EUROPEAN JOURNAL OF EMERGING REAL-TIME IOT AND EDGE INFRASTRUCTURES

MULTI-LAYERED FEATURE MODELS FOR ENHANCED IOT APPLICATION DEPLOYMENT IN EDGE ENVIRONMENTS

Dr. Caroline M. Rivers
Department of Communication, Utah State University, Logan, UT, USA

Dr. Jared E. Nolan
Department of Political Science, Western Carolina University, Cullowhee, NC, USA

V0LUME01 ISSUE01 (2024)

Published Date: 29 December 2024 // Page no.: - 49-68

ABSTRACT

The pervasive growth of Internet of Things (IoT) applications necessitates robust and efficient deployment strategies, particularly within the constrained and dynamic environments of edge computing infrastructures. Traditional cloud-centric models often suffer from high latency and bandwidth limitations, making edge computing a crucial paradigm for processing data closer to its source [6, 17, 45, 46]. This article explores the application of multi-layered feature models as a sophisticated approach to support and optimize the deployment of diverse IoT applications on heterogeneous edge-based infrastructures. Feature models, a cornerstone of Software Product Line Engineering (SPLE), provide a structured way to represent commonalities and variabilities within a system [12, 20]. By extending these models to multiple layers, we can capture the intricate interdependencies between IoT application features, underlying edge infrastructure capabilities, and deployment configurations. This approach facilitates automated reasoning, configuration, and optimization of deployment decisions, addressing challenges such as resource allocation, energy efficiency, and latency reduction in dynamic edge environments [24, 25, 34, 49]. We discuss the theoretical foundations, methodological considerations, potential benefits, and future research directions for leveraging multi-layered feature models to achieve flexible, scalable, and performant IoT deployments at the edge.

Keywords: IoT, Edge Computing, Multi-Layered Feature Models, Software Product Lines, Application Deployment, Variability Management, Resource Optimization, Latency, Energy Efficiency.

INTRODUCTION

The Internet of Things (IoT) has rapidly transformed various sectors, from smart homes and cities to industrial automation and healthcare, by connecting billions of devices and generating unprecedented volumes of data [13, 43]. The traditional centralized cloud computing model, while powerful, often struggles to meet the stringent requirements of many IoT applications, particularly those demanding real-time processing, low latency, and high bandwidth efficiency [6, 7, 45]. This has led to the emergence and rapid adoption of edge computing, which brings computation and data storage closer to the data sources, mitigating network congestion and improving response times [17, 46]. Edge computing environments are characterized by their heterogeneity, resource constraints, and dynamic nature, posing significant challenges for the effective deployment and management of diverse IoT applications [19, 23, 35].

Deploying IoT applications on edge infrastructures is a complex task involving decisions about where to place application components (e.g., sensors, actuators, processing modules, data storage), how to allocate limited resources, and how to adapt to changing network conditions or device availability [24, 25, 28, 50]. These challenges are further compounded by the inherent variability of IoT applications themselves, which can range from simple data collection to sophisticated real-time analytics, each with unique functional and non-functional requirements.

Software Product Line Engineering (SPLE) offers a principled approach to managing variability and achieving systematic reuse in software development [12]. At the heart of SPLE are feature models, hierarchical structures that represent the common and variable features of a software system or product line [20, 31]. While feature models have been widely used for modeling software systems, their application to the complex domain of IoT application deployment on edge infrastructures, particularly in a multi-layered context, remains an area with significant potential. Multi-layered feature models, as explored by researchers like Reiser and Weber [15], Rabiser et al. [14], and Lettner et al. [5], allow for the representation of variability at different levels of abstraction, from high-level application features down to low-level infrastructure capabilities. This paper proposes that by integrating these multi-layered models, we can create a powerful framework for supporting the automated and optimized deployment of IoT applications on diverse edge infrastructures.

The objective of this article is to present a comprehensive overview of how multi-layered feature models can be leveraged to address the complexities of IoT application deployment in edge environments. We aim to articulate the benefits of this approach, outline a conceptual methodology, discuss the types of problems it can solve, and identify key research challenges and opportunities.

2. Background and Related Work

To fully appreciate the novelty and potential of multi-layered feature models in IoT edge deployment, it is crucial to understand the existing landscape of edge computing, IoT application deployment strategies, and variability management techniques. This section provides a comprehensive overview of relevant background concepts and surveys significant related work.

2.1. The Edge Computing Paradigm

Edge computing has emerged as a critical architectural paradigm to address the limitations of centralized cloud computing for latency-sensitive and bandwidth-intensive IoT applications.

2.1.1. Evolution from Cloud to Edge

Traditional cloud computing offers centralized processing and storage capabilities, ideal for batch processing and large-scale data analytics. However, for real-time IoT applications like autonomous vehicles, industrial control systems, or augmented reality, transmitting all raw data to the cloud for processing introduces unacceptable latency and places a heavy burden on network infrastructure. This led to the concept of fog computing, which extends the cloud to the edge of the network, bringing computation closer to the data sources. Edge computing is often used interchangeably with fog computing, or as a more general term for any computing performed at the "edge" of the network, away from the centralized cloud [6, 17, 45, 46].

2.1.2. Characteristics of Edge Environments

Edge environments present unique characteristics that differentiate them from traditional cloud data centers:

- Heterogeneity: Edge infrastructures comprise a diverse range of devices, from resource-constrained IoT devices (sensors, actuators) to more powerful edge servers (gateways, micro-data centers). These devices vary significantly in terms of CPU, memory, storage, and networking capabilities [19, 23, 35].
- Resource Constraints: Unlike the virtually limitless resources of the cloud, edge devices typically have limited computational power, memory, storage, and energy. This necessitates careful resource management and optimization.
- Dynamic Nature: Edge environments are highly dynamic. Devices can join or leave the network, network connectivity can fluctuate, and resource availability can change rapidly due to varying workloads or device failures [23].
- Geographical Distribution: Edge nodes are geographically dispersed, often across vast areas, leading to varying network conditions and localized data processing needs.
- Security and Privacy: Processing sensitive data closer to its source at the edge raises specific security and privacy concerns, requiring robust security mechanisms and data governance policies [19].
- Intermittent Connectivity: Some edge devices may experience intermittent network connectivity, requiring applications to function robustly even when disconnected from the cloud or other edge nodes.

2.1.3. Edge Computing Architectures

Various architectural models exist for edge deployments, including:

- Device Edge: Computation directly on the IoT device itself (e.g., smart sensors with embedded processing).
- Gateway Edge: A local gateway aggregates data from multiple IoT devices and performs preliminary processing before forwarding to the cloud or another edge layer.
- Micro Data Centers/Edge Servers: More powerful servers located close to the data sources, capable of hosting more complex applications and services.
- Cloudlets: Small-scale data centers offering cloud-like services in close proximity to mobile users [45].
- Hierarchy of Edge Nodes: Complex deployments often involve multiple layers of edge nodes, forming a hierarchy from devices to local gateways to regional edge data centers, eventually connecting to the centralized cloud.

2.2. IoT Application Deployment Challenges

Deploying IoT applications in edge environments is inherently challenging due to the characteristics outlined above.

2.2.1. Resource Management and Allocation

Optimal allocation of limited CPU, memory, and network resources across diverse edge devices is critical. Poor resource management can lead to performance degradation, increased latency, and energy inefficiency [24, 25].

2.2.2. Latency and Bandwidth Optimization

Minimizing latency is paramount for real-time IoT applications. This involves strategic placement of application components, efficient data routing, and intelligent task offloading decisions [8, 29, 30, 34, 49, 50]. Similarly, reducing bandwidth consumption by processing data at the edge before sending aggregated results to the cloud is a key driver for edge adoption.

2.2.3. Energy Efficiency

Many IoT devices are battery-powered, making energy consumption a critical non-functional requirement. Deployment strategies must consider the energy footprint of computational tasks and data transmission to maximize device longevity [18, 24, 26, 51].

2.2.4. Heterogeneity and Interoperability

Ensuring seamless operation across a wide array of heterogeneous devices, operating systems, and communication protocols is a major hurdle. Applications need to be adaptable to different underlying hardware and software stacks.

2.2.5. Security, Privacy, and Trust

Distributing computation and data across many edge nodes introduces new attack vectors and complicates security management. Ensuring data privacy, integrity, and authenticity at the edge is a complex task [19].

2.2.6. Scalability and Elasticity

IoT deployments can involve millions of devices. The deployment strategy must be scalable to accommodate growth and elastic enough to adapt to fluctuating workloads and device availability.

2.3. Variability Management in Software Engineering

Variability is a fundamental aspect of software systems, especially when developing families of related products.

2.3.1. Software Product Line Engineering (SPLE)

Software Product Line Engineering (SPLE) is a systematic approach to developing software systems by leveraging commonalities and managing variabilities across a set of related products [12]. The core idea is to develop a common core asset base (e.g., reusable components, architectures, processes) that can be configured to produce various products in a product line. SPLE aims to increase productivity, reduce time-to-market, and improve software quality.

2.3.2. Feature Models

Feature models are a cornerstone of SPLE. They are hierarchical, tree-like structures used to represent the common and variable features of a software system or product line [20, 31].

- Features: Represent distinct characteristics or functionalities of a system.
- Relationships:
  - Mandatory: A feature that must be included if its parent is included.
  - Optional: A feature that may or may not be included if its parent is included.
  - Alternative (XOR): Exactly one feature from a group must be selected.
  - Or: One or more features from a group must be selected.
- Cross-Tree Constraints: Additional rules (e.g., "A requires B," "C excludes D") that define dependencies or incompatibilities between features that are not directly related in the hierarchy.

Feature models provide a formal and structured way to:

- Define the scope of a product line.
- Guide the configuration process.
- Enable automated reasoning about valid product configurations.

2.3.3. Multi-Level and Multi-Dimensional Variability

Traditional feature models often focus on a single level of abstraction. However, complex systems like IoT deployments necessitate modeling variability at multiple levels.

- Multi-Level Feature Trees: Reiser and Weber [15] introduced the concept of multi-level feature trees, where features at one level can refine or constrain features at a lower level. This allows for a hierarchical decomposition of variability.
- Multi-Dimensional Variability Modeling: Rosenmüller et al. [16] discuss multi-dimensional variability, where variability is modeled across different dimensions (e.g., functional, non-functional, deployment). This provides a more holistic view of system variability.
- Multi-Purpose, Multi-Level Feature Modeling: Rabiser et al. [14] extended these concepts for large-scale industrial software systems, emphasizing the reusability of feature models themselves for different purposes (e.g., design, configuration, testing) and at different levels of abstraction.
- Two-Layered Feature Models with Attributes: Lettner et al. [5] explored automated analysis of two-layered feature models with feature attributes, which is directly relevant to our proposed approach. Attributes allow for the quantification of non-functional properties associated with features.

2.4. Existing Approaches to IoT Deployment and Optimization

While not explicitly using multi-layered feature models, various research efforts have addressed aspects of IoT application deployment and optimization.

2.4.1. Task Offloading and Resource Management

Many studies focus on intelligent task offloading decisions in mobile edge computing (MEC) to optimize for energy efficiency, latency, or a combination thereof [8, 29, 30, 34, 49, 51, 52]. These often employ mathematical programming or heuristic algorithms to determine which parts of an application should run on the device, at the edge, or in the cloud.

2.4.2. Containerization and Orchestration

The use of containerization technologies (e.g., Docker) and orchestration platforms (e.g., Kubernetes) is widespread in modern edge deployments [11, 27, 42]. These tools facilitate packaging applications and managing their deployment and scaling. However, they typically operate at a lower level of abstraction and require manual configuration of deployment manifests.

2.4.3. Model-Driven Engineering (MDE) for IoT

MDE approaches have been used to generate deployment artifacts or configurations for IoT systems. While some MDE tools incorporate notions of variability, they often lack the formal reasoning capabilities and multi-layered structure provided by feature models for comprehensive variability management.

2.4.4. Software Product Lines in IoT (General)

Some research has applied SPLE principles to general IoT software development [38], but few have focused specifically on the intricate deployment challenges in heterogeneous edge environments using multi-layered models for optimization. Abbas et al. [1] explored multi-objective optimum solutions for IoT-based feature models of software product lines, which is a significant related work that highlights the potential for optimization within an SPL context.

In summary, while edge computing, IoT deployment, and variability management are well-established fields, the explicit integration of multi-layered feature models as a holistic framework for automated, optimized, and flexible IoT application deployment on heterogeneous edge infrastructures represents a significant gap that this article aims to address. The existing work provides the necessary foundation and highlights the pressing need for such a comprehensive approach.

3. Materials and Methods: A Multi-Layered Feature Model Methodology

To systematically support IoT application deployment on edge infrastructures using multi-layered feature models, a structured methodology is required. This section outlines the conceptual framework, the types of models involved, the relationships and constraints between them, and the analytical and optimization techniques that can be employed.

3.1. Conceptual Framework: Aligning Variability Dimensions

The core idea is to align the variability inherent in IoT applications with the variability of edge infrastructure capabilities and deployment configurations. This alignment is achieved through a multi-layered feature modeling approach, where each layer captures a specific dimension of variability and its intricate interdependencies with other layers. This framework moves beyond a simplistic application-to-infrastructure mapping by introducing a dedicated deployment layer, enabling more nuanced and optimized decisions.

The proposed conceptual framework comprises three primary, interconnected feature model layers:

3.1.1. Application Layer Feature Model (ALFM)

This model describes the functional and non-functional features of the IoT application itself. It encapsulates the application's inherent variability, allowing for different versions or configurations of a single IoT application. The ALFM represents the "what" the application does and its requirements.

Key features and attributes at this layer include:

- Core Functionality:
  - Data Collection: Specifies types of sensors (e.g., temperature, humidity, acceleration, video), data formats (e.g., JSON, Protocol Buffers), and data sources (e.g., internal device sensors, external peripherals).
  - Data Processing: Defines the complexity of processing, such as Simple Aggregation, Filtering, Anomaly Detection, Machine Learning Inference (e.g., image recognition, predictive maintenance), Stream Analytics.
  - Actuation: Specifies types of actuators (e.g., motors, lights, valves) and control logic (e.g., On/Off Control, Proportional Control).
  - Communication Protocols: MQTT, CoAP, HTTP/S, AMQP, WebSockets.
- Non-Functional Requirements (NFRs): These are often quantified as attributes associated with features.
  - Latency Tolerance: Real-time (e.g., <100ms), Near Real-time (e.g., <1s), Batch Processing (e.g., hours).
  - Throughput Requirements: Data rate (e.g., Mbps), number of transactions per second.
  - Storage Requirements: Local Persistent Storage, Ephemeral Storage, Cloud Archival.
  - Security Features: Data Encryption (e.g., TLS, end-to-end), Authentication (e.g., OAuth, X.509 certificates), Access Control (e.g., RBAC), Secure Boot.
  - Energy Consumption Targets: Low Power, Standard Power.
  - Reliability/Availability: High Availability (e.g., redundant components), Fault Tolerance.
  - Computational Complexity: Quantifies CPU cycles,

memory usage per operation.

○ Data Sensitivity: Personal Data, Critical Infrastructure Data.

Example ALFM Structure (Partial):

```
IoT Application
├── Data Collection (mandatory)
│  ├── Sensor Type (Or)
│  │  ├── Temperature Sensor
│  │  ├── Humidity Sensor
│  │  └── Camera
│  └── Sampling Rate (attribute: Hz)
├── Data Processing (optional)
│  ├── Aggregation
│  ├── Filtering
│  ├── ML Inference (optional)
│  │  └── Model Size (attribute: MB)
│  │  └── Inference Frequency (attribute: per_sec)
├── Communication (mandatory)
│  ├── Protocol (XOR)
│  │  ├── MQTT
│  │  └── CoAP
│  └── Encryption (optional)
│     └── TLS
└── Quality of Service (QoS)
   ├── Latency (attribute: ms_max)
   └── Energy Efficiency (attribute: J_per_op)
```

Variability in this layer reflects different versions or configurations of an IoT application, tailored to specific use cases or user preferences.

3.1.2. Infrastructure Layer Feature Model (ILFM)

This model captures the variability of the available edge infrastructure. It describes the capabilities and characteristics of individual edge devices, network segments, and available software components within the edge environment. The ILFM represents the "where" the application can run and the resources available.

Key features and attributes at this layer include:

● Edge Device Features:

○ Processor: CPU Architecture (e.g., ARM, x86), Clock Speed (e.g., GHz), Number of Cores, GPU (presence/type), NPU (presence/type).

○ Memory: RAM Capacity (e.g., MB, GB), Memory Speed.

○ Storage: Storage Type (e.g., eMMC, SSD, HDD), Storage Capacity (e.g., GB, TB).

○ Battery Capacity (e.g., mAh), Power Consumption Profile.

○ Network Interfaces: Wi-Fi (standards: 802.11n, ac, ax), Cellular (e.g., 4G, 5G), Ethernet, Bluetooth, LoRa.

○ Sensors/Actuators (Native): Sensors/actuators directly integrated into the edge device (e.g., a smart camera with an onboard microphone).

○ Operating System: Linux (e.g., Ubuntu Core, Raspbian), FreeRTOS, Android Things.

○ Virtualization Support: Container Runtime (e.g., Docker, containerd), Hypervisor (e.g., KVM, Xen), Unikernel support [11, 22].

● Network Features:

○ Bandwidth: Upload/Download Speed (e.g., Mbps).

○ Latency: Round-trip Time (e.g., ms).

○ Reliability: Packet Loss Rate.

○ Connectivity Type: Wireless, Wired.

● Software Stack Features:

○ Middleware: Apache Kafka, RabbitMQ.

○ AI/ML Frameworks: TensorFlow Lite, OpenVINO, PyTorch Mobile.

○ Security Features: Hardware Security Module (HSM), Trusted Platform Module (TPM).

Example ILFM Structure (Partial):

```
Edge Infrastructure
├── Edge Node (Or: multiple nodes can exist)
│  ├── Device Type (XOR)
│  │  ├── IoT Sensor Node
│  │  ├── Edge Gateway
│  │  └── Edge Server
│  ├── CPU (mandatory)
│  │  ├── Architecture (attribute: ARM/x86)
│  │  └── Cores (attribute: int)
│  ├── Memory (attribute: MB)
│  ├── Network Interface (Or)
│  │  ├── Wi-Fi (attribute: standard)
│  │  ├── 5G
│  │  └── Ethernet
│  ├── OS (XOR)
│ │ ├── Linux
│ │ └── FreeRTOS
│ └── ML Inference Engine (optional)
│     └── TensorFlow Lite
└── Network Connectivity
    ├── Bandwidth (attribute: Mbps)
    └── Latency (attribute: ms)
```

This layer allows for modeling the diverse capabilities of different edge nodes and the network segments connecting them, forming a comprehensive resource catalog.

3.1.3. Deployment Layer Feature Model (DLFM)

This layer acts as the bridge between the application and infrastructure models, representing the decisions related to how application features are mapped to infrastructure features. It captures the "how" the application is deployed. This layer contains the configuration choices and strategies.

Key features and attributes at this layer include:

● Task Offloading Decisions:

○ Local Processing: Entire application or specific modules run on the end device.

○ Edge Offloading: Computation offloaded to an edge gateway or server [8, 29, 30, 34, 41, 51, 52].

○ Cloud Offloading: Computation offloaded to the central cloud.

○ Hybrid Offloading: Distributed execution across multiple layers.

● Resource Allocation Strategies:

○ CPU Cores Allocation: Number of cores assigned to an application module.

○ Memory Limits: Maximum RAM assigned.

○ Storage Allocation: Persistent storage volumes.

○ Network Bandwidth Allocation.

● Data Routing Paths:

○ Direct to Cloud.

○ Edge Pre-processing, then Cloud.

○ Edge-to-Edge Communication.

● Deployment Configuration Parameters:

○ Container Orchestration Settings: Replicas, Resource Requests/Limits, Node Affinity, Pod Anti-Affinity (for Kubernetes [27]).

○ Virtual Machine/Unikernel Deployment Settings: VM Size, Image Selection.

○ Security Policies: Firewall Rules, Network Segmentation.

○ Fault Tolerance Mechanisms: Redundancy, Failover.

● Monitoring and Logging Configuration: Local Logging, Centralized Logging.

Example DLFM Structure (Partial):

```
Deployment Configuration
├── Task Placement (XOR)
│ ├── Local Device Execution
│ ├── Edge Offloading (requires: Edge Node in ILFM)
│ │ └── Offloading Target (reference to specific Edge Node)
│ └── Cloud Offloading (requires: Cloud Connectivity in ILFM)
├── Resource Limits (Optional)
│ ├── CPU Limit (attribute: mCPU)
│ └── Memory Limit (attribute: MB)
├── Data Path (XOR)
│ ├── Direct Cloud Uplink
│ └── Edge Pre-processing Path
│     └── Edge Node for Pre-processing (reference to specific Edge Node)
└── Orchestration (optional)
    ├── Kubernetes Deployment
    │ ├── Number of Replicas (attribute: int)
    │ └── Node Selector (reference to ILFM Node Type)
    └── Standalone Deployment
```

The concept of multi-level feature trees [15] and multi-dimensional variability modeling [16] are foundational to this approach, allowing for the explicit representation of relationships and constraints between features across these distinct layers. Rabiser et al. [14] further discuss multi-purpose, multi-level feature modeling for large-scale industrial systems, which is highly relevant to complex IoT deployments. This layered approach provides the necessary granularity and abstraction to manage the complexity of IoT deployments at the edge.

3.2. Model Relationships and Cross-Layer Constraints

A critical aspect of multi-layered feature models is defining the relationships and constraints between features in different layers. These constraints are the essence of how the models interact and enable intelligent deployment decisions. They ensure that selected application features are compatible with available infrastructure and that deployment choices respect both application requirements and infrastructure limitations.

3.2.1. Types of Cross-Layer Constraints

Constraints can be categorized based on their nature and the features they relate:

● Requires/Excludes Constraints: These are fundamental boolean logic constraints:

○ Application Requires Infrastructure: An application feature might necessitate the presence of a specific infrastructure capability. For example, the ML Inference feature in the ALFM might require GPU-enabled edge device in the ILFM. Similarly, a Real-time Analytics feature could require an Edge Server (device type) to meet its latency demands.

○ Infrastructure Excludes Application: An infrastructure limitation might preclude certain application features. For instance, an IoT Sensor Node (ILFM) with Low Memory might exclude an ML Inference application feature (ALFM) if the model size is too large.

○ Deployment Requires/Excludes Application/Infrastructure: A deployment choice might require or exclude certain features. For example, Edge Offloading (DLFM) naturally requires the presence of an Edge Node (ILFM) and specific Data Processing features (ALFM) to offload.

○ Attribute-based Requires/Excludes: Constraints can depend on attributes. Application.Latency < 100ms might require Network.Latency < 50ms.

● Attribute-based Constraints (Quantitative Constraints): Features often have associated attributes with quantifiable values (e.g., CPU usage, memory footprint, power consumption, latency impact). Constraints can be defined over these attributes, often involving inequalities or arithmetic expressions [5]. These are crucial for optimization.

○ Resource Matching: "The total required CPU for application components (sum of ALFM.Processing.CPU_req) must not exceed the available CPU on the selected edge device (ILFM.EdgeNode.CPU.Cores * ILFM.EdgeNode.CPU.ClockSpeed)."

○ Performance Matching: "The cumulative latency introduced by ALFM.Communication.Protocol.Latency_overhead plus ILFM.Network.Latency plus ALFM.Processing.Execution_latency must be less than ALFM.QoS.Latency.ms_max."

○ Energy Budget: "The total estimated energy consumption for a given deployment (DLFM.Energy_cost) must be less than the ILFM.Battery.Capacity or a predefined Deployment.Energy_Budget."

○ Data Throughput: "The data sampling rate of the ALFM.Data Collection.Sampling Rate must be supported by the ILFM.Network.Bandwidth and ILFM.Edge Node.Processing Capacity."

● Cross-Layer Propagation and Derivation: Changes or selections in one layer can propagate constraints or derive new requirements for other layers.

○ Application-driven Propagation: Selecting a Low-power mode feature in the ALFM might constrain the choice of edge devices to those with specific Energy-efficient Processors in the ILFM. It might also influence the Task Offloading Decision in the DLFM to prioritize local processing for minimal communication energy.

○ Infrastructure-driven Constraints: If an Edge Gateway (ILFM) is selected, its limited Memory Capacity might constrain the number of ML Inference Models (ALFM) that can be simultaneously deployed on it.

○ Deployment-driven Refinement: Choosing Container Orchestration (DLFM) implies that selected edge nodes must have a Container Runtime (ILFM). It also opens up options for Replicas and Resource Limits features in the DLFM, which then influence resource consumption from the ILFM.

3.2.2. Formalizing Constraints

Constraints are typically formalized using logical expressions. For quantitative constraints, they often involve arithmetic expressions and comparisons.

Example Constraint Formalization:

1. ML Inference Requires GPU:

○ select(ALFM.Data_Processing.ML_Inference) IMPLIES select(ILFM.Edge_Node.GPU_enabled)

2. Total CPU Requirement vs. Available CPU:

○ ALFM.Data_Processing.CPU_req + ALFM.Data_Collection.CPU_req <= ILFM.Edge_Node.CPU.Cores * ILFM.Edge_Node.CPU.Clock_Speed (where _req are attributes representing CPU demands, and Cores, Clock_Speed are attributes of the infrastructure feature).

3. Latency Constraint:

○ DLFM.Offloading_Decision.Latency_Impact + ALFM.Communication.Protocol.Latency_Impact + ILFM.Network.Latency_ms <= ALFM.QoS.Latency.ms_max

These relationships enable the system to identify valid and optimal configurations for deploying IoT applications, ensuring coherence and feasibility across the application, infrastructure, and deployment choices [2, 36]. The precise definition and management of these constraints are crucial for the effectiveness of the proposed methodology. This formalization is what allows for automated reasoning and optimization.

3.3. Automated Analysis and Optimization Techniques

Once the multi-layered feature models and their intricate cross-layer constraints are defined, automated reasoning techniques become indispensable tools for supporting deployment decisions. These techniques enable verification of configuration validity, generation of feasible deployments, and optimization based on various non-functional objectives.

3.3.1. Satisfiability Modulo Theories (SMT) Solvers

SMT solvers are powerful engines for checking the satisfiability of logical formulas over various background theories (e.g., integers, arrays, bit-vectors, real numbers) [21, 32]. They extend the capabilities of boolean satisfiability (SAT) solvers to handle more complex logical expressions involving mathematical constraints.

How SMT Solvers are Applied:

● Configuration Validation: Given a proposed deployment configuration (a specific selection of features from all three layers), an SMT solver can determine if this configuration is valid, meaning it satisfies all defined intra-layer and cross-layer constraints. This is particularly valuable for quickly identifying incompatible deployments before actual resource allocation.

○ Input: A boolean formula representing the selected features and their combined constraints.

○ Output: SAT (satisfiable, configuration is valid) or UNSAT (unsatisfiable, configuration is invalid).

● Configuration Generation: SMT solvers can also be used to generate all valid deployment configurations that satisfy a given set of desired application features and available infrastructure. This is effectively querying the model space for all feasible solutions. For large models, this might yield a vast number of configurations.

● Constraint Discovery and Explanation: When a configuration is UNSAT, SMT solvers can often provide a "core" of conflicting constraints, helping developers understand why a particular deployment is invalid.

● Optimization (with MaxSMT/Optimization Extensions): Many SMT solvers or extensions (e.g., MaxSMT, Opt-SMT) can find configurations that optimize certain non-functional properties. This is achieved by formulating the optimization goal as a series of satisfiability checks or by directly integrating with optimization algorithms. For instance, one might seek to:

○ Minimize Latency: Find a configuration where the Total_Latency attribute is minimized.

○ Maximize Energy Efficiency: Find a configuration where the Total_Energy_Consumption attribute is minimized, or Device_Lifetime is maximized.

○ Minimize Cost: Find a deployment that uses the least expensive combination of infrastructure resources.

○ Maximize Throughput: Find a configuration that allows for the highest data processing rate.

Popular SMT Solvers: Z3 [21, 32], CVC4, Yices.

3.3.2. Metaheuristic Algorithms

For complex optimization problems with large search spaces, especially those involving multiple, often conflicting, objectives (e.g., minimizing latency and maximizing energy efficiency), metaheuristic algorithms offer a practical approach to finding near-optimal solutions. Unlike SMT solvers that guarantee optimality for solvable problems, metaheuristics provide good solutions within a reasonable time, even for NP-hard problems.

Common Metaheuristic Algorithms:

● Genetic Algorithms (GAs) [39]: Inspired by natural selection, GAs evolve a population of candidate solutions (deployments) over generations. Each solution is evaluated based on a fitness function that quantifies how well it meets the optimization objectives (e.g., a weighted sum of latency and energy efficiency). Solutions are combined (crossover) and mutated to explore the search space.

● Particle Swarm Optimization (PSO): Inspired by bird flocking or fish schooling, PSO iteratively tries to improve a candidate solution with regard to a given measure of quality.

● Ant Colony Optimization (ACO): Mimics the behavior of ants finding paths, where "pheromones" guide the search for optimal solutions.

● Bees Algorithm [10]: Inspired by the foraging behavior of honey bees, it balances exploration and exploitation of the search space.

Application in Multi-Layered Feature Models:

Metaheuristics operate on an encoding of the feature model configuration. The fitness function would be derived from the attribute-based constraints and optimization objectives defined in the models. For example, a "chromosome" in a genetic algorithm could represent a sequence of selected features and their attribute values from all three layers. The fitness function would then calculate the total latency, energy consumption, or a composite score for that configuration.

3.3.3. Domain-Specific Languages (DSLs) and Tool Support

Managing large and complex multi-layered feature models requires robust tool support.

● FAMILIAR [3]: A prominent DSL and tool for managing large-scale feature models. It facilitates their creation, manipulation, and analysis. Tools like FAMILIAR can be instrumental in implementing and maintaining the multi-layered models proposed here, offering capabilities for model composition, decomposition, and analysis of feature model properties.

● FeatureIDE: A widely used open-source tool for feature model creation, configuration, and analysis. It supports various automated analysis techniques.

- Custom Tooling: For highly specific requirements or unique constraint types, custom tooling built on top of SMT solvers or metaheuristic libraries might be necessary.

3.3.4. Integration with Machine Learning (ML)

While not a direct analysis technique for feature models, ML can complement the framework, especially for dynamic environments.

- Predictive Models: ML models can predict future network conditions, device load, or energy availability. These predictions can then be fed as attributes into the feature models, enabling more proactive optimization (e.g., predicting an increase in network latency to trigger a pre-computed re-deployment).

- Reinforcement Learning (RL): RL agents could learn optimal deployment strategies over time by interacting with the edge environment and receiving rewards based on meeting QoS objectives. The feature model could define the "state space" or "action space" for the RL agent, guiding its learning process.

The combination of these techniques allows for a systematic and automated approach to managing the inherent variability in IoT application deployment on edge infrastructures, moving beyond manual, error-prone configuration processes. This comprehensive methodology forms the backbone for achieving flexible, scalable, and performant IoT deployments at the edge.

4. Results: Demonstrated Benefits and Use Cases

The application of multi-layered feature models to IoT application deployment on edge infrastructures yields several significant results, demonstrating profound improvements in efficiency, flexibility, adaptability, and decision-making. These benefits extend across the entire lifecycle of IoT application management, from initial design to dynamic run-time adjustments.

4.1. Enhanced Deployment Automation and Validation

By formalizing application, infrastructure, and deployment variabilities within a unified feature modeling framework, the process of deploying IoT applications can be largely automated. This automation stems from the explicit representation of all relevant parameters and their interdependencies, enabling rigorous, machine-driven checks.

- Error Reduction: SMT solvers, as discussed in the methodology [20, 32], can quickly validate whether a given deployment configuration (a specific combination of features selected from all layers) is feasible. This means checking if it satisfies all defined functional and non-functional constraints. This significantly reduces manual configuration errors, which are common in complex, heterogeneous edge environments. For instance, if an IoT application requires a specific processing capability (e.g., for augmented reality tasks [25]) and an edge device lacks it, or if its latency requirements cannot be met by the network, the model can automatically flag this incompatibility. This prevents failed deployments and costly debugging in real-world scenarios, shifting validation from post-deployment testing to pre-deployment design.

- Guaranteed Feasibility: The system can guarantee that any generated configuration is inherently valid according to the defined constraints. This eliminates guesswork and ensures that deployed applications are compatible with the target edge environment.

- Accelerated Deployment Cycles: Automating the configuration and validation process drastically reduces the time required to deploy new IoT applications or update existing ones. Instead of manually configuring YAML files or deployment scripts, the system can generate validated deployment artifacts directly from the selected feature configurations.

- Deployment Blueprint Generation: The validated configurations can serve as a blueprint for generating actual deployment scripts or manifests for platforms like Kubernetes [27, 42], Docker Swarm, or even custom provisioning tools. This bridges the gap between high-level architectural decisions and low-level operational details.

4.2. Optimized Resource Allocation and Performance

Multi-layered feature models enable the formulation of deployment as an optimization problem, leading to highly efficient resource utilization and improved application performance across various metrics.

- Quantitative Optimization: By associating features with quantifiable attributes (e.g., CPU usage, memory footprint, power consumption, latency impact, bandwidth requirements), optimization algorithms (SMT solvers with optimization extensions, metaheuristics like genetic algorithms [39]) can identify configurations that optimize specific objectives.

    - Latency Minimization: Crucial for real-time applications, the models can identify offloading strategies and resource allocations that minimize end-to-end latency [8, 29, 34, 49, 50]. This might involve placing computation closer to the data source (edge offloading) or distributing tasks across multiple edge nodes to parallelize processing.

    - Energy Efficiency Maximization: Especially vital for battery-powered IoT devices, the models can prioritize configurations that minimize energy usage [18, 24, 26, 51]. This can involve intelligent task offloading decisions (e.g., offloading to a more powerful, less battery-constrained edge server), selecting less power-hungry edge nodes, or adjusting application parameters (e.g., data sampling frequency, inference frequency) to reduce computational load. Studies have shown that task offloading decisions in mobile edge computing can be optimized for energy efficiency or reduced completion time [8, 29, 34, 51, 52]. With feature models, these decisions can be systematically derived. Cañete et al. [24] demonstrated how a software product line approach, underpinned by variability modeling, can achieve energy-efficient deployment of IoT applications in edge infrastructures.

○ Resource Utilization: Optimizing for efficient resource utilization (e.g., CPU, memory) prevents under-provisioning (leading to performance bottlenecks) and over-provisioning (leading to wasted resources and increased costs). The framework can balance the workload across available edge nodes, preventing single points of congestion.

○ Cost Optimization: In scenarios where edge infrastructure costs vary (e.g., public edge clouds), the models can optimize deployments to minimize operational expenditure while meeting performance targets.

● Multi-Objective Optimization: Real-world deployments often involve conflicting objectives (e.g., reducing latency might increase energy consumption). The framework, particularly with metaheuristic algorithms, supports multi-objective optimization, allowing the system to find Pareto-optimal solutions that represent the best trade-offs between different objectives [1]. This provides decision-makers with a set of balanced options rather than a single "optimal" solution.

4.3. Improved Variability Management and Adaptability

The multi-layered approach provides a structured, hierarchical, and comprehensive way to manage the vast and intricate variability inherent in both IoT applications and the underlying edge infrastructures.

● Systematic Reuse: Instead of managing countless individual application versions or infrastructure types, developers can define a core set of features and their variations, along with the rules for their combination. This enables systematic reuse of software components, architectural patterns, and deployment knowledge across an entire IoT product line [12, 38]. This leads to faster development cycles, reduced maintenance overhead, and higher quality deployments by ensuring consistency.

● Simplified Evolution: When new edge devices become available, existing infrastructure is upgraded, or application requirements change, the feature models can be updated incrementally. The system can then automatically re-evaluate optimal deployment strategies based on the modified models. This adaptability is crucial for the dynamic nature of edge computing environments, where devices may join or leave the network, or resource availability may fluctuate [23].

● Customization and Personalization: The explicit modeling of variability allows for easy customization of IoT applications for different user needs, geographical regions, or specific industrial requirements. For example, a smart city surveillance application could be easily configured for different camera types, processing capabilities (e.g., simple motion detection vs. advanced facial recognition), and local regulations.

● Clearer Communication: Feature models serve as a common language for stakeholders (developers, architects, operations teams, business analysts) to understand the capabilities and constraints of the IoT system. They provide a high-level overview of the product line's scope and the allowable variations. The ability to model multi-dimensional variability [16] and multi-purpose, multi-level feature models [14] makes this approach particularly robust for complex industrial systems.

4.4. Enhanced Energy Efficiency

As previously highlighted, energy consumption is a paramount concern in edge computing, especially for battery-powered IoT devices and environmentally conscious deployments.

● Direct Optimization for Energy: By explicitly including energy-related attributes in both the application layer (e.g., Power_draw_per_operation for a processing module) and infrastructure layer (e.g., Battery_capacity, Power_consumption_profile for an edge device), the optimization algorithms can directly prioritize configurations that minimize energy usage.

● Intelligent Task Offloading: The models can guide decisions on where to execute tasks (on device, edge, or cloud) to minimize overall energy consumption. For instance, offloading compute-intensive tasks to a mains-powered edge server might be more energy-efficient than running them on a small battery-powered IoT sensor, even with communication overhead [51]. Conversely, for very small tasks, local execution might be more efficient.

● Dynamic Power Management: The framework can suggest adjustments to application parameters (e.g., data sampling frequency, sensor duty cycle, ML inference frequency) that directly impact energy consumption, based on the current energy budget and performance requirements. The work by Cañete et al. [26] on energy-efficient adaptation engines for Android applications, while not directly on feature models, highlights the importance of such considerations, which can be integrated into the attribute-based reasoning of feature models.

4.5. Support for Software Product Lines in IoT

The methodology naturally aligns with and strongly reinforces Software Product Line (SPL) principles [12], transforming how IoT solutions are developed and managed.

● IoT as a Product Line: An IoT solution ecosystem can be rigorously viewed as a product line, where different "products" correspond to various configurations deployed on diverse edge infrastructures. This paradigm shift enables systematic development and evolution.

● Systematic Reuse of Assets: Beyond just code components, the approach enables the systematic reuse of:

○ Architectural patterns: Common deployment topologies for edge computing.

○ Configuration knowledge: Best practices for configuring specific application features on particular infrastructure types.

○ Testing assets: Test cases can be derived from feature models, ensuring adequate coverage for all valid configurations.

● Faster Time-to-Market: By automating configuration and leveraging reusable assets, new IoT applications or variations can be deployed much faster, accelerating time-to-market for businesses.

● Increased Quality: Formalizing variability and enabling automated validation leads to higher quality deployments with fewer errors and better performance. Multi-objective optimization techniques, as applied to IoT-based feature models of software product lines, can further enhance the selection of optimal solutions considering various trade-offs [1]. This ensures that not only are deployments valid, but they also meet critical quality attributes.

In conclusion, the results demonstrate that multi-layered feature models provide a powerful, systematic, and automated framework for navigating the complexities of IoT application deployment in heterogeneous edge environments. They transform a traditionally manual and error-prone process into an efficient, optimized, and adaptable engineering discipline.

5. Discussion: Challenges, Opportunities, and Future Directions

The results highlight the transformative potential of applying multi-layered feature models to IoT application deployment on edge infrastructures. This approach moves beyond traditional manual configuration and provides a systematic, automated, and optimizable framework for managing complexity and variability. However, like any advanced methodology, it comes with its own set of challenges and opens up numerous avenues for future research and development.

5.1. Model Complexity and Scalability

While powerful, multi-layered feature models can become inherently complex, especially for large-scale IoT ecosystems with numerous application variants, diverse edge devices, and intricate cross-layer dependencies.

● Exponential Growth: The number of features, attributes, and cross-layer constraints can grow exponentially with the increasing scale of IoT deployments. This proliferation directly impacts the performance of automated reasoning tools, potentially leading to long computation times for validation or optimization tasks [3, 48].

● Maintenance and Evolution: Managing the evolution of these complex models over time, as new technologies emerge, existing devices are updated, or application requirements change, presents a significant maintenance challenge [33, 37]. Ensuring consistency and correctness across model versions is critical.

● Modularity and Composition: To mitigate complexity, techniques for modularizing feature models [2] and using hierarchical or compositional approaches are essential. For example, breaking down a large infrastructure model into sub-models for different geographical regions or device types could improve manageability. Tools like FAMILIAR [3] provide DSLs for managing large-scale feature models, facilitating their creation, manipulation, and analysis. However, advanced methods for composing and decomposing multi-layered models are still an active area of research.

● Scalability of Solvers: Further research is needed on efficient algorithms and solver technologies capable of analyzing extremely large and dynamic multi-layered feature models within acceptable timeframes. This might involve distributed solving or approximation techniques for highly complex optimization problems.

● Knowledge Representation: Developing more expressive and flexible knowledge representation formalisms beyond traditional boolean feature models, possibly incorporating probabilistic or fuzzy logic, could handle uncertainties and incompleteness often present in real-world edge environments.

5.2. Run-time Adaptability and Dynamic Reconfiguration

The current framework primarily focuses on design-time optimization or static configuration. However, edge environments are inherently dynamic. Network conditions fluctuate, devices may fail or become overloaded, and application requirements can change in real-time (e.g., an autonomous vehicle needing higher processing power in complex traffic).

● Real-time Re-evaluation Overhead: While the models can be re-evaluated, the computational overhead of re-solving complex optimization problems (e.g., using SMT solvers or metaheuristics) at run-time might be prohibitive, especially for latency-critical applications.

● Lightweight Adaptation Mechanisms: Future work should explore lightweight mechanisms for dynamic reconfiguration based on the insights derived from the feature models. This could involve:

○ Pre-computed Configuration Set: Pre-computing a set of optimal or near-optimal configurations for anticipated run-time scenarios (e.g., "low bandwidth mode," "high load mode"). The system could then quickly switch between these pre-validated configurations.

○ Incremental Reasoning: Developing techniques that allow solvers to efficiently update solutions when only small changes occur in the environment or requirements, rather than re-solving from scratch.

○ Feedback Loops: Integrating monitoring data from the edge environment (e.g., CPU utilization, network latency, energy levels) into a feedback loop that triggers adaptation. This could involve adaptive agents that use simplified decision models derived from the comprehensive feature models.

○ Energy-efficient Adaptation Engines: Approaches for energy-efficient adaptation engines [26] suggest avenues for real-time adjustments, possibly guided by feature model-derived policies.

● Uncertainty and Probabilistic Reasoning: Edge environments are often characterized by uncertainty (e.g., unreliable network links, unpredictable device availability). Incorporating probabilistic reasoning or uncertainty quantification into feature models could allow for more robust deployment decisions under dynamic conditions.

5.3. Integration with Existing Edge Orchestration Platforms

Modern edge deployments heavily leverage containerization and orchestration platforms like Kubernetes [27, 42]. The proposed multi-layered feature modeling approach needs to seamlessly integrate with these existing technologies to ensure practical applicability.

● From Model to Manifests: A key challenge is translating the optimized deployment configurations derived from the feature models into executable deployment manifests (e.g., Kubernetes YAML files, Docker Compose files, Ansible playbooks). This requires robust code generation or configuration generation capabilities.

● API and Tooling Interoperability: Developing APIs and connectors that allow feature modeling tools to interact with and influence existing orchestration platforms is crucial. This could involve extending existing schedulers (e.g., Kubernetes schedulers) or developing custom operators that consume feature model outputs. Research on low-carbon Kubernetes schedulers [42] and container orchestration surveys [27] indicate the practical relevance of such integration.

● Runtime Reconfiguration with Orchestrators: How does a run-time adaptation decision based on feature models translate into a re-orchestration event (e.g., scaling a deployment, moving a pod) on platforms like Kubernetes? This requires mapping abstract feature model changes to concrete orchestration commands.

● Standardization: As the field matures, there might be a need for standardization of feature model representation and their interfaces with orchestration platforms to foster broader adoption and interoperability.

5.4. Empirical Validation and Case Studies

While the theoretical benefits are clear, extensive empirical validation is crucial to demonstrate the practical applicability, performance gains, and limitations of the proposed methodology in real-world scenarios.

● Benchmarking and Performance Metrics: This involves applying the proposed methodology to real-world IoT applications and heterogeneous edge infrastructures. Key performance indicators (KPIs) such as latency reduction, energy savings, resource utilization efficiency, deployment success rate, and configuration time should be measured and compared against baseline approaches (e.g., manual configuration, heuristic-based deployments).

● Diverse Use Cases: Conducting case studies across various IoT domains (e.g., smart agriculture, industrial IoT, smart cities, healthcare, autonomous systems) would provide valuable insights into the generalizability and specific challenges faced in different contexts. Case studies, such as those involving augmented reality tasks [25] or SLAM for mobile robots [44], would provide valuable insights into the practical applicability and limitations of the framework.

● Scalability Testing: Rigorous testing on large-scale synthetic or real-world datasets of feature models and infrastructure instances to understand the limits of the automated reasoning tools and the methodology itself.

● User Experience and Tooling: Beyond technical performance, evaluating the usability of the feature modeling tools for architects and developers is important for adoption. How intuitive is it to define complex multi-layered models and constraints?

5.5. Security and Dependability in Multi-Layered Models

As IoT applications become more critical, security and dependability are paramount [19]. Feature models offer an opportunity to embed these non-functional requirements into the deployment process from the outset.

● Security Feature Modeling: Feature models can explicitly incorporate security-related features and constraints (e.g., Data Encryption requirements, Access Control policies, Secure Boot, Intrusion Detection). The model can ensure that sensitive data is only processed on secure edge nodes or that communication paths meet specific encryption standards.

● Threat Modeling Integration: Future research could explore integrating threat modeling techniques with multi-layered feature models to automatically identify potential vulnerabilities arising from specific feature combinations or deployment choices.

● Dependability and Resilience: The dependability of edge computing itself is an evolving area [19]. Feature models could contribute to designing more resilient deployments by explicitly modeling redundancy, failover mechanisms, and fault tolerance strategies. For example, a feature High Availability in the ALFM could require Redundant Edge Nodes in the ILFM and Automatic Failover Configuration in the DLFM.

● Privacy-by-Design: With increasing data privacy regulations (e.g., GDPR), the ability to model privacy-related features (e.g., Local Data Anonymization, Data Retention Policies) and ensure their enforcement through deployment choices is crucial.

The ongoing evolution of edge computing, including advancements in mobile edge computing [7], ubiquitous computing paradigms [35], and its integration with 5G and beyond-5G networks [51], further underscores the need for sophisticated, automated deployment strategies. Multi-layered feature models offer a promising avenue to navigate this complexity, providing a structured and optimizable approach to realizing the full potential of IoT at the edge. The future work discussed here will solidify the practical utility and robustness of this powerful paradigm.

6. Case Studies and Exemplar Implementations (Conceptual)

To illustrate the practical application and benefits of multi-layered feature models, this section outlines conceptual case studies in diverse IoT domains. While detailed implementation is beyond the scope of this article, these examples demonstrate how the proposed methodology would operate in real-world scenarios.

6.1. Smart City Surveillance System

Scenario: A city wants to deploy a distributed surveillance system across various locations (parks, intersections, public buildings) using diverse camera types and processing requirements, with varying network connectivity. The goals are to optimize for real-time anomaly detection, minimize bandwidth usage, and ensure privacy.

Multi-Layered Feature Models in Action:

● Application Layer (ALFM):

○ Core Functionality: Video Capture (resolutions: 1080p, 4K), Motion Detection, Facial Recognition, License Plate Recognition, Crowd Density Estimation.

○ Data Processing: Local Edge Processing (for immediate alerts), Cloud Archival (for long-term storage and advanced analytics).

○ Non-Functional Requirements: Real-time Latency (<200ms for alerts), Privacy Compliance (e.g., Face Anonymization, Data Retention Policy), Bandwidth Optimization.

● Infrastructure Layer (ILFM):

○ Edge Devices: High-End Camera (onboard GPU, high processing power), Mid-Range Camera (basic CPU), Edge Gateway (Xeon CPU, multiple network interfaces), 5G Base Station (with MEC capabilities).

○ Network: Fiber Optic Connectivity (high bandwidth, low latency), 4G/5G Wireless (variable bandwidth, moderate latency), Wi-Fi Mesh (lower bandwidth, higher latency).

● Deployment Layer (DLFM):

○ Task Offloading: Local Motion Detection on Mid-Range Cameras; Facial/License Plate Recognition offloaded to Edge Gateway (for privacy-preserving local processing) or 5G MEC (for ultra-low latency). Raw 4K video processing offloaded to Cloud only if Fiber Optic connection available.

○ Resource Allocation: Allocate higher CPU/GPU limits to Facial Recognition modules on Edge Gateways.

○ Data Routing: Anonymized event data routed to central city control, Raw video (if not anonymized) routed to specific secure Cloud Storage only after local processing.

○ Security: TLS encryption for all communication paths; Role-Based Access Control for video feeds.

Optimization Example: If Real-time Facial Recognition is selected in the ALFM, the system will search for High-End Cameras or Edge Gateways with GPU capabilities in the ILFM and suggest a Local Processing or Edge Offloading decision in the DLFM to meet latency. If a location only has a Mid-Range Camera and 4G connectivity, the model might propose only Motion Detection locally and offloading Facial Recognition (with higher latency) to the cloud, or prompt the need for infrastructure upgrade. The Privacy Compliance feature would enforce the selection of Face Anonymization if raw video is processed at the edge.

6.2. Smart Agriculture: Precision Farming

Scenario: A large farm aims to monitor crop health, soil conditions, and livestock using various sensors and drones, requiring localized data analysis and rapid decision-making for irrigation or pest control.

Multi-Layered Feature Models in Action:

● Application Layer (ALFM):

○ Data Collection: Soil Moisture Sensors, Weather Stations, Drone Imagery (RGB, NDVI), Livestock Trackers.

○ Data Processing: Crop Health Analysis (from NDVI), Soil Nutrient Analysis, Animal Behavior Anomaly Detection, Automated Irrigation Control.

○ Non-Functional Requirements: Energy Efficiency (for battery-powered sensors), Robustness to Intermittent Connectivity, Data Freshness (e.g., hourly updates for soil, daily for drone imagery).

● Infrastructure Layer (ILFM):

○ Edge Devices: Battery-powered Wireless Sensors (LoRaWAN), Farm Gateway (ruggedized, solar-powered, with embedded PC), Drone (onboard compute for immediate image processing).

○ Network: LoRaWAN (low bandwidth, long range), Wi-Fi (local connectivity), Satellite Internet (for remote areas with no cellular).

● Deployment Layer (DLFM):

○ Task Offloading: Initial image stitching and NDVI calculation on Drone (if capable), Comprehensive Crop Health Analysis offloaded to Farm Gateway. Long-term historical data analysis offloaded to Cloud.

○ Resource Allocation: Prioritize Farm Gateway CPU for Automated Irrigation Control if Soil Nutrient Analysis indicates immediate need.

○ Energy Optimization: Adjust Soil Moisture Sensor reporting frequency based on Battery Level (ILFM) and Crop Criticality (ALFM).

Optimization Example: If a remote field has LoRaWAN connectivity and Battery-powered Sensors (ILFM), the ALFM's Energy Efficiency feature will guide the DLFM to select Low Data Sampling Rate and Local Processing of simple thresholds, with infrequent data upload to the Farm Gateway. If a Drone captures 4K NDVI imagery and has Onboard Compute (ILFM), the Crop Health Analysis (ALFM) will be configured for Local Edge Processing on the drone to immediately detect issues and trigger Automated Irrigation Control via the DLFM, avoiding delays from cloud round-trips.

6.3. Industrial IoT: Predictive Maintenance

Scenario: A manufacturing plant wants to implement predictive maintenance on critical machinery to reduce downtime, requiring continuous sensor data monitoring and real-time anomaly detection.

Multi-Layered Feature Models in Action:

● Application Layer (ALFM):

○ Data Collection: Vibration Sensors, Temperature Sensors, Acoustic Sensors.

○ Data Processing: Data Filtering, Feature Extraction (e.g., FFT for vibration data), Machine Learning Model Inference (e.g., LSTM for anomaly detection), Alert Generation.

○ Non-Functional Requirements: Ultra-low Latency for critical alerts (<50ms), High Reliability (no data loss), Data Security (IP of machinery), High Throughput (continuous sensor streams).

● Infrastructure Layer (ILFM):

○ Edge Devices: Industrial IoT Gateway (ruggedized, high-performance CPU, often fanless), Programmable Logic Controller (PLC) with edge capabilities, Industrial PC.

○ Network: Industrial Ethernet (e.g., EtherCAT, PROFINET), Private 5G Network (low latency, high bandwidth, reliability), Wi-Fi 6.

● Deployment Layer (DLFM):

○ Task Offloading: Raw sensor data collection and basic filtering on PLC/Industrial IoT Gateway. Feature Extraction and ML Inference executed on Industrial PC acting as a local edge server. Model retraining and long-term trend analysis offloaded to On-premise Cloud or Public Cloud.

○ Resource Allocation: Dedicate CPU cores and memory on the Industrial PC for critical ML Inference tasks.

○ Security: Network Segmentation for OT (Operational Technology) network, Mutual TLS for all communications, Intrusion Detection System deployed on gateway.

○ High Availability: Redundant Industrial IoT Gateways and Failover Mechanisms (DLFM) if High Reliability is selected in ALFM, requiring compatible infrastructure in ILFM.

Optimization Example: For Ultra-low Latency Alert Generation (ALFM), the system would identify Industrial IoT Gateways or Industrial PCs with powerful CPUs and Private 5G Network connectivity (ILFM). The DLFM would then configure Local Processing and Edge Offloading strategies, minimizing hops to ensure the alert reaches operators within milliseconds. If High Throughput from many vibration sensors is needed, the system might recommend deploying multiple Industrial PCs configured for parallel Feature Extraction and ML Inference.

These conceptual case studies demonstrate how multi-layered feature models provide a holistic and systematic approach to design, validate, and optimize IoT application deployments across diverse and challenging edge environments, effectively translating high-level requirements into concrete, performant, and reliable configurations.

7. Implementation Considerations and Technical Aspects

Translating the multi-layered feature model methodology into a practical system requires careful consideration of various technical aspects, including toolchain integration, data representation, and the development of specific algorithms. This section delves into these implementation details.

7.1. Model Representation and Storage

The feature models themselves need to be represented in a machine-readable and parsable format.

● XML/JSON Formats: Feature models can be represented using structured data formats like XML or JSON. This allows for easy parsing by software tools and enables serialization and deserialization for storage and exchange.

○ Example (Simplified JSON Structure):

JSON

```
{
 "application_model": {
  "name": "SmartHomeMonitoring",
  "features": [
   { "id": "F_VideoFeed", "type": "optional", "attributes": {"resolution": "1080p", "fps": 30} },
   { "id": "F_TempSensor", "type": "mandatory", "attributes": {"sampling_rate": "10s"} }
  ],
  "constraints": [
   "F_VideoFeed requires I_GPU_Support",
   "F_VideoFeed.fps > 25 implies D_Edge_Offloading"
  ]
 },
 "infrastructure_model": {
  "name": "HomeEdgeNetwork",
  "devices": [
   { "id": "I_Edge_Cam_1", "type": "HighEndCamera", "attributes": {"cpu_cores": 4, "gpu_support": true, "network_type": "WiFiAC"} },
   { "id": "I_Temp_Sensor_Node_1", "type": "BasicSensor", "attributes": {"cpu_cores": 1, "memory_mb": 16, "network_type": "LoRa"} }
  ],
  "network_segments": [
   { "id": "N_WiFiAC_Home", "attributes": {"bandwidth_mbps": 100, "latency_ms": 10} }
  ]
 },
 "deployment_model": {
  "name": "SmartHomeDeployment",
  "features": [
   { "id": "D_Edge_Offloading", "type": "optional" },
   { "id": "D_Cloud_Archival", "type": "optional" }
  ],
  "constraints": [
   "select(D_Edge_Offloading) implies select(I_Edge_Cam_1)"
  ],
  "mappings": [
   {"app_feature": "F_VideoFeed", "deploy_action": "offload_to_edge", "target_infra": "I_Edge_Cam_1"},
   {"app_feature": "F_TempSensor", "deploy_action": "local_process", "target_infra": "I_Temp_Sensor_Node_1"}
  ]
 }
}
```

● Ontologies and Semantic Web Technologies: For highly complex and evolving IoT ecosystems, representing feature models and their relationships using ontologies (e.g., OWL, RDF) could offer greater expressiveness, semantic richness, and reasoning capabilities. This allows for more sophisticated query answering and interoperability.

● Dedicated Feature Modeling Tools: Leveraging existing tools like FeatureIDE or developing extensions for FAMILIAR could streamline the model creation and management process. These tools often provide graphical editors, syntax checking, and basic consistency validation.

7.2. Constraint Representation and Solver Integration

The precise definition and efficient processing of cross-layer constraints are paramount.

● Propositional Logic and First-Order Logic: Simple requires/excludes constraints can be directly translated into propositional logic. Attribute-based constraints require First-Order Logic (FOL) or extensions like Satisfiability Modulo Theories (SMT) to handle numerical comparisons and arithmetic.

● SMT-LIB Format: SMT solvers typically accept input in a standardized format called SMT-LIB. A core component of the implementation would be a model translator that converts the multi-layered feature model and its constraints (from XML/JSON/ontology) into an SMT-LIB formula.

● Optimization Objectives: For optimization, the fitness function for metaheuristics or the objective function for SMT solvers (e.g., using MaxSMT) needs to be defined programmatically based on the quantitative attributes and desired optimization goals. This often involves defining weighted sums or multi-objective functions.

● Solver API Integration: The chosen SMT solver (e.g., Z3) or metaheuristic library (e.g., DEAP for Python) would be integrated via its respective API. This allows the system to programmatically pose queries, retrieve solutions, and analyze results.

7.3. Deployment Artifact Generation

The ultimate goal is to generate executable deployment artifacts.

● Template-Based Generation: This involves using templating engines (e.g., Jinja2, Go templates) to generate deployment files (e.g., Kubernetes YAML, Docker Compose, Ansible playbooks). The selected features and their attributes from the valid configuration would populate these templates.

● Domain-Specific Translators: For highly specialized IoT platforms or custom edge runtimes, a dedicated translator might be required to convert the high-level deployment decisions into specific low-level commands or configurations.

● Configuration Management Tools: Integration with configuration management tools like Ansible, Puppet, or Chef can automate the provisioning and configuration of edge devices based on the generated deployment manifests.

7.4. Runtime Monitoring and Adaptation (Conceptual)

While a detailed runtime system is future work, conceptual considerations for integration are important.

● Monitoring Agents: Lightweight agents deployed on edge devices collect real-time data (CPU usage, memory, network latency, battery level, application-specific metrics).

● Data Aggregation and Analysis: A central or distributed component aggregates and analyzes monitoring data to detect deviations from desired performance or resource availability.

● Re-evaluation Trigger: If a significant deviation is detected (e.g., sustained high latency, low battery), it triggers a re-evaluation process:

1. The current state of the edge environment is used to update attributes in the ILFM.

2. The optimization problem is re-solved with the updated environmental context.

3. A new optimal or near-optimal deployment configuration is proposed.

● Dynamic Reconfiguration Engine: This engine translates the proposed new configuration into live changes on the edge infrastructure. This is the most challenging part, requiring careful state management, minimal disruption, and rollback capabilities. This could involve Kubernetes kubectl apply commands for containerized deployments or device-specific API calls.

7.5. Toolchain and Ecosystem Considerations

A complete implementation would involve several integrated components:

● Feature Model Editor: A user interface for graphically defining and managing the multi-layered feature models.

● Constraint Editor: A mechanism for defining cross-layer constraints, ideally with auto-completion and validation.

● Model Repository: A database or version control system for storing and managing the evolution of feature models.

● Reasoning Engine: The SMT solver and/or metaheuristic algorithms, integrated as a backend service.

● Deployment Generator: The component responsible for generating platform-specific deployment artifacts.

● Monitoring & Feedback Loop: (Future) Components for real-time data collection, analysis, and triggering re-adaptation.

The development of such a system would likely be an iterative process, starting with core functionality (design-time validation and generation) and progressively adding more advanced capabilities (optimization, runtime adaptation). Leveraging existing open-source tools and libraries wherever possible would accelerate development and foster community adoption.

8. Broader Impact and Ethical Considerations

The application of multi-layered feature models for IoT application deployment in edge environments, while offering significant technical advantages, also carries broader implications for society, the economy, and ethical practices. It is crucial to consider these impacts as the technology evolves.

8.1. Economic Impact

● Reduced Operational Costs: By optimizing resource allocation and energy efficiency, organizations can significantly reduce the operational costs associated with large-scale IoT deployments, including energy bills and infrastructure maintenance.

● Faster Time-to-Market: Automated and validated deployments accelerate the development and release cycles of new IoT products and services, leading to increased competitiveness and innovation.

● New Business Models: The flexibility and adaptability enabled by this approach can foster new business models, such as "IoT as a Service" or highly customized vertical IoT solutions, allowing providers to tailor deployments precisely to client needs.

● Workforce Transformation: While automating some aspects of deployment, it also creates a need for new skill sets in variability modeling, formal methods, and complex system optimization. This could lead to a shift in the job market, requiring reskilling initiatives.

● Democratization of IoT Deployment: By simplifying complex deployment decisions, the methodology could make advanced IoT deployments accessible to a broader range of organizations, including SMEs, who may lack specialized expertise.

8.2. Environmental Impact

● Enhanced Energy Efficiency: A core benefit of the approach is the ability to explicitly optimize for energy consumption. This can lead to significant reductions in the carbon footprint of IoT infrastructure, particularly in large-scale deployments where small optimizations per device can sum up to substantial savings.

● Reduced Electronic Waste: By allowing for more efficient utilization of existing edge hardware and guiding appropriate deployment choices, it could potentially extend the lifespan of devices or reduce the need for constant hardware upgrades, contributing to a reduction in electronic waste.

● Sustainable Edge Computing: The methodology contributes to the broader goal of sustainable edge computing by enabling environmentally conscious design and deployment choices from the outset.

8.3. Societal Impact

● Improved Quality of Life: More efficient and reliable IoT applications can improve various aspects of daily life, from smarter homes and cities to more responsive healthcare systems and safer industrial environments.

● Increased Accessibility: Optimized deployments can bring advanced IoT capabilities to remote or underserved areas with limited infrastructure, bridging digital divides.

● Privacy Concerns: While the models can incorporate privacy features (e.g., local data anonymization), the very efficiency of data processing at the edge raises concerns about data collection and potential misuse. Robust ethical guidelines and regulatory frameworks are necessary.

● Dependability and Trust: By enabling more resilient and fault-tolerant deployments, the methodology can increase public trust in critical IoT applications, such as autonomous vehicles or smart grids.

8.4. Ethical Considerations

The deployment of IoT systems, especially those with advanced processing capabilities at the edge, raises several critical ethical considerations that must be addressed proactively.

● Data Privacy and Security:

○ Data Minimization: Can the feature models be used to enforce data minimization principles, ensuring only necessary data is collected and processed?

○ Purpose Limitation: How can the models ensure that data processed at the edge is only used for its stated purpose, preventing mission creep?

○ Consent and Transparency: How can the complexity of edge deployments be communicated transparently to users, allowing for informed consent regarding data collection and processing?

○ Homomorphic Encryption/Federated Learning: Future work could explore how to integrate features related to privacy-preserving technologies into the models to ensure privacy-by-design.

● Bias and Fairness in AI/ML at the Edge: If the deployed IoT applications involve AI/ML inference at the edge, there is a risk of propagating biases present in the training data or models.

○ Model Selection: Can the feature models guide the selection of AI/ML models that are known to be fairer or less biased for specific edge contexts?

○ Explainability: Can the deployment process ensure that edge AI inferences are explainable, particularly in critical applications where decisions could have significant impact?

● Accountability and Governance:

○ Responsibility: Who is accountable when an optimized deployment fails or causes harm due to a complex interaction of features and constraints across layers?

○ Auditability: Can the feature models and their configuration logs provide an auditable trail of deployment decisions and their underlying rationale?

○ Human Oversight: While automation is key, ensuring appropriate human oversight and intervention points, especially in critical systems, is essential.

● Environmental Justice: Are there risks that the benefits of optimized edge deployments disproportionately favor certain regions or populations, while the environmental burden (e.g., from e-waste if not properly managed) falls on others?

● Digital Divide: While potentially democratizing access, there's also a risk that the complexity of designing and managing such systems creates new digital divides for those without the expertise or resources.

Addressing these ethical considerations requires a multidisciplinary approach, integrating insights from computer science, ethics, law, and social sciences. The framework of multi-layered feature models, with its explicit representation of system properties, provides a strong foundation for building more ethical and responsible IoT deployments by allowing these considerations to be embedded into the design and deployment process from the very beginning. It allows for the formalization of ethical "features" and "constraints" that must be satisfied by a valid deployment.

9. CONCLUSION

The proliferation of IoT applications and the increasing adoption of edge computing necessitate advanced methodologies for efficient and optimized deployment. This article has presented a compelling case for leveraging multi-layered feature models as a powerful paradigm to address the inherent complexities and variabilities in deploying IoT applications on heterogeneous edge infrastructures.

By defining distinct feature models for the application, infrastructure, and deployment layers, and by establishing explicit relationships and constraints between them, we can achieve automated validation, configuration, and optimization of deployment decisions. This approach offers significant advantages, including enhanced deployment automation, optimized resource allocation, improved performance (e.g., reduced latency, increased energy efficiency), and better management of variability across the entire IoT software product line. The integration of advanced reasoning techniques like SMT solvers and metaheuristic algorithms enables the systematic identification of optimal configurations that meet diverse functional and non-functional requirements.

While challenges related to model complexity and scalability, run-time adaptability, and seamless integration with existing orchestration platforms remain, the foundational benefits of this approach are clear. Future research should critically focus on developing scalable modeling techniques, efficient run-time adaptation mechanisms, and robust empirical validation in real-world IoT scenarios across various domains. Furthermore, proactively addressing the broader economic, environmental, and societal impacts, including crucial ethical considerations such as data privacy, bias in AI, and accountability, will be paramount for the responsible and successful adoption of this methodology.

As edge computing continues to evolve as a critical enabler for the next generation of IoT applications, multi-layered feature models will play an increasingly vital role in ensuring their effective, efficient, and resilient deployment. This framework provides the intellectual scaffolding necessary to transform the complex art of IoT edge deployment into a more precise, automated, and optimizable engineering science.

REFERENCES

1. Abbas, A., Farah Siddiqui, I., Lee, S.U., Kashif Bashir, A., Ejaz, W., Qureshi, N.M.F., 2018. Multi-objective optimum solutions for IoT-based feature models of software product line. IEEE Access 6, 12228–12239.
2. Acher, M., Collet, P., Gaignard, A., Lahire, P., Montagnat, J., France, R.B., 2012. Composing multiple variability artifacts to assemble coherent workflows. Softw. Qual. J. 20 (3), 689–734.
3. Acher, M., Collet, P., Lahire, P., France, R.B., 2013. FAMILIAR: A domain-specific language for large scale management of feature models. Sci. Comput. Program. 78 (6), 657–681.
4. Cañete, A., Amor, M., Fuentes, L., 2022. The Journal of Systems & Software 183, 111086.
5. Lettner, M., Rodas, J., Galindo, J.A., Benavides, D., 2019. Automated analysis of two-layered feature models with feature attributes. J. Comput. Lang. 51, 154–172.
6. Liu, F., Tang, G., Li, Y., Cai, Z., Zhang, X., Zhou, T., 2019. A survey on edge computing systems and tools. Proc. IEEE 107 (8), 1537–1562.
7. Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B., 2017. A survey on mobile edge computing: The communication perspective. IEEE Commun. Surv. Tutor. 19 (4), 2322–2358.
8. Melendez, S., McGarry, M.P., 2017. Computation offloading decisions for reducing completion time. In: 2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC). pp. 160–164.
9. Niewiadomski, A., Skaruz, J., Penczek, W., Szreter, M., Jarocki, M., 2014. SMT versus genetic and OpenOpt algorithms: Concrete planning in the PlanICS framework. Fund. Inform. 135, 451–466.
10. Özbakir, L., Baykasoğlu, A., Tapkan, P., 2010. Bees algorithm for generalized assignment problem. Appl. Math. Comput. 215 (11), 3782–3795.
11. Plauth, M., Feinbube, L., Polze, A., 2017. A Performance Survey of Lightweight Virtualization Techniques. pp. 34–48.
12. Pohl, K., Böckle, G., Linden, F., 2005. Software Product Line Engineering: Foundations, Principles, and Techniques.
13. Premsankar, G., Di Francesco, M., Taleb, T., 2018. Edge computing for the internet of things: A case study. IEEE Internet Things J. 5 (2), 1275–1284.
14. Rabiser, D., Prähofer, H., Grünbacher, P., Petruzelka, M., Eder, K., Angerer, F., Kromoser, M., Grimmer, A., 2016. Multi-purpose, multi-level feature modeling of large-scale industrial software systems. Softw. Syst. Model. 17.
15. Reiser, M.-O., Weber, M., 2007. Multi-level feature trees. Requir. Eng. 12 (2), 57–75.
16. Rosenmüller, M., Siegmund, N., Thüm, T., Saake, G., 2011. Multi-dimensional variability modeling. In: Fifth International Workshop on Variability Modelling of Software-Intensive Systems, Namur, Belgium, January 27-29, 2011. Proceedings. In: ACM International Conference Proceedings Series, ACM, pp. 11–20.
17. Ai, Y., Peng, M., Zhang, K., 2018. Edge computing technologies for Internet of Things: a primer. Digit. Commun. Netw. 4 (2), 77–86.

18. Al-Shuwaili, A., Simeone, O., 2017. Energy-efficient resource allocation for mobile edge computing-based augmented reality applications. IEEE Wirel. Commun. Lett. 6 (3), 398–401.

19. Bagchi, S., Siddiqui, M.-B., Wood, P., Zhang, H., 2020. Dependability in edge computing. Commun. ACM 63 (1), 58–66.

20. Benavides, D., Trinidad, P., Ruiz-Cortés, A., 2005. Automated reasoning on feature models. In: Advanced Information Systems Engineering. Springer Berlin Heidelberg, pp. 491–503.

21. Bjørner, N., Phan, A.-D., Fleckenstein, L., 2015. νZ - AN optimizing SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 194–199.

22. Bratterud, A., Walla, A., Haugerud, H., Engelstad, P.E., Begnum, K., 2015. IncludeOS: A minimal, resource efficient unikernel for cloud services. In: 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). pp. 250–257.

23. Bulej, L., Bures, T., Filandr, A., Hnetynka, P., Hnetynková, I., Pacovsky, J., Sandor, G., Gerostathopoulos, I., 2021. Managing latency in edge-cloud environment. J. Syst. Softw. 172, 110872.

24. Cañete, A., Amor, M., Fuentes, L., 2020. Energy-efficient deployment of IoT applications in edge-based infrastructures: A software product line approach. IEEE Internet Things J. 1.

25. Cañete, A., Amor, M., Fuentes, L., 2019. Optimal assignment of augmented reality tasks for edge-based variable infrastructures. In: 13th Int. Conf. on Ubiquitous Computing and Ambient Intelligence, UCAmI 2019, Toledo, Spain, December 2-5, 2019. In: MDPI Proceedings, vol. 31, MDPI, p. 28.

26. Cañete, A., Horcas, J.-M., Ayala, I., Fuentes, L., 2020. Energy efficient adaptation engines for android applications. Inf. Softw. Technol. 118, 106220.

27. Casalicchio, E., 2019. Container orchestration: A survey. In: Systems Modeling: Methodologies and Tools. Springer International Publishing, pp. 221–235.

28. Cecchinel, C., Mosser, S., Collet, P., 2016. Automated deployment of data collection policies over heterogeneous shared sensing infrastructures. In: 23rd Asia-Pacific Software Engineering Conference, APSEC 2016, Hamilton, New Zealand, December 6-9, 2016. IEEE Computer Society, pp. 329–336.

29. Chen, M., Dong, M., Liang, B., 2016. Joint offloading decision and resource allocation for mobile cloud with computing access point. In: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 3516–3520.

30. Chen, M., Yixue, H., 2018. Task offloading for mobile edge computing in software defined ultra-dense network. IEEE J. Sel. Areas Commun. PP, 1.

31. Czarnecki, K., Helsen, S., Eisenecker, U., 2005. Formalizing cardinality-based feature models and their specialization. Softw. Process: Improv. Pract. 10 (1), 7–29.

32. De Moura, L., Bjørner, N., 2008. Z3: An efficient SMT solver. In: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. In: TACAS'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg, pp. 337–340.

33. Dhungana, D., Grünbacher, P., Rabiser, R., Neumayer, T., 2010. Structuring the modeling space and supporting evolution in software product line engineering. J. Syst. Softw. 83 (7), 1108–1122, SPLC 2008.

34. Dinh, T.Q., Tang, J., La, Q.D., Quek, T.Q.S., 2017. Offloading in mobile edge computing: Task allocation and computational frequency scaling. IEEE Trans. Commun. 65 (8), 3571–3584.

35. Elazhary, H., 2018. Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions. J. Netw. Comput. Appl. 128, 105–140.

36. Farahani, E., Habibi, J., 2019. Feature model configuration based on two-layer modelling in software product lines. Int. J. Electr. Comput. Eng. 9, 1–11.

37. Gámez, N., Fuentes, L., 2013. Architectural evolution of FamiWare using cardinality-based feature models. Inf. Softw. Technol. 55 (3), 563–580.

38. Geraldi, R.T., Reinehr, S., Malucelli, A., 2020. Software product line applied to the Internet of Things: A systematic literature review. Inf. Softw. Technol. 124, 106293.

39. Guo, J., White, J., Wang, G., Li, J., Wang, Y., 2011. A genetic algorithm for optimized feature selection with resource constraints in software product lines. J. Syst. Softw. 84 (12), 2208–2221.

40. Holl, G., Grünbacher, P., Rabiser, R., 2012. A systematic review and an expert survey on capabilities supporting multi product lines. Inf. Softw. Technol. 54 (8), 828–852.

41. Huang, M., Liu, W., Wang, T., Liu, A., Zhang, S., 2020. A cloud–MEC collaborative task offloading scheme with service orchestration. IEEE Internet Things J. 7 (7), 5792–5805.

42. James, A., Schien, D., 2019. A low carbon kubernetes scheduler. In: Proceedings of the 6th International Conference on ICT for Sustainability, ICT4S 2019, Lappeenranta, Finland, June 10-14, 2019. In: CEUR Workshop Proceedings, vol. 2382, CEUR-WS.org.

43. Köksal, O., Tekinerdogan, B., 2019. Architecture design approach for IoT-based farm management information systems. Precis. Agric. 20 (5), 926–958.

44. Sarker, V.K., Peña Queralta, J., Gia, T.N., Tenhunen, H., Westerlund, T., 2019. Offloading SLAM for indoor mobile robots with edge-fog-cloud computing. In: 2019 1st Int. Conf. on Advances in Science, Engineering and Robotics Technology (ICASERT). pp. 1–6.

45. Satyanarayanan, M., 2017. The emergence of edge computing. Computer 50 (1), 30–39.

46. Shi, W., Pallis, G., Xu, Z., 2019. Edge computing [scanning the issue]. Proc. IEEE 107 (8), 1474–1481.

47. Spinczyk, O., Beuche, D., 2004. Modeling and building software product lines with eclipse. In: Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications. In: OOPSLA '04, ACM, New York, NY, USA, pp. 18–19.

48. Sundermann, C., Thüm, T., Schaefer, I., 2020. Evaluating #SAT solvers on industrial feature models. In: Proceedings of the 14th Int. Conference on Variability Modelling of Software-Intensive Systems. In: VAMOS '20, ACM, New York, NY, USA.

49. Tran, T.X., Pompili, D., 2019. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. IEEE Trans. Veh. Technol. 68 (1), 856–868.

50. Wang, J., Pan, J., Esposito, F., Calyam, P., Yang, Z., Mohapatra, P., 2019. Edge cloud offloading algorithms: Issues, methods, and perspectives. ACM Comput. Surv. 52 (1), 2:1–2:23.

51. Zhang, K., Mao, Y., Leng, S., Zhao, Q., Li, L., Peng, X., Pan, L., Maharjan, S., Zhang, Y., 2016. Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks. IEEE Access 4, 5896–5907.

52. Zhang, W., Wen, Y., Wu, D.O., 2013. Energy-efficient scheduling policy for collaborative execution in mobile cloud computing. In: 2013 Proceedings IEEE INFOCOM. pp. 190–194.