In-Network Machine Learning for Real-Time Patient Monitoring on IoMT Edge Gateways

Aristide Tanyi-Jong Akem, *Member, IEEE,* Huiqi Yvonne Lu, *Senior Member, IEEE,* and Noa Zilberman, *Senior Member, IEEE*

***Abstract*—The Internet of Medical Things is transforming healthcare from reactive, hospital-based care to preventive and continuous remote monitoring. However, current solutions often depend on cloud or mobile platforms for data aggregation and analysis, introducing latency, privacy risks, and financial burden. In-network machine learning offers an opportunity to address these challenges by enabling real-time analytics directly within the network infrastructure. In this work, we present `VISTA`, an in-network computing framework for health analytics that enables real-time patient monitoring through in-network machine learning within edge gateways. `VISTA` integrates P4-based data plane modules that asynchronously aggregate heterogeneous sensor data and execute machine learning inference locally within the gateway, eliminating the reliance on cloud offloading. Implemented on a Dell Edge Gateway and evaluated on a 2000-patient dataset for early detection of sepsis and heart failure, `VISTA` achieves up to 95% detection accuracy, 2 milliseconds average latency, and 90% reduction in communication overhead, compared with cloud-based baselines. These results demonstrate the potential of in-network machine learning for scalable, low-latency, and privacy-preserving remote patient monitoring.**

***Index Terms*—In-network computing, machine learning, P4, remote patient monitoring, vital signs, wearable devices, Internet of Medical Things.**

I. Introduction

THE healthcare landscape is rapidly evolving, driven by advancements in artificial intelligence, mobile computing, and sensor technologies, particularly in low-resource settings such as low- and middle-income countries (LMICs) [1]. Traditional hospital-based, reactive patient care is giving way to community and home-based, continuous, and personalized monitoring [2]. This transformation is enabled by the Internet of Medical Things (IoMT), which integrates diverse wearable and ambient devices for continuous data collection [3]. However, most existing patient monitoring systems rely on mobile or cloud-based platforms for data aggregation and analysis, which introduces challenges related to latency, reliability, privacy, cost, and connectivity dependence [4]. Network disruptions or intermittent access to cloud services can delay alerts or degrade monitoring continuity, limiting their suitability for real-time and resource-constrained healthcare environments.

Edge computing has emerged as a widespread solution in healthcare to address latency concerns, bringing data processing closer to patients by leveraging Internet of Things (IoT) devices such as wearables, smartphones, gateways, and dedicated fog servers in the case of fog computing [5]–[8]. Despite its adoption, many edge-based healthcare applications still process data at higher software layers rather than within the network stack itself, resulting in residual latency limitations [9]–[12]. This has prompted exploration into more efficient processing paradigms.

In parallel, advances in Software-Defined Networking (SDN) and programmable data planes have shown how computation can be moved into the network for ultra-low-latency processing. Domain-specific languages such as P4 [13] have further enabled offloading computing tasks directly to the data plane [14], allowing network devices to perform operations beyond simple packet forwarding. This paradigm has given rise to in-network machine learning (ML), where trained ML models are deployed on programmable network hardware, *e.g.*, switches, for real-time inference within the network itself [15].

Initially explored in data center and networking contexts, in-network ML has recently been extended to IoT systems, where low-latency decision-making is equally critical. Recent studies demonstrate that IoT gateways equipped with programmable data planes can execute ML inference directly on streaming sensor data for tasks such as attack detection, anomaly detection, and traffic classification [16], [17]. By operating directly in the data plane, these systems avoid data movement between the network and application layers seen in traditional gateway-based inference, thereby reducing processing latency, software overhead, and dependence on centralized cloud analytics, ultimately improving responsiveness.

The same principles hold great promise for IoMT-based patient monitoring, where continuous sensing and rapid response are essential for patient safety. Real-time and low-latency inference in IoMT gateways at the network edge could enable earlier detection of clinical deterioration and improve continuity of monitoring in resource-constrained environments.

Consider a care home where dozens of residents are continuously monitored through wearable and ambient sensors producing streams of heart rate, oxygen saturation, temperature, and activity data. Each resident may have more than ten connected sensors, resulting in hundreds or even thousands of concurrent data streams within the facility. While it is possible to process some of this data on personal devices *e.g.*, smartphones, these devices are restricted to a single patient and often lack the resources and connectivity to handle multiple sensor streams simultaneously. As a result, they frequently depend on cloud services for computation. While a single reading, such as a sharp heart-rate drop, may indicate an

This work was partly funded by the Leverhulme Trust and EU Horizon SmartEdge (101092908, Innovate UK 10056403).

The authors are with the Department of Engineering Science, University of Oxford, OX1 2JD Oxford, U.K. (e-mail: aristide.akem@eng.ox.ac.uk; yvonne.lu@eng.ox.ac.uk; noa.zilberman@eng.ox.ac.uk).

A.T-J. Akem is also with the School of Electronics and Computer Science, University of Southampton, SO17 1BJ Southampton, U.K. (email: a.t-j.akem@soton.ac.uk). A.T-J. Akem and H.Y. Lu are co-first authors.

acute event requiring urgent attention, detecting gradual deterioration often requires observing correlated changes across multiple vital signs. Moreover, aggregating data from several residents enables the detection of common patterns that can reveal the early spread of infection, which is difficult to achieve on personal devices. The local IoMT gateway, which already collects traffic from all sensors in the care home, provides an ideal vantage point for such analysis. By aggregating sensor data streams and running ML inference directly at the gateway, we can support real-time, multi-patient monitoring while reducing latency and preserving data privacy.

Despite this potential, existing in-network ML systems have remained limited to network analytics and security use cases, leveraging traffic features extracted from packets [16]. Such systems are not designed for continuous, privacy-sensitive sensing or multi-sensor data aggregation as required in healthcare monitoring. To the best of our knowledge, *no prior work has explored in-network computing or ML within a programmable data plane specifically for healthcare applications.*

To address this gap, we propose VISTA, a novel in-network computing framework that embeds sensor data aggregation and ML inference directly within the network data plane of IoMT edge gateways. Our goal is to deliver real-time, scalable, privacy-preserving, and cost-effective ubiquitous patient monitoring and alert generation that integrates seamlessly with existing healthcare networks and sensor devices.

Scaling IoMT for healthcare demands, addressing three fundamental challenges around privacy, scalability, and resilience.

Privacy and Data Locality. Patient data collected by IoMT devices must remain secure and compliant with healthcare privacy regulations (*e.g.*, GDPR [18], HIPAA [19]). By performing aggregation and inference within the gateway's network data plane, VISTA keeps sensitive information local, reducing exposure to external networks while supporting regulatory compliance and maintaining patient trust.

Scalability and Real-Time Performance. Healthcare environments involve many patients generating continuous, heterogeneous data streams. Cloud-based solutions struggle with network delays and bandwidth limits, while mobile devices have limited compute capacity. VISTA enables scalable, real-time feature extraction and millisecond-level inference directly at the gateway, supporting real-time health monitoring.

Resilience and Reliability. Continuous patient monitoring demands reliable operation even under network disruptions or device failures. Recent AWS and Microsoft cloud outages have shown how dependence on a few cloud providers can disrupt many online services; about a quarter of the top 10,000 websites would become unreachable during an AWS outage [20]. By executing data aggregation and inference locally within the gateway, independent of external networks, VISTA sustains operation during outages, ensuring uninterrupted monitoring and timely alerting in critical settings.

By proposing VISTA, we address these challenges through the following key contributions:

- We identify the opportunity for in-network ML on IoMT gateways to enable continuous, privacy-preserving, and low-latency patient monitoring.
- We propose and implement VISTA, an in-network computing framework that embeds clinical ML models directly within the gateway data plane for real-time inference and alert generation. VISTA features a P4-based design that performs stateful, line-rate aggregation of heterogeneous sensor streams into structured feature vectors, enabling efficient in-network feature extraction and ML inference. We prototype VISTA as an open-source[1] system on a commercial DELL Edge Gateway.
- We demonstrate the feasibility and performance of VISTA using a clinical dataset comprising vital signs from 2000 patients. Experimental results show up to 95% accuracy in patient deterioration prediction, average inference latency of 2 ms, and communication overhead reductions of up to 90% compared to cloud-based approaches.

The rest of the paper is organized as follows. Section II reviews background and related work. Section III introduces the system design, and Section IV details the in-network data aggregation module. Section V presents the ML model preparation and encoding, followed by evaluation in Section VI, discussion in Section VII, and conclusions in Section VIII.

II. Background and Related Work

A. Early-Warning Systems and Remote Patient Monitoring

Patient monitoring has evolved from bedside, reactive observation to remote and continuous assessment. In traditional hospital settings, at-risk patients are monitored using bedside devices and rule-based scoring systems such as the National Early Warning Score 2 (NEWS2), which aggregates deviations in vital signs into a single risk indicator [21], [22]. While interpretable, these tools rely on manual input and intermittent measurements rather than continuous IoMT data streams. As patient volumes increase, timely identification of deterioration becomes challenging; for example, chronic heart failure patients face over 20% readmission within 30 days [23], [24], and sepsis mortality exceeds 25% due to delayed treatment [25], [26]. These trends underscore the need for automated, data-driven monitoring solutions that enable earlier intervention.

Advances in sensing, wireless communication, and embedded intelligence have enabled continuous monitoring across hospital, home, and community settings [1], [3]. The Internet of Medical Things (IoMT) integrates wearable, implantable, and ambient sensors to collect physiological data such as heart rate, temperature, and oxygen saturation for real-time assessment. Extending monitoring beyond hospitals, IoMT systems often transmit data via mobile gateways or cloud platforms for processing and alert generation [4], [10]–[12]. While cloud- and mobile-based designs ease deployment and support large-scale storage, they depend on reliable connectivity, are susceptible to latency and congestion, and raise privacy and regulatory concerns. These limitations motivate processing closer to the data source, where low-latency and privacy-preserving computation can ensure reliable and continuous operation.

[1]Our source code is available at https://github.com/ox-computing/VISTA

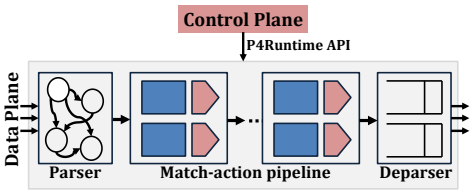


Figure 1: P4-programmable PISA pipeline

B. Edge Computing in IoMT

Edge computing has become a cornerstone technology in IoMT, enabling low-latency analytics and improved privacy by processing data near its source. Numerous application-driven studies have demonstrated its promise for healthcare monitoring as captured in recent surveys [5]–[8]. Building on this foundation, fog and smart-edge frameworks have shifted computation toward IoT gateways for local storage, temporal analytics, and real-time decision-making, reducing latency and bandwidth use [27]–[30]. Some further integrate on-device or gateway-level ML for personalized monitoring [31].

These studies illustrate the growing role of edge and fog computing in improving responsiveness, scalability, and privacy for healthcare IoT. However, most of these frameworks rely on application-layer software running on general-purpose processors, which might introduce additional overhead and limit their ability to process data streams at network line rate. As a result, they cannot fully exploit programmable network data planes for data processing and in-network inference.

In parallel, edge-based lightweight analytics and decentralized learning have been applied primarily to network security and anomaly detection [32]–[34]. These systems demonstrate the feasibility of privacy-preserving edge intelligence but focus on network and security analytics, are not applicable to clinical monitoring, and do not leverage programmable data planes for real-time, in-network processing.

While edge and fog architectures reduce cloud dependence and improve responsiveness, they typically perform analytics at the application layer on general-purpose processors [27], [28]. In these designs, sensor data is processed by software services outside the packet-processing pipeline. In contrast, `VISTA` embeds structured feature aggregation and inference directly within the programmable data plane, enabling deterministic per-packet processing and real-time execution. Moreover, prior IoMT edge systems often target per-device analytics or centralized edge servers, whereas `VISTA` adopts a gateway-centric, multi-patient model that supports concurrent aggregation and inference across diverse physiological streams.

C. Programmable Data Planes

Software-Defined Networking (SDN) transformed traditional networking by introducing a programmable control plane, enabling a centralized controller to manage network device forwarding behavior [35] dynamically. Recent advancements, notably the Protocol-Independent Switch Architecture (PISA) and the P4 language [13], have extended this programmability to the data plane, enhancing packet processing flexibility. As depicted in Figure 1, PISA consists of three key components: (i) a parser that extracts packet headers and metadata, (ii) a reconfigurable Match-Action (M/A) pipeline for processing packets, and (iii) a deparser that reconstructs packets for forwarding. The P4 language defines packet processing logic through M/A tables, configurable at runtime by a control program via interfaces like the P4Runtime API [36]. Available data plane targets today range from software switches, *e.g.*, bmv2 [37], to hardware targets like ASICs *e.g.*, Intel Tofino [38], and SmartNICs [39]. A software switch such as bmv2 can run on devices with general-purpose CPUs, enabling gateway or edge platforms to execute P4 programs for customized packet processing by equipping them with a programmable data plane.

D. In-Network and Network-Assisted Machine Learning

Programmable data planes have enabled two distinct classes of ML applications [15], [40]. The first, often termed *network-assisted ML*, uses programmable switches to accelerate distributed training via in-network operations such as gradient aggregation. Representative systems include DAIET [41], ATP [42], and SwitchML [43], which improve training throughput in data center environments by reducing communication overhead between workers and parameter servers. These approaches optimize distributed training but do not execute inference within the data-plane pipeline, and are therefore orthogonal to this work, which focuses on embedding real-time inference directly into the programmable data plane.

The second class, *in-network ML*, embeds inference logic directly within the match-action pipeline of programmable data planes [15], [40]. This enables line-rate prediction, *i.e.*, inference at packet forwarding speed, eliminating reliance on centralized ML servers and significantly reducing latency [44]. However, such designs are constrained by limited on-chip memory, restricted arithmetic operations, and bounded per-packet processing stages [15]. As a result, most in-network ML systems adopt decision-tree models and ensemble variants such as random forests and XGBoost, which offer a practical balance between accuracy and hardware feasibility. Although neural networks have also been explored in data-plane pipelines [45]–[47], tree-based models remain the most scalable under current architectural constraints.

Existing in-network ML frameworks have primarily targeted programmable switches and SmartNICs, focusing on applications such as traffic classification, anomaly detection, and intrusion prevention [48]–[52]. Only a few efforts have considered deploying such processing on IoT gateways, where the data plane can be implemented in software. Notably, P4Pir [16] integrates programmable data planes into IoT gateways for in-network traffic analysis, and FLIP4 [17] extends this concept with federated learning to reduce communication overhead during model updates. These studies demonstrate that gateway-level data planes can support in-network inference, but they do not address clinical sensing workloads, structured feature aggregation across heterogeneous physiological streams, or integration with patient monitoring and early warning systems.

E. Gaps and Proposed Solution

Existing remote patient monitoring systems rely largely on cloud or mobile platforms for data aggregation and in-

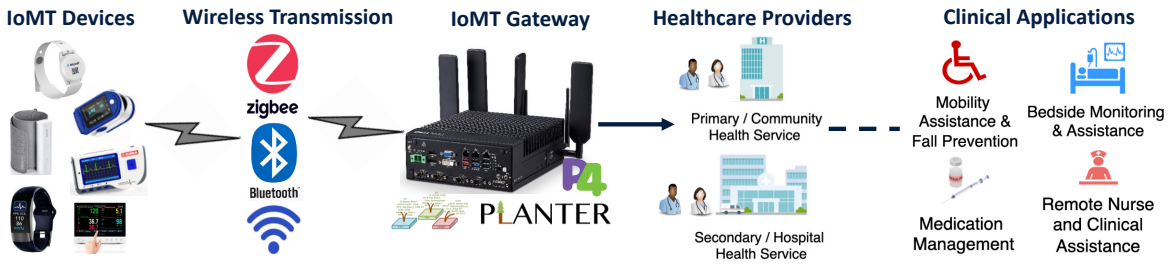


Figure 2: Overview of patient monitoring with VISTA. IoMT devices collect physiological data and transmit it to the IoMT gateway via wireless technologies. The gateway performs local processing and alert generation before forwarding alerts to healthcare providers for clinical applications.

ference [10]–[12], introducing network delays and potential privacy concerns. Reported round-trip times often range from 100 to 300 ms [27], which is on the higher end for real-time or latency-sensitive clinical alerts. Application-layer edge and fog architectures bring computation closer to the data source but might incur software overhead and bandwidth limitations [27]–[30]. Federated Learning (FL) has been widely adopted for privacy-preserving distributed model training across edge devices [17]. While FL could be integrated for initial model training and periodic retraining, it does not address the architectural constraints of executing inference within the programmable match-action pipeline, where computation must satisfy strict per-packet latency and resource limits.

Recent in-network ML frameworks demonstrate the potential of executing ML inference directly within programmable data planes, achieving line-rate packet processing for network analytics [15]. However, these efforts primarily target domains such as traffic classification, intrusion detection, and anomaly detection, not physiological sensing or healthcare [16], [48]–[50]. They lack mechanisms for stateful aggregation of heterogeneous sensor streams for structured feature construction, and ML-driven health-event inference within the network gateway.

This work addresses these gaps by proposing VISTA, a framework which leverages in-network ML to embed both sensor data aggregation and low-latency ML inference directly within IoMT gateways. By integrating these capabilities into the programmable data plane, VISTA enables real-time detection and alert generation for patient deterioration while preserving privacy and reducing reliance on cloud infrastructure. Unlike FL, which addresses the training phase of the ML lifecycle, VISTA focuses on enabling real-time inference execution within the programmable data plane of IoMT gateways. VISTA is therefore complementary to FL-based approaches.

III. Proposed Framework

We propose VISTA, a framework for continuous, privacy-preserving remote patient monitoring at the edge, by leveraging in-network computing on IoMT edge gateways. This section outlines the system overview, design considerations, architecture, and data flow, emphasizing real-time clinical deterioration detection and alerting.

A. System Overview

The patient monitoring workflow of VISTA is illustrated in Figure 2. IoMT sensors continuously collect physiological data such as heart rate, oxygen saturation, and temperature from wearable and bedside devices. These data streams are transmitted via short-range wireless technologies, such as Zigbee, Bluetooth, or WiFi to an IoMT gateway located within the patient's vicinity. A single strategically positioned gateway could serve multiple patients within its range.

The gateway serves as the local computation node and communication hub. It aggregates incoming sensor streams and executes embedded ML models and early warning score calculations within a P4-programmable data plane running in a Docker container. This enables rapid detection of abnormal conditions while maintaining data privacy by avoiding unnecessary cloud transmission of sensitive patient data.

When an abnormal event is detected, alerts are generated and forwarded to healthcare providers. VISTA supports both on-premise alerts, where notifications are delivered to nearby or co-located healthcare staff, and remote alerts transmitted over the Internet to distant or centralized care providers. The resulting insights support various clinical applications such as mobility assistance and fall prevention, bedside monitoring, medication management, and remote clinical support.

B. Design Considerations

1) Design Challenges: Implementing continuous patient monitoring at the edge using programmable network hardware with P4 presents several challenges.

Asynchronous Sensor Data Arrival. Wearable sensors transmit vital sign data (*e.g.*, heart rate, oxygen saturation) asynchronously, with potential delays or missing data due to network variability or sensor failures. This requires robust, memory-aware aggregation mechanisms and stateful operations to maintain context and ensure reliable inference.

Low-Latency and Real-Time Workflow. Timely detection and response are critical for clinical monitoring. Inference and alert-generation pipelines must operate at low latency to support real-time decision-making. This requires an agile in-network workflow that minimizes end-to-end processing delay while maintaining clinical reliability, avoiding false positives or negatives that could compromise patient safety.

Scalability. A single patient may generate data from more than ten sensors, and a ward with several patients can produce dozens of continuous data streams simultaneously. Supporting such workloads on resource-constrained gateways is challenging, especially when executing compute-intensive operations such as ML inference or scoring models like NEWS2. The framework must efficiently scale across patient contexts while maintaining inference accuracy and responsiveness.

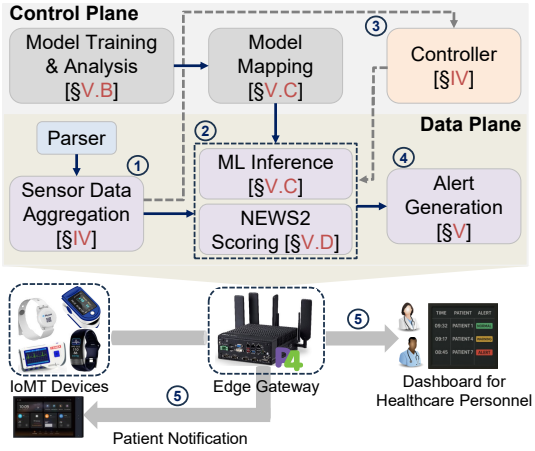


Figure 3: End-to-end VISTA system architecture

Trust and Interpretability. Clinical decision support requires transparency in alert generation alongside privacy measures that keep patient data local to the gateway. ML models should produce interpretable, traceable outcomes that clinicians can validate against recognized standards. Building trust further involves explainability and post-decision auditing while preserving model performance and data confidentiality.

Constrained In-Network Deployment. Deploying clinical scoring protocols (*e.g.*, NEWS2) and ML models in programmable data planes is challenging due to the limited computational and memory resources of network devices. P4's match-action abstraction is optimized for fast packet processing rather than general-purpose computation, requiring careful optimization to achieve full functionality.

2) Design goal: To address these challenges, the goal of VISTA is to develop a patient monitoring framework that robustly handles asynchronous and missing sensor data, achieves millisecond-level latency for ML inference and alert generation, and efficiently scales to multiple patients on IoMT gateways. The system must operate within constrained in-network environments, preserve data privacy through local processing, and provide interpretable, trustworthy outputs that align with clinical expectations.

C. System Architecture

VISTA operates within a P4-programmable data plane on an IoMT edge gateway, enabling real-time processing of sensor data. The architecture, shown in Figure 3, combines a P4-programmable data plane pipeline with a control plane and a healthcare personnel dashboard for continuous monitoring. Wearable IoMT devices collect vital signs, such as temperature, oxygen saturation, pulse rate, systolic blood pressure, respiratory rate, consciousness, and supplemental oxygen, transmitting them to the edge gateway via wireless protocols like Bluetooth or Wi-Fi. The core of VISTA is the P4 pipeline, implemented on a programmable software switch running in a Docker container on the gateway. It comprises: (*i*) an In-Network Sensor Data Aggregation module, (*ii*) an In-Network ML module, (*iii*) a NEWS2 Scorer, and (*iv*) a Controller. Generated alerts are fed to an on-site or remote dashboard for monitoring by clinicians.

The parser extracts features from sensor packets and passes them to the in-network sensor data aggregator, depicted by step ① in Figure 3, which aggregates asynchronous data within a defined time *window*, handling missing or late packets by triggering timeouts for incomplete windows, which are sent to the controller for imputation using historical patient data.

Once there is complete data, the in-network ML module, denoted ② in Figure 3, executes lightweight ML models, such as XGBoost, using Planter [49] to predict the patient medical condition, like sepsis risk or heart failure risk, in real time, and generates alerts for monitoring in step ④. The NEWS2 scorer, also in ②, leverages the same data to compute the NEWS2 score for the patient, based on lookup tables that implement the standard scoring chart. It also enables the generation of *alerts* in step ④, employing a tricolor scheme (Green: NEWS2 $\leq$ 3, Amber: NEWS2 $\geq$ 5 or single parameter score = 3, Red: NEWS2 $\geq$ 7) to distinguish alerts sent as notifications to healthcare personnel and patients in step ⑤.

The controller, a Python program running on the gateway's CPU, interacting with it via the P4Runtime API [36], handles data collection windows with incomplete sensor data. It is marked ③ in Figure 3. At runtime, when there is missing data, it receives a packet with the partial data from the data plane with which it communicates via a dedicated port, does imputation for the missing features by filling them with their average values from the patient's historical data, and emits a packet back to the data plane, which then runs inference and sends out the corresponding alert in step ⑤. It also periodically sends out *heartbeat packets* to the data plane, to trigger the closing of expired windows and the clearing of stale state. Finally, the dashboard for healthcare personnel receives and displays real-time alerts and patient status, enabling healthcare providers to monitor and make timely decisions. Alerts can also be simultaneously sent to the patient's location, ensuring prompt response when healthcare providers are distant.

In long-running deployments, model updates can be managed through the existing control-plane mechanisms of programmable data planes. Prior work has extensively explored safe runtime rule replacement, consistent table updates, and control-plane-driven reconfiguration in P4-based systems [16], [17], [53], and these techniques can be directly applied in VISTA. Accordingly, periodic retraining may be performed offline using newly collected data, after which the updated model is compiled into P4-compatible table entries and re-installed via the runtime interface. Given the relatively low data rates typical of IoMT sensing, brief buffering packets during updates is feasible if needed. As such, model lifecycle management would not introduce new architectural challenges in VISTA and is orthogonal to our primary contribution, which is enabling efficient in-network aggregation and inference within the programmable data plane for patient monitoring.

D. Data Flow and Operation

The data flow in VISTA, illustrated in Figure 4, ensures efficient processing from sensor data collection and aggregation to inference, NEWS2 scoring, and clinical alert generation.

Sensor Data Collection and Packet Parsing: Wearable sensors collect and transmit vital sign data in packets contain-

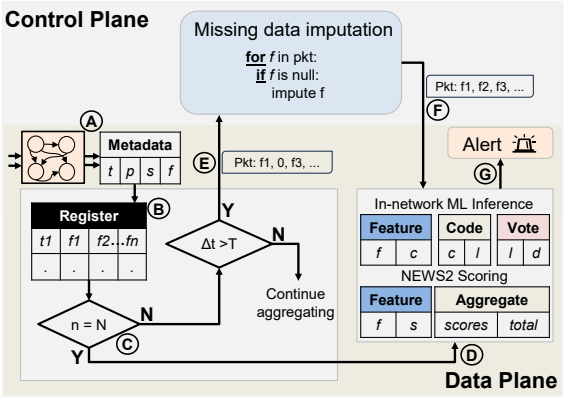


Figure 4: Data flow for real-time patient monitoring with VISTA

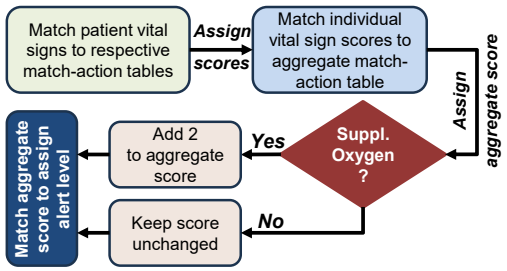


Figure 5: Flowchart for in-network NEWS2 score calculation

ing patient ID, sensor ID, timestamp, and a single feature value (*e.g.*, temperature). The parser in the P4 pipeline parses these packets as shown in step (A) in Figure 4, identifying them by their respective Ethernet types (*e.g.*, 0x1235), and extracting the patient ID, sensor ID, and feature value. The features are then passed on to the match-action pipeline of the data plane, where the rest of the processing happens.

Feature Aggregation and Window Management: The in-network sensor data aggregation module stores incoming feature values in per-feature registers, depicted by (B) in Figure 4, whose memory addresses are indexed with the patient IDs. For each patient, a data collection *window* is initiated upon receiving the first feature, with the timestamp stored in a register to help track windows and manage state. Features are written to dedicated registers based on sensor ID, which stipulates what kind of feature was in the packet. A one-bit register is employed to set a feature presence flag to 1, indicating that the feature is now available. When a feature arrives after the first but within the window, its value is used to update the corresponding register; late packets (arriving after the window ends but within a set *quiet time*) are dropped. The *quiet time* represents the idle interval between consecutive data collection windows, analogous to the pause between routine clinical measurements. During this period, sensors are expected to have completed transmission, and any packets arriving afterward are discarded to maintain data consistency and prevent overlap between adjacent windows.

If a *heartbeat packet* (a periodic control signal from the controller) arrives from the controller during the window, it is simply dropped but if within the *quiet time*, the pipeline checks for any present features and, if found, forwards a Planter packet (Ethernet type 0x1234) with all collected features to the control plane (step (E) in Figure 4) via a dedicated CPU port for the missing features to be imputed and returned to the data plane. Details are provided in Section IV.

NEWS2 Scoring: When all features are collected within the window *i.e.*, when the condition at (C) returns yes, or upon receiving a Planter packet from the control plane as illustrated by (F) in Figure 4, the pipeline computes individual NEWS2 scores for each vital sign using match-action tables, as shown in Figure 5. For instance, temperature (scaled by 10 for non-floating point data plane operation) is mapped to a score of 0–3 based on ranges (*e.g.*, 36.1–38.0°C yields 0). The scores for the main variables, respiratory rate, oxygen saturation, systolic blood pressure, pulse rate, consciousness (AVPU), and temperature are summed, with supplemental oxygen adding 2 or 0 to the aggregate NEWS2 score, based respectively on whether it is in use or not, as shown in Figure 5. The aggregate score is matched on a final table to determine the alert level, indicating how urgent the patient needs attention.

In-Network ML Inference: For complete feature sets, made available through either (D) or (F), the in-network ML module, leveraging Planter [49], applies lightweight XGBoost models to predict patient conditions (Normal, Sepsis Risk, or Heart Failure Risk) and trigger alert generation. Planter [49] is a modular framework designed to map ML models to programmable data plane pipelines. It leverages the P4 match-action pipeline to execute complex ML inference with trained models directly within the network data plane, while maintaining high accuracy with minimal resource overhead. Additional details are provided in Section V-C.

Control Plane Processing: Due to the inherent constraints of programmable data plane pipelines and the P4 language, model training happens in the control plane or a dedicated ML server, as shown in Figure 3. The trained models are then deployed from the control plane as match-action table entries adapted to the dedicated P4 program [15]. At runtime, for incomplete feature sets, triggered by a heartbeat packet or window expiry, a packet with the Planter header is received in the controller as a *PacketIn* message (step (E) in Figure 4). This custom header has the full feature list as its main fields. The controller parses the packet using Scapy [54], imputes missing features (*e.g.*, by filling them with average values from the patient's medical history), and returns the packet as *PacketOut* message (step (F)) to the data plane pipeline for NEWS2 scoring and ML inference.

Alert Generation and Forwarding: Upon inference and scoring, an Alert packet (Ethernet type 0x1236) is generated (step (G) in Figure 4) containing the patient ID, timestamp, patient deterioration alert level, and ML predictions for sepsis and heart failure risk. This packet is forwarded to the monitoring sites for display on the clinician's dashboard.

Clinician Notification: Alerts are received and displayed on a monitoring dashboard, showing patient ID, timestamp, NEWS2 patient deterioration alert level (Green, Amber, Red), and ML predictions of early warning of adverse health events (*e.g.*, NEWS2 $\geq$ 7 or predicted sepsis/heart failure risk). The dashboard is a Python-based program that runs on the monitoring host, receiving and displaying the alerts. It enables healthcare providers to respond to critical conditions concur-

rently in real time. Patients can also be alerted to enable a more prompt reaction by a bedside carer in critical cases.

This workflow leverages the P4 pipeline's stateful registers and match-action tables to achieve millisecond-scale processing for complete feature sets, with predictable timeouts and control plane integration for incomplete data, ensuring high accuracy, low latency, and privacy, by processing all data locally on the edge gateway. The entire VISTA workflow executes within the gateway, where both feature aggregation and ML inference are performed internally. Only compact alert messages are transmitted to healthcare providers when abnormalities are detected. This ensures that sensitive health data is confined to the gateway, preserving patient privacy.

IV. In-Network Data Aggregation

The stateful in-network data aggregation mechanism of the VISTA framework is designed to support real-time, privacy-preserving patient monitoring by processing asynchronous sensor data directly on programmable edge gateways using the P4 programming language. The approach leverages stateful data structures and time-based windowing to handle variable data arrival patterns while ensuring scalability and clinical accuracy for patient monitoring.

A. Asynchronous Sensor Data Handling

In realistic sensor environments, IoMT devices collecting vital signs such as temperature, oxygen saturation, and pulse rate often transmit data asynchronously due to network variability, device limitations, or intermittent connectivity, making it essential to have a robust aggregation scheme to ensure data completeness. This necessity is addressed in VISTA's Sensor Data Aggregation module. A data collection window[2] is initiated for each patient upon receiving the first sensor packet, with the timestamp stored in a stateful register to track the window's start, as outlined in Algorithm 1. In a human-based data collection system, the window is analogous to the time the nurse needs to collect all measurements from the ward, which typically spans several minutes. In contrast, with automated IoMT sensing and continuous wireless transmission, the data collection period can be much shorter since sensor updates occur automatically and more frequently.

Incoming sensor packets, identified by an Ethernet type of 0x1235 in our implementation, are aggregated into feature registers (*e.g.*, `reg_temperature`) indexed by the patient ID. Another register tracks the presence of each feature with a 1-bit flag, enabling the pipeline to detect incomplete data sets. If a packet arrives within the window, it updates the corresponding register, then checks if all features are now available, which either triggers inference or keeps the window open in case more features are being awaited. As clinical data collection typically happens at predefined time windows, between data collection windows, there is a *quiet time*, when no new data is expected to arrive. Late packets arriving after the close of the window but within the *quiet time* are dropped to prevent data corruption by unexpected packets.

[2]We set the window size to 60 seconds. It can be adapted as needed.

Given that the programmable data plane's packet processing pipeline is packet-triggered, it is impossible to initiate any operations without the arrival of a packet. It is possible that a sensor malfunctions, or for some other reason fails to send its data for a long time. As no packet will arrive with similar identification to trigger the checking of its register index, stale state could accumulate in such register locations. To address this, per-patient *heartbeat* packets from the control plane regularly trigger the check of all registers to identify and clear stale state, signalling the closure of windows. They also trigger the pipeline to check for any present features and forward them to the controller, which does data imputation (*i.e.*, filling in missing feature values using default values from historical patient data) and sends the packet back to the data plane with the complete feature set, ensuring robust handling of asynchronous streams for multiple patients.

B. Data Plane Implementation

VISTA's data aggregation mechanism is implemented as a P4 program running on a containerized programmable software switch, optimizing resource-constrained edge devices for real-time processing, as detailed in Algorithm 1.

Algorithm 1 Data Aggregation Workflow

Require: Packet with patient ID p, timestamp t_{now}, sensor with ID s_{id}, and value v

Load state: state$[p] \leftarrow (t_{\text{start}}, F, V, n, \texttt{runInf})$ ▷ F: feature flags, V: feature values, n: feature count, `runInf`
$\Delta \leftarrow t_{\text{now}} - t_{\text{start}}$
if $s_{\text{id}} = H_{\text{b}}$ **then** ▷ Heartbeat packet for patient p
  if $\Delta > T_{\text{timeout}}$ **&** $\Delta < T_{\text{timeout}} + T_{\text{quiet}}$ **&** $n > 0$ **then**
    Send packet to CPU: $(p, F, V,$ timeout=true$)$
    Reset $F, V, n, t_{\text{start}}$
  else
    drop()
  return
else if $n = 0$ **then** ▷ First packet for patient p
  $F[s_{\text{id}}] \leftarrow 1$, $V[s_{\text{id}}] \leftarrow v$, $n \leftarrow 1$
  $t_{\text{start}} \leftarrow t_{\text{now}}$
  return
else if $\Delta < T_{\text{timeout}}$ **then** ▷ Packet in current window
  if $F[s_{\text{id}}] = 0$ **then**
    $F[s_{\text{id}}] \leftarrow 1$, $V[s_{\text{id}}] \leftarrow v$, $n \leftarrow n + 1$
  if $n = N_{\text{features}}$ **then** ▷ All features received
    `runInf` $\leftarrow 1$
    Send ML features to inference pipeline: (p, V)
    Reset $F, V, n, t_{\text{start}}$
  return
else if $\Delta < T_{\text{timeout}} + T_{\text{quiet}}$ **then**
  drop() ▷ Drop unexpected packet
  return
else ▷ Quiet time passed, new window begins
  if $n > 0$ **then**
    Send packet to CPU: $(p, F, V,$ expired=true$)$
  Reset F, V, n
  $F[s_{\text{id}}] \leftarrow 1$, $V[s_{\text{id}}] \leftarrow v$, $n \leftarrow 1$, $t_{\text{start}} \leftarrow t_{\text{now}}$
  return
if `CPU_pkt` **then** ▷ Complete packet from CPU
  $V \leftarrow v$, $F \leftarrow \vec{1}$, $n \leftarrow N_{\text{features}}$, `runInf` $\leftarrow 1$
  Send ML features to inference pipeline: (p, V)
  Reset $F, V, n, t_{\text{start}}$
Save state: state$[p] \leftarrow (t_{\text{start}}, F, V, n,$ `runInf`$)$

Each incoming packet carries a patient identifier (p), a timestamp, a sensor ID (s_{id}), and the sensor's value (v). The packet's arrival time is denoted (t_{now}). The system maintains per-patient registers rows: a start timestamp (t_{start}) for the current data window, a bit vector (F) marking received features, a vector (V) for feature values, and a counter (n) of observed features. For each patient, the system maintains a persistent state represented by the tuple $(t_{\text{start}}, F, V, n, \texttt{runInf})$, `runInf` is a flag indicating whether all features have been collected and inference can proceed.

When a heartbeat packet (identified by $s_{\text{id}} = H_{\text{b}}$) arrives for a specific patient p, the gateway checks how much time has elapsed since the last feature window started. If the timeout period T_{timeout} has passed but it is still within the *quiet time* extension T_{quiet}[3], if any features have already been received ($n > 0$), a packet is packed with all existing features and sent to the CPU for imputation. n is updated via the 1-bit register that tracks the presence of features. After sending the packet to the controller, all registers for the patient are reset. If no features are present or the window is still open, the packet is simply dropped.

If the packet is not a heartbeat, it is processed as a regular sensor packet. The system again computes the time difference Δ from the start of the current window. If this is the first packet for the patient in the current cycle ($n = 0$), a new window is initialized: the received feature is stored, its presence flag is marked, n is updated, and t_{start} is updated to the current time. If the packet is not the first but arrives within the timeout threshold ($\Delta < T_{\text{timeout}}$), and the feature it carries has not yet been observed ($F[s_{\text{id}}] = 0$), it is added to the vector V, the bit vector F is updated, and the counter n is incremented. If this update results in all features being present ($n = N_{\text{features}}$), where N_{features} is the number of features required, inference is triggered by sending the complete feature vector (p, V) to the inference pipeline. The state is then reset for the next window.

Packets that arrive after the timeout period but before the end of the *quiet time* ($T_{\text{timeout}} \leq \Delta < T_{\text{timeout}} + T_{\text{quiet}}$) are dropped silently. This avoids adding stale data and allows the heartbeat packets to finalize the window. If a packet arrives after the entire *quiet time* has passed ($\Delta \geq T_{\text{timeout}} + T_{\text{quiet}}$), the previous window is considered expired. If it had any collected features, a packet $(p, F, V, \text{expired=true})$ is sent to the CPU. The system then initializes a new window using the current packet's data and zeros for the other features and flags.

Special handling is provided for packets from the CPU, denoted `CPU_pkt` in Algorithm 1. These packets have a Planter header as described in Section III-D, and carry a full set of features which are used to enforce inference directly. Upon receiving a `CPU_pkt` packet, the system populates V with the provided values, sets F to all ones, sets n to N_{features}, and enables the `runInf` flag. The feature vector is then sent for inference, and the state is reset. At the end of any packet's processing, the updated state tuple $(t_{\text{start}}, F, V, n, \texttt{runInf})$ is saved to the per-patient state register row to ensure consistency across future arrivals.

[3]Quiet time was set to 30 seconds for most experiments, but is adjustable.

C. Missing feature imputation

`VISTA` addresses missing sensor values through a lightweight imputation mechanism integrated within the controller. When a feature vector $\mathbf{V} = [f_0, f_1, f_2, \ldots, f_6]$ is received, missing values are represented as zeros, *e.g.*, $\mathbf{V} = [f_0, f_1, 0, f_3, 0, f_5, f_6]$. For each missing feature $f_i = 0$, the controller applies a hierarchical imputation policy.

If sufficient patient-specific historical data are available, the controller retrieves prior non-null measurements from the patient's historical record $\mathcal{H}_p$. Let $H_{p,i} = \{h_1, h_2, \ldots, h_n\}$ denote the set of historical values for patient p and feature i. When $n > 0$, the feature is imputed using the patient-specific historical mean, $\hat{f}_i = \text{mean}(H_{p,i})$. This averaging strategy is commonly used in the literature due to its simplicity and robustness in healthcare datasets [55], [56].

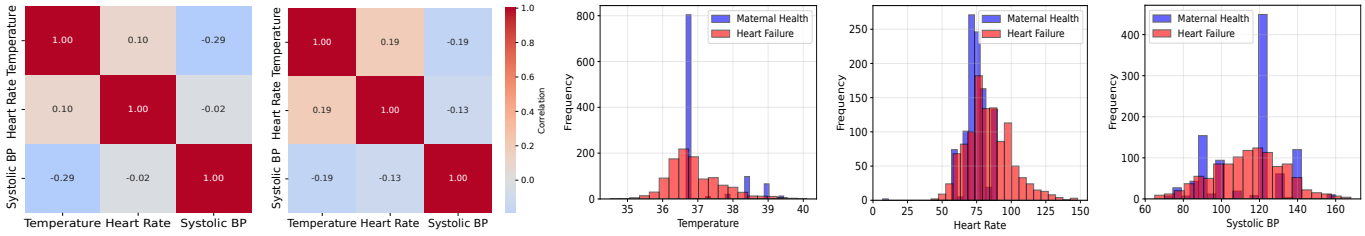
If historical data are unavailable or insufficient, the system substitutes a clinically informed default value defined for each feature. These defaults correspond to medically reasonable reference values and ensure stable inference under missing-sensor conditions. In rare cases where neither patient history nor predefined defaults are available, the population mean computed during model training is used as a final fallback.

The resulting imputed feature vector from the control plane $\mathbf{V}' = [f_0, f_1, \hat{f}_2, f_3, \hat{f}_4, f_5, f_6]$ is then reinjected into the data plane for inference and alert generation. This hierarchical strategy prioritizes preservation of individual physiological baselines while ensuring predictable and bounded behavior when sensor readings are unavailable, ensuring robustness.

While imputation techniques have significantly advanced in recent years, particularly through deep learning-based methods [57] and interpretable ML approaches [58], such methods introduce additional computational complexity and design considerations. These are beyond the scope of this work. Our focus is to demonstrate how `VISTA` enables efficient real-time data aggregation, preprocessing, and patient monitoring within the data plane. The adopted historical averaging strategy provides a lightweight, explainable, and deployment-friendly solution aligned with this system-level objective.

V. In-Network Patient Monitoring with ML

The training and testing of ML models for in-network patient monitoring constitute a preliminary step in the `VISTA` workflow. Due to the limited memory and restricted mathematical operations supported by programmable data planes, these tasks are performed offline on a dedicated ML server or within the control plane, where sufficient computational resources are available to handle complex learning and evaluation processes. The trained models are then deployed to the P4-programmable data plane in a Docker container on the IoMT gateway, which lacks the resources for training [41], but excels at real-time inference on packets. This approach ensures efficient use of edge resources while maintaining privacy and low latency. As we utilize Planter [49] for in-network deployment, we focus on testing ML models compatible with its framework, prioritizing those that can be optimized for edge environments.



(a) Maternal Health (b) Heart Failure (c) Temperature (d) Heart Rate (e) Systolic BP

Figure 6: Maternal Health (real) and Heart Failure (synthetic) physiological data; (a) and (b) present the correlation matrices for both datasets, showing consistent directional relationships among temperature, heart rate, and systolic blood pressure; (c) to (e) show the marginal distributions of these variables, illustrating substantial overlap and clinically plausible value ranges in the synthetic data.

A. Evaluation Datasets

We evaluate VISTA using two complementary datasets: a real-world clinical dataset to assess practical applicability, and a larger clinically informed synthetic dataset to enable controlled, large-scale experimentation.

1) Real-World Clinical Dataset: To ground our evaluation in real physiological measurements, we use the publicly available Maternal Health Risk dataset [59]. The dataset comprises 1,014 fully anonymized patient records collected from IoT-enabled monitoring systems and maternity clinics in Bangladesh. It includes six attributes: Age, Systolic Blood Pressure, Diastolic Blood Pressure, Blood Sugar, Body Temperature, and Heart Rate, along with a categorical target variable indicating low, mid, or high maternal health risk. This anonymized dataset has been used in several AI-for-health works [60]–[62], and is therefore useful for evaluating VISTA to assess its performance on real-world data.

2) Clinically-Informed Synthetic Dataset: Obtaining large-scale real-world patient data collected from IoMT sensors for extensive evaluation is challenging due to privacy regulations and data-sharing restrictions. Consequently, several prior works on healthcare-focused edge and fog systems have also employed synthetic datasets for evaluation [27], [63]. Following this approach, a synthetic dataset was generated based on published vital sign parameter ranges reported in clinical studies [21], [64]–[66], ensuring that the simulated data remain within clinically realistic physiological ranges. Together with the NEWS2 early-warning criteria [67], we emulated a hospital-at-home and virtual ward monitoring scenario with 2,000 patients. Each patient's data covers a continuous 24-hour period, with one sample every 15 minutes, yielding 96 samples per patient and a total of 192,000 samples. The six physiological features summarized in Table I show vital sign ranges of healthy individuals (Normal Range) and patients with one of the four discharge conditions (Simulation Range).

In addition, three patient-specific attributes: supplemental oxygen use, age, and sex, are included to reflect patient-specific conditions that influence risk stratification. These demographic and clinical context variables change infrequently and can be entered by healthcare staff as needed.

The dataset includes two binary labels representing simulated clinical deterioration: sepsis and heart failure. The risk of each condition is modelled based on physiological patterns described in the literature [21], [64], [65], such as rising temperature and heart rate for sepsis, or declining oxygen saturation and fluctuating blood pressure for heart failure. These condition labels enable two binary classification tasks: detecting imminent sepsis and detecting heart failure onset, using only physiological and demographic features.

Table I: Summary of Physiological Features and Ranges

Feature	Unit	Normal Range	Simulation Range
Temperature	°C	36.1–37.2	34.0–41.0
Oxygen Saturation (SpO_2)	%	95–100	80–100
Pulse Rate	bpm	60–100	40–180
Systolic BP	mmHg	90–120	60–200
Respiratory Rate	breaths/min	12–20	5–40
AVPU Score	–	0–3	0–3

To assess the realism of the generated synthetic data, we compare overlapping physiological variables between the Maternal Health and Heart Failure datasets, namely temperature, heart rate, and systolic BP, as shown in Figure 6. The marginal distributions in Figures 6c to 6e exhibit substantial overlap and similar central tendencies, demonstrating that the generated values remain within clinically plausible ranges. Moreover, the correlation matrices in Figures 6a and 6b show consistent directional relationships across datasets, including a weak positive association between temperature and heart rate and a negative coupling between temperature and systolic BP.

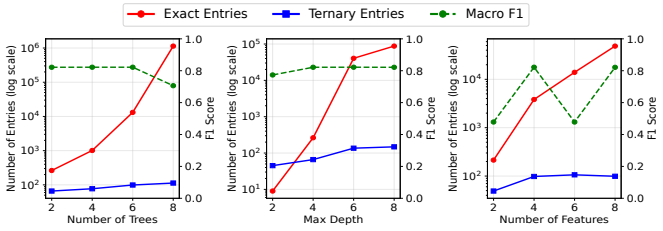
Although the Maternal Health dataset displays greater variability and differs in specific numerical ranges, these differences are expected given the distinct clinical contexts and prediction targets. The synthetic dataset nonetheless preserves inter-feature relationships and physiological ranges, supporting its suitability for controlled system-level evaluation. The complete data description, code, and analysis are available on GitHub: https://github.com/ox-computing/VISTA.

B. Offline ML Model Analysis

We train and evaluate Planter-supported ML models for predicting sepsis and heart failure, selecting the best-performing ones for in-network deployment in VISTA. The synthetic dataset is preprocessed to support two binary classification tasks: Normal vs Sepsis and Normal vs Heart Failure. It comprises 156,768 normal, 18,624 sepsis, and 16,608 heart failure samples. The data is split into training (70%), test (20%), and validation (10%) sets using a patient-stratified approach, ensuring validation patients (183 for Sepsis, 181 for Heart Failure) are independent of training and test sets, with distributions maintained. All ten available features are used for model training and testing. In VISTA's in-network workflow, using all 10 features corresponds to when all sensor data for a

Table II: Model Performance Comparison

	Sepsis		Heart Failure	
Model	Accuracy	F1 Score	Accuracy	F1 Score
XGBoost	0.926	0.727	0.953	0.829
Random Forest	0.919	0.691	0.950	0.817
Logistic Regression	0.919	0.675	0.943	0.799
MLP Classifier	0.919	0.676	0.949	0.812
Gaussian NB	0.918	0.675	0.894	0.723
KNN Classifier	0.912	0.670	0.948	0.809
Support Vector Machine	0.892	0.472	0.941	0.796
Decision Tree	0.885	0.710	0.918	0.773
KMeans Clustering	0.617	0.507	0.614	0.504



(a) (b) (c)

Figure 7: Trade-off between model complexity and resource usage

patient has been received and aggregated. Using their default parameters, we evaluate 9 Planter-compatible classification models, namely: Decision Tree (DT), Random Forest (RF), XGBoost (XGB), Gaussian Naive Bayes (NB), K-Means, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Multi-Layer Perceptron (MLP), and Logistic Regression (LR).

1) Model performance metrics: We evaluate model performance using **accuracy** and **F1 score**. Accuracy measures the proportion of correct predictions, computed as $\text{Accuracy} = (\text{TP}+\text{TN})/(\text{TP}+\text{TN}+\text{FP}+\text{FN})$, where TP, TN, FP, and FN denote true positives, true negatives, false positives, and false negatives, respectively. The F1 score is the harmonic mean of precision and recall, defined as $\text{F1} = 2 \times (\text{Precision} \times \text{Recall})/(\text{Precision}+\text{Recall})$, with $\text{Precision} = \text{TP}/(\text{TP}+\text{FP})$ and $\text{Recall} = \text{TP}/(\text{TP}+\text{FN})$.

2) Model performance results: Table II summarizes the test performance of all evaluated models. The results show that XGB achieves the highest accuracy and Macro F1 scores, followed closely by RF, demonstrating the strength of ensemble tree-based methods in capturing non-linear feature interactions and reducing overfitting. Simpler models such as LR and NB deliver consistent but lower performance, while MLP and KNN achieve competitive accuracy at a higher computational cost. Unsupervised methods like K-Means perform substantially worse due to the absence of label guidance. From an in-network computing perspective, tree-based models such as XGB and RF offer an attractive balance between predictive accuracy and computational efficiency, as their inference involves lightweight conditional operations rather than intensive arithmetic computations. Accordingly, XGB is adopted as the predictive model for integration within the VISTA framework using the Planter toolchain [68], which enables compact model generation and efficient in-network deployment for real-time inference on resource-constrained gateways.

3) Model complexity - resource usage trade-off: We quantify model complexity as $M = f(t, d, F)$, where t denotes the number of trees, d the maximum tree depth, and F the number of input features. To establish the relationship between learning complexity and programmable data-plane resource consumption, we analyze the Heart Failure prediction task and measure resource usage as the number of generated match-action table entries after compilation with Planter [49].

Figure 7 illustrates how each complexity parameter affects the resulting model footprint while reporting the corresponding predictive performance. When increasing the number of trees, t, performance saturates at a small ensemble size, whereas table entries grow super-linearly and eventually become infeasible for deployment when millions of entries are generated, as shown in Figure 7a. A similar trend is observed when increasing tree depth, d, which results in near-exponential growth in exact match entries (Figure 7b). For example, increasing depth from 2 to 4 yields a moderate increase in entries, while further increasing depth beyond 4 leads to orders-of-magnitude growth with negligible improvement in performance. Feature dimensionality exhibits non-linear scaling behavior as shown in Figure 7c. While moderate increases in F can improve predictive performance, further expansion might increase rule fragmentation and table entries without gains in accuracy.

Across all three dimensions, predictive performance saturates early, whereas data-plane resource usage continues to grow rapidly. These results provide empirical guidance for hardware-aware model configuration: for the evaluated workload, shallow trees with a limited number of estimators achieve near-optimal performance while maintaining a compact match-action footprint. This motivates our choice of the model configurations in Table III for in-network deployment.

C. In-Network Model Implementation

As introduced in Section III-D, VISTA employs the Planter framework [49] to deploy trained ML models within programmable data planes. Planter maps models such as XGBoost to the P4 match-action pipeline through an *Encode-Based* (XGB_{EB}) approach that replaces arithmetic operations with table lookups for efficient in-network inference.

Figure 8 illustrates the mapping of a single decision tree. In general, an XGBoost model with N trees and M features is represented using $N + M$ tables: M feature tables and N code tables. The feature tables match input feature values to threshold ranges and assign encoded branch decisions. The resulting codes match onto the code tables, which store precomputed probabilities for each class. A final match-action table aggregates these probabilities across all trees to determine the predicted label. This design enables line-rate inference while keeping the pipeline stage count independent of tree depth.

D. In-Network NEWS2 Scoring

We use in-network computing to calculate NEWS2 scores within the data plane, employing a series of match-action tables. The workflow is illustrated in Figure 5. The scoring uses six vital signs: heart rate, respiratory rate, temperature, systolic blood pressure, oxygen saturation, and level of consciousness, plus the need for supplemental oxygen. Each vital sign is implemented in a dedicated match-action table, which matches it at runtime and assigns a score from 0 to 3. Then,

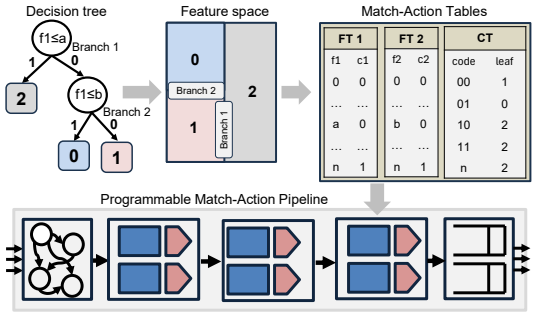


Figure 8: Decision tree mapping to programmable data planes. XGBoost comprises multiple mapped trees.

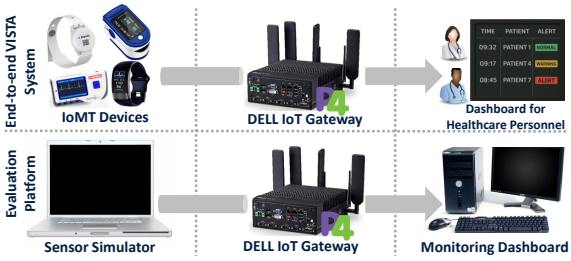


Figure 9: VISTA end-to-end system and evaluation platform

another match-action table aggregates the individual scores, plus the use of supplemental oxygen (0 or 1), taking these 7 numbers as exact match table keys, and then assigns a final score and the alert level based, triggering real-time tricolor alerts (Green: ≤ 3, Amber: ≥ 5 or any score = 3, Red: ≥ 7). This stateless approach ensures continuous monitoring while reducing network load and latency.

VI. Evaluation

We conduct a comprehensive evaluation of VISTA with both datasets described in Section V-A, to demonstrate its capabilities for real-time patient monitoring applications.

A. Experimental Settings

Our experimental setup is shown in Figure 9. We prototype VISTA in P4 using the Behavioral Model v2 (bmv2) software switch [37], deployed in a Docker container on a Dell IoT Edge Gateway 5200 [69]. The gateway is designed for robust IoT applications and is equipped with a 9th Gen Intel Core processor running at 2.2 GHz and 16 GB of DDR4 SO-DIMM RAM. Its CPU executes the VISTA controller, implemented in Python, while network traffic is processed through two Ethernet ports bound to bmv2. The gateway also provides wireless connectivity via Wi-Fi 6 and Bluetooth 5.3. Sensor data stream generation is performed on a laptop equipped with an Intel Core i7 165U processor @ 2.10 GHz and 16 GB of RAM. This machine runs a Scapy-based [54] simulator, crafting and transmitting sensor packets periodically to the gateway. Monitoring is facilitated by a Dell server with an Intel Core i7-9700 processor clocked @ 3.0 GHz and 16 GB of RAM. It runs a Python script which receives alerts from the gateway and displays them on-screen for analysis and action if necessary. ML model training and evaluation are conducted using Scikit-Learn [70] and XGBoost [71] Python libraries.

Table III: Model configurations

Prediction	Model	Estimators	Depth	Max Leaf Nodes
Sepsis	Offline	6	–	–
	VISTA	4	5	500
Heart Failure	Offline	4	–	–
	VISTA	2	5	500

We use the synthetic dataset described in Section V-A for our evaluation. In experiments requiring more patients or data samples, such as the scalability evaluation, we leverage a Python script to loop through existing data samples.

B. Performance Metrics

Two types of performance metrics are considered, covering ML inference performance and system performance.

1) ML inference performance: We evaluate the in-network ML model performance using two key metrics: **accuracy** and **F1 score**, previously defined in Section V-B1.

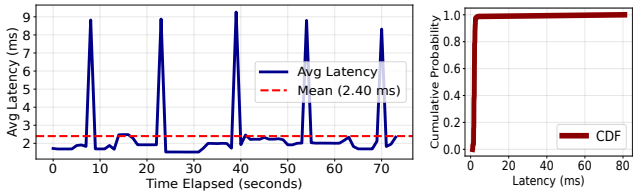
2) Latency: We measure the packet processing delay within the gateway. This is the time taken by VISTA to process aggregated features and make an inference decision. We measure this time by taking the difference between when the packet got into the ingress pipeline of the data plane and the time when a result packet got to the egress pipeline. This enables the quantification of any additional delay on the gateway's packet processing induced by VISTA.

3) Resource usage: We assess the system's efficiency by monitoring resource consumption, focusing on: **CPU usage**, which is the percentage of CPU resources used by VISTA, and **memory usage**, measuring the amount of RAM consumed. These metrics reflect the computational load on the hardware. Both measurements were taken on the gateway using the *pidstat* tool [72], with the `-u` and `-r` flags.

C. ML Inference Performance

We use the validation data to measure classification accuracy. Complete sets of per-patient sensor packets with no missing data are transmitted to the gateway, enabling a direct comparison of inference performance between in-network and offline models. The gateway receives the packets, parses them, aggregates the features progressively, computes the NEWS2 score, and makes ML predictions with VISTA. Offline models were trained and evaluated on a server using grid search over 1–100 estimators while keeping all other parameters at their default XGBoost settings. The in-network VISTA variants executed directly within the IoMT gateway's data plane using compact models optimized for real-time inference by Planter [49]. All models use 10 features, with the other configurations summarized in Table III.

The results of the ML performance are summarized in Table IV. VISTA's inference performance closely matches that of the offline server-based XGBoost models. Accuracy differs by less than 0.001 for both prediction tasks: 0.9303 vs. 0.9300 for sepsis and 0.9586 vs. 0.9579 for heart failure. Similarly, the macro-F1 scores vary by less than 0.002, with 0.6815 vs. 0.6809 for sepsis and 0.8223 vs. 0.8204 for heart failure, with weighted F1 scores following the same trend. The minor discrepancies arise from the training and encoding of



(a) (b)

Figure 10: Time series and cumulative distribution function of latency

the trained models with Planter for data plane execution. These results indicate that deploying inference within the gateway's data plane preserves model fidelity despite its strict constraints.

Table IV: Summary of ML Inference Performance

Prediction	Metric	Offline	VISTA
Sepsis	Accuracy	0.9303	0.9300
	F1 Score (Macro)	0.6815	0.6809
	F1 Score (Weighted)	0.9107	0.9105
Heart Failure	Accuracy	0.9586	0.9579
	F1 Score (Macro)	0.8223	0.8204
	F1 Score (Weighted)	0.9521	0.9515

D. System Performance Evaluation

We conduct experiments to validate the ability of VISTA to enable scalable real-time inference within the edge gateway.

1) Latency: The test involved generating packets with the full feature set (all sensors, demographic, and context variables), representing complete windows where all features arrive as expected, and injecting them into the gateway. These complete windows were processed entirely in the data plane, generating immediate alerts. In total, we sent over 17,500 packets from 183 different patients and measured each packet's processing time after the alert was generated.

The latency test results are summarized in Table V. On average, it takes the gateway just about 2.4 ms to parse the packets, make an ML prediction, compute the NEWS2 score, and generate an alert, with almost 75% of packets processed within 2 ms. The standard deviation is 5 ms, arising from the peaks that can be observed in Figure 10a, which are separated by 15 seconds after which heartbeat packets are sent from the controller to clear stale state for all patients, causing a momentary surge in the number of packets in the gateway. These peaks can be reduced by spreading heartbeat packets over the tunable 15-second window. Also, the 99th percentile is only 19.42 ms, which still outperforms existing edge or cloud-based solutions as shown in Table VI. The millisecond-level latency for complete windows enables real-time monitoring, suitable for timely clinical interventions in remote health monitoring settings.

Table V: Latency Summary Statistics. P25 = 25th Percentile, P75 = 75th Percentile, STDEV = Standard Deviation

	P25	Mean	P75	STDEV
Latency (ms)	1.689	2.401	2.030	5.119

The steep Cumulative Distribution Function (CDF) shown in Figure 10b highlights the system's consistency under ideal conditions on the Dell IoT Edge Gateway. While 99% of packets are processed within 20 ms, a small long tail reaching up to 80 ms appears due to the infrequent controller heartbeat packet injections and transient queueing in the software switch. These results demonstrate VISTA's suitability for real-world deployment on IoT gateways, balancing low-latency performance with robust handling of incomplete data, thus enhancing patient safety and reducing alert latency.

Comparison to the state-of-the art. Table VI summarizes latency values reported in related remote patient-monitoring systems. Rahmani *et al.* [28] report 21–33 ms for fog-based and 161–176 ms for cloud-based deployments using Wi-Fi and BLE sensors, representing the full sense-to-actuation path that includes wireless sensor transmission, gateway processing, and cloud communication. They also observed gateway-to-cloud transfer delays of about 100–200 ms under different Wi-Fi conditions. Similarly, Alsahfi *et al.* [12] measure average network delays of 163 ms for regional and 363 ms for cloud configurations, also including end-to-end wireless and network delays. In contrast, VISTA achieves an average packet-processing latency of 2.4 ms within the IoMT gateway. Because our evaluation uses simulated, wired sensor inputs rather than real wireless sensors, we report only the data-plane processing and inference latency, excluding network transmission delays. Although the measurement scope differs, isolating gateway processing highlights the efficiency of in-network ML execution, providing a foundation for low-latency clinical inference once integrated with live sensor networks.

Table VI: Latency measurements from related studies and VISTA

Work	Deployment	Scope	Latency (ms)
[28]	Fog (Wi-Fi)	End-to-end	21
	Fog (BLE)	End-to-end	33
	Cloud (Wi-Fi)	End-to-end	161
	Cloud (BLE)	End-to-end	176
[12]	Regional Server (Wi-Fi)	End-to-end	163
	Cloud (Wi-Fi)	End-to-end	363
VISTA	**Wired simulator → Gateway**	**Within gateway**	**2.4**

2) Resource usage: We monitor the CPU and memory usage of the `simple_switch` process running VISTA, which handles core packet processing and inference. We use the same settings as the latency experiments.

CPU Utilization. As VISTA runs in a Docker container on the gateway, **only one CPU core** is used. Statistics on the CPU usage are presented in Table VII. In steady functional mode, the usage pattern shows a mean CPU utilization of 9.2% during sustained operation, with peaks of up to 30%, as shown in Figure 11. This represents a clear periodic behavior, with the peaks corresponding to when windows open and sensors start sending data, or when the controller injects per-patient heartbeat packets, and low cycles corresponding to quiet times before window expiry. The 95th percentile utilization of 10.00 % in stable conditions indicates that the system operates well within acceptable bounds for edge deployment, maintaining sufficient headroom for handling traffic spikes while ensuring real-time responsiveness for critical healthcare alerts. We also experiment with bursts of traffic, sending multi-patient data, starting from 100 patients to 10,000 patients. This results in periodic hikes, reaching up to 74.00% during intensive processing phases with high patient numbers.

Metric	Value
CPU Core Utilization	
Mean CPU Core Usage	9.20 %
Peak CPU Core Usage	30.00 %
95th Percentile	10.00 %
Standard Deviation	5.56 %
Memory Utilization	
Resident Set Size (RSS)	580.26 MB
Memory Stability	Near Constant

Table VII: Use of gateway resources

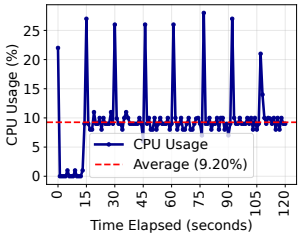
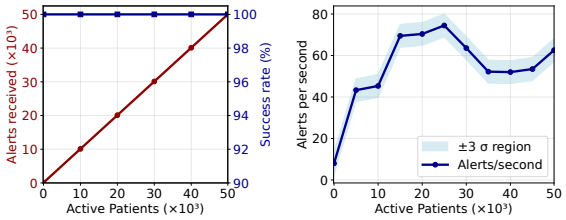


Figure 11: CPU usage over time



(a) (b)

Figure 12: Alert rate and reliability under scaling load

Memory Usage. The system demonstrates exceptional memory stability with a constant resident set size (RSS) of 580.26 MB throughout the test period, as shown in Table VII. The constant value is possibly due to the gateway assigning a fixed memory to the process once it is started. Register memory was allocated for 50,000 patients during these tests, implying that VISTA requires just 11KB per patient, demonstrating VISTA's ability to scale without overly consuming switch memory. This consistent memory footprint is crucial for edge deployment scenarios, like in the IoT gateway, where memory resources are limited.

3) Response to missing data: We conduct tests to validate VISTA's behavior when dealing with incomplete data, and the reliability of its timeout mechanism. The test involves systematically evaluating incomplete data scenarios with critical feature sets (vital signs only), random missing data patterns, and minimal data cases (*e.g.*, only 1–3 features).

We observe a deterministic binary threshold behavior: complete feature sets (10 features) trigger direct inference, averaging 2.4 ms response time, while any incomplete set consistently activates the timeout mechanism with processing times tightly bounded between 62–72s, due to the 60s timeout, which is tunable. This demonstrates the robust reliability of the controller heartbeat and timeout mechanism.

Furthermore, the ML model degrades predictably with data completeness. When most features are available (7–10 features), model performance remains high, and NEWS2 scores closely match the ground truth. With partial input (4–6 features), the model selectively relies on critical sensors to sustain reasonable predictions. Severe data loss (1–3 features) leads to degraded accuracy, as most features must be imputed, which may cause deviations from the actual patient condition.

4) Scalability: We evaluate VISTA's ability to handle multiple patient sensor streams simultaneously, a key requirement for scaling to large-scale monitoring scenarios. We begin from a single patient and progressively increase the number of active patient data streams to 50,000. We measure alert generation rate and success rate, calculated as Success Rate = (Patients Generating Alerts / Total Patients) x 100%, reflecting the proportion of patients triggering alerts.

VISTA maintains a perfect success rate: 100% for all tests, up to 50,000 patients, as shown in Figure 12a. As expected, the number of alerts received grows linearly with the number of active patients, as a notification is generated upon inference in the gateway. The number of alerts per second increases with load, peaking at 74.48 alerts/second at 25,000 patients, as illustrated in Figure 12b, before levelling off. This non-linear trend likely reflects the cumulative effects of runtime overheads and resource contention at high load, where the system reaches a steady processing capacity rather than continuing to scale linearly. Overall, these results demonstrate that VISTA is robust and reliable even when handling large volumes of sensor data streams on a single edge gateway.

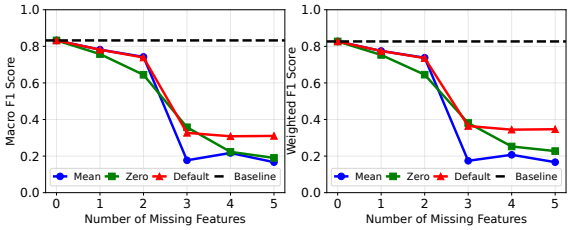
5) Reduction in communication overhead: We quantify the reduction in network data overhead achieved by performing inference directly at the gateway with VISTA by comparing two scenarios: (i) sending raw sensor data to a remote processing node, and (ii) performing inference locally and sending only alerts. Each sensor and alert packet is encapsulated in an Ethernet frame. Due to the minimum Ethernet frame size constraint of 64 bytes, each frame, regardless of the actual payload size, occupies at least 64 bytes on the wire. Without edge processing, ten sensor packets generate a minimum of $10 \times 64 = 640$ bytes of network traffic. In contrast, local inference at the gateway allows sending just one alert packet, resulting in only 64 bytes transmitted. This corresponds to a data overhead reduction of $\frac{640-64}{640} \times 100 = 90\%$. Such reduction illustrates the efficiency gains of edge-based inference with VISTA in bandwidth-constrained or latency-sensitive remote monitoring scenarios.

E. Evaluation on Real-World Clinical Data

Using the Maternal Health Risk dataset described in Section V-A, we evaluate the practical applicability of VISTA under real clinical data distributions. As with the other use cases based on synthetic data, we train (offline) and encode a tree-based model, which we deploy in the gateway to predict maternal health risk. We employed an 85%–15% train–test split on the dataset, using five features to train a decision tree with the maximum number of leaf nodes set to 100 and the maximum tree depth set to 12. We then employed the sensor simulator described in Section VI-A to generate and send sensor packets to the gateway for analysis.

1) Model performance: Our experiments show that the model achieved an overall accuracy of 0.82, with macro- and weighted F1-scores of approximately 0.82 and 0.81, respectively. The system demonstrates particularly strong performance in identifying high-risk cases (F1 score of 0.93), while maintaining reliable detection of low-risk cases (F1 score of 0.81). Performance on mid-risk cases is comparatively lower (F1 score of 0.71), reflecting the inherent overlap between adjacent clinical risk levels, increasing classification difficulty.

Since the VISTA architecture and deployment pipeline remain unchanged, system-level performance metrics such as latency, resource usage, and communication overhead are consistent with the results reported in Section VI-D. Overall,



(a) Macro F1 (b) Weighted F1

Figure 13: Performance of VISTA with imputed missing features

these results confirm that VISTA maintains reliable and stable inference performance when evaluated on real-world data.

2) Response to missing data: We assess the behavior of the imputation mechanism under incomplete inputs by progressively removing physiological features and evaluating the resulting prediction performance. Features are removed cumulatively, such that each configuration contains the previously removed features plus one additional missing feature, varying the number of missing inputs from one to five. The Age feature is always retained, as it is readily available from the patient record. We compare three imputation strategies: (i) mean imputation using training-set averages, (ii) zero substitution, and (iii) clinically informed default-value imputation, where missing features are replaced with medically reasonable reference values (120/80 mmHg for blood pressure, 5 mmol/L for blood sugar, 98° F for body temperature, and 72 bpm for heart rate). Because the dataset contains only a single record per patient, patient-specific historical averaging cannot be applied. Figures 13a and 13b summarize the results.

Performance degrades as more features are removed, though not strictly in proportion to their number but to their clinical importance. A pronounced drop occurs when three features are missing, corresponding to the removal of Blood Sugar (BS) in addition to Systolic BP and Heart Rate. As BS is a key indicator for maternal risk stratification, its absence substantially alters the model's decision boundaries. In this setting, mean imputation performs poorly, as replacing BS with a global average suppresses its discriminative signal, whereas zero and default-value substitution mitigate the impact to some extent. As additional features are removed and the model increasingly relies on Age alone, performance declines across all strategies. Overall, clinically informed default-value imputation provides the most stable and consistent behavior under high missingness, and is therefore adopted as the deployment strategy in VISTA.

VII. Discussion

Evaluation results show that VISTA enables real-time patient monitoring via in-network ML inference on IoMT gateways, operating reliably even without cloud connectivity. Local processing reduces network load and latency, providing a cost-efficient and responsive edge solution. These findings motivate a discussion of several key considerations.

Technical Novelty. VISTA introduces a new use case for in-network computing by enabling sensor data aggregation and ML inference for patient monitoring directly within the data plane. Unlike prior in-network ML systems focused on network analytics or security, VISTA applies these capabilities to continuous physiological sensing, for real-time health inference at the edge. From a healthcare perspective, it represents a new architectural approach that leverages existing network infrastructure to deliver low-latency and privacy-preserving patient monitoring close to where the data is generated. This cross-domain design bridges programmable networking and IoMT-based healthcare.

Scalability. VISTA scales efficiently from single-patient monitoring to large multi-patient deployments. Experimental results show reliable operation for thousands of sensor streams on one gateway with consistent alert generation and low latency. In practice, multiple gateways can be deployed across care homes or hospitals, operating independently or in coordination with cloud services for data backup and broader coverage. This hierarchical scaling preserves local responsiveness and privacy while extending monitoring capacity across distributed healthcare environments.

Data Privacy. VISTA enhances patient privacy by executing feature aggregation and ML inference locally within the IoMT gateway, eliminating the need to transmit raw physiological data to external servers. This design limits outbound communication to high-level alerts, reducing the risk of sensitive data exposure and supporting compliance with healthcare privacy frameworks such as GDPR [18] and HIPAA [19]. By following the principles of data minimization and locality, VISTA preserves confidentiality while maintaining real-time monitoring capabilities. Processing data within on-premise IoMT gateways ultimately strengthens both system security and patient trust in connected healthcare environments.

Resilience and Reliability. VISTA delivers reliable patient monitoring by operating entirely within the IoMT gateway, ensuring that critical analytics and alerting remain available even during network or cloud outages. Unlike cloud-dependent healthcare platforms that may be affected by large-scale service disruptions such as recent AWS and Microsoft outages that rendered large portions of the Internet unreachable [20], VISTA operates autonomously without external dependencies, except in cases where the provider is distant. This edge-centric design guarantees continuity of monitoring and alert generation in care environments, preventing the loss of vital observations that could jeopardize patient safety. By maintaining full local functionality during connectivity failures, VISTA strengthens the resilience of digital healthcare infrastructure and supports trusted, always-on patient monitoring.

Cost and Ease of Deployment. VISTA runs on commercially available IoT gateways combined with the open-source bmv2 software switch, lowering the barrier to entry for in-network ML deployment. This design removes the need for specialized hardware or constant cloud connectivity, avoiding the recurring costs of cloud analytics and improving reliability in constrained environments. By leveraging existing network infrastructure, VISTA can be deployed incrementally in diverse care settings, including low- and middle-income regions where affordable, autonomous monitoring solutions are most needed.

ML Model Choice. We use XGBoost and decision tree classifiers for predicting patient condition due to their balance between accuracy and computational efficiency. Other models, *e.g.*, deep neural networks, can also be deployed in

programmable data planes for potentially higher accuracy [46], [47]. However, there is an inherent trade-off between model accuracy and data-plane overheads. More sophisticated models may yield higher accuracy but increase memory and processing costs. For IoMT gateway deployments, lightweight tree-based models, *e.g.*, XGBoost and decision trees provide an effective balance for real-time, resource-constrained settings.

Latency. We computed the packet processing delay within the gateway, representing the time required to parse incoming data, perform ML inference, compute the NEWS2 score, and generate an alert. For complete feature sets, the gateway achieved an average latency of 2.4 ms, with almost 75% of packets processed within 2 ms. When feature sets were incomplete, the controller triggered the timeout mechanism, and alerts were generated after 60–72 seconds for a 60-second timeout value, which is tunable. Although the evaluation does not include sensor-to-gateway transmission delays, these are constant across similar systems and do not affect the measured inference time. The low millisecond-level latency confirms VISTA's suitability for real-time patient monitoring.

System Performance. VISTA demonstrates efficient and stable operation on the IoT gateway across multiple performance dimensions. CPU utilization remains low, averaging 9.2% with peaks of up to 30%, well within acceptable limits for edge deployment. Memory usage is highly stable, with a resident set size of about 580 MB while supporting up to 50,000 registered patients, equivalent to roughly 11 KB of state per patient. This consistent resource footprint shows that VISTA can scale without exceeding gateway capacity. Local inference also reduces network traffic by about 90% compared to transmitting raw sensor data to the cloud, highlighting its communication efficiency. Together, these results confirm that VISTA achieves robust performance on a commercial gateway while maintaining low computational and memory overhead.

Architectural Constraints and Practicality. VISTA runs on a Dell IoT Gateway using the v1model software data-plane architecture. Although this software target does not impose the strict match-action stage limits of ASIC-based switches such as Tofino, execution remains fundamentally constrained by the P4 programming model. As a result, inference must be expressed as acyclic, table-driven logic with fixed execution paths and preallocated state. Under these constraints, VISTA supports up to 50,000 concurrent patients using approximately 580 MB of memory, corresponding to about 11 KB per patient. The system is therefore well-suited to structured, low-dimensional clinical tasks such as early-warning scoring and physiological risk prediction, but is not intended for workloads requiring deep neural networks, high-dimensional signal processing, or image-based analytics, which exceed the practical expressiveness of P4-based data-plane execution.

Within these architectural limits, model complexity depends on the number of features, tree depth, and estimators, since each decision node increases match-action table entries. Using Planter [49], whose scalability has been evaluated in [73], tree-based models can scale to tens of numerical features, with decision tree and random forest models supporting up to about 59 features and XGBoost up to 55. Increasing tree depth significantly expands table entries and reduces the number of trees that can fit; for up to 6 features, ensembles of up to 20 trees are feasible, while deeper trees constrain ensemble size under fixed memory constraints. Our deployed models operate within these limits as shown in Table III, with a packet-processing latency floor of approximately 1.7 ms, and an accuracy ceiling determined by the equivalent offline model. Kernel-bypass targets (*e.g.*, T4P4S [74] or eBPF-based compilers [75]) could further reduce latency by eliminating networking overhead, but introduce trade-offs in compiler maturity, portability, and supported P4 features.

Limitations and Future Work. VISTA is a proof of concept demonstrating the feasibility of in-network ML inference for patient monitoring on IoMT gateways. Although both synthetic and real-world clinical data have been used for evaluation, sensor inputs are currently simulated rather than originating from live IoMT sensors. This allows controlled testing without hardware and wireless variability. Future work will extend VISTA to fully operational sensor deployments to evaluate long-term stability and real-world communication dynamics. The current prototype runs the software-based P4 switch (bmv2) inside a Docker container on the IoT gateway.

While VISTA is primarily designed for IoMT gateways where latency and cost efficiency are paramount, evaluating it on hardware P4 targets such as Tofino switches would help validate feasibility beyond the software switch prototype and explore extensions at aggregation points or clinical network backbones where programmable switches could enhance large-scale data handling. Another direction is to integrate collaborative and adaptive learning mechanisms, such as federated learning, allowing multiple gateways to update shared models while maintaining data privacy. Finally, implementing automated model updates and monitoring pipelines will enable continuous adaptation to evolving data distributions, improving long-term performance in dynamic environments.

VIII. Conclusion

We proposed VISTA, an edge computing framework for real-time patient monitoring on IoMT gateways. The key contributions of VISTA include a novel asynchronous, multi-sensor data plane aggregation module and a P4-based in-network ML module for real-time patient condition inference. Prototyped on a Dell Edge Gateway, VISTA processes data from wearable and ambient sensors at variable sampling rates, integrating with clinical guidelines to generate early warning alerts for NEWS2, heart failure, and sepsis. Experimental evaluation demonstrated how VISTA can conduct ML inference within 2.4 ms on average in the gateway, reducing communication overhead by up to 90%, and enabling privacy-preserving remote patient monitoring. By embedding intelligence in the IoMT network fabric, VISTA enables a new generation of decentralized healthcare systems that enhance chronic disease management and strengthen global health resilience.

Ethics Statement: This study uses only publicly available datasets, processes no personal information, and complies with our institution's guidelines for human-subjects research.

REFERENCES

[1] J. McCool et al. Mobile health (mhealth) in low- and middle-income countries. *Annual Review of Public Health*, 43:525–539, 2022.

[2] NHS England. Virtual wards. https://www.england.nhs.uk/virtual-wards/, 2023. Accessed: 24 May 2025.

[3] P. He et al. A survey of internet of medical things: technology, application and future directions. *Digit. Commun. Netw. (DCN)*, 2024.

[4] A. Balasundaram et al. Internet of things (IoT)-based smart healthcare system for efficient diagnostics of health parameters of patients in emergency care. *IEEE Internet Things J.*, 10(21):18563–18570, 2023.

[5] A. Rocha et al. Edge AI for internet of medical things: A literature review. *Comput. Electr. Eng.*, 116:109202, 2024.

[6] Y. A. Qadri et al. The future of healthcare Internet of Things: A survey of emerging technologies. *IEEE Commun. Surveys Tuts.*, 22(2):1121–1167, 2020.

[7] H. Habibzadeh et al. A survey of healthcare Internet of Things (HIoT): A clinical perspective. *IEEE Internet Things J.*, 7(1):53–71, 2020.

[8] L. Greco et al. Trends in IoT based solutions for health care: Moving AI to the edge. *Pattern recognition letters*, 135:346–353, 7 2020.

[9] D. Koutras et al. Security in IoMT communications: A survey. *Sensors (Basel)*, 20(17):4828, 2020.

[10] N. Nigar et al. IoMT meets machine learning: From edge to cloud chronic diseases diagnosis system. *J. Healthc. Eng.*, 2023:9995292, 2023.

[11] G. Yang et al. IoT-based remote pain monitoring system: From device to cloud platform. *IEEE J. Biomed. Health Inform.*, 22(6):1711–1719, 2018.

[12] T. Alsahfi et al. Optimizing healthcare big data performance through regional computing. *Scientific Reports*, 15(1):3129, 1 2025.

[13] P. Bosshart et al. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, jul 2014.

[14] E. F. Kfoury et al. An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends. *IEEE Access*, 9:87094–87155, 2021.

[15] C. Zheng et al. In-network machine learning using programmable network devices: A survey. *IEEE Commun. Surveys Tuts.*, 26(2):1171–1200, April 2024.

[16] M. Zang et al. Toward continuous threat defense: in-network traffic analysis for IoT gateways. *IEEE Internet Things J.*, 11(6):9244–9257, 2024.

[17] M. Zang et al. Federated in-network machine learning for privacy-preserving IoT traffic analysis. *ACM Trans. Internet Technol.*, 29, 2024.

[18] European Parliament and Council of the European Union. Council regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data. https://www.legislation.gov.uk/eur/2016/679, April 2016.

[19] U.S. Congress. Health insurance portability and accountability act of 1996. https://aspe.hhs.gov/reports/health-insurance-portability-accountability-act-1996, August 1996.

[20] S. Wang et al. Measuring the consolidation of DNS and web hosting providers. arXiv:2110.15345 [cs.NI], 2024.

[21] Royal College of Physicians. National early warning score (NEWS) 2: Standardising the assessment of acute-illness severity in the NHS. Royal College of Physicians, London, UK, 2017.

[22] D. Bell et al. A trend-based early warning score can be implemented in a hospital electronic medical record to effectively predict inpatient deterioration. *Critical Care Medicine*, 49(10):e961–e967, 2021.

[23] K. Dharmarajan et al. Diagnoses and timing of 30-day readmissions after hospitalization for heart failure, acute myocardial infarction, or pneumonia. *JAMA*, 309(4):355–363, 2013.

[24] C. W. Yancy et al. 2017 ACC/AHA/HFSA focused update of the 2013 ACCF/AHA guideline for the management of heart failure. *Circulation*, 136(6):e137–161, 2017.

[25] M. Singer et al. The third international consensus definitions for sepsis and septic shock (sepsis-3). *JAMA*, 315(8):801–810, 2016.

[26] M. Cecconi et al. Sepsis and septic shock. *Lancet*, 392(10141):75–87, 2018.

[27] P. Verma and S. K. Sood. Fog assisted-IoT enabled patient health monitoring in smart homes. *IEEE Internet Things J.*, 5(3):1789–1796, 2018.

[28] A. M. Rahmani et al. Exploiting smart e-health gateways at the edge of healthcare internet-of-things: A fog computing approach. *Future Generation Computer Systems*, 78:641–658, 2018.

[29] R. K. Pathinarupothi et al. IoT-based smart edge for global health: Remote monitoring with severity detection and alerts transmission. *IEEE Internet Things J.*, 6(2):2449–2462, 2019.

[30] S. Khan et al. Digital-twins-based internet of robotic things for remote health monitoring of covid-19 patients. *IEEE Internet Things J.*, 10(18):16087–16098, 2023.

[31] K.-Y. Huang et al. Enhancing healthcare AI stability with edge computing and machine learning for extubation prediction. *Scientific Reports*, 15(1):17858, May 2025.

[32] M. Eskandari et al. Passban IDS: An intelligent anomaly-based intrusion detection system for IoT edge devices. *IEEE Internet Things J.*, 7(8):6882–6897, 2020.

[33] H. Wang et al. Non-IID data re-balancing at IoT edge with peer-to-peer federated learning for anomaly detection. In *Proc. ACM WiSec*, pp. 153–163. ACM, 2021.

[34] O. Shahid et al. Detecting network attacks using federated learning for IoT devices. In *Proc. IEEE ICNP*, pp. 1–6, 2021.

[35] N. Feamster et al. The road to SDN: an intellectual history of programmable networks. *SIGCOMM Comput. Commun. Rev.*, 44(2):87–98, April 2014.

[36] P4.org. P4Runtime Specification. https://p4.org/wp-content/uploads/sites/53/p4-spec/p4runtime/v1.2.0/P4Runtime-Spec.pdf, 2019.

[37] P4 Language Consortium. Behavioral Model v2 (bmv2): The Reference P4 Software Switch. https://github.com/p4lang/behavioral-model, 2025.

[38] Tofino Switch. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html. Accessed: 03-10-2025.

[39] Netronome Agilio SmartNICs. https://netronome.com/agilio-smartnics/. Accessed: 03-10-2025.

[40] R. Parizotto et al. Offloading machine learning to programmable data planes: A systematic survey. *ACM Comput. Surv.*, jun 2023.

[41] A. Sapio et al. In-network computation is a dumb idea whose time has come. In *In Proc. of ACM HotNets*, NY, USA, 2017. ACM.

[42] C. Lao et al. ATP: In-network aggregation for multi-tenant learning. In *In Proc. NSDI*, pp. 741–761, April 2021.

[43] A. Sapio et al. Scaling distributed machine learning with In-Network aggregation. In *In Proc. NSDI*, pp. 785–808, 2021.

[44] K. He et al. Measuring control plane latency in SDN-enabled switches. In *Proc. SOSR*, NY, USA, 2015. ACM.

[45] D. Sanvito et al. Can the network be the AI accelerator? In *Proc. NetCompute*, pp. 20–25, NY, USA, 2018. ACM.

[46] M. Saquetti et al. Toward in-network intelligence: Running distributed artificial neural networks in the data plane. *IEEE Communications Letters*, 25(11):3551–3555, 2021.

[47] Z. Zhao et al. RIDS: Towards advanced IDS via RNN model and programmable switches co-designed approaches. In *Proc. IEEE INFOCOM*, pp. 591–600, 2024.

[48] A. T.-J. Akem et al. Flowrest: Practical flow-level inference in programmable switches with random forests. In *Proc. IEEE INFOCOM*, pp. 1–10, 2023.

[49] C. Zheng et al. Planter: Rapid prototyping of in-network machine learning inference. *SIGCOMM Comput. Commun. Rev.*, 54(1):2–20, 2024.

[50] B. M. Xavier et al. MAP4: A pragmatic framework for in-network machine learning traffic classification. *IEEE Trans. Netw. Serv. Manag.*, 19(4):4176–4188, 2022.

[51] G. Zhou et al. An efficient design of intelligent network data plane. In *Proc. USENIX Security Symp.*, 2023.

[52] G. Xie et al. Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation. In *Proc. IEEE INFOCOM*, pp. 1938–1947, 2022.

[53] H. Yan et al. Linc: Enabling low-resource in-network classification and incremental model update. In *Proc. IEEE ICNP*, pp. 1–12, 2024.

[54] P. Biondi. Scapy: Interactive packet manipulation program. https://scapy.net/, 2025. Accessed on April 28, 2025.

[55] W.-C. Lin and C.-F. Tsai. Missing value imputation: a review and analysis of the literature (2006–2017). *Artificial Intelligence Review*, 53(2):1487–1509, Feb 2020.

[56] L. O. Joel et al. A comparative study of imputation techniques for missing values in healthcare diagnostic datasets. *International Journal of Data Science and Analytics*, 20(7):6357–6373, Nov 2025.

[57] M. Liu et al. Handling missing values in healthcare data: A systematic review of deep learning-based imputation techniques. *Artificial Intelligence in Medicine*, 142:102587, 2023.

[58] Z. Chen et al. Missing values and imputation in healthcare data: Can interpretable machine learning help? *Proc. of Machine Learning Research*, 209:2023, 2023.

[59] M. Ahmed. Maternal health risk [dataset]. UCI Machine Learning Repository: https://archive.ics.uci.edu/dataset/863/maternal+health+risk, 2023. Accessed: 2026-02-12.

[60] M. Ahmed et al. Review and analysis of risk factor of maternal health in remote area using the internet of things (IoT). *Lecture Notes in Electrical Engineering*, 2020.

[61] M. Ahmed and M. A. Kashem. IoT based risk level prediction model for maternal health care in the context of bangladesh. In *2020 2nd Int. Conf. on Sustainable Technologies for Industry 4.0 (STI)*, pp. 1–6, 2020.

[62] A. K. Malde et al. A machine learning approach for predicting maternal health risks in lower-middle-income countries using sparse data and vital signs. *Future Internet*, 17:190, 2025.

[63] N. Sehgal et al. Synthetic data generation with GenAI for privacy-preserving smart healthcare IoT systems. *Journal of Data Analysis and Critical Management*, 1:58–67, 8 2025.

[64] T. W. Jones et al. Sepsis with preexisting heart failure: Management of confounding clinical features. *Journal of Intensive Care Medicine*, 36(9):989–1012, 2021.

[65] A. Jalilian et al. Length of stay and economic sustainability of virtual ward care in a medium-sized hospital of the UK: a retrospective longitudinal study. *BMJ Open*, 14(1), 2024.

[66] NHS England. Frailty virtual ward (hospital at home for those living with frailty). https://www.england.nhs.uk/wp-content/uploads/2021/12/B1207-ii-guidance-note-frailty-virtual-ward.pdf, 2021.

[67] National Institute for Health and Care Excellence. NICE Guidelines. https://www.nice.org.uk/about/what-we-do/our-programmes/nice-guidance/nice-guidelines, 2025. Accessed: 2025-06-29.

[68] C. Zheng et al. Planter: In-network machine learning. https://github.com/In-Network-Machine-Learning/Planter, 2024.

[69] Dell Technologies Edge Gateway Spec Sheet. https://www.delltechnologies.com/asset/en-us/solutions/business-solutions/technical-support/dell-technologies-edge-gateway-spec-sheet.pdf, 2023.

[70] F. Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[71] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proc. ACM KDD*, pp. 785–794. ACM, 2016.

[72] Sébastien Godard. pidstat (part of sysstat utilities). https://github.com/sysstat/sysstat. Accessed on June 2, 2025.

[73] C. Zheng et al. Iisy: Hybrid in-network classification using programmable switches. *IEEE/ACM Trans. Netw.*, 32(3):2555–2570, 2024.

[74] P. Vörös et al. T4P4S: A target-independent compiler for protocol-independent packet processors. In *Proc. IEEE HPSR*, pp. 1–8, 2018.

[75] M. Simon et al. Honey for the ice bear - dynamic ebpf in p4. In *Proc. ACM eBPF*, pp. 44–50, NY, USA, 2024. ACM.