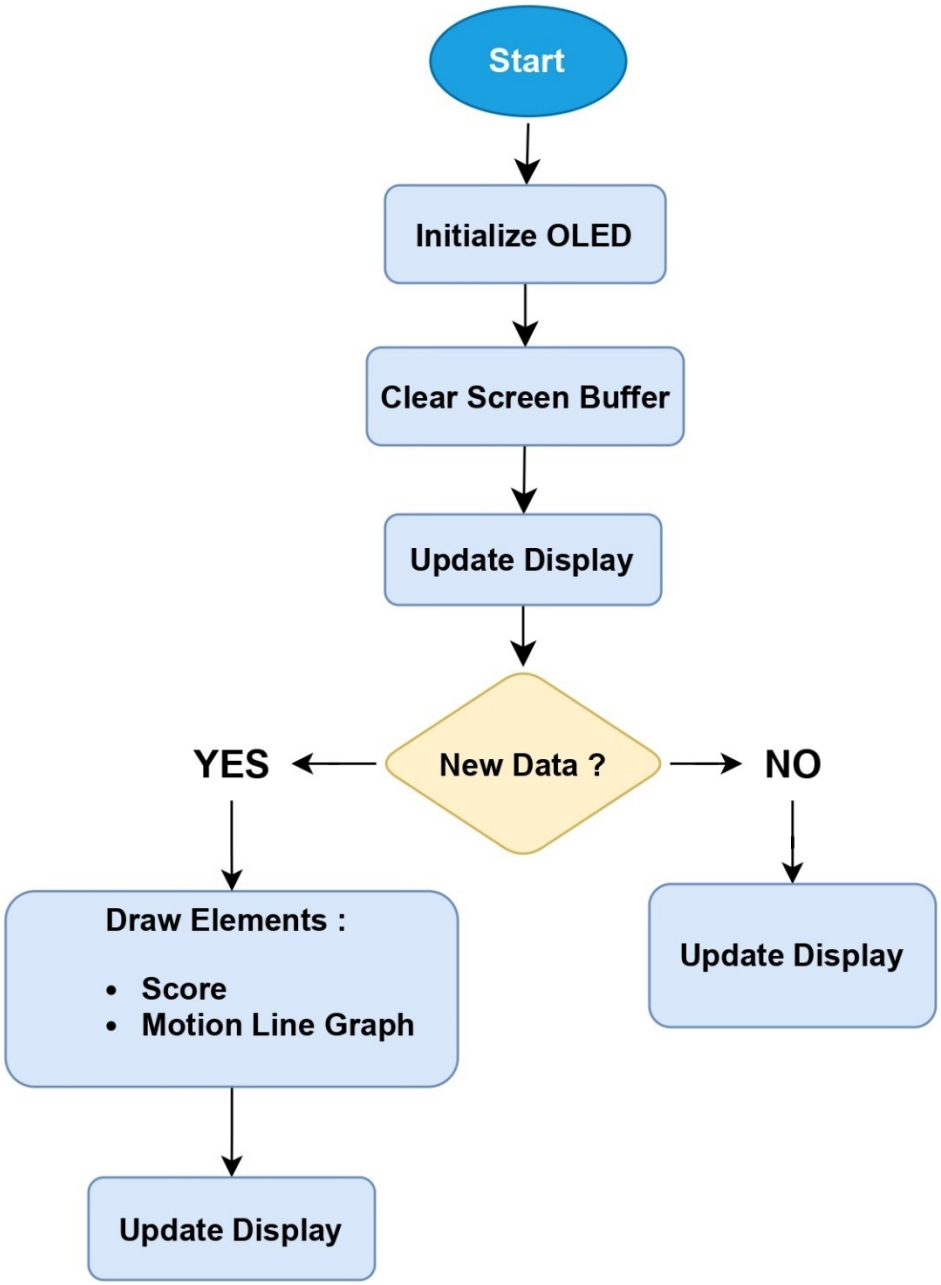


Figure 3.40: Display Output Workflow

3.5.7. CAN Transmission & ESP32 Web interface

In the CAN communication setup with the ESP32, the ESP32 sends a request to the Main Board via the CAN bus. The STM32 then responds with the current similarity score. Acting as a web server, the ESP32 initializes the CAN header once, transmits the request, and continuously listens for incoming packets. It parses the received scores and displays the results on a real-time dashboard accessible over Wi-Fi.

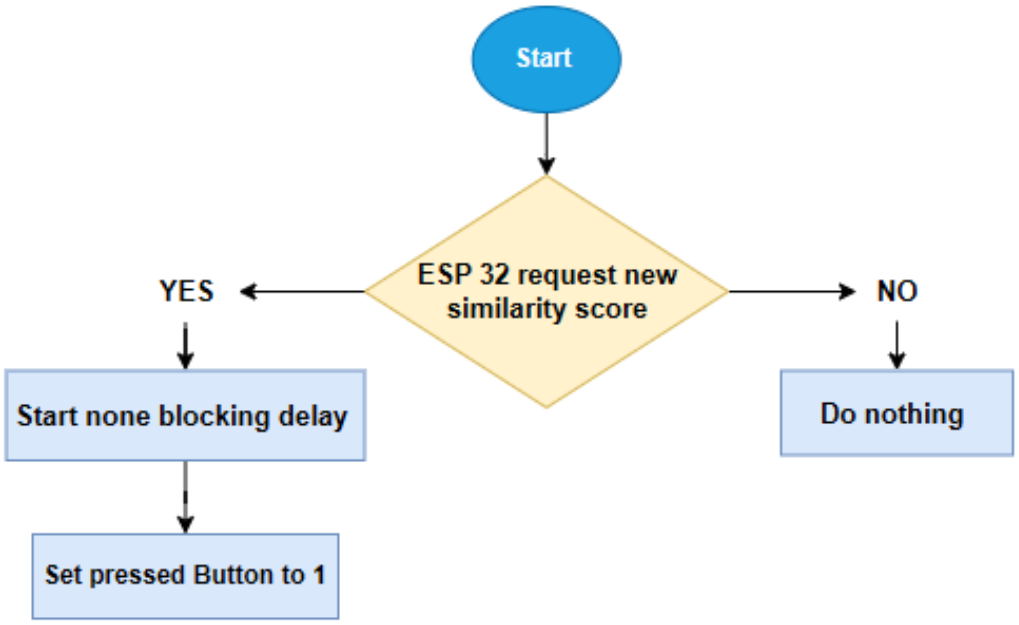


Figure 3.41: The similarity request

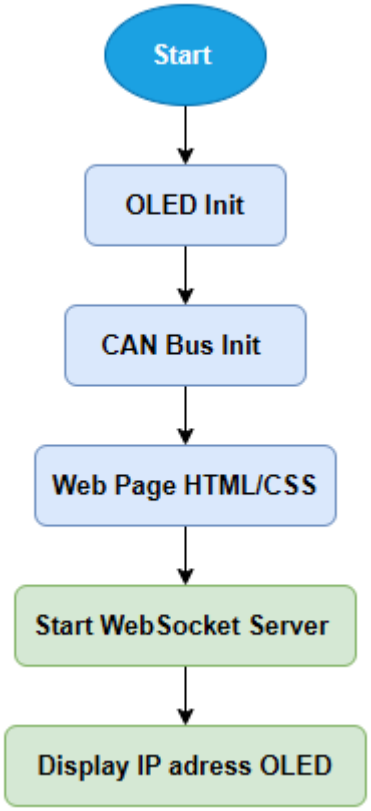


Figure 3.42: CAN Transmission Workflow

The following flowchart illustrates the main loop executed by the ESP32 in the project. In this loop, the ESP32 continuously communicates with the STM32 microcontroller to request and retrieve data, which is then broadcasted to connected clients via WebSocket in JSON format and used to update the web interface in real-time.

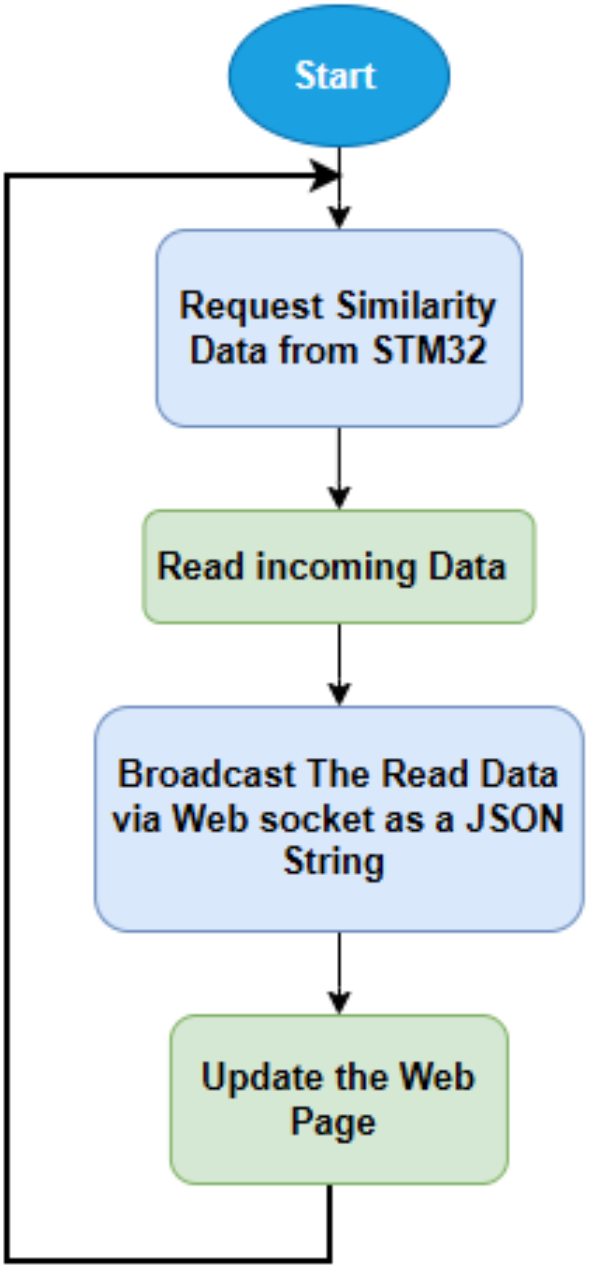


Figure 3.43: ESP32 Continuous Data Acquisition and WebSocket Process

3.6. NanoEdge AI Integration: Training and Evaluation

This section highlights the practical implementation of the AI component using **NanoEdge AI Studio**. After collecting and formatting our motion signals from the embedded system, we used NanoEdge AI Studio to import, preprocess, and train models tailored for our project's edge environment. This part documents that journey, beginning with how the signals were prepared and imported, then moving into benchmarking and model selection, and finally detailing validation results that confirmed our final choice.

3.6.1. Signal import summary

Once the Nano-Edge AI Studio project was configured, the next critical step was to provide it with real-world motion data representative of both normal and faulty behavior. The objective of this stage was to allow the AI to distinguish clearly between healthy system operation and potential signs of failure or disturbance. To achieve this, two distinct datasets were prepared:

Before beginning the data upload process, NanoEdge AI Studio provides a dedicated workspace to categorize signals. As shown in **Figure 3W,** separate channels are reserved for "Regular signals" and "Abnormal signals." This structure supports a clean and organized import workflow, ensuring each type of signal is treated accordingly for training and evaluation.

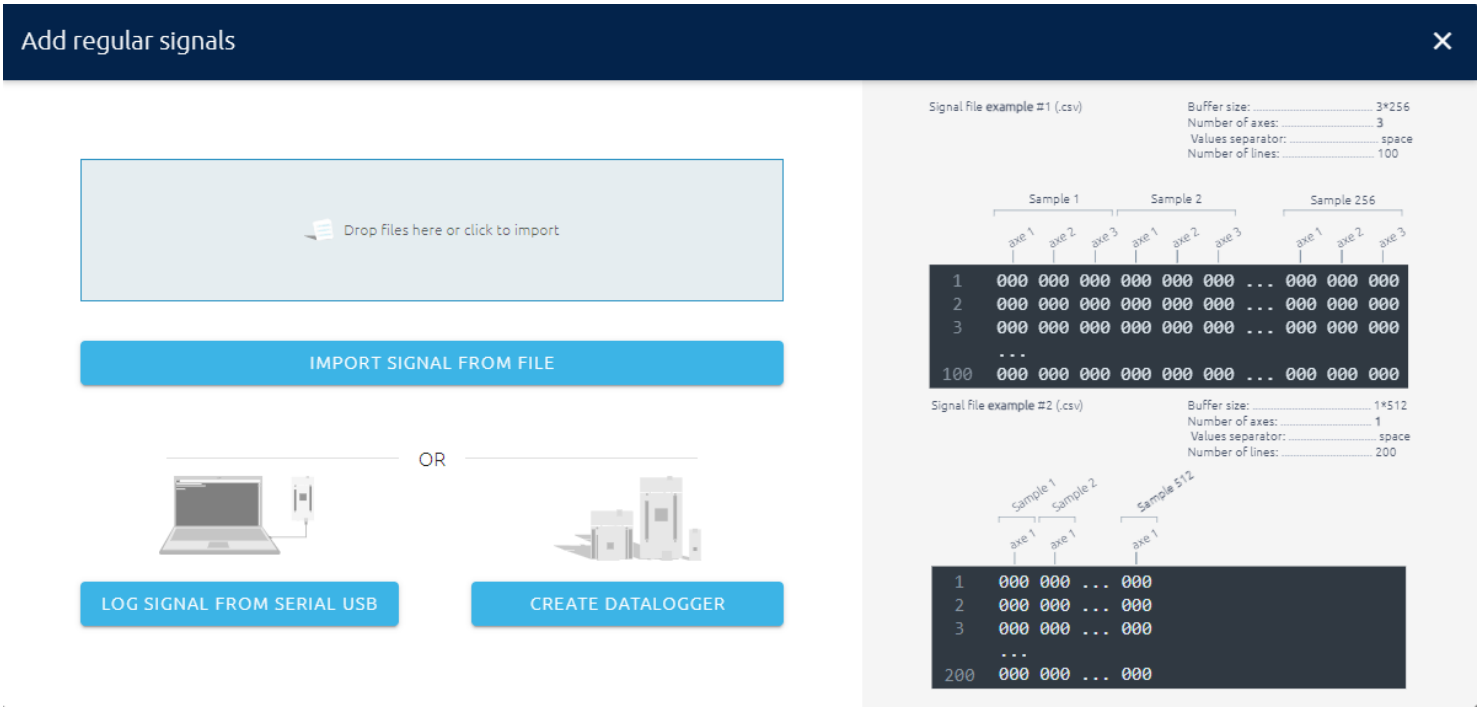


Figure 3. 44: Initial regular Signal Import Interface in NanoEdge AI Studio

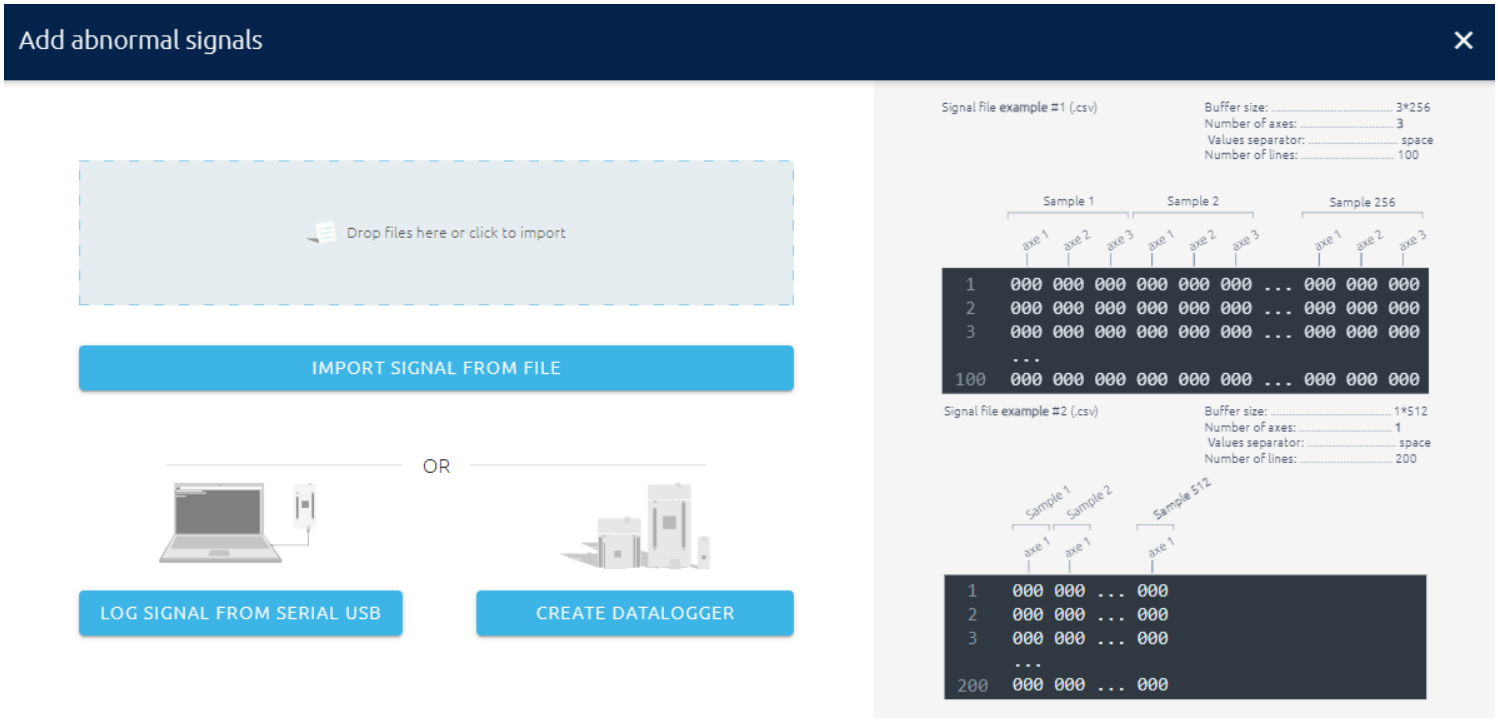


Figure 3.45: Initial abnormal Signal Import Interface in NanoEdge AI Studio

A) Regular (Nominal) Signals

These signals were collected while the system was functioning under stable, normal conditions. They served as the *baseline behavior* from which the AI would learn. The idea was to provide multiple examples of what "healthy" motion patterns looked like so the model could build a solid internal understanding of normality.

In the NanoEdge AI Studio interface, each signal appears as an orange waveform, representing a snapshot of motion data captured across the X, Y, and Z axes. We also enabled **Fast Fourier Transform (FFT)** preprocessing, which allowed us to examine the signals not just in the time domain, but also in the frequency domain. This provided deeper insights into the periodic characteristics of the motion data.

As shown in **Figure 3.**, the nominal signals displayed consistent, smooth periodic patterns across all axes, without significant noise or fluctuations. The FFT plots confirmed this stability, with predictable and narrow frequency distributions.

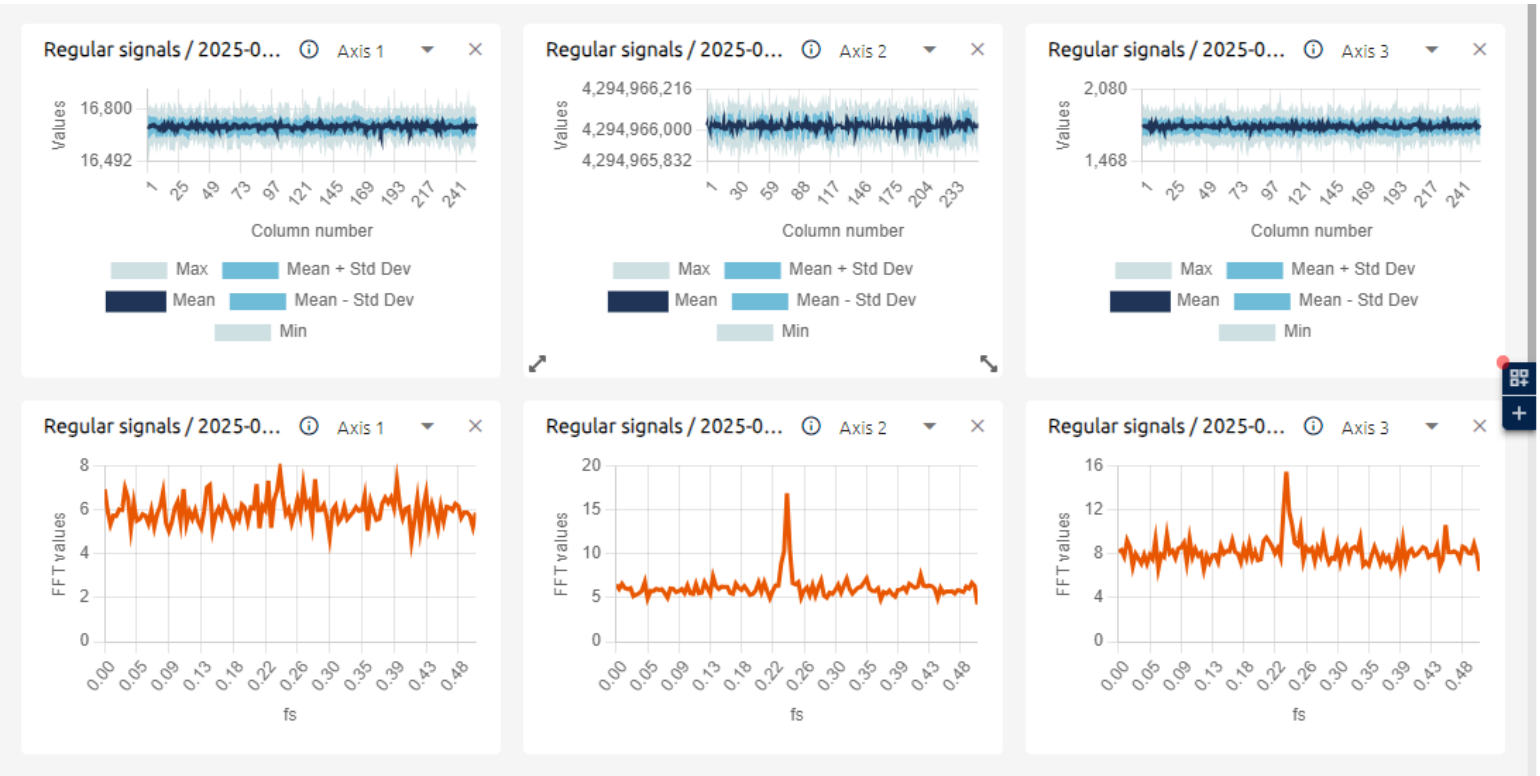


Figure 3.46: Regular signal overview in NanoEdge AI Studio

B) Abnormal (Fault-Induced) Signals

To test the model's ability to detect irregularities, we collected a second set of data under intentionally disturbed conditions. These included introducing physical vibrations, mechanical imbalance, or loose structural components. Importantly, these abnormal signals were **not** used for training; they were set aside for *validation and testing* only.

In contrast to the regular dataset, the abnormal waveforms exhibited visibly erratic behavior—higher amplitudes, irregular oscillations, and unpredictable frequency content. The FFT plots revealed scattered spectral components, indicating the presence of dynamic faults or instability in the system.

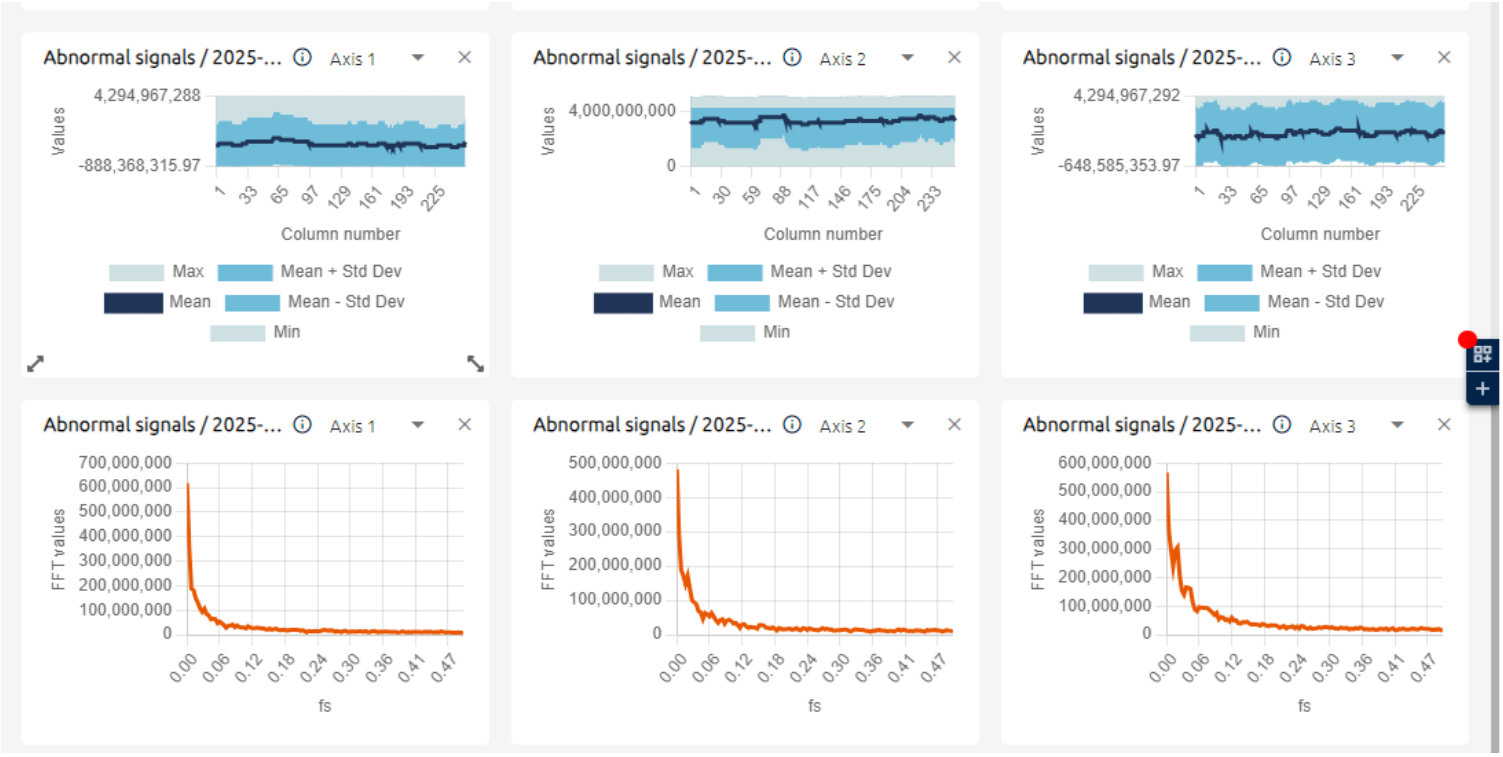


Figure 3.47: Abnormal signal overview in NanoEdge AI Studio

3.6.2. Benchmarking & Model Selection

Once the signal dataset was cleaned and uploaded, NanoEdge AI Studio performed a comprehensive benchmarking procedure to evaluate which machine learning library best matched the characteristics of our application. The benchmarking tool automatically tests thousands of model combinations, optimizing for accuracy, speed, and memory usage within the constraints of the STM32F103RB microcontroller.

A) Search Space and Benchmarking Process

The benchmarking process explored **19,852 AI libraries** in under **9 minutes**, systematically evaluating each library's performance against the training and validation datasets. The testing engine filtered candidates based on their ability to:

- Recognize anomalies in real-time
- Operate within strict memory constraints
- Deliver high detection reliability under noise or slight variations

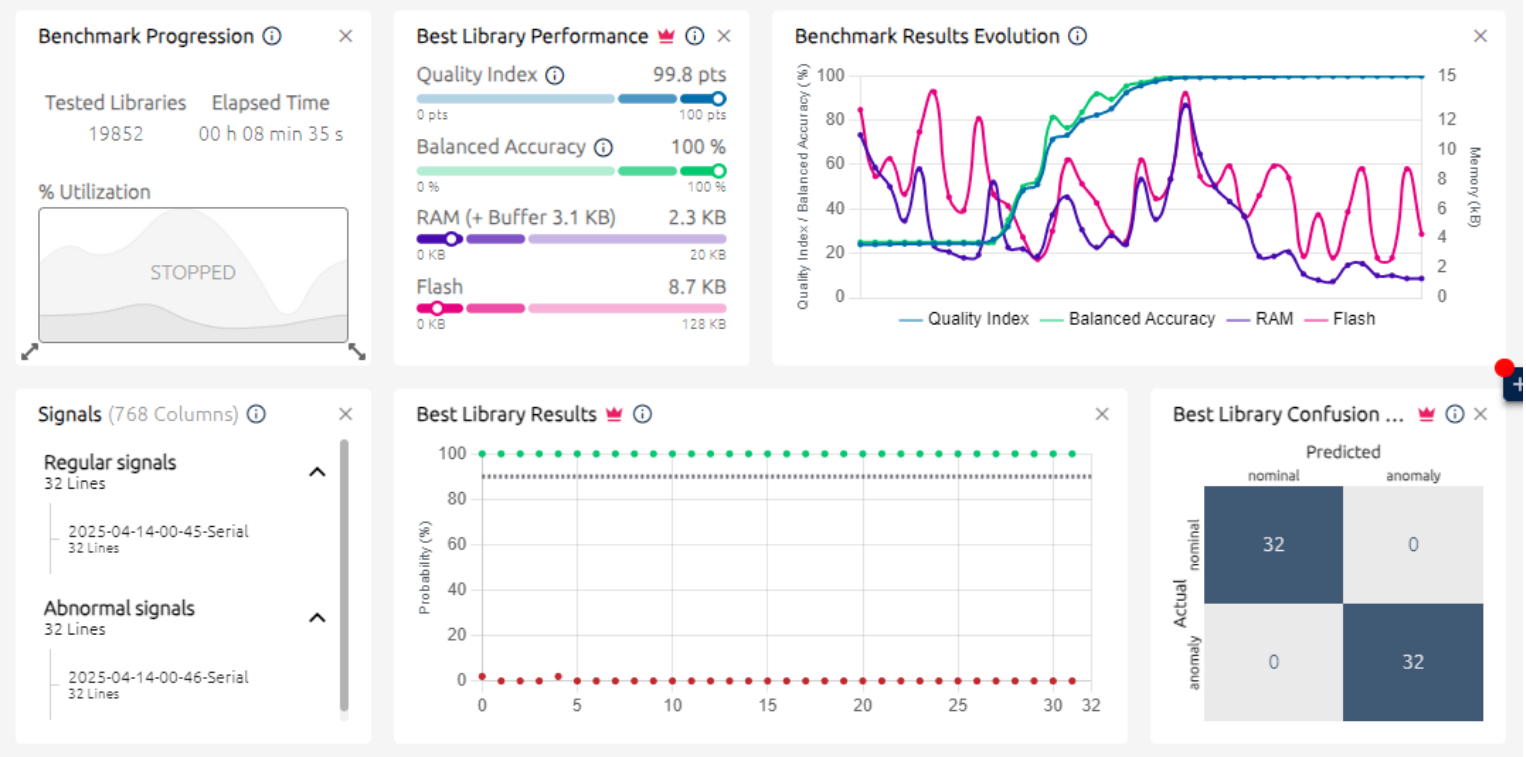


Figure 3.48: Benchmark Progression overview in NanoEdge AI Studio

➢ Models Tested

NanoEdge AI Studio benchmarked the following four model families:

Model	Description	Role
ICM *(Isolation Covariance Matrix)*	Learns statistical patterns of normal data and flags outliers based on covariance.	Detecting sudden, unpredictable anomalies (e.g., loose motor parts)
IPCA *(Incremental Principal Component Analysis)*	Compresses data by extracting main features, then updates models over time.	Detecting slow degradation (e.g., imbalance, wear)
ZSM *(Zero-Shot Model)*	Pre-trained, general-purpose model that works immediately.	Fast deployment and prototyping, minimal training
MML *(Miniaturized Machine Learning)*	Lightweight neural network optimized for microcontrollers.	Simultaneous anomaly scoring and state classification

Table 3.1: Summary Comparison of ICM, IPCA, ZSM, MML

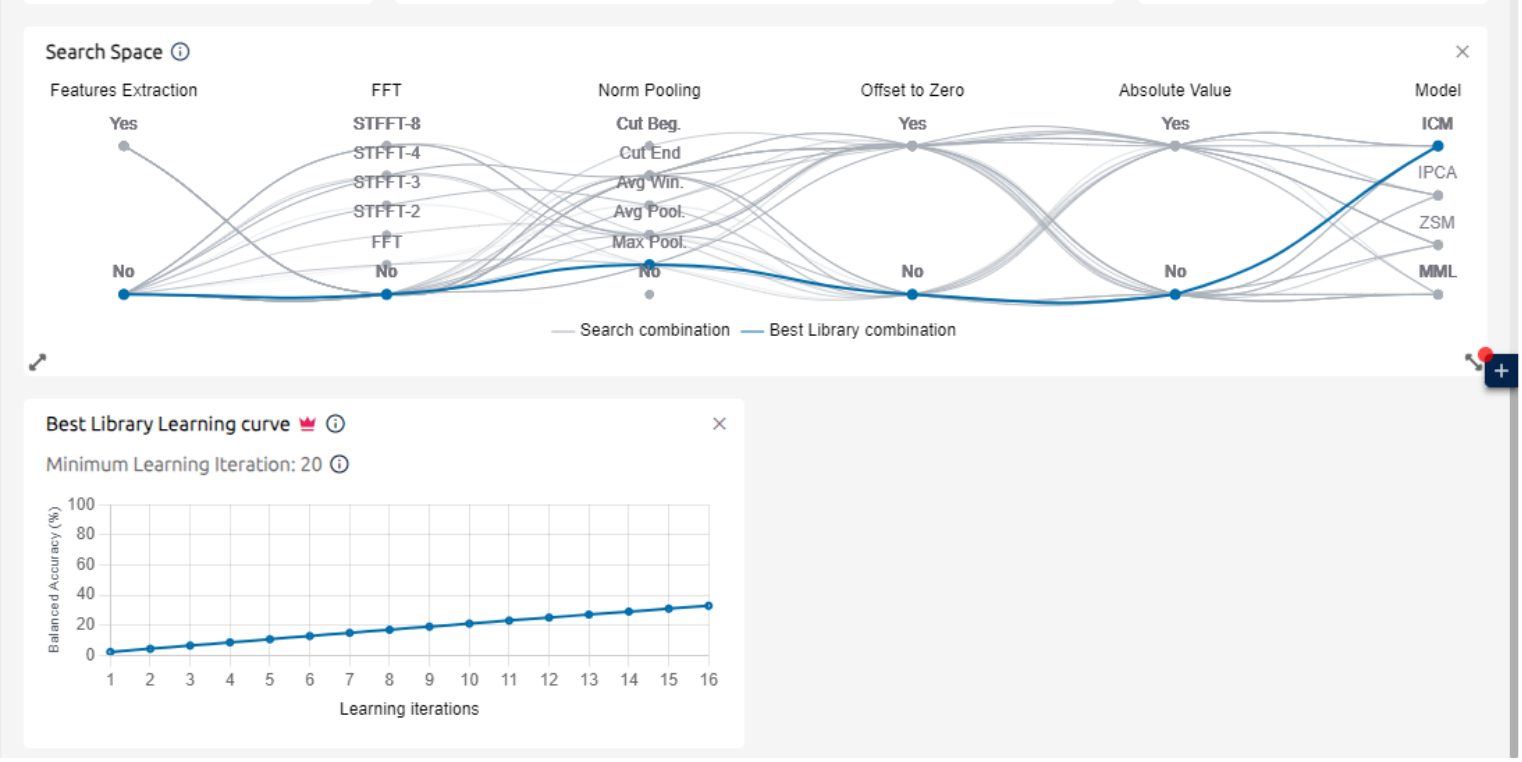


Figure 3.49: Search Space overview in NanoEdge AI Studio

3.6.3. Model Validation and Library Selection

After benchmarking more than 19,000 libraries across four AI model ***(ICM, IPCA, ZSM, MML),*** NanoEdge AI Studio automatically shortlisted 39 top-performing libraries based on metrics like inference speed, memory usage, and quality index. However, automatic rankings alone are not always sufficient especially for real-world applications where consistency**,** separation of states, and practical interpretability matter most.

.

Benchmark
2025-04-14-00-47-Benchmark
+ NEW EXPERIMENT
Select at least one library to start a new experiment.

	Library ID	Quality Index	Balanced Accuracy	RAM	Flash	Minimum Learning Iteration	Execution Time	Model	Report	Serial Emulator	Deployment Library
☐	30	99.8 pts	100.00 %	1.3 KB	8.7 KB	20	0.4 ms	MML			
☐	36	99.8 pts	100.00 %	1.3 KB	4.3 KB	20	0.7 ms	ZSM			
☐	33	99.8 pts	100.00 %	1.5 KB	2.7 KB	20	0.7 ms	ZSM			
☐	25	99.8 pts	100.00 %	1.5 KB	2.7 KB	20	0.1 ms	ZSM			
☐	29	99.8 pts	100.00 %	2.2 KB	5.8 KB	20	1.6 ms	MML			
☐	39	99.8 pts	100.00 %	2.3 KB	8.7 KB	20	0.4 ms	ICM			
☐	38	99.8 pts	100.00 %	1.1 KB	2.7 KB	20		IPCA			
☐	35	99.7 pts	100.00 %	1.2 KB	5.6 KB	20		IPCA			
☐	17	99.7 pts	100.00 %	1.6 KB	2.8 KB	20		ZSM			

Figure 3.50: Validation overview in NanoEdge AI Studio

To identify the most stable and accurate option, each model type was tested using NanoEdge's built-in **real-time emulator**, which connects to the STM32 board over a serial port and visualizes **similarity scores** between current signals and the "learned" baseline behavior.

We structured each test session into 4 physical states:

- **Idle Zone (Purple/Rose color)**

  This state reflects the condition used during the learning phase—no vibration, no touch. All models were expected to classify this region as *"regular."*
- **Vibration Zone (Blue)**

  We gently tapped the surface near the sensor, introducing consistent, mild vibrations. If such signals were not part of the learning set, they should be flagged as *"anomalous."*
- **Shaking Zone (Yellow)**

  Here, the entire board was shaken, simulating severe mechanical disturbance. This behavior clearly deviates from the learned state and must be identified as *"anomalous."*
- **Return to Idle (Orange)**

  The device was returned to a steady, motionless state. This final zone tests whether the model can recognize the return to normal conditions.

A) Results and Model Selection

Despite strong theoretical performance during benchmarking, models like **ZSM**, **MML**, and **IPCA** failed to distinguish between the above states in practice. As shown in the emulator graph below, these models maintained high similarity percentages across all zones, falsely classifying every state even violent shaking as *"regular."*

Figure 3.51: Failed Real-Time Classification by ZSM Model

No meaningful drop in similarity observed during vibration or shaking. This renders the model ineffective for anomaly detection.

In contrast, **ICM**, though **not ranked among the "Best Libraries" by NanoEdge**, demonstrated outstanding real-time performance. It effectively detected both moderate and severe anomalies, showing clear separation between signal regions. The emulator results showed:

- High similarity during idle zones (purple, orange)
- Sharp similarity drops in blue and yellow zones (vibration and shaking)
- Fast recovery to "regular" classification post-anomaly

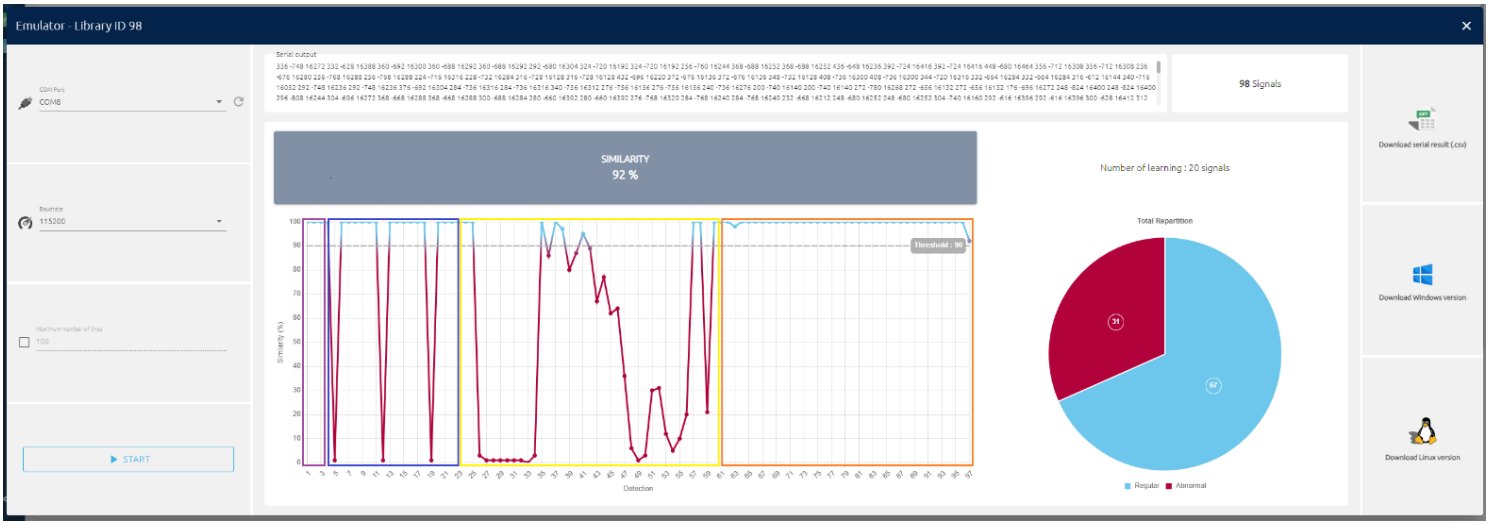


Figure 3.52: ICM Model Validation Showing Accurate State Separation

The blue region (light tapping) and red region (shaking) produced sharp similarity drops, while purple and orange regions (idle) were correctly classified as regular. Overall similarity score: ***92%.***

This outcome illustrates a key insight**:** real-world emulator validation is indispensable. Even models that rank lower on the quality index may outperform others in actual deployment scenarios. In our case, **ICM despite being overlooked by NanoEdge's automated selection, was the most reliable and accurate model.**

Once ICM was chosen, NanoEdge AI Studio automatically generated a comprehensive PDF report for the selected library, including metadata, performance metrics, and function prototypes. This report was archived for documentation and firmware integration purposes.

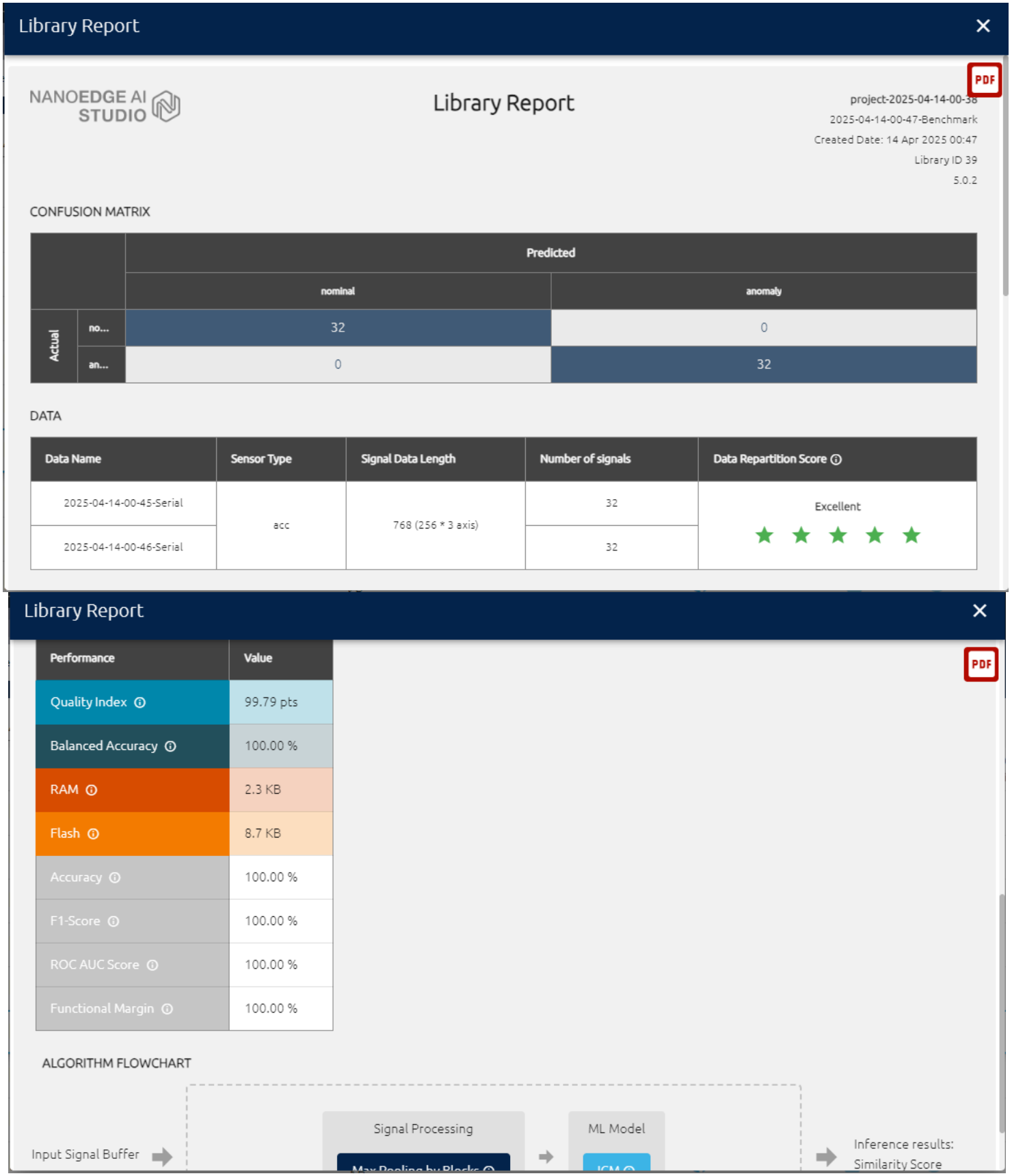


Figure 3.53: Final Library Report (ICM) Generated by NanoEdge AI Studio

3.7. Results and System Evaluation

This final section presents the real-world performance of the complete system, combining AI inference, embedded hardware, and user feedback mechanisms. Tests were conducted across various physical states (normal operation, induced vibration, idle, and shaking) to evaluate how well the system detects changes and displays results.

3.7.1. State Visualization and OLED Feedback

The SSD1306 OLED provided real-time state awareness through two dynamic elements:

- **Score Bar:** Horizontal fill reflecting NanoEdge AI's confidence (0–100%) in the current classification (e.g., 80% → likely suboptimal).
- **Vibration Waveform:** Live plot of processed accelerometer data (1000 Hz update rate).

Figure 3.52, 3.53, 3.54 shows a sequence of display frames captured during a state transition test. As the device moved from idle (no vibration) to light tapping (transient anomaly) to shaking (severe anomaly), the bar and graph both responded with increased activity. When the system returned to stability, the visuals adapted accordingly.

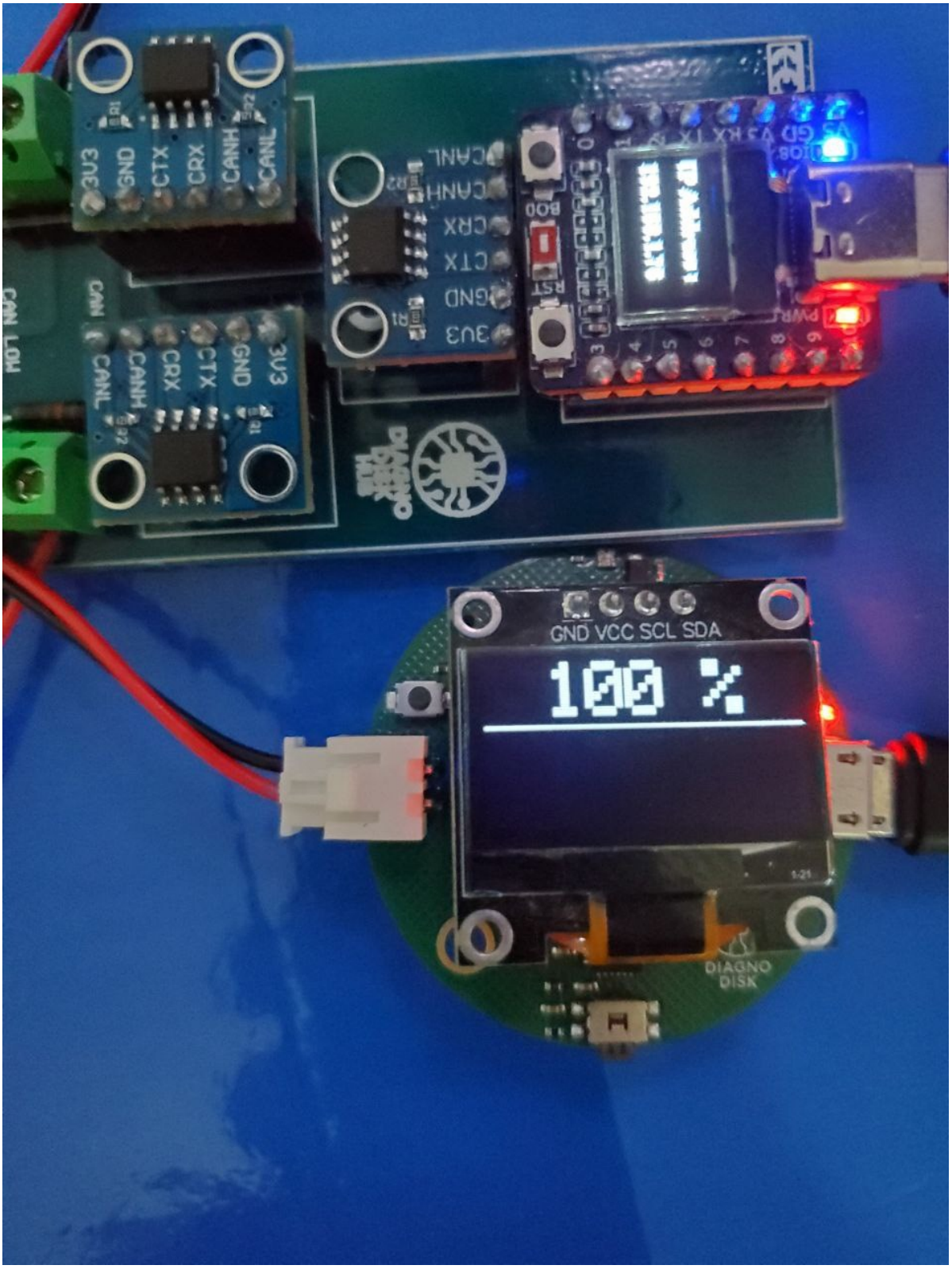


Figure 3.54: OLED Display Frames Under Various Conditions: idle

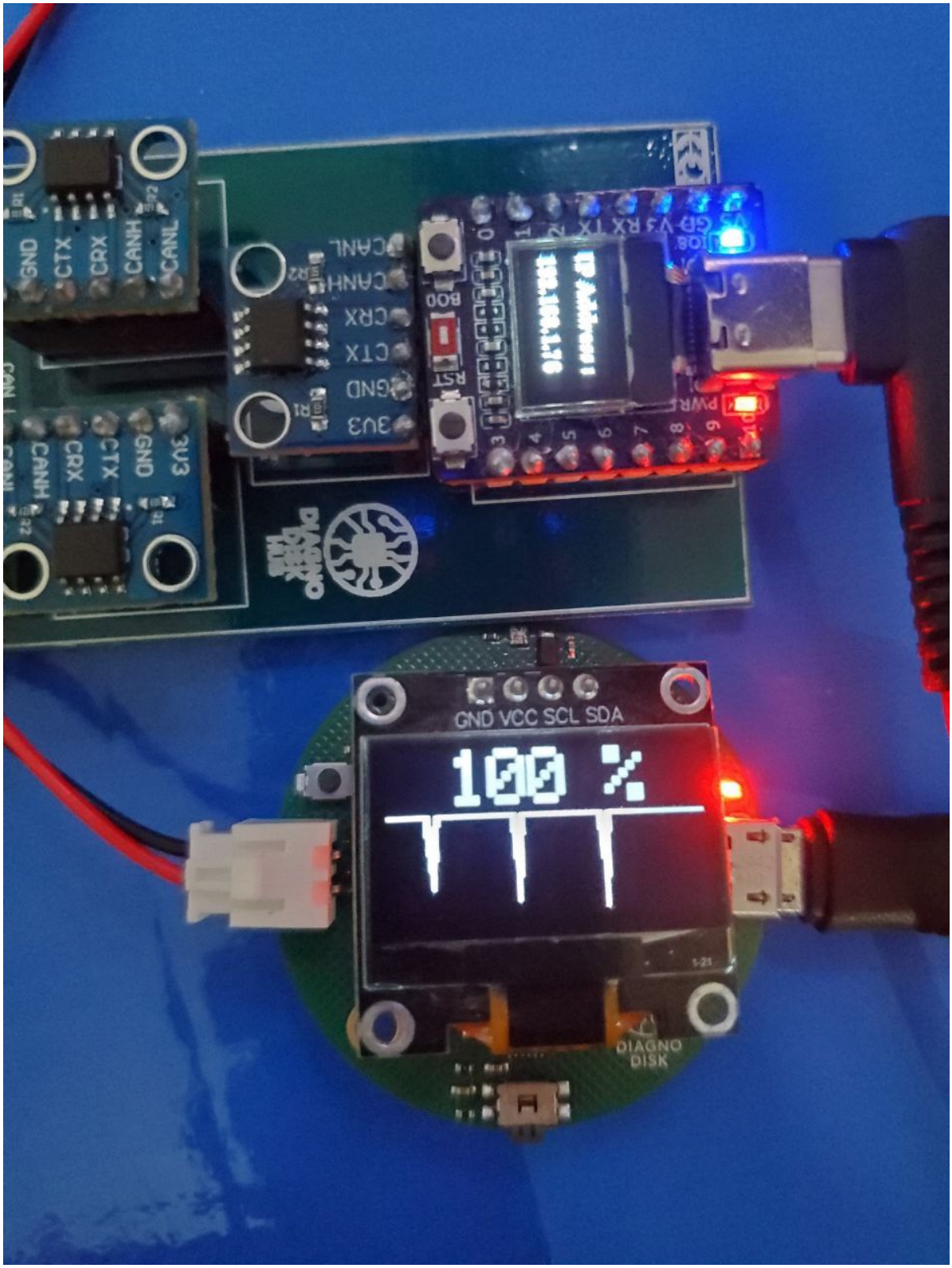


Figure 3.55: OLED Display Frames Under Various Conditions: Transient anomaly

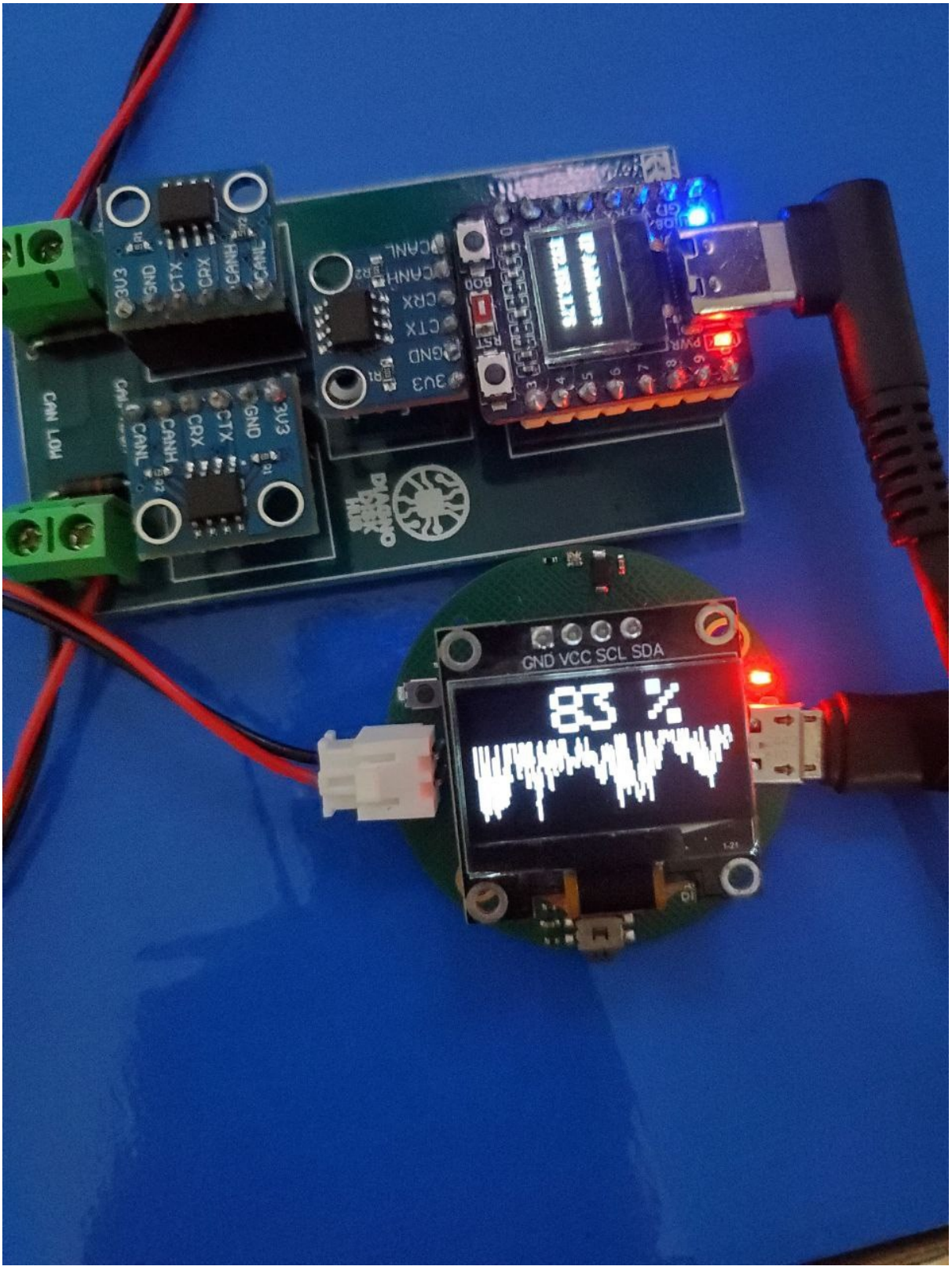


Figure 3.56: OLED Display Frames Under Various Conditions: Severe anomaly

3.7.2. Inference Behavior and Response Time

NanoEdge's ICM (Isolation Covariance Matrix) model was selected for deployment due to its superior performance in validation. The system consistently returned an anomaly score within 20 ms after each DMA-triggered data acquisition.

This low latency was achieved thanks to:

- DMA-enabled sensor readout, minimizing CPU load.
- Efficient memory usage: only 2.3 KB RAM and 8.7 KB flash for the AI library.
- Onboard inference without delay from external processing.

Real-time responsiveness was critical to providing meaningful display updates and timely web dashboard visualization.

3.7.3. Score Consistency and Model Accuracy

During physical experiments, the AI model demonstrated stable behavior with clear state transitions:

- In idle mode, the anomaly score remained under 5%, representing learned normal behavior.
- Under vibration, scores increased gradually, peaking at 60–80% depending on severity.
- Intense shaking caused spikes above 95%, which were correctly identified as strong anomalies.
- Recovery to idle resulted in a rapid return to low scores, showing the model's robustness to transients.

This result correlates with the quality index (99.9 pts) and balanced accuracy (100%) observed in NanoEdge AI Studio's benchmarking phase (Section 3.6.2).

3.7.4. Result Transmission and Web Visualization

In parallel with OLED output, the STM32 system transmitted the AI score to the secondary ESP32 board via the CAN bus. This ESP32 hosted a local Wi-Fi web server to render the anomaly score in a browser-accessible interface. Figure 3.55 shows a sample webpage displaying real-time status.

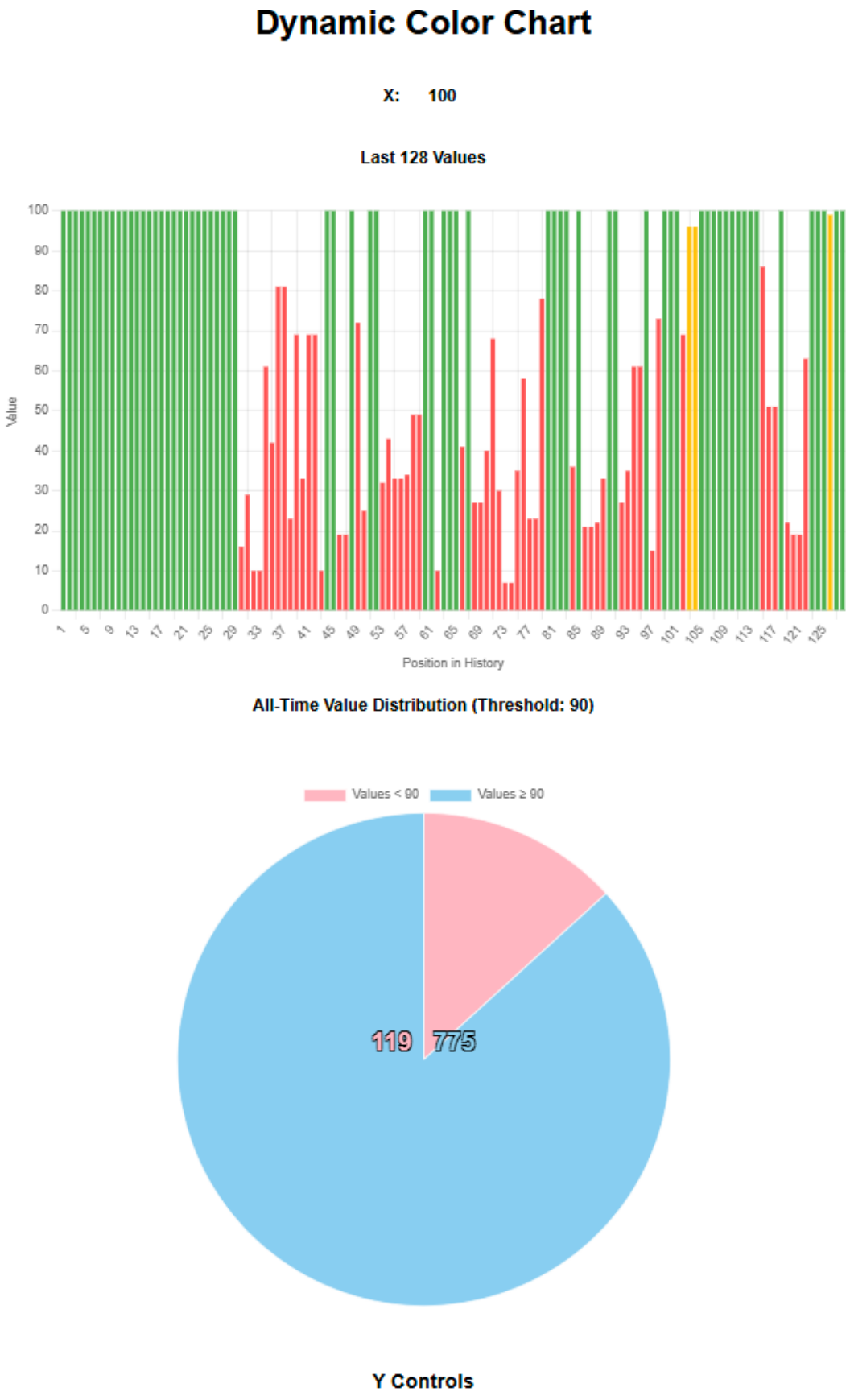


Figure 3.57: Web Dashboard Displaying Live Anomaly Score via ESP32 Module

The dynamic color chart displayed in the web interface **(Figure 3.55)** serves as an accessible, multi-user extension of the OLED graph, providing real-time visualization and analysis of similarity scores. Unlike the localized OLED display, this tool enables simultaneous remote access for collaborators or stakeholders, enhancing collaborative analysis and decision-making.

A) Key Features and Functionality

- **Real-Time Data Synchronization**:

The interface updates instantaneously as new similarity scores are generated, ensuring all users observe the same data without latency. This is critical for time-sensitive applications, such as anomaly detection or quality control.

- **Comprehensive Data Visualization**

The graph retains a rolling window of the *last 128 values*, offering immediate insight into recent trends. Additionally, the *all-time value distribution* categorizes results into normal **(≥90)** and abnormal **(<90)** signals, with cumulative counts (e.g., **775** normal vs**. 119** abnormal in **Figure 3.55**). This quantitative breakdown supports statistical evaluation of system performance.

- **Threshold-Based Classification**

A configurable threshold (default: 90) automatically flags anomalies, streamlining outlier identification. Color-coding (e.g., red for abnormal, green for normal) accelerates visual interpretation.

B) The Role of WebSocket Technology

The real-time capabilities of this interface are powered by **WebSocket**, a communication protocol that enables full-duplex, low-latency data exchange between the server and connected clients. Unlike traditional HTTP requests (which require repeated polling), WebSocket maintains a persistent connection, allowing the server to push updates to all users instantly when new data arrives.

This implementation ensures the system meets the demands of real-time monitoring, where delays or desynchronization could compromise analytical accuracy or operational responsiveness.

Advantages Over OLED Display

- **Accessibility**: Remote access eliminates dependency on physical hardware, enabling scalable deployment across devices (e.g., laptops, tablets).
- **Collaboration**: Multiple users can monitor, discuss, or annotate results concurrently, fostering teamwork.
- **Enhanced Analytics**: Aggregated counts (e.g., total anomalies) and distribution visualizations provide deeper insights than the OLED's real-time graph alone.

3.8. Conclusion

This chapter brought the system to life translating theoretical concepts and architectural plans into a fully functional embedded AI platform. Starting with hardware fabrication, we demonstrated how a carefully engineered 6-layer PCB improved mechanical reliability, electrical performance, and real-world integration. Software-wise, we developed a modular and efficient firmware stack capable of acquiring sensor data via DMA, performing on-device AI inference with NanoEdge AI, and presenting results both locally (via OLED and RGB LEDs) and remotely (via a wireless ESP32 interface).

Each subsystem sensor integration, AI processing, output logic, and communication was validated through structured testing. The chosen ICM anomaly detection model delivered highly accurate results in real time, reacting reliably to varying motion conditions. Benchmarking and emulator-based validation supported our design choices, while layered display feedback ensured interpretability and usability.

In sum, this chapter proved that embedded intelligence can be deployed on a compact, resource-constrained device to detect, analyze, and visualize machine behavior autonomously. With its modular architecture and low-latency response, the system offers a scalable foundation for smart diagnostics, making embedded AI not just feasible but practical and impactful.

General Conclusion

This thesis began with a practical question: how can we give embedded systems the ability to detect and interpret the behavior of machines in real time using only on-board resources, with no dependence on the cloud? The challenge was not just to make it work, but to make it fast, efficient, interpretable, and ready for the constraints of the real world.

Through the development of a complete hardware-software pipeline, this project answered that question by building a fully functional diagnostic platform. At its core is the STM32F103RB microcontroller, supported by carefully selected components and a custom 6-layer PCB that ensured signal integrity, mechanical stability, and compact design. Paired with the MPU6050 sensor and powered by efficient clock scaling and DMA-based I^2C communication, the system was able to continuously capture motion data with minimal CPU overhead.

On the intelligence side, we used NanoEdge AI Studio to train lightweight embedded models directly from real sensor data. By working with different algorithm types and evaluating them side-by-side in benchmarking and simulation tools, we identified which ones aligned best with our goals. The embedded AI library we deployed allowed the system to recognize familiar signal patterns and distinguish new or irregular behaviors effectively enabling the microcontroller to make decisions based on its own learned knowledge, rather than hard-coded thresholds.

Those decisions were made visible in two ways. Locally, the OLED screen offered a clear and intuitive visualization of the current machine state, along with responsive feedback that changed in real time. Remotely, the system used a dedicated ESP32-based module connected over CAN bus, allowing the same output to be viewed on a dynamic web interface. This dual-layer display architecture made the system scalable, accessible, and user-friendly.

The results showed that this approach works. The AI model achieved a quality index above 99%, with a compact memory footprint well below the MCU’s limits. In live testing, the system reliably distinguished between idle, light, and aggressive vibrations. The integration of DMA, interrupt handling, and power-efficient design further strengthened the system’s ability to operate stably over time even in the presence of motion and environmental noise.

In building this system, we demonstrated that real-time state detection can be embedded into compact, standalone hardware without sacrificing performance or clarity. The modular architecture we created, from the tactile switch to the CAN interface, makes it reusable in other contexts and ready for further development.

In the end, what we built wasn't just a technical solution. It was a small, smart, self-aware system capable of learning, sensing, interpreting, and informing and one that reflects the growing power of embedded AI when it's paired with thoughtful engineering.

References

[1] Intel. What is edge computing? [Internet]. Intel. [date unknown]. Available from: https://www.intel.com/content/www/us/en/learn/what-is-edge-computing.html

[2] ENTSO-E. Cloud and edge computing. [Internet]. ENTSO-E; [date unknown]. Available from: https://www.entsoe.eu/technopedia/techsheets/cloud-and-edge-computing/

[3] Luzmo. Embedded AI: what it is and how it works. [Internet]. Luzmo; [date unknown]. Available from: https://www.luzmo.com/blog/embedded-ai

[4] «A Survey on The Latest Techniques for Implementing Machine Learning and Artificial Intelligence Algorithms on Resource-constrained Embedded Devices» April 2025 | IJIRT | Volume 11 Issue 11 |
[research paper] https://ijirt.org/publishedpaper/IJIRT174969_PAPER.pdf

[5] International Journal of Research Publication and Reviews. [Title unknown: IJRPR45571]. [Internet]. IJRPR; [date unknown]. Available from: https://ijrpr.com/uploads/V6ISSUE5/IJRPR45571.pdf

[6] Imagimob. What is tinyML? [Internet]. Imagimob, 2024 Feb 6. Available from: https://www.imagimob.com/blog/what-is-tinyml

[7] CoreTigo. What is condition monitoring? Explanation & examples. [Internet]. CoreTigo,[date unknown]. Available from: https://www.coretigo.com/what-is-condition-monitoring-explanation-examples/

[8] IBM. What is condition monitoring? [Internet]. IBM; [date unknown]. Available from: https://www.ibm.com/think/topics/condition-monitoring

[9] Graceport. What is condition monitoring? [Internet]. Graceport; [date unknown]. Available from: https://www.graceport.com/condition-monitoring-sensors/what-is-condition-monitoring

[10] STMicroelectronics. AI:NanoEdge AI Studio – Basic format and use. [Internet]. STMicroelectronics, [28 June 2024]. Available from: https://wiki.st.com/stm32mcu/wiki/AI:NanoEdge_AI_Studio#Basic_format

[11] STMicroelectronics. STM32StepByStep: STM32MCU basics – Varieties of STM32 MCUs. [Internet]. STMicroelectronics; [2022 September 21]. Available from: https://wiki.st.com/stm32mcu/wiki/STM32StepByStep:STM32MCU_basics#Varieties_of_STM3

2_MCUs

[12] STMicroelectronics. STM32F103C8 datasheet. [Internet]. STMicroelectronics; 2013 Aug. Available from: https://www.st.com/resource/en/datasheet/stm32f103c8.pdf

[13] STMicroelectronics. STM32 System Direct-memory access controller (DMA). [Internet]. STMicroelectronics; [date unknown]. Available from: https://www.st.com/resource/en/product_training/STM32G0-System-Direct-memory-access-controller-DMA.pdf

[14] DFRobot. What are the Types of Motion Sensors?. [Internet]. DFRobot; [2023 june 1]. Available from: https://www.dfrobot.com/blog-13308.html

[15] Dwyer Omega. Vibration sensors and their importance in vibration monitoring of rotating and/or reciprocating machines. [Internet]. Dwyer Omega; [date unknown]. Available from: https://www.dwyeromega.com/en-us/resources/vibration-sensors-in-vibration-monitoring/

[16] ElectronicWings. SSD1306 OLED Display Complete Guide with Arduino Interfacing. [Internet]. ElectronicWings; 2023. Available from: https://www.electronicwings.com/sensors-modules/ssd1306-oled-display

[17] Lascar Electronics. Understanding human–machine interfaces: bridging humans and technology. [Internet]. Lascar Electronics; [date unknown]. Available from: https://lascarelectronics.com/understanding-human-machine-interfaces-briding-humans-and-technology/

[18] InvenSense Inc. MPU Datasheet. [Internet]. [publisher unknown]; 2013 august 19. Available from: https://atta.szlcsc.com/upload/public/pdf/source/20140425/1457707046159.pdf

[19] STMicroelectronics. STM32CubeIDE: Introduction – What is STM32CubeIDE. [Internet]. STMicroelectronics; [2025 april 14]. Available from: https://wiki.st.com/stm32mcu/wiki/STM32CubeIDE:Introduction_to_STM32CubeIDE#What_is_STM32CubeIDE--

[20] KiCad Documentation. Introduction to KiCad (version 9.0) [Internet]. KiCad; 2025 Feb 18. Available from: https://docs.kicad.org/9.0/en/introduction/introduction.pdf

[21] JLCPCB. [Internet]. JLCPCB; [2006,]. Available from: https://jlcpcb.com

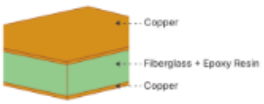
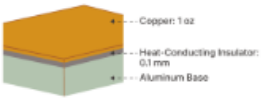
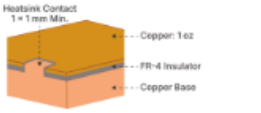
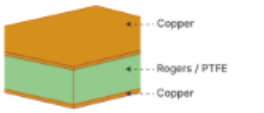
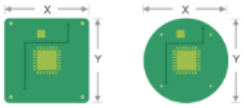
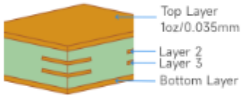
[22] SNC ALMItech. À propos – SNC ALMItech. [Internet]. SNC ALMItech; [2025]. Available from: https://pcbproduction.almitech-dz.com/a-propos/

[23] JLCPCB. PCB Manufacturing & Assembly Capabilities. [Internet]. JLCPCB; [date unknown]. Available from: https://jlcpcb.com/capabilities/pcb-capabilities

Appendix

Supplementary Tables Related to Section 3.3.4 Capabilities of a 6-Layer Rigid PCB (via JLCPCB)

PCB Specifications

Features	Capability	Description	Patterns
Layer count	1-32 Layers	The number of copper layers in the PCB	
Controlled Impedance	4/6/8/10/12/14/16/18/20/.../32 layers	User Guide to the JLCPCB Impedance Calculator JLCPCB Impedance Calculator	
Impedance Tolerance	±10%		
Material	FR-4	Grade A laminates from suppliers including Nan Ya, KB, Shengyi and etc.	
	Aluminum-Core	1-layer Aluminum-core PCBs	
	Copper-Core	1-layer copper-core PCBs with direct heatsink contacts to core (≥ 1 × 1 mm)	
	RF PCB	1 oz copper, 2-layer RF PCBs with Rogers and PTFE cores	
FR-4 Dielectric Constants	4.5 (2-Layer PCB)	7628 Prepreg 4.4 3313 Perpreg 4.1 2116 Perpreg 4.16	
Maximum Dimensions	FR4 PCB: 670 × 600 mm Rogers / PTFE Teflon PCB: 590 × 438 mm Aluminum PCB: 602 × 506 mm Copper PCB: 480 × 286 mm	These limits apply to PCBs with thickness ≥ 0.8 mm. The thinner FR4 PCBs are 500 × 600 mm maximum. 2-layer FR4 PCBs can reach a maximum size of 1020 × 600 mm.	
Minimum Dimensions	Regular: 3 × 3 mm. Castellated / Plated Edges: 10 × 10 mm.	These limits apply to PCBs with thickness ≥ 0.6 mm. Manual review required for thinner PCBs. Panelization is recommended for small-sized boards.	
Dimension Tolerance	±0.1mm	±0.1mm(Precision) and ±0.2mm(Regular) for CNC routing, and ±0.4mm for V-scoring	
Thickness	0.4 - 4.5 mm	Thickness for FR4 are: 0.4/0.6/0.8/1.0/1.2/1.6/2.0 mm (2.5 mm and above are for 12+ layer PCBs only)	
Thickness Tolerance (Thickness≥1.0mm)	± 10%	e.g. For the 1.6mm board thickness, the finished board thickness ranges from 1.44mm(T-1.6×10%) to 1.76mm(T+1.6×10%)	
Thickness Tolerance (Thickness<1.0mm)	± 0.1mm	e.g. For the 0.8mm board thickness, the finished board thickness ranges from 0.7mm(T-0.1) to 0.9mm(T+0.1).	
Finished Outer Layer Copper	1 oz / 2 oz (35um / 70um)	Finished copper weight of outer layer is 1oz or 2oz.	

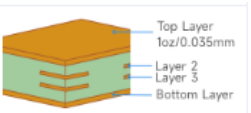
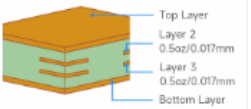
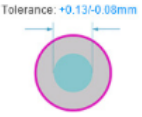
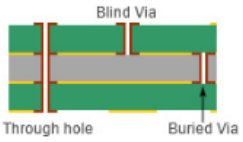
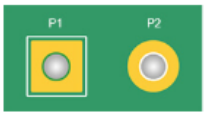
Finished Outer Layer Copper	1 oz / 2 oz (35um / 70um)	Finished copper weight of outer layer is 1oz or 2oz.	
Finished Inner Layer Copper	0.5 oz / 1 oz / 2 oz (17.5um / 35um / 70um)	Finished copper weight of inner layer is 0.5oz by default.	
Soldermask	Green, Purple, Red, Yellow, Blue, White, and Black.	We use LPI (Liquid Photo Imageable) solder mask. This is the most common type of mask used today. Heat-cured ink soldermask is usually found on low-cost, single-sided PCBs.	
Surface Finish	HASL (leaded / lead-free), ENIG, OSP (copper core boards only)	FR4 has all three finishes available, 6+ layers and RF boards only have ENIG. Aluminium core boards only have HASL. Copper core boards only have OSP.	

Table 1: Core PCB Manufacturing Specifications [23]

Drilling

Features	Capability	Description	Patterns
Drill Diameter	1-layer: 0.3 – 6.3 mm 2-layer: 0.15 – 6.3 mm Multilayer: 0.15 – 6.3 mm	Holes with diameter ≥ 6.3 mm are CNC routed from a smaller drilled hole. Min. drill diameter for 2- or more-layer PCBs is 0.15 mm (more costly!) Min. drill diameter for aluminum-core PCBs is 0.65 mm Min. drill diameter for copper-core PCBs is 1.0 mm	
Hole size Tolerance (Plated)	Through-holes: +0.13 / -0.08 mm Press-fit holes: ±0.05 mm (multilayer ENIG boards only – mention the specific holes in PCB Remark)	e.g. for the 0.6mm hole size, the finished hole size between 0.52mm to 0.73mm is acceptable.	
Hole size Tolerance (Non-Plated)	±0.2mm	e.g. for the 1.00mm Non-Plated hole, the finished hole size between 0.80mm to 1.20mm is acceptable.	
Average Hole Plating Thickness	18μm		
Blind/Buried Vias	Not supported	Currently we don't support Blind/Buried Vias, only make through holes.	
Min. Via hole size/diameter	0.15mm / 0.25mm	1-layer (NPTH only): 0.3 mm hole size / 0.5 mm via diameter 2-layer: 0.15mm hole size / 0.25mm via diameter Multilayer: 0.15 mm hole size / 0.25 mm via diameter ① Via diameter should be 0.1mm(0.15mm preferred) larger than Via hole size. ② Preferred Min. Via hole size: 0.2mm	
Min. Non-plated holes	0.50mm	Please draw NPTHs in the mechanical layer or keep out layer.	
Min. Plated Slots	0.5mm	The minimum plated slot width is 0.5mm, which is drawn with a pad.	
Min. Non-Plated Slots	1.0mm	The minimum Non-Plated Slot Width is 1.0mm, please draw the slot outline in the mechanical layer(GM1 or GKO)	

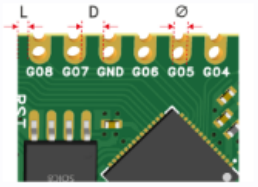
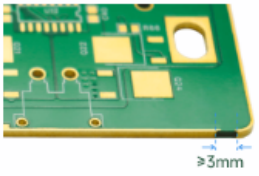
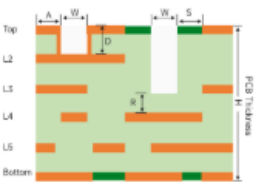
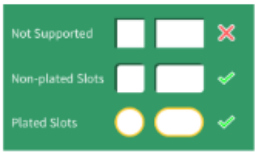
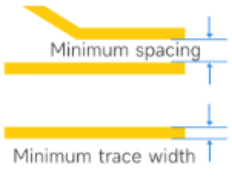
Via Hole-to-Hole Spacing	0.2mm		
Pad Hole-to-Hole Spacing	0.45mm		
Min. Castellated Holes	0.5mm	Castellated holes are metalized half-holes on PCB edges, commonly used on daughter boards to be soldered onto carrier PCBs. ① Hole diameter (Φ): ≥ 0.5 mm ② Hole to board edge (L): ≥ 1 mm ③ Hole to hole (D): ≥ 0.5 mm ④ Min. PCB size: 10 × 10 mm ⑤ Min. PCB thickness: 0.6 mm	
Plated Edges	10 x 10mm	Plated edges are copper-plated and ENIG treated. HASL is not supported. ① Min. PCB size: 10 × 10 mm ② Min. PCB thickness: 0.6 mm ③ At least 3 breaks (more for larger PCBs) in the edge plating are required for support tab connections	
Blind Slot		① Blind slot width (W): ≥1.0mm ② Blind slot depth (D): ≥0.2mm ③ Blind slot annular width (A): ≥0.3mm (The pad width of PTH blind slots) ④ Safety distance (S): ≥0.2mm (The distance from NPTH blind slots to pad/traces/copper plane) ⑤ Blind slot remaining thickness (R): ≥0.2mm (The distance from the bottom of the blind slot to the nearest inner copper layer/surface substrate) ⑥ Supports 2-32 layer FR4 boards with a thickness of ≥0.8mm	
Rectangular Holes / Slots	Not supported	Rectangular holes and slots without rounded corners are not supported.	

Table 2: Drill Capabilities and Hole Parameters [23]

Traces

Features	Capability	Description	Patterns
Min. track width and spacing (1 oz)	0.10 / 0.10 mm (4 / 4 mil)	1- and 2-layer: 0.10 / 0.10 mm (4 / 4 mil) Multilayer: 0.09 / 0.09 mm (3.5 / 3.5 mil). 3 mil is acceptable in BGA fan-outs.	
Min. track width and spacing (2 oz)	0.16 / 0.16 mm (6.5 / 6.5 mil)	2-layer: 0.16 / 0.16 mm (6.5 / 6.5 mil) Multilayer: 0.16 / 0.20 mm (6.5 / 8 mil)	

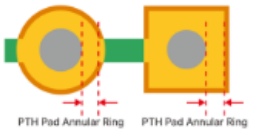
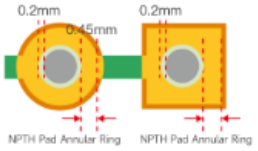
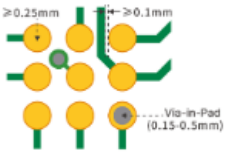
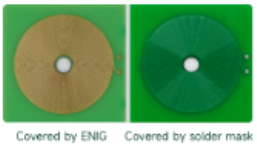
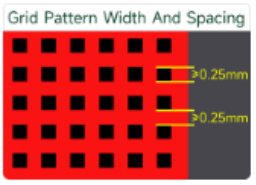
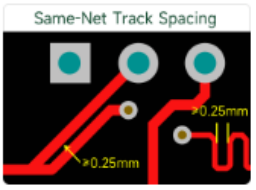
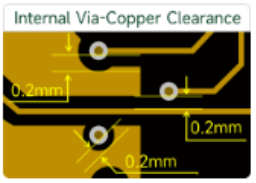
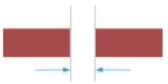
PTH annular ring	≥0.20mm	2-layer: 1 oz: Recommended 0.25 mm or above; absolute minimum 0.18 mm 2 oz: 0.254 mm or above Multilayer: 1 oz: Recommended 0.20 mm or above; absolute minimum 0.15 mm 2 oz: 0.254 mm or above	
NPTH pad annular ring	≥0.45mm	Recommended 0.45 mm or more. This is to allow a 0.2 mm ring of copper to be removed around the hole for the sealing film to attach. Pad sizes smaller than the recommended value can result in the annular ring being very thin or completely missing.	
BGA	0.25mm	① BGA pad diameter ≥ 0.25 mm ② BGA pad to trace clearance ≥ 0.1 mm (min. 0.09 mm for multilayer boards) ③ Vias can be placed within BGA pads using filled and plated-over vias	
Trace coils	0.15/0.15mm	Minimum trace width/clearance: 0.15/0.15mm, when traces are covered by solder mask (1oz). Minimum trace width/clearance: 0.25/0.25mm, when traces are NOT covered by solder mask (1oz). ENIG only(high risk of short circuit with HASL)	
Hatched grid width and spacing	0.25 mm		
Same-net track spacing	0.25mm		
Inner layer via hole to copper clearance	0.2mm		
Inner layer PTH pad hole to copper clearance	0.3mm		
Pad to track clearance	0.1mm	Min. 0.1 mm (stay well above if possible). Min. 0.09 mm locally for BGA pads	
SMD pad to pad clearance (different nets)	0.15mm	More details of SMD pad spacing: SMD Components Minimum Spacing	

Table 3: Trace and Spacing Constraints [23]

Soldermask

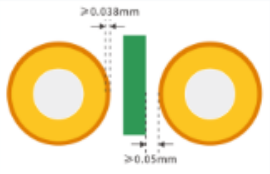
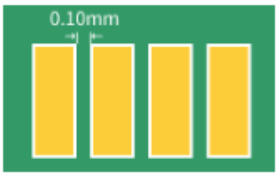
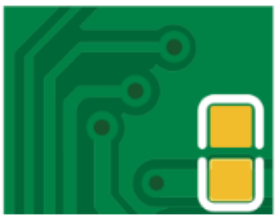
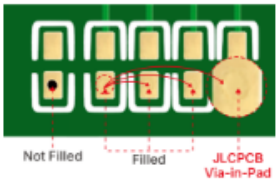
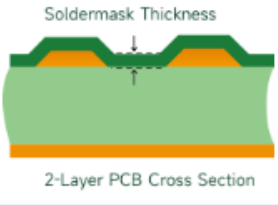
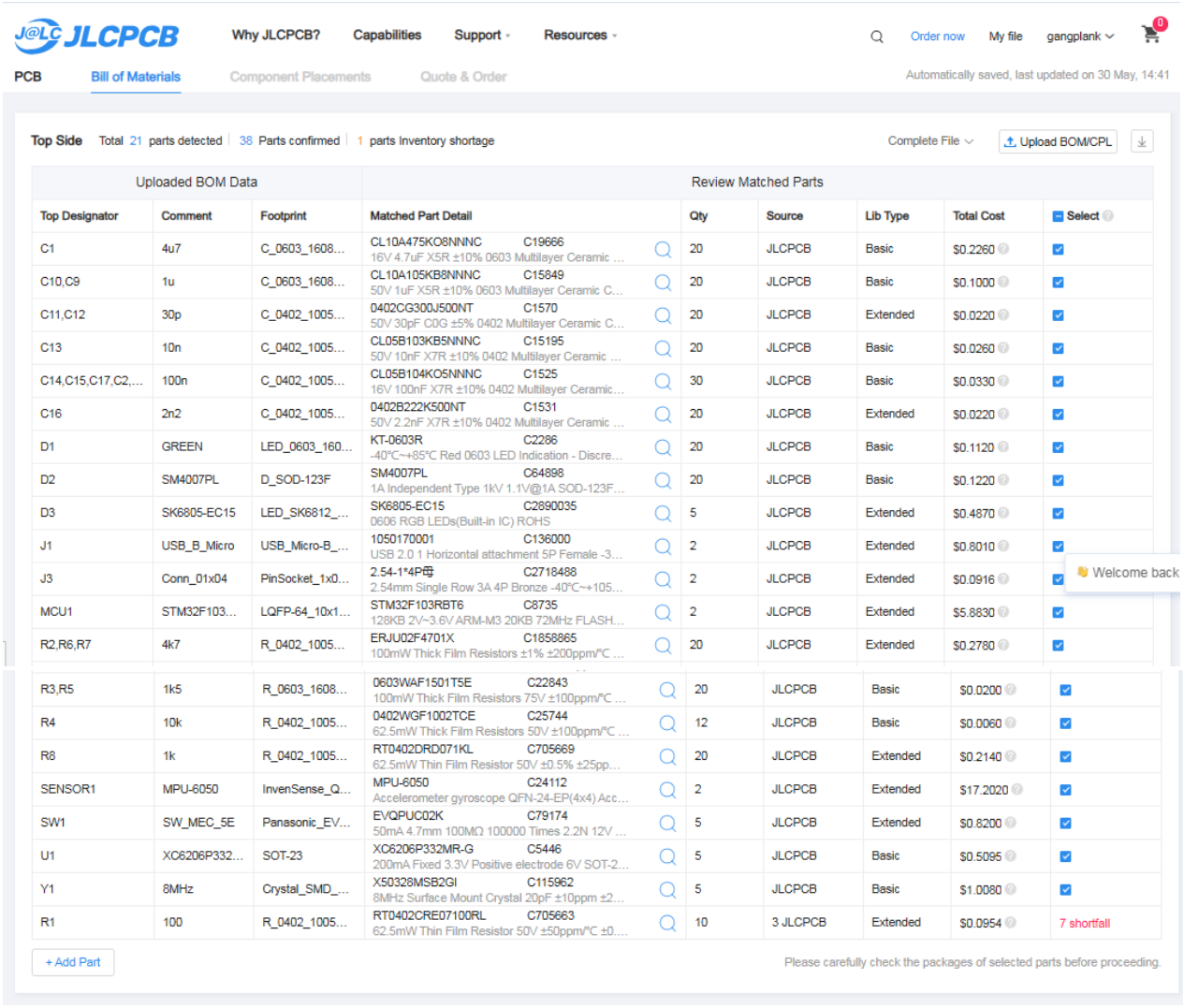
Features	Capability	Description	Patterns
Soldermask Expansion	0.038mm	2-layer: 0.038 mm expansion on each side of a pad. Keep at least 0.05 mm clearance between soldermask openings and neighboring traces. Multilayer: No expansion required Note: this rule does not conflict with Pad to track clearance Min. 0.1 mm	
Soldermask bridge	0.10mm	2-layer (1 oz): Min. pad spacing: 0.20 mm (green, red, yellow, blue, purple) Min. pad spacing: 0.23 mm (black, white) Multilayer (1 oz): Min. pad spacing: 0.10 mm (green, red, yellow, blue, purple) Min. pad spacing: 0.13 mm (black, white)	
Plugged vias	Filled with soldermask	Vias are filled with soldermask for an opaque finish. Click for detailed explanation ① Filled vias must not have soldermask openings on either side ② Filled vias should have ≥ 0.35 mm clearance from other soldermask openings (e.g. pads) ③ Filled vias must be no wider than 0.5 mm diameter	
JLCPCB Via-in-Pad Process	Epoxy Filled & Capped Copper paste Filled&Capped	Vias are filled with epoxy resin or copper paste and then plated over to achieve an opaque and smooth finish. Click for detailed explanation ① Vias are filled and plated over. Choose copper paste filling for applications requiring high thermal conductivity. ② This process is the default for 6-layer and above multilayer boards. ③ Compatible with via diameters from 0.15 to 0.5 mm.	
Solder mask dielectric constant	3.8		
Solder mask ink thickness	≥ 10μm		

Table 4: Solder Mask Capabilities [23]



JLCPCB | Why JLCPCB? | Capabilities | Support | Resources | Order now | My file | gangplank

PCB | Bill of Materials | Component Placements | Quote & Order

Automatically saved, last updated on 30 May, 14:41

Top Side Total 21 parts detected | 38 Parts confirmed | 1 parts Inventory shortage

Complete File | Upload BOM/CPL

Uploaded BOM Data			Review Matched Parts					
Top Designator	Comment	Footprint	Matched Part Detail	Qty	Source	Lib Type	Total Cost	Select
C1	4u7	C_0603_1608...	CL10A475KO8NNNC C19666 16V 4.7uF X5R ±10% 0603 Multilayer Ceramic ...	20	JLCPCB	Basic	$0.2260	☑
C10,C9	1u	C_0603_1608...	CL10A105KB8NNNC C15849 50V 1uF X5R ±10% 0603 Multilayer Ceramic C...	20	JLCPCB	Basic	$0.1000	☑
C11,C12	30p	C_0402_1005...	0402CG300J500NT C1570 50V 30pF C0G ±5% 0402 Multilayer Ceramic C...	20	JLCPCB	Extended	$0.0220	☑
C13	10n	C_0402_1005...	CL05B103KB5NNNC C15195 50V 10nF X7R ±10% 0402 Multilayer Ceramic ...	20	JLCPCB	Basic	$0.0260	☑
C14,C15,C17,C2,...	100n	C_0402_1005...	CL05B104KO5NNNC C1525 16V 100nF X7R ±10% 0402 Multilayer Ceramic...	30	JLCPCB	Basic	$0.0330	☑
C16	2n2	C_0402_1005...	0402B222K500NT C1531 50V 2.2nF X7R ±10% 0402 Multilayer Ceramic ...	20	JLCPCB	Extended	$0.0220	☑
D1	GREEN	LED_0603_160...	KT-0603R C2286 -40°C~+85°C Red 0603 LED Indication - Discre...	20	JLCPCB	Basic	$0.1120	☑
D2	SM4007PL	D_SOD-123F	SM4007PL C64898 1A Independent Type 1kV 1.1V@1A SOD-123F...	20	JLCPCB	Basic	$0.1220	☑
D3	SK6805-EC15	LED_SK6812_...	SK6805-EC15 C2890035 0606 RGB LEDs(Built-in IC) ROHS	5	JLCPCB	Extended	$0.4870	☑
J1	USB_B_Micro	USB_Micro-B_...	1050170001 C136000 USB 2.0 1 Horizontal attachment 5P Female -3...	2	JLCPCB	Extended	$0.8010	☑
J3	Conn_01x04	PinSocket_1x0...	2.54-1*4P母 C2718488 2.54mm Single Row 3A 4P Bronze -40°C~+105...	2	JLCPCB	Extended	$0.0916	☑
MCU1	STM32F103...	LQFP-64_10x1...	STM32F103RBT6 C8735 128KB 2V~3.6V ARM-M3 20KB 72MHz FLASH...	2	JLCPCB	Extended	$5.8830	☑
R2,R6,R7	4k7	R_0402_1005...	ERJU02F4701X C1858865 100mW Thick Film Resistors ±1% ±200ppm/°C ...	20	JLCPCB	Extended	$0.2780	☑
R3,R5	1k5	R_0603_1608...	0603WAF1501T5E C22843 100mW Thick Film Resistors 75V ±100ppm/°C ...	20	JLCPCB	Basic	$0.0200	☑
R4	10k	R_0402_1005...	0402WGF1002TCE C25744 62.5mW Thick Film Resistors 50V ±100ppm/°C ...	12	JLCPCB	Basic	$0.0060	☑
R8	1k	R_0402_1005...	RT0402DRD071KL C705669 62.5mW Thin Film Resistor 50V ±0.5% ±25pp...	20	JLCPCB	Extended	$0.2140	☑
SENSOR1	MPU-6050	InvenSense_Q...	MPU-6050 C24112 Accelerometer gyroscope QFN-24-EP(4x4) Acc...	2	JLCPCB	Extended	$17.2020	☑
SW1	SW_MEC_5E	Panasonic_EV...	EVQPUC02K C79174 50mA 4.7mm 100MΩ 100000 Times 2.2N 12V ...	5	JLCPCB	Extended	$0.8200	☑
U1	XC6206P332...	SOT-23	XC6206P332MR-G C5446 200mA Fixed 3.3V Positive electrode 6V SOT-2...	5	JLCPCB	Basic	$0.5095	☑
Y1	8MHz	Crystal_SMD_...	X50328MSB2GI C115962 8MHz Surface Mount Crystal 20pF ±10ppm ±2...	5	JLCPCB	Basic	$1.0080	☑
R1	100	R_0402_1005...	RT0402CRE07100RL C705663 62.5mW Thin Film Resistor 50V ±50ppm/°C ±0....	10	3 JLCPCB	Extended	$0.0954	7 shortfall

+ Add Part

Please carefully check the packages of selected parts before proceeding.

Welcome back

Table 5: Summary Bill of Materials for the 6-Layer Diagno disc PCB