

LOCUS: AI-Driven Household Context-Awareness for Predictive Cleaning

Hanyeong Go
Department of Information Systems
Hanyang University
Seoul, South Korea
lilla9907@hanyang.ac.kr

Dayeon Lee
Department of Sports Science
Hanyang University
Seoul, South Korea
ldy21@hanyang.ac.kr

Junhyung Kim
Department of Information Systems
Hanyang University
Seoul, South Korea
combe4259@hanyang.ac.kr

Seunghwan Lee
Department of Information Systems
Hanyang University
Seoul, South Korea
shlee5820@hanyang.ac.kr

Group name : LOCUS

Abstract — Current robot vacuums remain "reactive" tools, dependent on pre-set schedules or direct user commands. This represents a fundamental limitation in their inability to effectively address dynamically occurring contamination scenarios in real home environments, such as "the kitchen immediately after cooking" or "the dining table after a meal." This project proposes the LOCUS (Learning On-device Context and User-specific Schedules) system to solve these problems. LOCUS operates based on On-device AI, processing all data internally on the device, thereby ensuring the complete protection of users' sensitive personal life information. The system fuses multi-modal sensor data—including a spatial map built in the Webots simulation environment, YOLO (visual), and SED (auditory)—to perceive the home's 'context' in real-time. Furthermore, it learns each household's unique living patterns, such as "cooking followed by dining in the kitchen," through a lightweight GRU model and a Personalized Federated Learning (PFL) architecture. Ultimately, LOCUS aims to establish a 'Proactive Partner' robot vacuum system that anticipates the occurrence of contamination and actively automates the cleaning process.

Federated Learning, Edge AI, On-device AI, IoT, Artificial Intelligence

Role Assignments (till the middle of the term):

Roles	Name	Task description and etc.
Development Manager	Seunghwan Lee	(General Management) Establish project schedule and manage milestones. Chair weekly team meetings and track issues. (Documentation) Final compilation and LaTeX formatting of the Requirements Specification, Detailed Specification, and Design documents. (Administration) Establish GitHub

		repository branching strategy and manage the main branch. Manage final deliverables.
Software Developer (AI/ML)	Hanyeong Go	(Research) Explore and benchmark the performance of lightweight YOLO, SED, and GRU models suitable for the local PC environment (for initial prototyping) and the on-device environment (e.g., Raspberry Pi) for final porting. (Design) Design the Personalized Federated Learning (PFL) system based on the FedPer architecture. Define the input/output data schema for the GRU model. (Implementation) Implement the model prototype in a local environment (Python/TensorFlow) and test for lightweight conversion (Quantization) using TensorFlow Lite. Implement prototypes for the PFL central server and client logic.
Software Developer (Core System)	Junhyung Kim	(Environment Setup) Set up the common local development environment (Python, virtual environment). Build a Webots-based robot data simulator

		<p>(virtual position, sensor values) for AI testing.</p> <p>(Design) Design the multi-modal sensor data fusion pipeline (Camera, Mic, Simulated Position). Define the API for communication with the AI module (Team Member B).</p> <p>(Implementation) Implement the sensor data collection and real-time pre-processing module.</p>
User / Customer	Dayeon Lee	<p>(Planning) Detailed creation of user personas and core Use Case scenarios.</p> <p>(Requirements) Draft FR/NFR (Functional Requirements / Non-Functional Requirements) and lead the team review.</p> <p>(Data) Generate and manage Mock data (e.g., virtual event sequences) necessary for initial AI model training.</p> <p>(Testing) Write functional and integration test cases for each development stage. Conduct usability testing on the final prototype, write bug reports, and track issues.</p>

I. INTRODUCTION

A. Background and Motivation

The adoption rate of robot vacuums in modern homes is rapidly increasing, establishing them as essential appliances. Despite this popularization, most products remain constrained by first-generation limitations. This is the 'reactive' approach, which relies on direct user commands or static schedules like "clean the living room every day at 3 PM." Recently, a major technological inflection point has occurred in this trend. Hardware performance for on-device AI is exponentially improving, exemplified by Apple increasing the base RAM in the latest iPhones and Roborock equipping its robot vacuums with powerful NPU (Neural Processing Units). This technological leap is directly impacting the robot vacuum sector. While older robot vacuums merely 'avoided' obstacles by relying on LiDAR or laser sensors, a paradigm shift is now occurring, where high-performance cameras and AI chipsets are used to specifically 'perceive' objects and determine appropriate suction strength. In essence, we have entered an era of visual context awareness—from sensing "there is something" to recognizing "that is a 'power cable'" or "a 'person' is 'sitting' on the 'sofa'." However, currently commercialized products primarily utilize this powerful, newly acquired 'perception' capability for the passive task of 'reactive obstacle avoidance.' The motivation of this project is to

leverage this strong 'perception' ability to fundamentally transform the operational mode of the robot vacuum into a 'predictive' one.

B. Problem Statement

This project seeks to solve the problem: "How can a robot vacuum utilize its perception capabilities to autonomously understand real-life context without user intervention, and proactively perform cleaning by predicting the time and area where contamination will occur?" Two major challenges exist in solving this problem. First, the system must accurately perceive the complex situation within the home (who, what, where) and learn the sequential relationships (patterns) between them. Second, because the system deals with visual/auditory data from the 'home,' an inherently private space, an architecture that absolutely guarantees user privacy is essential.

C. Proposed Solution

We propose the LOCUS (Learning On-device Context and User-specific Schedules) system as the solution. LOCUS is a next-generation on-device AI software technology built upon the aforementioned hardware advancements. The core philosophy of LOCUS is to solve the privacy issue fundamentally by processing all sensitive data exclusively on the device, without transmitting it externally. The LOCUS system is designed to operate through a three-stage multi-modal data processing pipeline:

Stage 1: Real-time Context Perception First, the system perceives semantically segmented spaces like 'kitchen' or 'dining table' using the spatial map built in the Webots environment (FR1). Within this space, lightweight YOLO (visual) and SED (auditory) models detect "who" is doing "what" in real-time (FR2). The output of this stage is a stream of fragmented real-time information, such as ('Kitchen', 'Person Detected', 'Cooking Sounds').

Stage 2: Time-Series Context Vector Transformation To learn 'patterns' over time rather than a mere list of events, the information collected in Stage 1 is fused into a structured context vector with a timestamp (FR2.3). For example, a high-dimensional vector in the form of [time: 19:05, location: 'kitchen table', visual_event: 'person_eating', audio_event: 'clinking_sounds'] is generated every minute or upon every event occurrence.

Stage 3: Sequential Pattern Learning and Prediction The sequence of these context vectors is input into the core system of LOCUS: a lightweight GRU (Gated Recurrent Unit) model (FR3). The GRU was chosen because its simpler structure compared to LSTM allows for powerful time-series pattern learning with fewer resources in an on-device (NFR3) environment. The GRU model learns the probabilistic relationship where a specific event sequence, such as [Cooking Sound (T-30) → Eating (T-10) → Quiet (T-0)], leads to the outcome ["Contamination occurred under the dining table"]. Ultimately, this model outputs a concrete prediction value, such as "85% probability of contamination under the dining table 30 minutes after cooking sounds begin," which provides the clear rationale for the system to actively automate cleaning (FR5), thereby solving the problem while simultaneously addressing privacy concerns.

D. Related Works

1. Multi-modal Sensor Fusion and Context-Aware Robotics

Recent research in robot vacuums focuses on 'Context Awareness' through multi-sensor fusion. Zhou et al. (2025) integrated a YOLO-SBA object detection algorithm and laser SLAM into the ROS2 framework, implementing real-time context recognition for home service robots.

Specifically, this study showed an improvement of 9-10% in dynamic obstacle avoidance performance [1]. This is a case where visual information (camera) and geometric sensors (LiDAR) are fused to enhance real-time robot navigation, but it concentrates on reactive tasks like 'safety' and 'avoidance.' Han et al. (2025)'s latest robot vision survey systematically reviewed multi-modal fusion and Vision-Language Model (VLM) techniques for SLAM, 3D object detection, and object manipulation [2]. The ConceptFusion and similar frameworks discussed in this survey enable multi-modal understanding of the environment by fusing visual, language, and auditory data, an approach similar to LOCUS's YOLO (visual) + SED (auditory) fusion pipeline (FR2). However, these studies also concentrate on improving basic robot capabilities (SLAM, manipulation) and do not fully address the level of 'predictive cleaning automation through user pattern learning' that LOCUS targets.

2. Personalized Federated Learning

The seminal paper for the FedPer (Federated Learning with Personalization Layers) architecture adopted by LOCUS, Arivazhagan et al. (2019), presented a key approach of separating the Base layers (common feature extraction) and Head layers (personalized decision-making) to solve the "statistical heterogeneity" problem in federated learning [3]. While standard Federated Averaging suffers severe performance degradation when user device data distributions are dissimilar (Non-IID), FedPer effectively balances personalization and generalization. This serves as the direct theoretical foundation for LOCUS's FR4 (PFL) module. Building on this, Yang et al. (2024) further enhanced PFL efficiency by combining Reinforcement Learning (RL)-based client selection with local personalization strategies in heterogeneous environments [4]. Since each household's unique living pattern data inevitably has statistically distinct characteristics, Yang et al.'s approach strongly supports the suitability of this architecture for learning household-specific time-series data like 'cleaning patterns' targeted by LOCUS.

3. Privacy Threats and Communication Design for Domestic Robots

Regarding the privacy concerns raised by domestic robot cameras/mics, a USENIX SOUPS 2024 paper by Windl et al. (2024) demonstrated serious privacy threats through a large-scale user study of 1,720 participants [5]. The team found that (1) users' privacy concerns increase exponentially as the robot's mobility (linear \rightarrow planar \rightarrow 3D movement) increases, and (2) users feel "there is no safe space" when the robot can automatically move into private areas like bedrooms and bathrooms. More importantly, the research team developed 86 privacy communication patterns that allow the robot to clearly inform users of its camera/mic/internet connection status. This highlights the importance for LOCUS to clearly communicate to users that "data does not leave the device" through on-device processing, thereby building trust.

4. On-device AI and Lightweight Models: Realization at the Edge

4.1 Real-time Audio Event Classification using YAMNet

TensorFlow's official documentation demonstrates running YAMNet with TensorFlow Lite on a Raspberry Pi environment to detect 521 pre-trained classes (Speech, Dishes, Music, etc.) from a 16kHz audio stream in real-time [6]. This is an empirical precedent that simultaneously satisfies LOCUS's

FR2.2 (SED - Sound Event Detection) requirement and NFR3 (Raspberry Pi Porting).

4.2 GRU vs LSTM: Lightweight and Performance in Time-Series Modeling

A recent study by Yang et al. (2025) applied GRU-KAN (Kolmogorov-Arnold Networks-based GRU) and LSTM-KAN to the problem of early loan default prediction [7]. The results demonstrated that GRU achieved over 92% prediction accuracy with fewer parameters than LSTM. Specifically, this study recorded "over 88% accuracy even for early prediction 8 months prior," showing that in resource-constrained edge environments (Raspberry Pi), GRU can simultaneously satisfy both "lightweight nature" and "long-term time dependency modeling" performance. This academically supports the rationale for LOCUS's FR3.2 (GRU selection).

5. Predictive Smart Home Automation: Active Control Based on Pattern Learning

Existing robot vacuums rely on static schedules or explicit user commands. However, recent research demonstrates the potential for predictive automation. Gad et al. (2025) deployed a LightGBM machine learning model on a Raspberry Pi 5 environment to predict user activities (Watch TV, Cook Dinner, Sleeping, etc.) using spatio-temporal features. They proposed the EL-HARP system for automatically controlling lights and appliances based on these predictions [8]. This study directly proves the feasibility of "predictive automation through pattern learning" on edge devices, sharing the same philosophy as LOCUS's FR5 (Predictive Automation Engine).

Our Differentiation: Evolution to an Intelligent Policy Engine LOCUS's originality lies in integrating and advancing the reviewed preceding studies toward the single goal of 'predictive cleaning automation.'

6. Our Differentiation: Evolution to an Intelligent Policy Engine

LOCUS's originality lies in integrating and advancing the reviewed preceding studies toward the single goal of 'predictive cleaning automation.'

6.1 Paradigm Shift from Reactive to Predictive

While existing studies [1, 2] focused on reactive tasks like robot 'hazard detection' and 'obstacle avoidance,' LOCUS extends context awareness technology to 'living pattern prediction' and 'active cleaning automation.'

6.2 New Application Domain for PFL

We specifically apply the effective PFL approach in heterogeneous environments demonstrated by preceding studies [3, 4]—which typically address power consumption prediction or health monitoring—to the 'cleaning prediction problem based on visual/auditory multi-modal data.'

6.3 Privacy-Centric Design Philosophy

We adopt the requirements of privacy studies [5] as the core system architecture (NFR1), implementing the principles of "all computations on-device" and "no transmission of raw data" to secure both technical performance and user trust.

6.4 Cleaning Automation Engine

While existing predictive automation research [8] focuses on simple appliance control, LOCUS implements complex policies that take the user's situation into account.

E. References

1. L. Zhou, "Research and Simulation of A Home Robot Navigation System Based on YOLO-SBA and ROS2," in *Proceedings of the 2025 2nd International Conference on Mechanics, Electronics Engineering and Automation (ICMEEA 2025)*, Atlantis Press, vol. 272, pp. 465–479, Aug. 2025, doi: 10.2991/978-94-6463-821-9_48.
2. X. Han, S. Chen, Z. Fu, et al., "Multimodal Fusion and Vision-Language Models: A Survey for Robot Vision," *arXiv preprint arXiv:2504.02477*, Apr. 2025. To appear in *Information Fusion*, vol. 126, Feb. 2026.
3. M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, "Federated learning with personalization layers," *arXiv preprint arXiv:1912.00818*, Dec. 2019.
4. H. Yang, J. Li, M. Hao, W. Zhang, H. He, and A. K. Sangaiah, "An efficient personalized federated learning approach in heterogeneous environments: a reinforcement learning perspective," *Sci. Rep.*, vol. 14, article 28877, 2024, doi: 10.1038/s41598-024-80048-3.
5. M. Windl, J. Leusmann, A. Schmidt, S. S. Feger, and S. Mayer, "Privacy communication patterns for domestic robots," in *Proceedings of the 2024 Symposium on Usable Privacy and Security (SOUPS 2024)*, USENIX, Philadelphia, PA, USA, pp. 121–138, Aug. 2024. [Online]. Available: <https://www.usenix.org/system/files/soups2024-windl.pdf>
6. TensorFlow, "Transfer learning with YAMNet for environmental sound classification," TensorFlow Documentation, 2024. [Online]. Available: https://www.tensorflow.org/tutorials/audio/transfer_learning_audio
7. Y. Yang, Z. Su, Y. Zhang, C. C. Goh, Y. Lin, A. G. Bellotti, and B. G. Lee, "Kolmogorov–Arnold networks-based GRU and LSTM for loan default early prediction," *arXiv preprint arXiv:2507.13685*, Jul. 2025.
8. M. M. Gad, W. Gad, T. Abdelkader, and K. Naik, "Personalized smart home automation using machine learning: Predicting user activities," *Sensors*, vol. 25, no. 19, article 6082, Oct. 2025, doi: 10.3390/s25196082.

II. REQUIREMENTS A

For the successful development and operation of the LOCUS system, the following Functional Requirements (FR) and Non-Functional Requirements (NFR) are defined. LOCUS consists of the following components:

- 1) **Semantic Mapping Module (FR1)**
 - Spatial structure and semantic information extraction
- 2) **Context Recognition Module (FR2)**
 - Context recognition using visual and auditory input
- 3) **Sequential Pattern Learning Module (FR3)**
 - Time-series pattern learning and prediction
- 4) **Personalized Federated Learning Module (FR4)**
 - Federated learning and personalization
- 5) **Predictive Automation Engine (FR5)**

- Prediction-based action automation

6) **Dataset Management Module (FR6)**

- Learning and simulation data management

B. Functional Requirements (FR)

FR1: Semantic Space Mapping and Management (Semantic Mapping)

ID	Requirement
FR1.1 (Base Map Data Assumption)	The project does not perform real-time map generation via the robot's own SLAM (Simultaneous Localization and Mapping). Instead, the system must load and use an existing building floor plan file (e.g., 2D image) provided by the user (similar to the LG ThinQ app) as the base map.
FR1.1a (Simulation Environment)	This loaded floor plan must be constructed as a Virtual World within the Webots (Cyberbotics) simulation environment. The robot simulator will run within this Webots environment.
FR1.1b (Data Utilization)	This Webots-based virtual map shall be used as the Ground Truth Map for robot environment perception and absolute position estimation.
FR1.2 (User-Defined Zone Setting)	Users must be able to create polygonal or circular areas on the floor plan (FR1.1). Each area must be assigned a semantic label (e.g., "Kitchen Table," "Living Room Sofa Area," "Entrance Hall"), which serves as the base data for spatial recognition and action triggers. Creation, modification, and deletion of zones must all be possible.
FR1.3 (Real-time Position Reception)	The system must receive the robot's real-time (x, y) coordinates from the Webots robot simulator. The reception frequency must be at least every 0.5 seconds, and the coordinate values must be transformed and synchronized to the Webots World Coordinate System and the loaded floor plan's coordinate system (FR1.1).
FR1.4 (Space Mapping and Zone Lookup)	The system must judge in real-time whether the received coordinates (FR1.3) are included in a user-defined zone (FR1.2). The matching result must be returned in the form of "Coordinate → Semantic Label," e.g., (3.1, 2.5) → "Kitchen Table." This must be instantly reflected in subsequent activity recognition and event triggers (FR2.2).

FR2: Multi-modal Real-time Context Awareness (Multimodal Context Awareness)

ID	Requirement
----	-------------

FR2.1.1 (Model Selection - Visual)	To suit the on-device environment, a lightweight object detection model, YOLOv8n or YOLOv8n-pose (including pose estimation), shall be used after quantization with TensorFlow Lite.
FR2.1.2 (Detection Classes - Visual)	The model must detect at least person, pet, sofa, table, chair. If the Pose model is used, it must be able to infer activities such as person_sitting, person_standing, person_lying_down.
FR2.1.3 (Composite Recognition)	The system must combine the location information from FR1.4 with the detection results from FR2.1.2 to generate a composite context, such as "person_lying_down" detected in the "Living Room Sofa Area" (e.g., "Watching TV or Resting").
FR2.1.4 (Contamination Detection - Extension)	(Extension Feature) The YOLO model can be custom-trained (Fine-tuning) to detect specific forms of contamination, such as crumbs or liquid_spill.
FR2.2.1 (Model Selection - Auditory)	Considering the on-device environment, the pre-trained lightweight audio event classifier YAMNet (TensorFlow Lite version) shall be used as a base.
FR2.2.2 (Detection Classes - Auditory)	Utilizing YAMNet's base classes, the system must focus on and filter for events critical to context understanding, such as "Speech," "Television," "Music," "Dishes, pots, and pans," and "Silence."
FR2.3.1 (Data Schema - Fusion)	The results from FR1.4 (Location), FR2.1 (Visual), and FR2.2 (Auditory) must be fused into a formalized 'Context Vector' with a timestamp. (Example: {"timestamp": 1678886400, "zone": "kitchen_table", "visual_events": [...], "audio_events": [...]})
FR2.3.2 (Data Storage)	This Context Vector must be stored in a sequence format in a lightweight on-device database (e.g., SQLite) for time-series analysis (FR3).

FR3: Sequential Life Pattern Learning (Sequential Pattern Learning)

ID	Requirement
FR3.1 (Input Data)	The on-device GRU model must accept a sequence of 'Context Vectors' (e.g., 30 vectors aggregated per minute over the last 30 minutes) stored in FR2.3.2 as input.

FR3.2 (Model Architecture)	The model must be configured with 1 to 2 layers of GRU (Gated Recurrent Unit) with 32 to 128 hidden units, followed by several Dense (FC) layers to predict the 'probability of contamination occurrence,' considering the resource constraints of NFR3 (Raspberry Pi level).
FR3.3 (Output (Prediction))	The model must output the "probability of contamination occurring within the next 15 minutes" for each semantic zone (FR1.2), as a value between 0.0 and 1.0. (Example: {"predict_kitchen_table": 0.85, "predict_living_room_sofa": 0.15})
FR3.4 (Local Training)	The training of this GRU model must be performed on-device according to the PFL policy defined in FR4.2 and FR4.3.

FR4: Model Update via Personalized Federated Learning (PFL)

ID	Requirement
FR4.1 (Architecture)	The system shall adopt the FedPer (Federated Learning with Personalization Layers) architecture, clearly separating the FR3.2 GRU model into 'Base' and 'Head' layers.
FR4.2.1 (Base Layer - Shared Knowledge)	The Base layers (GRU layers) are responsible for learning universal life patterns common across all households (e.g., "Cooking \rightarrow Dining \rightarrow Cleanup" sequences, or standard acoustic pattern temporal features).
FR4.2.2 (Base Training)	The weights of the Base layers shall be exchanged with the Central Server (FR4.4) according to the federated learning protocol. Local Base weights trained by each robot are sent to the server, which aggregates (FedAvg) them to redistribute an improved Global Base Model to all clients.
FR4.3.1 (Head Layer - Personalization)	The Head layers (Dense layers) receive the generalized features extracted by the Base and learn the unique habits and cleaning needs of that specific household (e.g., predicting 90% contamination probability under the table for House A vs. 70% near the living room sofa for House B, given the same "dinner sequence").
FR4.3.2 (Privacy Guarantee - Head)	The Head layer weights, containing household-specific patterns, must never be transmitted outside the device for privacy protection (adherence to NFR1.1). This layer is trained and stored only on-device using the local data (FR2.3.2).

FR4.4.1 (FL Framework - Server)	The lightweight open-source framework Flower shall be used for PFL coordination. The central coordination server shall be built on the cloud (AWS) and perform aggregation strategies like FedAvg.
FR4.4.2 (FL Framework - Client)	The on-device component (robot simulator) must implement the Flower Client logic.
FR4.4.3 (PFL Process)	The client must, upon server request: (1) Receive the Global 'Base' model from the server, (2) Train both 'Base' and 'Head' locally with local data, and (3) Transmit only the updated 'Base' model weights back to the server.

FR5: Intelligent Cleaning Policy Engine

The system must determine and execute the optimal cleaning policy based on the prediction results from FR3 (Pattern Learning), taking into account FR2 (Context) and accumulated feedback (Confidence).

ID	Requirement
FR5.1 (Confidence & Context Assessment)	Before triggering the auto-clean logic (FR5.2), the system must evaluate both the prediction probability from FR3.3 and the pattern's past 'Feedback Success Rate (Confidence Score).' Simultaneously, it must scan FR2 (Context) to check for a 'Do Not Disturb (DND)' state (e.g., "Watching TV").
FR5.2 (Dynamic Action Policy Decision)	Based on the assessment in FR5.1, the system must automatically select the most appropriate action: (a) Silent Auto-Clean (If confidence is very high and no DND state is active, immediately call FR5.7). (b) Suggestive Clean (If confidence is moderate/new pattern and no DND state is active, trigger the user suggestion FR5.3). (c) Delayed Clean (If DND state is detected, defer cleaning/suggestion execution until the DND state is cleared). (d) Ignore & Log (If prediction confidence is below the threshold, log the failure to the local DB without disturbing the user).
FR5.3 (User Suggestion & Explicit Feedback)	If FR5.2(b) is decided, a push notification ("Contamination is expected near the dining table. Start cleaning? [Y/N]") must be sent to the user's mobile app (NFR4.1). The user's [Y/N] response must be stored immediately as 'Explicit Feedback' in the FR5.6 DB.
FR5.4 (DND Mode Adherence)	If FR5.2(c) is decided, the system must automatically delay the FR5.7 API call or FR5.3 suggestion until it detects the context in FR2 has changed to a "Silence" state.

FR5.5 (Implicit Feedback Collection)	User follow-up actions after an FR5.2 decision must be tracked as 'Implicit Feedback' to prevent poor UX: (a) (Positive Inference): If a suggestion was ignored but the user manually commanded cleaning of 'that zone' within 1 hour, store as 'Yes (Cleaning Needed).' (b) (Negative Inference): If 'Silent Auto-Clean' was performed but the user issued a 'Stop Cleaning' command within 5 minutes, store as 'No (Prediction Error/Disturbance).' (c) (Negative Inference): If a prediction was 'Ignored,' but the contamination sensor (FR4) returned 'Low' during the next scheduled clean, store as 'No (Prediction Error).'
FR5.6 (Feedback-driven Local Retraining)	All collected feedback (FR5.3 Explicit, FR5.5 Implicit) must be considered 'Ground Truth Labels' and stored in the on-device DB (FR2.3.2). This data must be used for retraining the FR4.3 'Head Layer' and updating the FR5.1 'Confidence Score.'
FR5.7 (Core System Interface)	Once cleaning is confirmed (FR5.2(a) or [Y] response from FR5.3), the system must send a standardized JSON command (e.g., {"action": "clean", "zone_label": "kitchen_table", "power": "auto"}) to the Core System API Stub

FR6: Training Dataset Generation and Management

ID	Requirement
FR6.1 (Simulation Environment Provision)	The Core System Module must provide a 'Data Simulator' that simulates multi-modal data (Position, Visual, Auditory) defined in FR1.3, FR2.1, FR2.2, and generates it as a real-time stream conforming to the 'Context Vector' JSON schema defined in FR2.3.1.
FR6.2 (Scenario-based Mock Data Definition)	QA & Documentation Subject must define Ground Truth scenarios as Mock data (e.g., "A specific 'Context Vector' sequence occurring results in 'Contamination Occurrence (Label=1)' in a specific 'Zone'" based on core Use Cases (e.g., "Dining after Cooking," "Snacking while Watching TV").
FR6.3 (Initial Training Dataset Construction)	The simulation data from FR6.1 and the Mock data from FR6.2 must be combined to construct an Initial (Bootstrap) Labeled Time-series Dataset for the FR3 GRU model and FR4 PFL process, and stored in SQLite (FR2.3.2).

C. Non-Functional Requirements (NFR)

NFR1: Privacy & Security

ID	Requirement
NFR1.1	The user's raw camera video and microphone audio data must never be transmitted outside the device.
NFR1.2	All visual/auditory data processing must be completed On-device.
NFR1.3	Data transmitted to the server during Personalized Federated Learning (PFL) (FR4.3) must be restricted to non-personally identifiable model weights (gradients).

NFR2: Performance

ID	Requirement
NFR2.1 (Inference)	Real-time inference speed for the Context Recognition (FR2) YOLO, SED, and GRU models must be completed within a few seconds.
NFR2.2 (Training)	The Pattern Learning (FR3) and model update (FR4) processes must run in the background when the robot is charging or idle, so as not to affect the robot's primary cleaning performance.

NFR3: Resource Constraints

ID	Requirement
NFR3.1	The system must be developed and tested primarily in a standard PC local environment. Finally, all AI models must be lightweight and ported to be operable on hardware resources restricted to the level of [e.g., Raspberry Pi 4 or a similar embedded board] (CPU, low-capacity RAM).

NFR4: Usability

ID	Requirement
NFR4.1	Users must be able to easily edit the semantic map (FR1.2) via the application.
NFR4.2	Users must be able to easily configure the frequency of cleaning suggestions (FR5.2) and the Do Not Disturb (FR5.3) options.

NFR5: Error Handling & Robustness

ID	Requirement
----	-------------

NFR5.1 (Network Exception)	If network connectivity is interrupted during PFL (FR4.4), local training must continue, and the 'Base' model parameter transmission must be retried N times (e.g., 3 times) until the network is restored.
NFR5.2 (Model Loading Failure)	If any AI model (FR2.1, FR2.2, FR3) fails to load, the corresponding functionality must be disabled, the system must operate only in 'Manual Cleaning Mode,' and the user must be explicitly notified of the error state via the mobile app (NFR4.1).
NFR5.3 (Data Collection Failure)	If the reception of sensor data (FR1.3) is interrupted, the Context Recognition (FR2) and Prediction (FR3) modules must be temporarily paused.

III. DEVELOPMENT ENVIRONMENT

A. GitHub Repository

URL: https://github.com/dayeon-21/AI_Software_Project

B. Development Platform and Language

1) Platforms

-1. Simulation

Webots (Cyberbotics) will be used to simulate robot movement and sensor data (camera, position). This provides an environment for safely and repeatedly verifying algorithms without physical robot hardware.

-2. AI Development/Training

Initial development, training, and testing of AI models (YOLO, YAMNet, GRU) will take place in a Python-based Local PC Environment, utilizing powerful resources for rapid training and debugging.

-3. On-device Porting

The primary goal is to port developed AI models to a Raspberry Pi 5 (8GB RAM) to verify performance in the low-power, low-specification environment defined in NFR3. This is a key step to confirm real-world operability.

-4. Server

The PFL Central Server (Flower) will be hosted using the AWS Free Tier, enabling cost-efficient operation of the coordination server for PFL computations.

-5. UI/UX

A Mobile Application (React Native or Flutter) will be implemented (NFR4.1) to handle user interaction (map setup, notifications). This interface allows users to set up zones (FR1.2) and respond to prediction alerts (feedback) (FR5.2).

2) Programming Language: Python (Rationale)

-1. AI/ML Ecosystem

It has an overwhelmingly strong library ecosystem (TensorFlow, Keras, etc.) essential for AI model development, allowing for rapid integration and testing of models like YOLO and TFLite.

-2. Rapid Prototyping

Its concise syntax allows for quickly implementing and testing ideas.

-3. Library Compatibility

It seamlessly supports all core project libraries: Flower (PFL), TensorFlow Lite (Lightweighting), Webots (Simulation API), and Paho-MQTT (Communication), enabling integrated development across all modules using a single language.

3) Deployment and Lightweighting Path

The project follows a phased lightweighting path:

- **Local PC:** Used for functional implementation and initial model performance verification.
- **Raspberry Pi (SoC):** Models will be converted/quantized using TensorFlow Lite to test real on-device inference performance within the Raspberry Pi 5 environment.
- **(Future Work) Microcontroller:** After performance verification on the Raspberry Pi, the project will explore expansion to smaller RAM microcontroller environments (e.g., Arduino, ESP32) using TensorFlow Lite for Microcontrollers.

C. Development Environment Details

- Operating Systems: Windows 11, macOS 14 (Sonoma)
- IDE: Visual Studio Code
- Python Version: 3.10.x or higher
- AI/ML Frameworks: TensorFlow 2.15+, Keras 3.x, TensorFlow Lite
- PFL Framework: Flower (flwr) 1.7+
- Simulation: Webots R2023b+
- Database: SQLite 3
- Communication: Paho-MQTT (MQTT Broker)

D. Cost Estimation

The total estimated cost aims to be minimized by utilizing open-source and free-tier cloud resources.

- **Hardware:** One Raspberry Pi 5 (8GB RAM) (Approx. ¥150,000).
- **Software:** All core software, including Webots, Python, and TensorFlow, will be open-source (¥0).
- **Cloud:** PFL Central Server (FR4.4.1) hosting will utilize the AWS Free Tier. The goal is to operate within the monthly free usage limits, resulting in an additional cost of ¥0.
- **Total Estimated Cost:** Approx. ¥150,000.

E. Software in Use (Summary)

- 1) Webots: Robot and virtual environment simulation (FR1)
- 2) YOLOv8n: Lightweight object/pose detection (FR2.1)
- 3) YAMNet: Audio event classification (FR2.2)
- 4) GRU (Gated Recurrent Unit): Lightweight time-series pattern learning, replacing LSTM (FR3)

5) Flower: PFL implementation based on the FedPer architecture (FR4)

IV. SPECIFICATIONS

This section describes in detail "How" each requirement (FR/NFR) defined in Section II. Requirements will be achieved, using detailed architecture and pseudocode.

A. System Architecture & Data Flow

The system consists of six core modules, with each module communicating asynchronously via MQTT topics.

1) [FR1] Semantic Mapping Module

Role: Converts the robot's physical (x, y) coordinates into semantic space labels ("kitchen").

Input: Robot (x, y) coordinates from the Webots simulator (FR1.3). Zone definitions set via the mobile app (FR1.2). Output: MQTT Publish (Topic: 'locus/current_zone') - {"zone": "kitchen_table"}

2) [FR2] Context Recognition Module (AI/ML)

Role: Performs real-time inference on visual (YOLO) and auditory (YAMNet) sensor data, generating a formalized "Context Vector" with current location information.

Input: MQTT Subscribe (Topic: 'locus/current_zone') (FR1.4). Webots Camera Stream (FR2.1), Webots Mic Stream (FR2.2).

Output: MQTT Publish (Topic: 'locus/context') - Formalized Context Vector (FR2.3.1).

3) [FR3] Sequential Pattern Learning Module (AI/ML)

Role: Receives/stores the Context Vector sequence and periodically predicts future contamination probability using the GRU model.

Input: MQTT Subscribe (Topic: 'locus/context') (FR2.3.2). Output: MQTT Publish (Topic: 'locus/prediction') - {"predict_kitchen_table": 0.85, ...} (FR3.3).

4) [FR4] PFL Module (AI/ML)

Role: Trains the [FR3] GRU model on-device according to the FedPer architecture (Base/Head) and exchanges only Base model weights with the central server.

Input: [FR3]'s Local DB (FR2.3.2), [FR5]'s Feedback DB (FR5.6). Global model request from the Flower Central Server (AWS) (FR4.4.3).

Output: Transmits updated Base model weights to the Flower server (NFR1.3).

5) [FR5] Predictive Automation Engine (Core System)

Role: Synthesizes prediction probability (FR3), current context (FR2), and feedback (FR5.5) to determine the final action policy, such as 'Immediate Clean,' 'Suggestion,' or 'Delay.'

Input: MQTT Subscribe (Topic: 'locus/prediction') (FR3.3). MQTT Subscribe (Topic: 'locus/context') (FR5.1). Explicit feedback [Y/N] from the mobile app (FR5.3).

Output: MQTT Publish (Topic: 'locus/command') - {"action": "clean", ...} (FR5.7). Mobile app push notification (FR5.3).

6) [FR6] Dataset Management Module (Simulation)

Role: Generates a stream of virtual Context Vectors (FR6.1) and Ground Truth labels (FR6.3) based on

defined scenarios (FR6.2) for initial model bootstrap training.

Input: Use Case Scenarios (FR6.2).

Output: Initial locus_context.db for the [FR3] module (FR6.3).

B. FR1: Semantic Mapping

The SemanticMapper module receives physical coordinates from Webots via MQTT, converts them into semantic Zone labels, and publishes the result.

// [FR1] Module (Core System)

// 1. System Initialization

// FR1.2: Load Zone Polygon information defined in the mobile app

```
GLOBAL          ZONES          =  
Load_Zone_Polygons_From_Config("zone_config.json")
```

// 2. Event Handler called upon receiving coordinates from Webots (FR1.3)

```
FUNCTION On_Robot_Position_Update(x, y):  
    current_point = Create_Point(x, y)  
    current_zone = "None" // Default value
```

// 3. FR1.4: Zone Matching

```
FOR EACH zone_name, zone_polygon IN ZONES:  
    IF zone_polygon CONTAINS current_point:  
        current_zone = zone_name  
    BREAK // Use the first matching zone
```

// 4. Publish the queried current Zone label

```
payload = {"zone": current_zone, "timestamp":  
Get_Current_Timestamp()}  
PUBLISH "locus/current_zone" PAYLOAD payload
```

// 5. Module Startup

// Subscribe to the position publication topic from the Webots simulator (Example)

```
SUBSCRIBE "webots/robot_position" CALLBACK  
On_Robot_Position_Update
```

C. FR2: Multimodal Context Awareness

The ContextAwarenessModule subscribes to locus/current_zone, executes TFLite models, and publishes the locus/context vector every second.

// [FR2] Module (AI/ML)

// 1. Global Variables and Model Initialization

```
GLOBAL current_zone = "None"  
GLOBAL last_visual_events = []  
GLOBAL last_audio_events = []  
GLOBAL          yolo_model          =  
Load_TFLite_Model("yolov8n.tflite") // FR2.1.1  
GLOBAL          yamnet_model         =  
Load_TFLite_Model("yamnet.tflite") // FR2.2.1
```

// 2. Event Handler Definition

```
FUNCTION On_Zone_Update(payload):  
    current_zone = payload.zone
```

FUNCTION On_Camera_Frame(frame):

```
    visual_results = yolo_model.Predict(frame) // FR2.1.2  
    // FR2.1.3: Parse composite recognition results like  
    "person_sitting"  
    last_visual_events          =  
    Parse_YOLO_Results(visual_results)
```

FUNCTION On_Audio_Chunk(chunk):

```
    audio_results = yamnet_model.Predict(chunk) //  
    FR2.2.2
```

// Robot's own cleaning noise filtering logic (Detail)

IF "Vacuum_cleaner" IN audio_results AND
confidence > 0.8:

```
    last_audio_events = [{"event": "Vacuum_cleaner",  
"confidence": 0.99}]
```

```
ELSE:  
    last_audio_events          =  
    Filter_Interesting_Audio(audio_results)
```

// 3. Main Loop (Separate process running every 1 second)

PROCEDURE Create_Context_Vector_Periodically():

```
    WHILE true:  
        // FR2.3.1: Generate Context Vector from the latest  
        collected information over 1 second  
        context_vector = {  
            "timestamp": Get_Current_Timestamp(),  
            "zone": current_zone,  
            "visual_events": last_visual_events,  
            "audio_events": last_audio_events  
        }
```

// FR2.3.1: Publish to the [FR3] module

```
PUBLISH          "locus/context"          PAYLOAD  
context_vector  
SLEEP 1.0 seconds
```

// 4. Module Startup

```
SUBSCRIBE "locus/current_zone" CALLBACK  
On_Zone_Update  
START_STREAM Webots_Camera CALLBACK  
On_Camera_Frame  
START_STREAM Webots_Audio CALLBACK  
On_Audio_Chunk  
START_PROCESS Create_Context_Vector_Periodically
```

D. FR3: Sequential Pattern Learning

The PatternLearningModule subscribes to locus/context, saves it to the DB, and publishes locus/prediction every minute.

// [FR3] Module (AI/ML)

// 1. Initialization

```
GLOBAL          DB_Connection          =  
Connect_SQLite("locus_context.db") // FR2.3.2  
GLOBAL          GRU_Model              =  
Load_TFLite_Model("gru_pattern_model.tflite") //  
FR3.2  
GLOBAL ZONES_TO_PREDICT = ["kitchen_table",  
"living_sofa"] // Synchronized with FR1.2 setting
```

```

// 2. Context Vector Reception and Storage
FUNCTION On_Context_Vector_Received(vector):
    // FR2.3.2: Save the received vector to the local DB
    DB_Connection.Save("context_vectors", vector)

// 3. Periodic Prediction (Separate process running every
1 minute)
PROCEDURE Predict_Mess_Probability_Periodically():
    WHILE true:
        // FR3.1: Load data from the last 30 minutes
        raw_vectors = DB_Connection.Query("SELECT *
FROM context_vectors WHERE timestamp > (NOW -
30_MINUTES)")

        // FR3.1: Aggregate into a time-series sequence (e.g.,
1-minute Max-pooling)
        // 1800 vectors (1s*60*30) -> Transform into a 30-
vector sequence
        input_sequence =
Aggregate_Vectors_By_Minute(raw_vectors, length=30)

        // FR3.2: Preprocess into a tensor for model input
        input_tensor =
Preprocess_For_GRU(input_sequence)

        // FR3.3: Model Inference
        prediction_output =
GRU_Model.Predict(input_tensor) // e.g., [0.85, 0.15]

        // FR3.3: Format Prediction Result
        probabilities =
Format_Prediction_With_Labels(prediction_output,
ZONES_TO_PREDICT)

        // Publish prediction result to the [FR5] module
        PUBLISH "locus/prediction" PAYLOAD
        probabilities

        SLEEP 60.0 seconds

// 4. Module Startup
SUBSCRIBE "locus/context" CALLBACK
On_Context_Vector_Received
START PROCESS
Predict_Mess_Probability_Periodically

```

E. FR4: Personalized Federated Learning

The implementation of this module is based on the Flower official documentation's "Quickstart TensorFlow" (or PyTorch) example. The FedPer (FR4.1) architecture is implemented using a helper function (BASE_MODEL, HEAD_MODEL) = load_fedper_model(...).

1) Core Strategy

Base: Flower official quickstart_tensorflow.py example.

Client: The example's CifarClient is changed to LocusClient.

Data: tf.keras.datasets.cifar10.load_data() is replaced with load_local_data_from_db() (loading data from FR2.3.2 and FR5.6).

FedPer Application: The LocusClient's get_parameters and fit methods are modified to exchange only BASE_MODEL weights.

// [FR4] Module (AI/ML)

```

// 1. Initialization (Model Separation)
// load_fedper_model: Custom helper function to load an
H5 file and separate into Base and Head
(BASE_MODEL, HEAD_MODEL) =
load_fedper_model("gru_pattern_model.h5",
"base_gru_layer_name")

```

```

// 2. Local Data Loader
FUNCTION load_local_data_from_db():
    // Load training data from FR2.3.2 (Context) and
FR5.6 (Feedback) DBs
    (X_train, y_train) = DB_Connection.Query("SELECT
sequences, labels FROM training_data")
    RETURN (X_train, y_train)

```

```

// 3. Flower Client Definition (Modified based on official
example)
// (Inherits from fl.client.NumPyClient)
CLASS LocusClient:

```

```

    FUNCTION get_parameters(config):
        // FR4.2.2: Transmit ONLY 'Base' model weights to
the server
        PRINT "PFL: Sending local base model to server."
        RETURN BASE_MODEL.get_weights()

```

```

    FUNCTION fit(parameters, config):
        // FR4.4.3 (1): Receive the global 'Base' model
(parameters) from the server
        PRINT "PFL: Received global base model."
        BASE_MODEL.set_weights(parameters)

```

```

        // FR4.4.3 (2): Load local data
        (X_train, y_train) = load_local_data_from_db()

        // FR4.3.2: Combine 'Base' and 'Head' for local
training
        PRINT "PFL: Starting local training (Base +
Head)..."
        combined_model =
Combine_Models(BASE_MODEL, HEAD_MODEL)
        combined_model.fit(X_train, y_train, epochs=1)
        PRINT "PFL: Local training complete."

```

```

        // FR4.2.2 (4): Return ONLY updated 'Base' weights
(Ensuring NFR1.1 privacy)
        RETURN BASE_MODEL.get_weights(),
len(X_train), {}

```

// (evaluate function omitted)

```

// 4. Client Execution (NFR2.2)
PROCEDURE Start_PFL_Client_If_Idle():

```

```

    WHILE true:
        IF Robot_Is_Idle() AND Network_Is_Available(): //
NFR5.1
            PRINT "PFL: Robot idle. Connecting to server..."
            // FR4.4.1: Connect to the Flower server built on
AWS
            fl.client.start_numpy_client(
                server_address="AWS_FLOWER_SERVER_I
P:8080",
                client=LocusClient()
            )
            SLEEP 10_MINUTES // Check idle status every 10
minutes

```

F. FR5: Predictive Automation Engine

The PolicyEngine subscribes to both locus/prediction and locus/context to determine the action policy (DND, auto-clean, suggest) defined in FR5.2.

// [FR5] Module (Core System)

// 1. Initialization

```

GLOBAL Feedback_DB =
Connect_SQLite("locus_feedback.db") // FR5.6
GLOBAL current_dnd_status = FALSE
GLOBAL current_context = {} // FR5.1 (For DND check)
GLOBAL PREDICTION_THRESHOLD = 0.70
GLOBAL HIGH_CONF_THRESHOLD = 0.95

```

// 2. Event Handlers

```

FUNCTION On_Context_Update(vector):
    current_context = vector
    // FR5.4: Update DND (Do Not Disturb) status
    current_dnd_status = Check_DND_Status(vector) //
e.g., Detect TV or Speech event

```

```

FUNCTION On_Prediction_Received(probabilities):
    // FR5.1: Evaluate prediction probability
    FOR EACH zone, probability IN probabilities:
        IF probability > PREDICTION_THRESHOLD:
            // FR5.1: Query 'Confidence Score' based on past
feedback
            pattern_id = Hash_Pattern(current_context) //
Identify current pattern
            confidence =
Feedback_DB.Get_Confidence_Score(zone, pattern_id)

```

```

        // FR5.2: Determine Action
        Determine_Action(zone, probability, confidence,
pattern_id)

```

// 3. Action Determination Logic

```

FUNCTION Determine_Action(zone, probability,
confidence, pattern_id):
    // FR5.4: DND status is priority 1
    IF current_dnd_status IS TRUE:
        PRINT f"Action (Delayed): DND mode. Delaying
clean for {zone}."
        Log_Action_To_DB("delayed_dnd", zone,
pattern_id)
    RETURN

```

```

// FR5.2(a): Silent Auto-Clean (Very High Confidence)
IF probability > HIGH_CONF_THRESHOLD AND
confidence > 0.9:
    PRINT f"Action (Auto-Clean): High confidence.
Cleaning {zone}."

```

```

        Execute_Clean_Command(zone) // FR5.7
        Log_Action_To_DB("auto_clean", zone, pattern_id)

```

// FR5.2(b): Suggestive Clean

```

ELSE:
    PRINT f"Action (Suggest): Sending suggestion for
{zone}."
    Send_Cleaning_Proposal(zone) // FR5.3
    Log_Action_To_DB("suggested", zone, pattern_id)

```

// 4. Feedback Processing

```

FUNCTION Send_Cleaning_Proposal(zone):
    // FR5.3: Push notification to mobile app (NFR4.1)
    Send_Push_Notification(user_id="...",
message=f"Contamination expected in '{zone}' area.
Start cleaning? [Y/N]")

```

```

FUNCTION On_Explicit_Feedback_Received(response):
    // [Y/N] received from mobile app
    zone = response.zone
    pattern_id = response.pattern_id // Pattern ID sent
with suggestion
    feedback = (response.answer == "Y") ? "Positive" :
"Negative"

```

```

    // FR5.6: Save feedback to DB -> Used for the next FR4
training cycle
    Feedback_DB.Save_Feedback_Label(zone, pattern_id,
feedback)

```

```

    IF feedback == "Positive":
        Execute_Clean_Command(zone) // FR5.7

```

```

FUNCTION Execute_Clean_Command(zone):
    // FR5.7: Publish cleaning command to Core System
    PUBLISH "locus/command" PAYLOAD {"action":
"clean", "zone_label": zone}

```

// 5. Implicit Feedback (Separate Process)

```

PROCEDURE
Monitor_Implicit_Feedback_Periodically():
    WHILE true:
        // FR5.5(a): Manual clean within 1 hour after
ignoring suggestion
        ignored_suggestions =
Feedback_DB.Find_Logs("suggested",
since=1_HOUR_AGO)
        FOR EACH sug IN ignored_suggestions:
            IF User_Manually_Cleaned(sug.zone) WITHIN
(sug.timestamp, 1_HOUR):
                Feedback_DB.Save_Feedback_Label(sug.zone,
sug.pattern_id, "Positive")

```

```

        // FR5.5(b): Stop command within 5 minutes after
auto-clean

```

```

    auto_cleans                                     =
Feedback_DB.Find_Logs("auto_clean",
since=5_MINUTES_AGO)
    FOR EACH clean IN auto_cleans:
        IF User_Stopped_Clean(clean.zone) WITHIN
(clean.timestamp, 5_MINUTES):
            Feedback_DB.Save_Feedback_Label(clean.z
one, clean.pattern_id, "Negative_Disturbed")

    SLEEP 5_MINUTES

// 6. Module Startup
SUBSCRIBE      "locus/prediction"      CALLBACK
On_Prediction_Received
SUBSCRIBE      "locus/context"        CALLBACK
On_Context_Update
LISTEN_FOR_APP_FEEDBACK(CALLBACK
On_Explicit_Feedback_Received)
START_PROCESS
Monitor_Implicit_Feedback_Periodically

```

G. FR6: Dataset Generation

The DatasetGenerator reads the scenario definition (FR6.2) and, according to FR6.1, publishes locus/context (data) and locus/ground_truth_label (ground truth). The DatasetBuilder subscribes to these to create the initial training DB (FR6.3).

```

// [FR6.2] Scenario Definition (scenario.json)
DEFINE SCENARIO "Dinner":
    // Phase 1: Cooking (18:00-18:30)
    PHASE 1 (DURATION 30_MINS):
        CONTEXT: {"zone": "kitchen", "visual":
"person_standing", "audio": "Dishes"}
        LABEL: {"zone": "kitchen", "label": 0}

    // Phase 2: Eating (18:30-19:00)
    PHASE 2 (DURATION 30_MINS):
        CONTEXT: {"zone": "kitchen_table", "visual":
"person_sitting", "audio": "Speech"}
        LABEL: {"zone": "kitchen_table", "label": 0}

    // Phase 3: Finished (19:01)
    PHASE 3 (AT T+61_MINS):
        CONTEXT: {"zone": "kitchen_table", "visual":
"None", "audio": "Silence"}
        LABEL: {"zone": "kitchen_table", "label": 1} //
Contamination Occurs

// [FR6.1] Data Simulator (Generator)
PROCEDURE Run_Simulation_Scenario():
    scenario = Load_Scenario("scenario.json")

    FOR EACH phase IN scenario:
        FOR 1 second UP TO phase.duration:
            // FR6.1: Publish virtual context vector
            PUBLISH      "locus/context"      PAYLOAD
phase.CONTEXT
            // FR6.3: Publish ground truth label alongside
            PUBLISH "locus/ground_truth_label" PAYLOAD
phase.LABEL

```

```

    SLEEP 1s
    PRINT "Simulation Complete."

// [FR6.3] Initial Training Dataset Builder
GLOBAL      DB                                     =
Connect_SQLite("locus_initial_training.db")

FUNCTION On_Context_Vector(vector):
    DB.Save("context_vectors", vector)

FUNCTION On_Ground_Truth(label_data):
    DB.Save("labels", label_data) // Store ground truth
label

// Builder Main Logic
SUBSCRIBE      "locus/context"      CALLBACK
On_Context_Vector
SUBSCRIBE      "locus/ground_truth_label" CALLBACK
On_Ground_Truth
PRINT "Dataset Builder running... Please execute the
[FR6.1] simulator."
// (Once simulation is complete, this script finishes the
initial training DB)
// (This DB is then loaded by [FR4] using
`load_local_data_from_db` for initial training)

```