R&D Project Proposals: OS, Cybersecurity, and Deep Learning

Himanshu Suri

1 Project A — FeatherOS (Lightweight Embedded OS)

1.1 Summary

FeatherOS is a minimal microkernel-inspired operating system designed for resource-constrained embedded devices (e.g., Raspberry Pi Zero, small ARM boards). It focuses on correctness, minimal trusted computing base, modular drivers, and predictable performance.

1.2 Objectives

- Boot a minimal kernel on QEMU/target board.
- Provide process scheduling, a small memory manager, and a message-passing IPC.
- Implement a module/driver loader allowing hot-pluggable drivers.
- Provide developer tools (cross-compile scripts, debug images).

1.3 Standardized Stack

- **Languages:** C (kernel), minimal Assembly for boot context switch.
- **Tools:** GNU toolchain (gcc cross-compiler), NASM, GNU Make, GDB, QEMU.
- **Version Control & CI:** GitHub, GitHub Actions for build+test.
- **Docs:** Doxygen + Markdown.

1.4 Weekly Milestones

1. Week 1: Setup cross-compilation environment, QEMU test boot.
2. Week 2: Basic kernel boot with UART output.
3. Week 3: Implement process scheduler.
4. Week 4: Add basic memory management.
5. Week 5: Implement message-passing IPC.
6. Week 6: Hot-pluggable driver loader.
7. Week 7: Stability and performance testing.

8. Week 8: Documentation and final demo.

1.5 Practical Experiments

- Test driver swapping without reboot.
- Run stability tests with multiple concurrent processes.
- Measure latency of IPC compared to baseline.

1.6 R&D Innovation Leap

We will create a tiny OS that allows adding or updating hardware drivers while the system is running — no reboot needed. This reduces downtime and improves flexibility for IoT and robotics. We will measure how fast and stable this process is compared to traditional systems.

2 Project B — SentinelIoT (IoT Security)

2.1 Summary

SentinelIoT is a lightweight, AI-assisted intrusion detection system (IDS) designed to run directly on IoT devices. It uses on-device learning to detect abnormal patterns in activity without relying on a central server.

2.2 Objectives

- Capture network and system metrics from IoT devices.
- Train a lightweight anomaly detection model.
- Implement real-time threat alerts.
- Run system entirely on-device for privacy and low latency.

2.3 Standardized Stack

- **Languages:** Python (ML logic), C/C++ (low-level hooks).
- **Libraries:** scikit-learn, NumPy, lightweight MQTT broker.
- **Hardware:** Raspberry Pi / ESP32.
- **Tools:** Wireshark, tcpdump, Mininet (for simulations).

2.4 Weekly Milestones

1. Week 1: Setup IoT device environment and packet capture.
2. Week 2: Implement baseline anomaly detection.
3. Week 3: Optimize model for on-device inference.
4. Week 4: Implement live alert system.
5. Week 5: Simulate attack scenarios in lab.

6. Week 6: Optimize CPU/memory usage.
7. Week 7: Field test with multiple devices.
8. Week 8: Documentation and security audit.

2.5 Practical Experiments

- Simulate DDoS and MITM attacks on a test IoT network.
- Measure detection speed and false positive rates.
- Test battery consumption with IDS enabled vs disabled.

2.6 R&D Innovation Leap

We will make IoT devices smart enough to detect attacks without a server. Our IDS will learn normal patterns and spot suspicious activity in real time. It will be tested for speed, accuracy, and low power usage.

3 Project C — Deep Learning for Edge Devices

3.1 Summary

This project focuses on deploying advanced deep learning models to small devices while maintaining high accuracy. The goal is to optimize models for low power and high speed.

3.2 Objectives

- Train a baseline image classification model.
- Apply model compression techniques.
- Deploy and run inference on Raspberry Pi-class devices.
- Compare performance to unoptimized versions.

3.3 Standardized Stack

- **Languages:** Python.
- **Libraries:** PyTorch, TensorFlow Lite.
- **Hardware:** Raspberry Pi, Jetson Nano.
- **Tools:** Jupyter, ONNX, TFLite Converter.

3.4 Weekly Milestones

1. Week 1: Dataset preparation and baseline model training.
2. Week 2: Implement Quantization-Aware Training.
3. Week 3: Test Adaptive Activation Functions.
4. Week 4: Deploy to Raspberry Pi and measure inference speed.

5. Week 5: Compare accuracy vs baseline.
6. Week 6: Optimize memory usage.
7. Week 7: Multi-device deployment tests.
8. Week 8: Documentation and final presentation.

3.5 Practical Experiments

- Compare model sizes before and after quantization.
- Measure inference speed on small devices.
- Test accuracy changes after optimization.

3.6 R&D Innovation Leap

We will shrink AI models so they run on tiny devices without losing much accuracy. We'll try two methods: making the model learn to use fewer bits (Quantization-Aware Training) and giving it smarter, self-adjusting functions (Adaptive Activation Functions). This lets small devices do big AI tasks.