CHALMERS
UNIVERSITY OF TECHNOLOGY

Interoperable Medical Device Analytics in Microservice Architectures

Development of a Hybrid Electrochemical and Gaussian Process Regression Framework for Lithium Carbon Monofluoride Battery Prognostics

Master's Thesis in Biomedical Engineering

DAVID ERIKMATS

RONI CELIKER

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2026

www.chalmers.se

Master's thesis 2026

Interoperable Medical Device Analytics in Microservice Architectures

Development of a Hybrid Electrochemical and Gaussian Process Regression Framework for Lithium Carbon Monofluoride Battery Prognostics

DAVID ERIKMATS
RONI CELIKER

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical engineering
Division of Signal Processing and Biomedical Engineering
Care@Distance - Remote and prehospital Digital Health
Chalmers University of Technology
Gothenburg, Sweden 2026

Interoperable Medical Device Analytics in Microservice Architectures
Development of a Hybrid Electrochemical and Gaussian Process Regression Framework for Lithium Carbon Monofluoride Battery Prognostics
DAVID ERIKMATS
RONI CELIKER

Supervisor: Mattias Seth, Department of Electrical Engineering
Supervisor: Johanna Stenqvist, Go North Medical AB
Examiner: Stefan Candefjord, Department of Electrical Engineering

Master's Thesis 2026
Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Care@Distance - Remote and prehospital Digital Health
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Interoperable Medical Device Analytics in Microservice Architectures
Development of a Hybrid Electrochemical and Gaussian Process Regression Framework for Lithium Carbon Monofluoride Battery Prognostics
DAVID ERIKMATS
RONI CELIKER
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Reliable battery prognostics for active implantable medical devices (IMDs), such as cardiac pacemakers and neurostimulators, are critical for safeguarding patient health and avoiding premature or delayed surgical replacements. However, primary lithium carbon monofluoride (Li/CFx) cells exhibit an exceptionally flat discharge plateau where the voltage gradient is minimal, making state-of-charge (SoC) estimation notoriously difficult. Furthermore, clinical telemetry data collected during routine hospital follow-ups is inherently sparse and irregular, causing traditional mechanistic tracking models to accumulate drift and pure data-driven estimators to fail.

To address these challenges, this thesis proposes an interoperable software architecture and hybrid prognostic framework deployed as a decoupled, stateless microservice. Exposing asynchronous RESTful endpoints via FastAPI, the service integrates Pydantic data models for schema validation at the network boundary and utilizes HL7 FHIR standards to support interoperable data exchange across clinical hospital systems. The core analytical engine couples a physical second-order equivalent circuit model (ECM) prior with a Gaussian Process Regression (GPR) statistical corrector to dynamically compensate for parameter mismatch and patient-specific load profiles. Programmatic thermodynamic guardrails are implemented to strictly enforce monotonic capacity depletion.

A containerized simulation environment orchestrated via Docker Compose was developed to validate the system. Across a retrospective cohort of $n = 87$ active devices, the bounded hybrid observer achieved a Mean Absolute Error (MAE) of 1.346% and a Root Mean Squared Error (RMSE) of 5.654%, outperforming mechanistic (4.611%) and data-driven (6.370%) baselines. In addition, automated stress-testing verified that the centralized exception boundary safely isolated 100% of injected network and data anomalies. Latency benchmarks demonstrated processing execution scaling from 0.382 s ($N = 1$) to 2.411 s ($N = 1000$) on a local container mesh. This demonstrates that the microservice architecture successfully balances robust software engineering with high-accuracy algorithmic execution, underscoring its potential utility as an analytical foundation for safety-critical clinical monitoring environments pending future in-vivo and clinical validation.

Keywords: Battery Prognostics, Gaussian Process Regression, Equivalent Circuit Model, State of Charge, Implantable Medical Devices, HL7 FHIR, Microservice Architecture, Hybrid Modeling.

Acknowledgements

This Master's thesis was conducted within the Division of Signal Processing and Biomedical Engineering, Department of Electrical Engineering at Chalmers University of Technology, in collaboration with Go North Medical. First and foremost, we would like to express our deepest gratitude to our supervisor, Johanna Stenqvist. Throughout this entire project, she has consistently been there to support us, immediately answering our questions and guiding us in the right direction to produce a high-quality academic thesis.
We also want to extend our sincere appreciation to our supervisor, Mattias Seth. His insights, deep expertise in biomedical services and software architecture design, and valuable guidance during key decision-making checkpoints throughout the project were instrumental in shaping our technical methodology.
Furthermore, we are highly grateful to our examiner, Stefan Candefjord, for his broader-scale guidance, constructive reviews, and continuous support, which helped us align our work with the highest standards of scientific research.
Finally, we would like to thank our families, friends, and peers for their continuous encouragement, support, and patience throughout this journey.

David Erikmats & Roni Celiker, Gothenburg, June 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

API	Application Programming Interface
CQRS	Command Query Responsibility Segregation
ECM	Equivalent Circuit Model
EHDS	European Health Data Space
EHR	Electronic Health Record
EOS	End of Service
ERI	Elective Replacement Indicator
FHIR	Fast Healthcare Interoperability Resources
GDPR	General Data Protection Regulation
GPR	Gaussian Process Regression
HIPAA	Health Insurance Portability and Accountability Act
HL7	Health Level Seven
IMD	Implantable Medical Device
LOINC	Logical Observation Identifiers Names and Codes
MAE	Mean Absolute Error
MDC	Medical Device Communication
MSA	Microservice Architecture
PCHIP	Piecewise Cubic Hermite Interpolating Polynomial
PICP	Prediction Interval Coverage Probability
RBF	Radial Basis Function
RMSE	Root Mean Squared Error
RUL	Remaining Useful Life
SEI	Solid Electrolyte Interphase
SoC	State of Charge
SoH	State of Health
SOMDA	Service-Oriented Medical Device Architecture
UCUM	Unified Code for Units of Measure
VNS	Vagus Nerve Stimulator
ZOH	Zero-Order Hold

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices and Sets

k	Sequence index for clinical telemetry sessions
t	Continuous time variable (seconds or days)
$\mathcal{D}$	Full database of patient telemetric histories
$\mathcal{O}_k$	Raw clinical database observation tuple at index k
$\mathcal{R}$	Set of reconstructed hybrid state trajectories

Parameters and Constants

Q_n	Nominal rated capacity of the battery cell (Ah)
R_0	Ohmic internal series resistance of the Equivalent Circuit Model (Ω)
R_1, R_2	Charge transfer and polarization resistances inside the ECM (Ω)
C_1, C_2	Electrochemical double-layer and polarization capacitances inside the ECM (F)
$I_{\text{quiescent}}$	Continuous baseline current drain of the implant circuitry (A)
τ_1, τ_2	Polarization time constants of the RC branches ($\tau_i = R_i C_i$) (s)
σ_n^2	Variance of the additive Gaussian measurement noise in the GPR model
$\boldsymbol{\mu}_x$	Vector of training feature means used for Z-score normalization
$\boldsymbol{\Lambda}$	Diagonal scaling matrix composed of feature standard deviations
σ_{SoC}	Standard deviation of the mechanistic state-of-charge feature
σ_I	Standard deviation of the average current drain feature
σ_Z	Standard deviation of the lead impedance feature

σ_{Age}	Standard deviation of the operational device age feature

Variables

$\text{SoC}(t)$	State of Charge of the battery at time t
SoC_{ecm}	State of Charge calculated by the mechanistic prior
SoC_{hybrid}	Enforced non-increasing hybrid State of Charge estimate
U_{oc}	Open circuit voltage of the electrochemical cell (V)
$U_{p,1}, U_{p,2}$	Transient polarization voltages across the RC branches (V)
U_{meas}	Measured terminal voltage under active device load (V)
I_{load}	Stimulation output pulse current amplitude (A or mA)
I_{avg}	Computed continuous baseline current drain (A)
PW	Stimulation pulse width (μs)
Z_{lead}	Stabilized electrical lead path impedance (Ω)
$\mathbf{x}_k$	Raw four-dimensional feature vector at visit k
$\mathbf{z}_k$	Standardized four-dimensional feature vector at visit k
μ_{gpr}	Posterior mean prediction of the residual GPR corrector
σ^2_{gpr}	Posterior epistemic variance of the residual GPR corrector
μ_{device}	Patient-specific calibration bias offset
$\widehat{\text{SoC}}$	Linearly interpolated capacity value used for validation

Contents

List of Figures

List of Tables

1

Introduction

Healthcare systems are undergoing a rapid digital transformation driven by the increasing availability of Electronic Health Records (EHR), wearable sensor streams, and other forms of Real-World Data (RWD) [1], [2]. As these data sources expand in scale and heterogeneity, modern clinical infrastructures must support secure, interoperable, and semantically consistent data exchange across organizational and technological boundaries [3], [4]. This shift is reinforced by emerging regulatory frameworks such as the European Health Data Space (EHDS), which mandates standardized mechanisms for the secondary use of health data in research, analytics, and clinical decision support [5]. Consequently, manufacturers and healthcare providers face increasing pressure to develop backend systems that are modular, interoperable, and engineered for long-term maintainability [6], [7].

A central challenge in this landscape is the extreme heterogeneity of biomedical data. Clinical systems must integrate structured measurements, unstructured notes, imaging data, and device telemetry—often originating from proprietary ecosystems with inconsistent data models [8]. Traditional monolithic architectures struggle to accommodate this diversity, as tightly coupled designs inhibit scalability, hinder interoperability, and complicate the integration of advanced analytical components [9], [10]. These limitations become particularly evident in the management of implantable medical devices (IMDs), where telemetry is sparse, irregular, and constrained by strict power budgets [11]. Analytical services, in the context of IMDs, should therefore operate in an *on-demand* fashion: activated only when new device data becomes available, performing their computation, and returning to an idle state without maintaining persistent connections [12].

The criticality of these analytical services is most evident in the management of the device's power source, which represents the ultimate bottleneck for IMD longevity [11]. For a biomedical engineer, battery life is not merely a technical specification but a primary determinant of patient safety and clinical intervention; while premature device replacement subjects the patient to unnecessary invasive surgery, an unexpected power failure can lead to a complete loss of life-critical therapy [13]. Estimating this Remaining Useful Life (RUL) is fundamentally difficult because the internal state of a battery cannot be measured directly through sensors and must instead be inferred from external signals like voltage and current [14], [15]. In safety-critical medical applications, this estimation requires a shift from simple voltage monitoring to complex physics-informed modeling.

This challenge is especially pronounced for IMDs powered by Lithium/Carbon Monofluoride (Li/CF_x) batteries. Manufacturers select Li/CF_x chemistry for its high energy density and low self-discharge rates, which enable the miniaturization of implants [13], [16]. However, these benefits come at the cost of "weak observability": Li/CF_x cells exhibit an exceptionally flat discharge plateau where terminal voltage remains nearly constant for up to 90% of the battery's life, followed by a sudden, steep drop as the cell reaches depletion [16], [17]. Furthermore, these cells are prone to "voltage delay"—a transient voltage dip after idle periods—which a naive system might misinterpret as an end-of-life signal, triggering a false clinical alarm [13], [16]. Additionally, the discharge process creates Lithium Fluoride, an electrical insulator that monotonically increases internal resistance, making the battery's behavior non-linear and state-dependent [16].

To address these challenges, this research develops a decoupled, microservice-oriented analytical backend that integrates a hybrid Equivalent-Circuit-Model–Gaussian Process Regression (ECM–GPR) framework for Li/CF_x battery RUL estimation. By separating analytical computation from ndata storage, orchestration, and security concerns, the system achieves modular scalability, robust fault isolation, and compliance with emerging interoperability standards such as HL7 FHIR [18], [19]. The analytical service serves as a case study demonstrating how lightweight, containerized microservices can support specialized biomedical analytics within the broader regulatory biomedical landscape.

1.1 Background

The development of interoperable and analytically capable biomedical backends requires a comprehensive understanding of how medical data exchange, software architectures, and implantable device integration have evolved over time. This background section outlines the historical progression from early, local device communication standards to modern cloud-native, microservice-based infrastructures. It highlights the architectural paradigms that shape contemporary clinical software and reviews the resilience and interoperability principles that underpin safety-critical analytical services. Crucially, this section also examines the mathematical and physical paradigms of battery state estimation, contextualizing the challenges of tracking the non-linear degradation of primary Lithium/Carbon Monofluoride (Li/CF_x) cells. Fusing software engineering and electrochemical modeling, these elements provide the conceptual foundation for the decoupled backend architecture and the hybrid battery estimation framework developed in this thesis.

Active implantable medical devices (IMDs), such as cardiac pacemakers, implantable cardioverter-defibrillators (ICDs), and vagus nerve stimulators (VNS), serve as life-sustaining therapies for millions of patients worldwide. Because these devices are surgically implanted inside the patient's body, their internal primary batteries cannot simply be recharged or swapped via external access. When a cell approaches depletion, the entire pulse generator must be surgically explanted and replaced. This creates a critical clinical dilemma: replacing a device prematurely wastes substantial battery capacity and subjects the patient to unnecessary surgical and infection risks,

whereas waiting too long risks sudden loss of therapy with potentially fatal consequences. Accurately tracking battery depletion and forecasting Remaining Useful Life (RUL) from sparse telemetric health checks is therefore paramount. However, bridging clinical device telemetry with advanced prognostic algorithms requires both mathematically sound state estimation and modern, secure, and interoperable digital health architectures.

1.1.1 The Evolution of Biomedical Software and Interoperability

Historically, medical software was designed using monolithic patterns, bundling data access, user interface, and analytical computation into a single deployable unit. This monolithic design offered clear advantages in early deployments, such as low network latency, straightforward local debugging, and simplified deployment schemes. However, as medical devices migrated out of isolated environments, the drawbacks of monoliths became apparent: tight coupling prevented independent module scaling, complicated updates to statistical runtimes, and introduced single points of failure where a crash in one computational subsystem could disable the entire clinical interface [10]. To address this, early networked medical environments turned to standardization. Frameworks such as IEEE 11073 established foundational domain models for point-of-care devices [20], which evolved into the Service-Oriented Medical Device Architecture (SOMDA) within the OR.NET project. Utilizing the Medical Devices Profile for Web Services (MDPWS), these early Service-Oriented Architectures (SOA) successfully established a secure "safety context" for plug-and-play device networking in local operating rooms [21], [22]. However, SOA often introduced centralized communication bottlenecks and heavy governance overhead.

This architectural landscape was further pressured by the rapid growth of distributed patient data and resource-constrained IoT healthcare devices (such as implantables and wearables), which transmit telemetry intermittently via protocols like MQTT or CoAP [11]. To handle these heterogeneous, on-demand workloads, software engineering transitioned toward Microservices Architecture (MSA). By decomposing the backend into fine-grained, stateless, and independently deployable containers, MSA allows computationally demanding analytical tasks to be isolated from core databases and security gateways [9], [10].

Crucially, these architectural shifts were not merely driven by technological convenience, but by the necessity of satisfying a rapidly emerging regulatory landscape, which directly governs the safety, trust, and accessibility of digital healthcare. Under the European Medical Device Regulation (MDR), software used in clinical diagnostics is held to strict safety and validation standards [23]. Decoupling the analytical engine into an independent microservice has a profound societal benefit: it allows clinical engineers to validate and deploy software updates for specific algorithms rapidly without undergoing full-system re-certification, ensuring patients receive safer, updated care without operational delays. Similarly, the European Health Data Space (EHDS) mandates patient data portability to eliminate proprietary vendor lock-in, enabling patients to transfer their historical medical device telemetry

seamlessly across healthcare providers and national borders [5]. Technically enabled by HL7 FHIR standardization, this portability democratizes access to specialized diagnostics and promotes cross-border health equity [18], [19]. Finally, regulations like the European AI Act classify safety-critical diagnostic algorithms as high-risk, demanding high transparency [24]. Building explainable, uncertainty-aware architectures within these frameworks ensures clinical trust: providing physicians with calibrated confidence bounds directly prevents both premature, risky replacement surgeries and catastrophic, unexpected device depletions.

1.1.2 Methodological Paradigms in Battery State Estimation

The deployment of battery analytical services within biomedical frameworks requires a deep understanding of the mathematical paradigms governing battery prognostics. Within implantable medical devices (IMDs), predicting remaining operational timelines is a safety-critical necessity [25]. To contextualize these paradigms, this section examines the clinical requirements of implantable power sources, outlines the classical modeling taxonomy, and highlights the implementation gap that motivates the development of containerized hybrid state observers.

Active implantable medical devices—such as cardiac pacemakers, implantable cardioverter-defibrillators (ICDs), and deep brain neurostimulators—are legally and clinically classified as Class III medical devices due to their role as life-support systems [25]. The clinical efficacy of these systems is fundamentally coupled to the reliability and continuous operation of their internal cells. In practice, a significant discrepancy frequently exists between pre-calculated battery service longevity projections and actual real-world operational lifespans [25]. This operational variance is heavily driven by the highly non-linear nature of patient-specific parameters. Specifically, the rate of capacity degradation is directly tied to the volume and frequency of therapy provided by the implant [25]. Because pacing thresholds, electrode-tissue interfaces, and individual patient conditions evolve dynamically over decadal horizons, static battery life estimations introduce risks of premature depletion. Consequently, modern healthcare infrastructures demand adaptive health-monitoring architectures capable of processing personalized telemetry logs to provide early, risk-calibrated replacement notifications [25].

To translate raw telemetric current and voltage measurements into reliable prognosis indicators, state tracking frameworks utilize formal mathematical abstracts. Battery health parameters are broadly divided into snapshot instant indicators, such as State of Charge (SoC), and dynamic degradation indicators, including State of Health (SoH) and Remaining Useful Life (RUL). As established in literature, the models used as inputs for these tracking algorithms are classified into four distinct structural taxa [26]:

1. Electrical Equivalent Circuit Models (ECMs): Also referred to as Randles or Thevenin abstractions, ECMs construct an arrangement of electrical components to mimic battery terminal behaviors. These systems incorporate an ideal voltage source representing the Open Circuit Voltage (U_{oc}) as a function of the SoC, coupled in series with an internal ohmic resistance (R_Ω) and one or more parallel resistor-capacitor ($R_p - C_p$) pairs. The RC networks mathematically isolate the transient polarization effects and diffusion voltage drops that surface under dynamic load current fluctuations [26].

2. Electrochemical Models: These frameworks apply advanced physical principles to model the actual chemical transport phenomena occurring inside the cell. They utilize systems of non-linear partial differential equations (PDEs) to continuously map solid-electrolyte interface (SEI) growth, phase-change boundaries, anode/cathode surface areas, and ion concentration gradients [26].

3. Mathematical and Empirical Lifecycle Models: These models deploy regression equations, phenomenological curve-fitting profiles, or statistical reliability distributions (such as the Wiener or Gamma processes) to map capacity degradation directly as a function of operational age, cumulative milliampere-hours drawn, or long-term storage configurations [26].

4. Pure Data-Driven Paradigms: These non-parametric architectures utilize statistical learning algorithms—such as Support Vector Regression, Relevance Vector Machines, and Gaussian Process Regression—to build mapping profiles from historical telemetry directly to the target health index. By manipulating multi-dimensional kernel metrics, they operate without explicit mechanical or structural parameter initialization [26]. To address the domain mismatch when moving from laboratory datasets to field operation, recent studies have successfully employed transductive transfer learning methods, such as kernel mean matching, to adapt SOH regression models to real-world applications without requiring direct field labels [27].

Selecting an optimal tracking configuration from this taxonomy introduces a critical engineering trade-off between mathematical precision and computational feasibility. Comprehensive reviews indicate that pure data-driven and machine learning estimators achieve excellent tracking precision, frequently keeping error boundaries below 1% to 2% [26]. This high precision is achieved by using kernel mapping transformations and matrix transposition routines to capture highly non-linear degradation trends without requiring explicit internal parameters. However, a severe implementation gap persists within the biomedical domain, particularly when managing primary Lithium Carbon Monofluoride (Li/CF_x) chemistries. The Li/CF_x cell exhibits an exceptionally flat thermodynamic voltage plateau across approximately 90% of its discharge life, introducing a severe weak observability constraint where small sensor noise translates into massive state estimation errors. While advanced machine learning approaches can theoretically map these subtle variations, they exhibit severe computational complexity ($O(N^3)$ matrix inversions) and depend on resource-intensive exponential operations [26]. Consequently, they are almost exclusively validated in abstract simulation environments using pre-processed laboratory datasets and are rarely deployed inside production healthcare networks or resource-constrained embedded medical architectures [25], [26].

To resolve this implementation bottleneck while maintaining tracking precision under uncertain clinical conditions, contemporary research emphasizes the development of integrated hybrid frameworks [25]. Hybrid state observers merge the structural benefits of adaptive filtering with data-driven statistical processes. By routing a lightweight mechanistic circuit model as a physical state prior, the framework enforces physical boundary constraints. Concurrently, an optimized machine learning corrector layer tracks the remaining residual model error, allowing the combined architecture to capture localized patient tracking variations without experiencing mathematical divergence. Fusing these hybrid algorithms with modern software engineering paradigms provides a scalable solution to the deployment problem. Packaging the unified analytical logic as an independent, containerized microservice decouples the heavy matrix execution overhead from both the physical medical

hardware and the upstream clinical system components. This architecture supports continuous delivery and isolation, allowing the complex estimation parameters to be optimized independently in the cloud while delivering on-demand, real-time diagnostic insight directly to clinical web networks.

1.2 Aim

The aim of this research is to develop a modular and interoperable biomedical system architecture capable of managing end-to-end data pipelines for implantable medical devices within the modern healthcare landscape. By engineering a generalized backend infrastructure that prioritizes syntactic and semantic interoperability, this work seeks to facilitate the integration of analytical modules into standardized health data ecosystems. Furthermore, this research aims to develop and implement a hybrid mechanistic-probabilistic framework for Li/CFx primary battery life estimation, aiming to demonstrate the architecture's capability to process non-linear, chaotic degradation signals and overcome clinical data scarcity through physics-informed modeling.

1.3 Research questions

This research aims to answer the following questions:

1. How can a decoupled backend infrastructure be engineered to facilitate the processing of heterogeneous medical device data while ensuring adaptability through modular, service-oriented design patterns?
2. To what extent can a hybrid modelling approach—integrating mechanistic Equivalent Circuit Models (ECM) with Gaussian Process Regression (GPR)—be utilized to accurately determine Remaining Useful Life (RUL) estimation for Li/CF_x implantable batteries?
3. What integration and validation strategies should be used for ensuring that a containerized analytical module (battery estimation framework) provides functionally robust and interoperable interactions in a simulated biomedical environment?

1.4 Limitations

This thesis focuses on modeling and developing an estimation algorithm based on historical clinical logs; consequently, direct physical testing and experimental characterization of the implantable medical hardware are excluded from the scope of this work.

The scope of this project is strictly limited to the backend software architecture and API development. The following aspects are explicitly excluded:

- **System-Level Interface Constraints:** The scope of this thesis excludes the development of consumer-facing user interfaces. System validation and interaction are performed exclusively from a backend perspective using automated API requests, focusing development resources on the core analytical engine and interoperability layers.
- **Hardware Development:** The work focuses entirely on software implementation; no hardware prototyping firmware modifications or destructive tests to the implantable medical devices will be performed.
- **Clinical Diagnostics:** While the system is validated using a de-identified clinical telemetry database, the project is technical in nature. It does not involve live clinical trials, real-time telemetry tracking for immediate treatment, or medical diagnosis. The focus is centered on the computational infrastructure required to process such data, rather than the direct clinical application of the prognosis.
- **Model Search Space and Project Timeline Constraints:** Due to the academic time constraints of the Master's project, the algorithmic investigation focused on a targeted set of model candidates (specifically a second-order Equivalent Circuit Model coupled with Gaussian Process Regression). Exhaustive benchmarking against alternative machine learning paradigms (such as Recurrent Neural Networks, Transformer architectures, or full electrochemical pseudo-two-dimensional models) was outside the achievable timeline of this thesis.

2

Theory

This chapter establishes the theoretical foundations for the hybrid battery estimation framework and its containerized software architecture. It begins by outlining the existing paradigms in state-of-charge (SoC) estimation, detailing the electrochemical characteristics of primary Lithium/Carbon Monofluoride (Li/CF_x) battery cells. It then presents the mathematical formulation of the second-order Equivalent Circuit Model (ECM) that serves as the deterministic physical prior. Next, the chapter details the mathematical framework of Gaussian Process Regression (GPR), which provides non-parametric residual correction and calibrated uncertainty estimation. Finally, it introduces the core architectural principles of microservices, system resilience, and containerization that govern the cloud-native deployment of safety-critical biomedical services.

2.1 Battery estimation paradigms

Accurate State of Charge (SoC) estimation is a fundamental requirement for the reliable operation of battery-powered systems [13], [14], [15]. Traditionally, these estimations have relied on mechanistic (white-box) models [13]. These models offer high theoretical predictability and physical insight; however, they necessitate a significant number of internal parameters, many of which can only be derived through sophisticated equipment and destructive testing [13].

As the field has evolved, data-driven (black-box) models—specifically machine learning—have gained prominence [13], [14], [15]. These methods establish complex non-linear relationships between observable variables using empirical data [13], [15]. While these models are powerful, data-driven approaches require vast amounts of high-quality, heterogeneous data to prevent overfitting and ensure accuracy across all operating conditions [13], [14]. Furthermore, their lack of physical transparency often complicates validation and limits reproducibility in safety-critical applications [13].

In response to these limitations, hybrid modeling frameworks have emerged [13], [14]. By combining the physical consistency of mechanistic models with the flexibility of machine learning, hybrid approaches improve both modeling accuracy and robustness [13], [14]. In such a framework, the physics-based model often acts as a feature extractor or a "prior" that guides the machine learning algorithm toward physically plausible trends [13], [14].

2.2 Core Principles of SoC Estimation

The State of Charge (SoC) represents the available capacity in a battery relative to its maximum capacity [14], [15]. It is formally defined as:

$$SoC(t) = \frac{Q_{remaining}(t)}{Q_n} \tag{2.1}$$

where $Q_{remaining}(t)$ denotes the battery's remaining charge at time t, and Qn represents the nominal rated capacity.

Because SoC cannot be directly measured via physical sensors, it must be inferred through the observation of external parameters such as terminal voltage, load current, and surface temperature [14], [15]. The relationship between these variables is highly non-linear and time-varying, particularly in Li/CFx chemistries where the discharge plateau is exceptionally flat [16], [17]. Consequently, a robust mathematical model is required to map these observable external signals to the internal chemical state of the battery [13].

2.2.1 Electrochemical Characteristics of Li/CFx Chemistry

A defining characteristic of the Li/CFx battery is its discharge profile, which exhibits a distinct voltage plateau. As illustrated in Figure 2.1(a), the terminal voltage remains relatively constant for approximately 80–90% of the battery's total capacity, followed by a steep drop as the cell approaches depletion [16], [17]. This profile poses a significant challenge for state estimation; since the derivative of voltage with respect to capacity (dU/dQ) is approximately zero during the plateau, the SoC becomes weakly observable from voltage measurements alone [13], [17].

Furthermore, Li/CFx cells are subject to a phenomenon known as "voltage delay." When a cell has remained idle for an extended period and a load is subsequently applied, the terminal voltage may drop significantly below the nominal operating level before gradually recovering [17]. This transient anomaly, illustrated in Figure 2.1(b), is particularly critical in medical applications. If not correctly modeled, this temporary dip can be misidentified as the onset of the End-of-Life (EOL) phase, potentially leading to premature device replacement or clinical false alarms [13].

Lastly, the discharge process in Li/CFx cells involves a chemical reaction that produces Lithium Fluoride (LiF). As LiF is an electrical insulator, its accumulation within the cathode structure—often forming a "shell" around the discharge products—leads to a monotonic increase in internal resistance over the battery's lifecycle [16]. Consequently, the parameters of an ECM for this chemistry cannot be treated as static constants; they must be identified as time-varying or state-dependent variables to maintain estimation accuracy [13], [16].

2.3 Equivalent Circuit Modeling

As stated in Chapter 1, the battery can be represented using an Equivalent Circuit Model (ECM) to provide a structured physical foundation for the modeling of SoC.

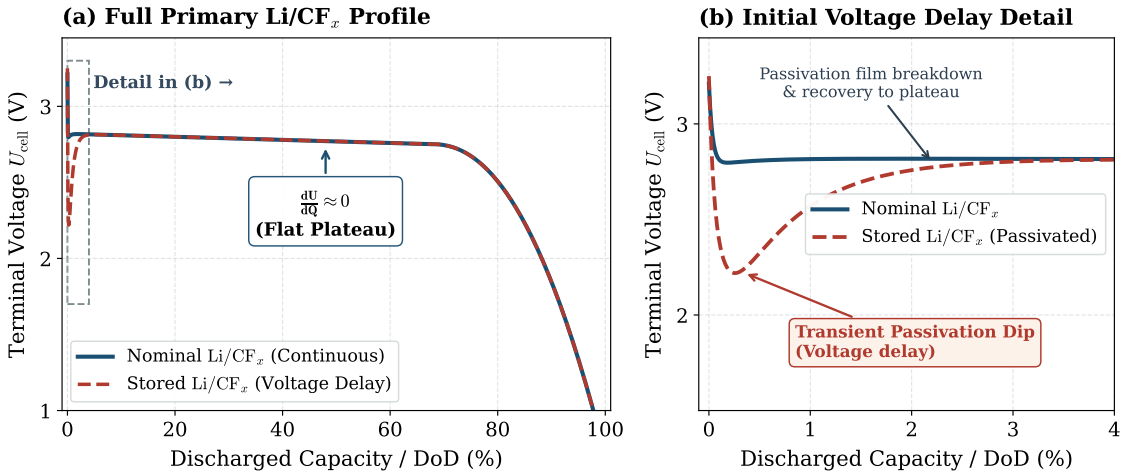


Figure 2.1: Discharge characteristics of a primary Lithium/Carbon Monofluoride (Li/CF_x) cell under load. (a) Terminal voltage profile showing the prolonged thermodynamic plateau spanning approximately 80–90% of the operating life, followed by a steep end-of-life depletion knee where dU/dQ drops sharply. (b) The initial transient "voltage delay" phenomenon caused by cathode passivation impedance after extended idle storage prior to breakdown and voltage stabilization.

Rather than functioning as a purely empirical black-box representation, the ECM serves as a reduced-order model where individual electrical components correspond directly to physical electrochemical processes within the cell. The second-order Thevenin model is frequently used due to its ability to accurately characterize the dynamic voltage response of Li/CF_x cells under pulsed loads, balancing mathematical complexity with predictive fidelity [13], [28].

2.3.1 Model and Physical Significance

The second-order Thevenin model consists of an ideal voltage source representing the Open Circuit Voltage U_{oc}, a series resistor (R_0) representing the internal ohmic resistance, and two parallel Resistor-Capacitor (RC) networks as shown in Figure 2.2 [13], [28].

Each component in the Thevenin model network corresponds to a specific electrochemical process:

- **Ohmic Resistance (R_0):** Represents the instantaneous voltage drop caused by the resistance of the electrolyte, current collectors, and separator [13], [28].
- **First RC Branch (R_1, C_1):** Captures the fast dynamics associated with the double-layer capacitance and charge transfer resistance at the electrode-electrolyte interface [13], [28].
- **Second RC Branch (R_2, C_2):** Accounts for the slow dynamics related to concentration polarization and mass transport (diffusion) effects within the active materials [13], [28].

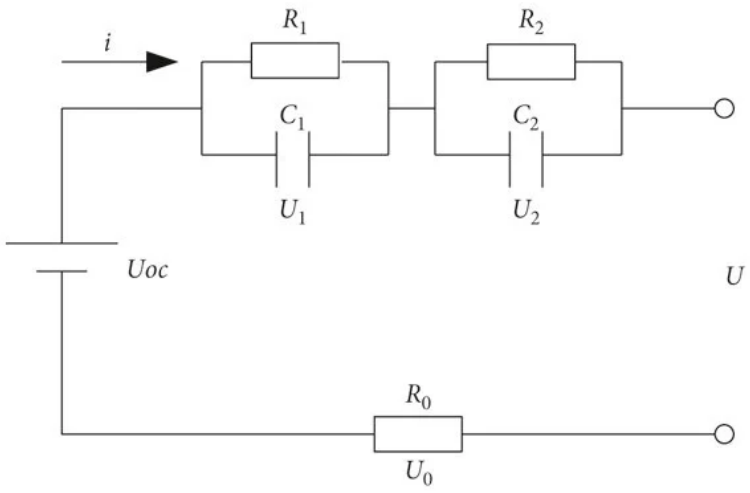


Figure 2.2: Second-order Thevenin equivalent circuit model for a battery cell.

The terminal voltage U_t is determined by the summation of the potential drops across the internal components. Following Kirchhoff's Voltage Law (KVL), the output equation is defined as [28]:

$$U_t(t) = U_{oc}(SoC) - I(t)R_0 - U_{p,1}(t) - U_{p,2}(t) \tag{2.2}$$

where $I(t)$ is the load current and $U_{p,i}$ is the polarization voltage across the i-th RC branch. The temporal evolution of the polarization voltages is governed by the following first-order differential equations [28]:

$$\frac{dU_{p,i}(t)}{dt} = -\frac{1}{R_i C_i} U_{p,i}(t) + \frac{1}{C_i} I(t), \quad i \in \{1, 2\} \tag{2.3}$$

By defining the time constants as $\tau_i = R_i C_i$, these equations describe how the internal overpotentials build up during a pulse and decay during periods of inactivity [28].

2.3.2 Discrete State-Space Formulation for Irregular Timestamps

A significant challenge in medical device data is the irregular sampling interval, where the duration between measurements, $\Delta t_k = t_{k+1} - t_k$, is non-constant. To implement the ECM in a digital framework, the equations are discretized using a Zero-Order Hold (ZOH) assumption on the current I_k over the interval Δt_k.

Following the formulation in Yi et al., the discrete-time update for the polarization voltages is expressed as [28]:

$$U_{p,1,k+1} = a_k U_{p,1,k} + c_k I_k \tag{2.4}$$

$$U_{p,2,k+1} = b_k U_{p,2,k} + d_k I_k \tag{2.5}$$

where the time-variant coefficients a_k, b_k, c_k, and d_k are calculated at each step k:

$$\begin{aligned} a_k &= \exp\left(-\frac{\Delta t_k}{R_1 C_1}\right), \quad c_k = R_1(1-a_k) \\ b_k &= \exp\left(-\frac{\Delta t_k}{R_2 C_2}\right), \quad d_k = R_2(1-b_k) \end{aligned} \tag{2.6}$$

This allow the system to be represented in a compact state-space form. Defining the state vector as $x_k = [U_{p,1,k}, U_{p,2,k}, SoC_k]^T$, the system is described by [28]:

$$x_{k+1} = A_k x_k + B_k I_k \tag{2.7}$$

$$U_{t,k} = U_{oc}(SoC_k) - R_0 I_k - C_{sys} x_k \tag{2.8}$$

where A_k and B_k are updated dynamically based on Δt_k:

$$A_k = \begin{bmatrix} a_k & 0 & 0 \\ 0 & b_k & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad B_k = \begin{bmatrix} c_k \\ d_k \\ -\frac{\Delta t_k}{Q_n} \end{bmatrix}, \quad C_{sys} = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \tag{2.9}$$

2.3.3 Parameter Identification Theory

The accuracy of the ECM depends on the precise identification of R_0, R_i, and C_i. Theoretically, these are determined through pulse discharge tests and relaxation analysis [28]. When current is removed ($I = 0$), the terminal voltage exhibits a characteristic recovery curve:

$$U_{recovery}(t) = U_{oc} - U_{p,1}(t_0)e^{-t/\tau_1} - U_{p,2}(t_0)e^{-t/\tau_2} \tag{2.10}$$

where t_0 is the time of interruption. By applying non-linear least squares fitting to this curve, the physical parameters can be isolated [28].

2.4 Gaussian Process Regression

The ECM serves as a mechanistic prior for the machine learning algorithm [13]. While capturing the fundamental physics of §Li/CF_x discharge, it deviates from reality due to aging and unmodeled effects [13], [16].In this hybrid structure, Gaussian Process Regression (GPR) is tasked with modeling the error, ensuring the final estimation remains constrained by electrochemical laws while accounting for complex non-linearities [13], [14].

Gaussian Process Regression (GPR) is a non-parametric Bayesian method for function approximation and uncertainty quantification [29], [30]. GPR defines a distribution over functions rather than estimating a single deterministic mapping, allowing both predictions and associated uncertainties to be inferred directly from the model structure [31], [32]. This capability is particularly valuable in battery degradation modeling, where nonlinearity, measurement noise, and cell-to-cell variability introduce significant uncertainty [33], [34], [35].

Beyond its practical advantages, GPR is grounded in a rigorous probabilistic framework in which the covariance function encodes prior beliefs about the smoothness and structure of the latent function [29], [36]. Because the posterior combines this prior with observed data through Bayes' theorem, GPR naturally provides calibrated uncertainty estimates that distinguish between epistemic uncertainty due to limited data and uncertainty arising from measurement noise, as visualized across the conditioning stages in Figure 2.3 [30], [32]. This probabilistic formulation is particularly advantageous in sparse telemetry settings, where reliable uncertainty quantification is essential for downstream clinical decision-making [37].

A Gaussian Process (GP) is formally defined as a collection of random variables, any finite number of which have a joint multivariate Gaussian distribution [29]. A GP is fully specified by its mean function $m(x)$ and covariance function (or kernel) $k(x, x')$:

$$f(x) \sim \mathcal{GP}\left(m(x), k(x, x')\right) \tag{2.11}$$

The prior mean function $m(x)$ represents the expected value of the function before observing data, and the covariance function $k(x, x')$ defines the correlation between the function values at different input points [29], [32].

Additionally the mean function $m(x)$ encodes prior beliefs about the global trend of the latent function. When the training data is normalized, the mean is often assumed to be zero ($m(x) = 0$), relying on the covariance function to model local variations [29], [31], [32]. However, when modelling physical degradation processes such as battery capacity fade, the function exhibits a long-term monotonic downward trend [33].

2.4.1 Mathematical Formulation of GPR

Let $f(x)$ denote the latent function we wish to infer from noisy observations y, defined as:

$$y = f(x) + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma_n^2) \tag{2.12}$$

where σ_n^2 represents the variance of the additive Gaussian measurement noise [29], [30], [31]. In battery applications, this noise term captures effects such as voltage measurement variability, load fluctuations, and other disturbances [34], [35].

Given a set of training inputs $X = [x_1, \ldots, x_n]^\top$ and corresponding observations $\mathbf{y} = [y_1, \ldots, y_n]^\top$, the joint prior distribution over the training outputs $\mathbf{y}$ and the function values f_* at a set of test inputs X_* is given by [29], [37]:

$$\begin{bmatrix} \mathbf{y} \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(X) \\ m(X_*) \end{bmatrix}, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix} \right) \tag{2.13}$$

where $K(X, X)$ is the covariance matrix evaluated over all pairs of training inputs, $K(X, X_*)$ is the cross-covariance matrix between training and test inputs, and I

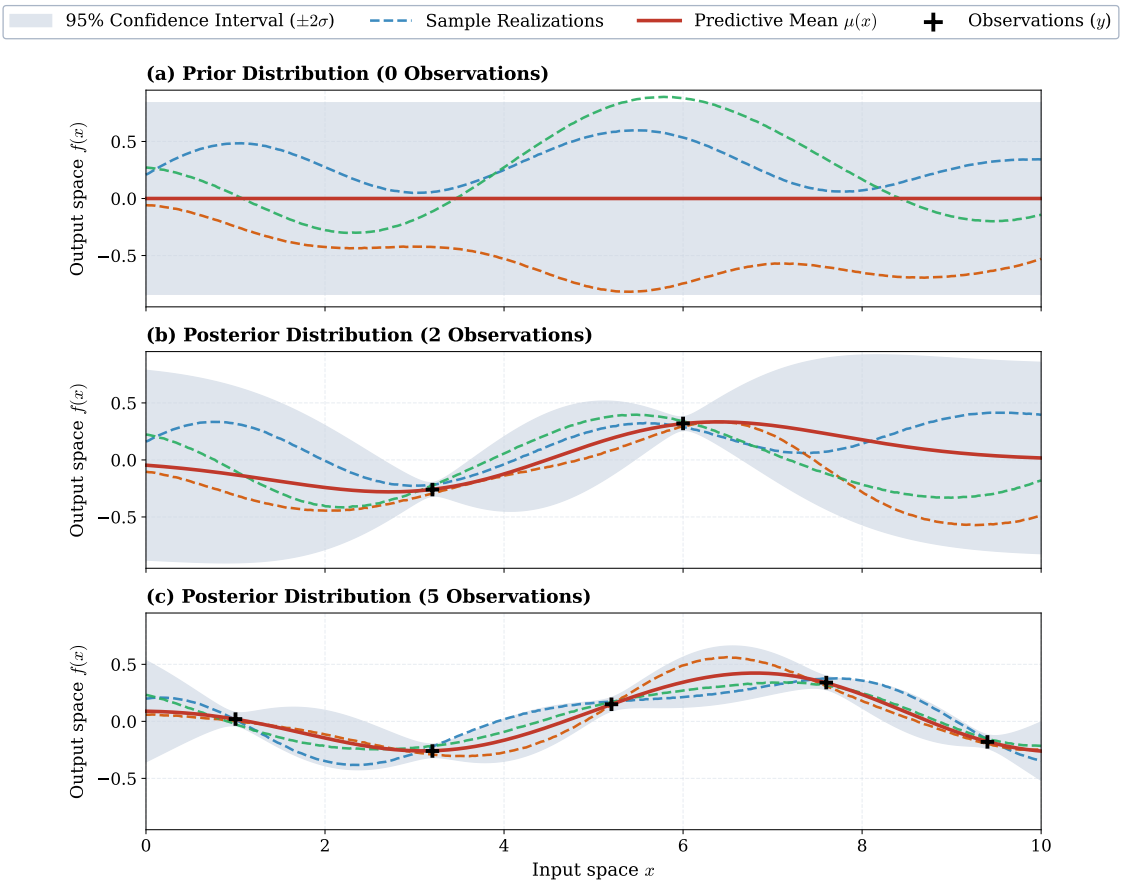


Figure 2.3: Sequential conditioning in Gaussian Process Regression (GPR) using an RBF covariance function: (a) zero-mean prior distribution with unconstrained epistemic uncertainty bands ($\pm 2\sigma$), (b) posterior predictive distribution conditioned on two discrete observations showing localized variance reduction, and (c) posterior distribution updated with five observations demonstrating progressive epistemic uncertainty contraction along the domain.

is the identity matrix [29]. Conditioning this joint Gaussian distribution on the observed training data yields the analytical posterior predictive distribution at the test inputs X_*. The predictive mean μ_* and predictive covariance Σ_* are expressed as [29], [32]:

$$\mu_* = m(X_*) + K(X_*, X)\left(K(X, X) + \sigma_n^2 I\right)^{-1}(\mathbf{y} - m(X)) \tag{2.14}$$

$$\Sigma_* = K(X_*, X_*) - K(X_*, X)\left(K(X, X) + \sigma_n^2 I\right)^{-1} K(X, X_*) \tag{2.15}$$

The diagonal elements of Σ_* represent the predictive variance at the test points, providing a measure of confidence. As the distance between a test point and the training data increases, the predictive variance expands, reflecting the epistemic uncertainty of the model [31]. The posterior mean and variance are obtained analytically, allowing GPR to provide calibrated uncertainty estimates[30], [32], [33], [35].

To perform inference, the hyperparameter vector $\theta = \{\ell, \sigma_f, \sigma_n, a, b\}$ governing the mean and covariance functions must be optimized. This is achieved by maximizing the log marginal likelihood of the observations [29], [31], [32]:

$$\log p(\mathbf{y} \mid X, \theta) = -\frac{1}{2}(\mathbf{y} - m(X))^\top (K_y)^{-1}(\mathbf{y} - m(X)) - \frac{1}{2}\log|K_y| - \frac{n}{2}\log 2\pi \tag{2.16}$$

where $K_y = K(X, X) + \sigma_n^2 I$ is the noisy covariance matrix. The first term measures data fit, the second term penalizes model complexity (acting as an implicit regularizer), and the third term is a normalization constant [29], [33].

2.4.2 Covariance Functions

The covariance function, or kernel, is the central mathematical engine of a Gaussian process, mapping how input similarities translate to correlations in the output space and defining geometric assumptions like differentiability, length-scale, and amplitude fluctuations [29], [31], [32]. To generate valid covariance matrices, the kernel must be positive semi-definite, and its hyperparameter vector—typically containing the characteristic length-scale ℓ and signal variance σ_f^2—is optimized by maximizing the marginal log-likelihood of the training data [29]. In battery diagnostics, the kernel selection is critical, as it must balance resolving rapid electrochemical changes near depletion with filtering high-frequency telemetric noise [34], [35].

To visualize the varying smoothness and characteristic sample paths generated by different covariance classes, the comparative simulation in Figure 2.4 was constructed. As illustrated in Figure 2.4(a), the Squared Exponential (RBF) kernel enforces mean-square infinite differentiability (C^∞), producing smooth, wave-like trajectories. While advantageous for well-behaved processes, this strict smoothness prior can over-smooth localized inflection points during sudden voltage drops. Conversely, the Matérn family introduces a parameter ν that explicitly dictates path

differentiability. The Matérn 3/2 kernel (Figure 2.4(b)) yields once-differentiable trajectories (C^1) that maintain continuous derivatives while accommodating localized gradient variations, making it well-suited for tracking degradation dynamics. Finally, the Matérn 1/2 kernel (Figure 2.4(c)) models continuous but non-differentiable (C^0) Ornstein–Uhlenbeck processes, resulting in rough sample paths that reflect high local volatility.

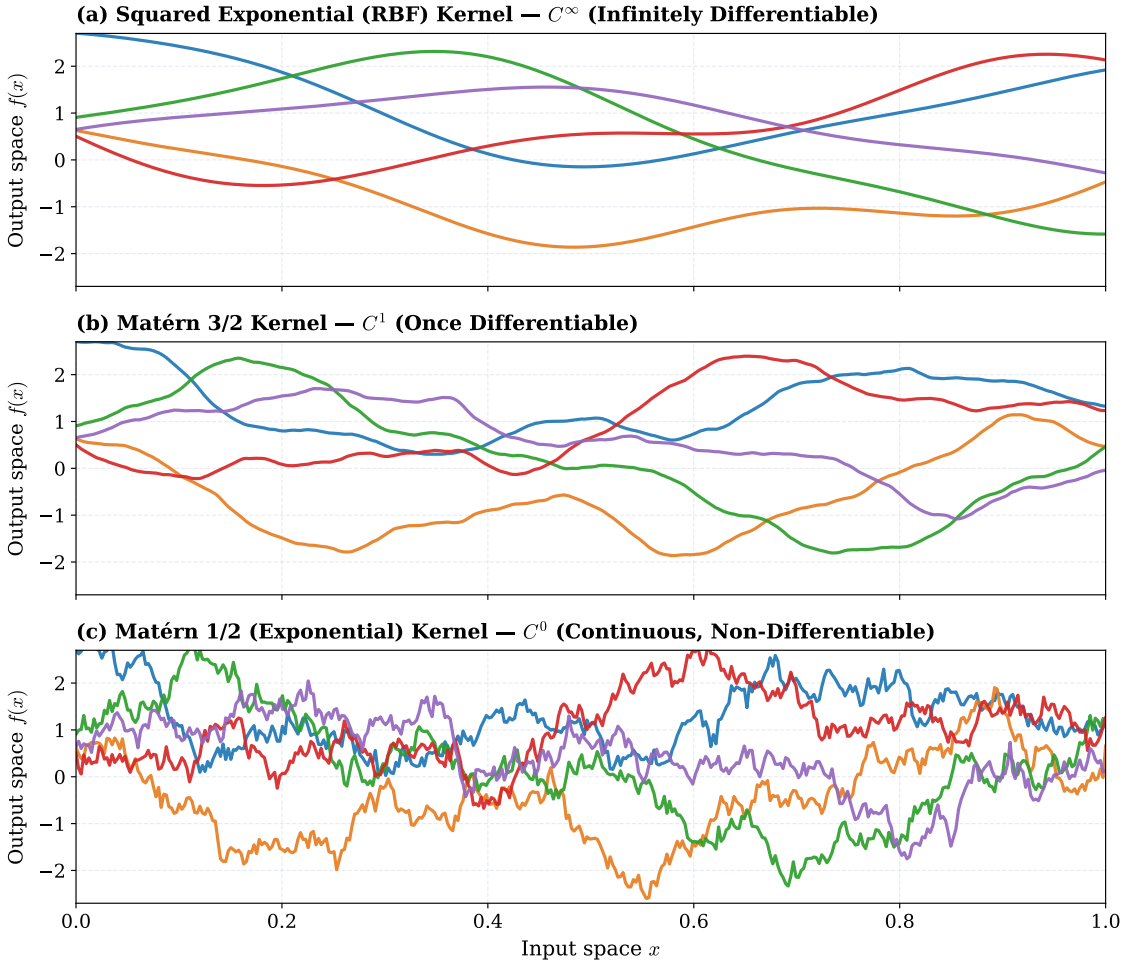


Figure 2.4: Sample function realizations illustrating the differentiability and smoothness characteristics of standard covariance functions across a normalized input space: (a) Squared Exponential (RBF) kernel generating infinitely differentiable (C^∞) paths; (b) Matérn 3/2 kernel yielding once-differentiable (C^1) sample paths suited for localized gradient changes; and (c) Matérn 1/2 (Exponential) kernel producing continuous but non-differentiable (C^0) rough trajectories modeling stochastic local fluctuations.

2.4.2.1 Squared Exponential Kernel

The Squared Exponential (or Radial Basis Function, RBF) kernel is a widely used baseline covariance function, defined as [29], [35]:

$$k_{\text{RBF}}(x, x') = \sigma_f^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right) \tag{2.17}$$

where ℓ is the characteristic length-scale and σ_f^2 is the signal variance. This kernel assumes that the underlying function is infinitely differentiable, which yields relatively smooth sample paths.

2.4.2.2 Matérn Class of Kernels

For physical degradation processes that exhibit sudden transitions or local variations, the infinitely smooth assumption of the RBF kernel can lead to over-smoothing at critical knees. The Matérn covariance class provides a parameter ν to control the differentiability of the sample paths [29], [34]:

$$k_{\text{Matérn}}(x, x') = \sigma_f^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\frac{|x - x'|}{\ell}\right)^{\nu} K_\nu \left(\sqrt{2\nu}\frac{|x - x'|}{\ell}\right) \tag{2.18}$$

where Γ is the Gamma function and K_ν is the modified Bessel function of the second kind.

For $\nu = 3/2$, the kernel yields once-differentiable sample paths, formulated as:

$$k_{\text{Matérn } 3/2}(x, x') = \sigma_f^2 \left(1 + \frac{\sqrt{3}|x - x'|}{\ell}\right) \exp\left(-\frac{\sqrt{3}|x - x'|}{\ell}\right) \tag{2.19}$$

This once-differentiable formulation ($\nu = 3/2$) is mathematically suited for modeling physical processes where the target function possesses a continuous first derivative but exhibits localized variations in its gradient. By allowing the first derivative to change relatively quickly, it prevents the over-smoothing of inflection points while maintaining a basic level of differentiability.

For $\nu = 5/2$, the kernel yields twice-differentiable sample paths, formulated as:

$$k_{\text{Matérn } 5/2}(x, x') = \sigma_f^2 \left(1 + \frac{\sqrt{5}|x - x'|}{\ell} + \frac{5|x - x'|^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}|x - x'|}{\ell}\right) \tag{2.20}$$

This twice-differentiable formulation ($\nu = 5/2$) generates smoother sample paths by ensuring the existence and continuity of both the first and second derivatives. It is mathematically designed for processes that are expected to be highly smooth, yet still require the flexibility to adapt to localized changes in the rate of acceleration (second derivative) without the extreme restriction of infinite differentiability.

For $\nu = 1/2$, the kernel is equivalent to the exponential covariance function:

$$k_{\text{Matérn } 1/2}(x, x') = \sigma_f^2 \exp\left(-\frac{|x - x'|}{\ell}\right) \tag{2.21}$$

This kernel yields continuous but non-differentiable sample paths, modeling rougher trajectories with high local variation.

2.4.2.3 Noise Kernel

Measurement uncertainty is represented using an additive white-noise kernel:

$$k_{\text{noise}}(x, x') = \sigma_n^2 \delta_{ij} \tag{2.22}$$

where δ_{ij} is the Kronecker delta, which models independent, identically distributed measurement noise [34].

2.5 Software Architecture and Service Design

The integration of analytical models into clinical workflows is situated within the broader digital transformation of healthcare infrastructures, driven by the increasing availability of wearable data and real-world clinical records [1], [2], [4], [6]. As distributed health networks evolve toward interoperable ecosystems, backend services must be designed to support modular scalability, fault isolation, and semantic consistency [3], [4], [7]. These requirements are particularly pronounced when working with active class III implantable medical devices (IMDs), where telemetry is retrieved intermittently as discrete snapshots due to strict power and communication constraints [11]. Analytical components should therefore operate as on-demand systems that are activated only when new data becomes available [12].

2.5.1 Microservices Architecture and Decoupling

Historically, clinical software was designed using monolithic patterns, where all functionalities were bundled into a single deployable unit. While straightforward to initialize, monoliths suffer from tight coupling, making modular scalability and technology updates difficult to manage [9], [10]. Service-Oriented Architecture (SOA) and the Service-Oriented Medical Device Architecture (SOMDA) frameworks emerged as intermediate steps, utilizing standard profiles to establish a safety context for clinical networking [21], [22].

Microservices Architecture (MSA) extends these principles by decomposing systems into fine-grained, independently deployable services that communicate through standardized RESTful or event-driven interfaces [10]. Recent frameworks structure clinical ecosystems around ingestion, analytics, orchestration, security, and API gateway management [4]. This decoupled approach provides several theoretical advantages:

- **Technology Heterogeneity:** Services can use optimized mathematical runtimes (e.g., Python-based GPR libraries) while interacting with general enterprise application frameworks.
- **Fault Isolation:** High-CPU analytical routines or memory exhaustion events are isolated to the specific service process, preventing cascading failures across the core clinical database or user interface [9], [10].
- **Decentralized Scalability:** Computational resources can be scaled horizontally through service replication to manage fluctuating request volumes without requiring monolithic database locks [10], [38].

Additionally, in distributed system theory, a service is defined as stateless if it does not store session context or client data between requests, relying entirely on the parameters provided in the incoming payload [12]. For implantable medical device telemetry, which is retrieved as sparse snapshots during patient check-ups rather than as continuous streams, a stateless on-demand pipeline is optimal [11]. Telemetry data from heterogeneous sources are normalized and validated at the service boundary [3], [8]. A stateless engine processes this normalized payload, serializes the resulting prognostic metrics, and returns to an idle state without maintaining persistent connections or local data states [9], [12].

To integrate these analytical models into clinical workflows, data exchange must adhere to standardized interoperability protocols. The HL7 Fast Healthcare Interoperability Resources (FHIR) standard represents the dominant framework for structuring clinical datasets, utilizing resources like `Bundle` for collection, `Observation` for clinical metrics, and `DiagnosticReport` for compiling diagnostic findings [39]. In production environments, FHIR R4 is the most widely adopted version due to extensive regulatory certification and vendor support [40]. However, the newer FHIR R5 specification introduces structural improvements, such as enhanced subscription frameworks and updated clinical resources, prompting active research into cross-version translation. Transforming schemas between R4 and R5 requires addressing non-backward-compatible breaking changes, often achieved programmatically using FHIR-native `StructureMap` resources and the FHIR Mapping Language (FML) to maintain data integrity during version migration [41].

2.5.2 Resilience and Robustness Patterns

Operating software within medical environments requires functional robustness—the ability of a system to maintain availability and isolate faults under unexpected or corrupted inputs [6], [42]. In distributed clinical systems, this robustness is not merely about preventing application crashes; it requires intercepting errors at the service boundary and propagating them as standardized, descriptive diagnostics. Rather than returning generic, opaque responses (such as standard `HTTP 500 Internal Server Error` codes) that mask the root cause, a robust system translates internal validation and mathematical failures into explicit, semantic error contracts. This transparency allows calling applications (such as hospital database proxies or programmer terminals) to identify structural or data-density anomalies programmatically and handle them without disrupting the wider network [6], [42].

Some of the more regular tools to make a system more robust and achieve this level of operational resilience include:

- **Fail-Fast Input Validation:** Inbound telemetry payloads are checked against strict schema definitions and physical domain boundaries immediately at the service perimeter [8]. Rejecting malformed or out-of-bounds data packets before triggering downstream processes prevents CPU-bound mathematical models from consuming thread pools or failing silently in downstream code blocks [12], [42].
- **Unified Exception Isolation:** A centralized error boundary intercepts framework-

level parsing anomalies and custom application-level domain errors. By routing all exceptions through a single handler, the service enforces a uniform diagnostic output format, ensuring that error responses are consistently structured, descriptive, and actionable for client integrations.

- **Circuit Breaker:** This pattern monitors communication timeouts and failure rates across service boundaries. If a service exceeds pre-defined failure thresholds, the circuit trips to immediately reject subsequent requests and prevent thread blocks from propagating and causing cascading hangs across clinical networks [6].

2.5.3 Containerization and edge-tunneling

Deploying specialized mathematical models requires a consistent runtime environment (library versions, dependencies, and OS packages). While hardware-level virtualization (Virtual Machines) runs a full guest operating system on top of a hypervisor, it introduces high CPU overhead and slow startup times.

Containerization utilizes operating system-level virtualization to package application code and dependencies into isolated, reproducible containers sharing the host kernel [10], [18]. For analytical microservices, containerization ensures that the exact versions of the regression kernels and physical parameters run identically across local development, validation, and cloud environments [6], [18]. This lightweight virtualization allows services to scale horizontally behind load balancers to manage thousands of devices without hypervisor overhead [9], [10].

In distributed deployment architectures, containerized services are frequently exposed using edge tunneling and secure reverse proxies. Secure public tunnels establish encrypted, temporary connections between isolated local hosts and public cloud infrastructure. This approach exposes local container endpoints to public networks, simulating cloud-native conditions. Through this mechanism, systems can evaluate external network latency, public routing overhead, and gateway firewall configurations before transitioning the microservice to production cloud infrastructure.[43]

3

Methods

This chapter details the implementation of the hybrid estimation framework, the data-driven preprocessing pipeline, the microservice-oriented deployment architecture, and the final validation strategies.

3.1 Hybrid Estimation Framework Implementation

The analytical core of this work is realized as a serial hybrid framework where a mechanistic physical prior is executed concurrently with a data-driven error corrector. Rather than running high-frequency, continuous-time circuit simulations—which are computationally prohibitive over decadal clinical prediction horizons—the framework translates the discrete-time formulations into a high-performance, object-oriented pipeline optimized for irregular and sparse telemetry snapshots. As the majority of the implementation is defined in code, the reader is referred to Appendix A for details on the software repository and implementation structure.

3.1.1 Data Ingestion Pipeline and Feature Engineering

The development of a robust data-driven error corrector requires transforming raw, heterogeneous telemetry logs stored within the clinical relational database into a standardized, multi-dimensional feature space. Telemetry snapshots gathered from long-term active implantable generators are inherently irregular, subject to transmission dropouts, and could contain mixed variable types.

Raw patient records are pulled asynchronously using a dedicated data ingestion module, fetching historical session tracking frames sorted sequentially by the clinical transmission timestamp t_k. Let the raw database observation at index k be defined as a tuple:

$$\mathcal{O}_k = (t_k, I_{load,k}, PW_k, Z_{lead,k}, \mathrm{SoC}_{telemetry,k}) \tag{3.1}$$

where $I_{load,k}$ is the recorded stimulus current in milliamperes, PW_k is the pulse width in microseconds, $Z_{lead,k}$ represents the secondary lead impedance in ohms (Ω), and $\mathrm{SoC}_{telemetry,k}$ denotes the raw battery fuel gauge percentage reported by the device's internal telemetry chip.

To eliminate non-representative anomalies, initialization noise, and telemetry artifacts, a deterministic sanitization pipeline is applied:

1. **Redundancy and High-Frequency Noise Filter:** Because immediate successive transmissions can occur due to clinical touchpoint testing or retry loops, a temporal delta check enforces a strict lower boundary where any session snapshot captured within $\Delta t_k = t_k - t_{k-1} < 3,600$ seconds of its predecessor is discarded to prevent high-frequency localized biasing.
2. **Outlier and Telemetry Error Culling:** Gross tracking anomalies—such as a vertical capacity drop or upward step jump where the difference between the true clinical capacity observation and the mechanistic state prior satisfies $|\mathrm{SoC}_{telemetry,k} - \mathrm{SoC}_{ecm,k}| > 0.50$—are flagged as system telemetry errors and culled from the training stream.

Following sanitization, the framework constructs a localized four-dimensional feature vector $\mathbf{x}_k \in \mathbb{R}^4$, formalized as:

$$\mathbf{x}_k = \begin{bmatrix} \mathrm{SoC}_{ecm,k} \\ I_{avg,k} \\ Z_{lead,k} \\ \mathrm{Age}_{days,k} \end{bmatrix} \tag{3.2}$$

where $\mathrm{SoC}_{ecm,k}$ represents the physical state-of-charge calculation generated by the 2-RC state prior, $I_{avg,k}$ is the moving average baseline current drain, $Z_{lead,k}$ is the stabilized system lead path impedance, and $\mathrm{Age}_{days,k}$ is the operational device lifespan calculated explicitly from the initial system implant activation date:

$$\mathrm{Age}_{days,k} = \frac{t_k - t_{implant}}{86,400} \tag{3.3}$$

To enforce statistical parity across input dimensions with vastly different numerical orders of magnitude, the feature matrix undergoes a linear affine Z-score transformation:

$$\mathbf{z}_k = \mathbf{\Lambda}^{-1} (\mathbf{x}_k - \mu_x) \tag{3.4}$$

where $\mu_x \in \mathbb{R}^4$ represents the column-wise vector of training feature means and $\mathbf{\Lambda} \in \mathbb{R}^{4\times4}$ denotes a diagonal scaling matrix composed of the corresponding computed feature sample standard deviations (σ_{SoC}, σ_I, σ_Z, σ_{Age}). The transformation centers the operational space around a zero mean ($\mu = 0$) and maps the variation uniformly onto unit variance ($\sigma^2 = 1$). The standardization parameters ($\mu_x, \mathbf{\Lambda}$) are serialized directly inside the containerized microservice package to guarantee that incoming on-demand real-time REST requests are scaled identically to the validation array prior to model execution.

3.1.2 Object-Oriented ECM Implementation

The discrete-time state-space formulation derived in the theory chapter is implemented within a dedicated Python abstraction class, `SecondOrderECM`. This component encapsulates the battery's electrical properties and maintains the state vector x_k as a persistent internal matrix defined as:

$$x_k = \begin{bmatrix} U_{p,1,k} \\ U_{p,2,k} \\ \mathrm{SoC}_k \end{bmatrix} \tag{3.5}$$

To handle the highly irregular sampling intervals (Δt_k) characteristic of real-world clinical generator logs, the analytical Zero-Order Hold (ZOH) coefficients are re-evaluated dynamically at each integration step. The non-linear relationship between the Open Circuit Voltage (U_{oc}) and the State of Charge (SoC) is captured using a Monotonicity-Preserving Piecewise Cubic Hermite Interpolating Polynomial (PCHIP). This mathematical approach preserves the shape of the cell characterization data without introducing artificial oscillations, ensuring safe gradient evaluation.

The non-linear chemical exhaustion phase (the characteristic rapid capacity fade or "voltage knee" encountered as primary Li/CF_x cells approach depletion) is engineered directly into the discrete input transition arrays. When the internal state variable SoC_k drops below a specified critical threshold ($\mathrm{SoC}_k \leq 0.20$), the model applies a parabolic pessimism multiplier to the nominal ohmic resistance R_0, defined by:

$$R_{eff} = R_0 \cdot \left(1.0 + 25.0 \cdot (0.20 - \mathrm{SoC}_k)^2\right) \tag{3.6}$$

This state-dependent amplification effectively maps the accumulation of insulating Lithium Fluoride (LiF) within the cathode lattice, forcing the physical prior to output a conservative, safety-first baseline.

Furthermore, the RC-block components included in the ECM are derived from normative values found in literature. This entails that a destructive test was not done to extract battery specific values.

3.1.3 Residual GPR and Personalization Pipeline

The data-driven corrective layer is managed by a structural `ResidualGPR` wrapper class that interfaces with the pre-trained global population kernel. Incoming device telemetry is highly heterogeneous; missing data coordinates—such as missing secondary lead impedance (Z) readings—are stabilized using a localized Zero-Order Hold forward-filling policy to avoid introducing artificial step-discontinuities into the multidimensional feature matrix.

Within this class, the Gaussian Process model utilizes a composite covariance kernel designed to capture both broad population-level trends and highly localized, patient-specific variations. The composite kernel is mathematically formulated as:

$$k(\mathbf{x}, \mathbf{x}') = k_{\text{global}}(\mathbf{x}, \mathbf{x}') + k_{\text{local}}(\mathbf{x}, \mathbf{x}') + k_{\text{noise}}(\mathbf{x}, \mathbf{x}') \tag{3.7}$$

where the global covariance k_{global} and the local covariance k_{local} are both modeled using the Matérn 5/2 kernel defined in Equation 2.20, scaled by constant coefficients. The global kernel is initialized with a large length-scale ($\boldsymbol{\ell} = [50.0]^4$) to model stiff, long-term degradation patterns, whereas the local kernel is initialized with a smaller length-scale ($\boldsymbol{\ell} = [1.0]^4$) to capture localized variations around the clinical visits. The noise component k_{noise} is modeled using a white noise kernel to absorb sensor quantization errors and random fluctuations in terminal voltage readings.

To adjust the generalized global population knowledge learned by the composite kernel during training to a singular, specific patient device without incurring the risk of local overfitting or destroying the hyperparameter optimization balance, a two-stage alignment pipeline was developed:

1. **Global Feature Mapping:** The pre-trained Gaussian Process models the normalized four-dimensional feature space $\mathbf{x} = [\text{SoC}_{ecm}, I_{avg}, Z_{lead}, \text{Age}_{days}]$ to output a generalized population residual error expectation μ_{gpr} and its associated variance σ^2_{gpr}.
2. **Device-Specific Bias Integration:** A local state observer monitors deviations at active measurement coordinates. To prevent vertical "sawtooth" jumps or step-wise snap discontinuities at clinical touchpoints, an integrated device offset (μ_{device}) is computed as the moving average deviation across historical data. This calibration bias is added uniformly to shift the posterior mean curve smoothly through the sparse observations.

To guarantee that the personalized output remains physically consistent, the hybrid estimates are subsequently subjected to thermodynamic monotonicity and boundary clamping constraints. While the residual Gaussian Process layer provides high-fidelity tracking of individual parameter variations, unconstrained stochastic estimators are entirely decoupled from physical conservation laws. Given that the implantable generators operate on non-rechargeable primary Li/CF_x chemistries, the true remaining battery capacity is thermodynamically constrained to be a strictly non-increasing, monotonic function over time. High-frequency sensor anomalies, pacing load fluctuations, or quantization noise within the clinical logs can cause the unconstrained GPR posterior mean prediction (μ_{gpr}) to artifactually jump upward between successive telemetry check-ups.

To preserve physical validity and prevent non-invertible state tracking errors, the hybrid engine subjects the converged posterior distribution to an explicit algorithmic monotonicity envelope and absolute boundary constraint. At each simulated integration sub-step t, the preliminary hybrid state estimate is evaluated against the historically preserved state trajectory:

$$\text{SoC}_{temp}(t) = \text{SoC}_{phys}(t) + \mu_{gpr}(t) + \mu_{device} \tag{3.8}$$

$$\text{SoC}_{hybrid}(t) = \min\left(\text{SoC}_{hybrid}(t - \Delta t_{step}),\ \max\left(0.0,\ \min\left(\text{SoC}_{temp}(t),\ 1.0\right)\right)\right) \tag{3.9}$$

By nesting the affine transformation inside a hard floor-and-ceiling envelope ($[0.0, 1.0]$) alongside a backward-looking historical minimizer, the framework programmatically filters out transient telemetry artifacts. This structural configuration transforms the free-roaming black-box regression corrector into a bounded, safety-critical state observer, ensuring that the tracking engine remains electrochemically stable under highly sparse or corrupted input streams.

3.1.4 The Hybrid Simulation Integration Engine

The sequential execution of the physical prior and the probabilistic corrector is orchestrated by a dedicated simulation pipeline. To guarantee numerical stability

of the exponential decay coefficients within the state transition matrix $\mathbf{A}_k$ across long clinical forecasting horizons, a strict temporal sub-stepping loop is enforced. The total temporal gap between two arbitrary clinical telemetry sessions (Δt_{gap}) is parsed and sliced into uniform, day-long internal steps capped at $\Delta t_{step} = 86,400$ seconds.

The comprehensive execution flow for the asynchronous state reconstruction, error matching, and forward state propagation is formalized explicitly in Algorithm 1.

Algorithm 1 Hybrid Asynchronous State Estimation and Prognosis Engine

Require: Telemetric History Array $\mathcal{D}$, Serialized Localized Kernel Weights $\mathcal{GP}$, Nominal Cell Parameters

Ensure: Array of reconstructed hybrid state trajectories $\mathcal{R}$

Instantiate `SecondOrderECM` class constructor object
Assign starting state coordinate: $x_k[2, 0] \leftarrow \mathcal{D}$.iloc[0]['battery_pct']
$\mu_{device} \leftarrow$ `personalize_to_device`($\mathcal{D}$) {Extract telemetric historical baseline error bias}
for $i = 1$ **to** length($\mathcal{D}$) $- 1$ **do**
  row$_{prev} \leftarrow \mathcal{D}$.iloc[$i - 1$]
  row$_{curr} \leftarrow \mathcal{D}$.iloc[$i$]
  $\Delta t_{gap} \leftarrow$ total_seconds(row$_{curr}$['visit_time'] $-$ row$_{prev}$['visit_time'])
  $\tau_{remaining} \leftarrow \Delta t_{gap}$
  while $\tau_{remaining} > 0$ **do**
    $\Delta t_{step} \leftarrow \min(86400, \tau_{remaining})$
    $\_, \text{SoC}_{phys} \leftarrow$ `ECM.step`(row$_{prev}$['i_avg'], Δt_{step}) {Advance mechanistic physical state}
    $\tau_{remaining} \leftarrow \tau_{remaining} - \Delta t_{step}$
    $\text{Age}_{days} \leftarrow$ calculate_current_device_age()
    $\mu_{gpr}, \sigma_{gpr} \leftarrow \mathcal{GP}$.`predict`(SoC$_{phys}$, row$_{prev}$['i_avg'], row$_{prev}$['lead_z'], Age$_{days}$)
    $\text{SoC}_{hybrid} \leftarrow \text{clip}(\text{SoC}_{phys} + \mu_{gpr} + \mu_{device}, 0.0, 1.0)$ {Compute Hybrid Posterior Distribution}
    Append $\{t, \text{SoC}_{phys}, \text{SoC}_{hybrid}, \sigma_{gpr}\}$ to $\mathcal{R}$
  end while
  $\mathcal{R}[-1]$['actual_soc'] $\leftarrow$ row$_{curr}$['battery_pct'] {Anchor ground-truth coordinate for validation profiling}
end for
return $\mathcal{R}$

3.2 Software Architecture Development Process

The software architecture for the battery estimation microservice was developed using an iterative, chronological engineering lifecycle. This section chronicles the sequential phases of development, tracking the system from conceptual layering and containerization to component-level integration and final semantic alignment.

3.2.1 Architecture Setup

The initial stage of development focused on defining system boundaries, initializing the physical codebase workspace, and establishing an isolated, multi-container runtime testing network. This phase translated conceptual architectural blueprints directly into a testable infrastructure designed from the outset to support native clinical interoperability and complex mathematical models.

To ensure the physical codebase structured the separation of concerns right from the beginning, an initial functional mapping was conducted. Reference designs from service-oriented and layered architecture models in collaborative healthcare systems research—specifically the Service-Oriented Medical Device Architecture (SOMDA) frameworks investigated within the OR.NET project [22] and the modular five-layer integration model outlined by Warrier [4]—informed the design. This research provided the framework to decompose the microservice's synchronous runtime lifecycle into five distinct logical domains: an entrance portal, a semantic validation tier, an abstract mathematical coordination block, a response formatting component, and a centralized fault containment layer. The logical flow of a request through these service layers is visualized in Figure 3.1. The responsibilities mapped to each functional layer are outlined in Table 3.1.

Following this mapping, a root package directory named `MSA/` was initialized. The codebase was divided into subdirectories that directly mirrored these functional layers, providing a structured environment for independent component implementation. The initialized directory tree and the definitive core files created to establish the working framework are presented in Table 3.2.

This layout was constructed to natively support an asymmetric data exchange pipeline. The framework utilized the `FastAPI` web framework and the `Pydantic` validation engine to accept an inbound HL7 FHIR R4 JSON packet over an HTTP connection, route the request through gateway dependency stubs within `routers/deps.py` in preparation for future cryptographic token validation, parse the multi-tiered model hierarchy via `fhir_ingress.py`, and securely forward the continuous load vectors directly to the active `HybridOrchestrator` simulation block within the execution core.

To keep the development of this infrastructure fully isolated and controlled, a containerized execution network was configured using Docker and Docker Compose before expanding any internal application logic. This setup isolated dependencies and prevented direct local filesystem pollution through a two-tiered configuration model:

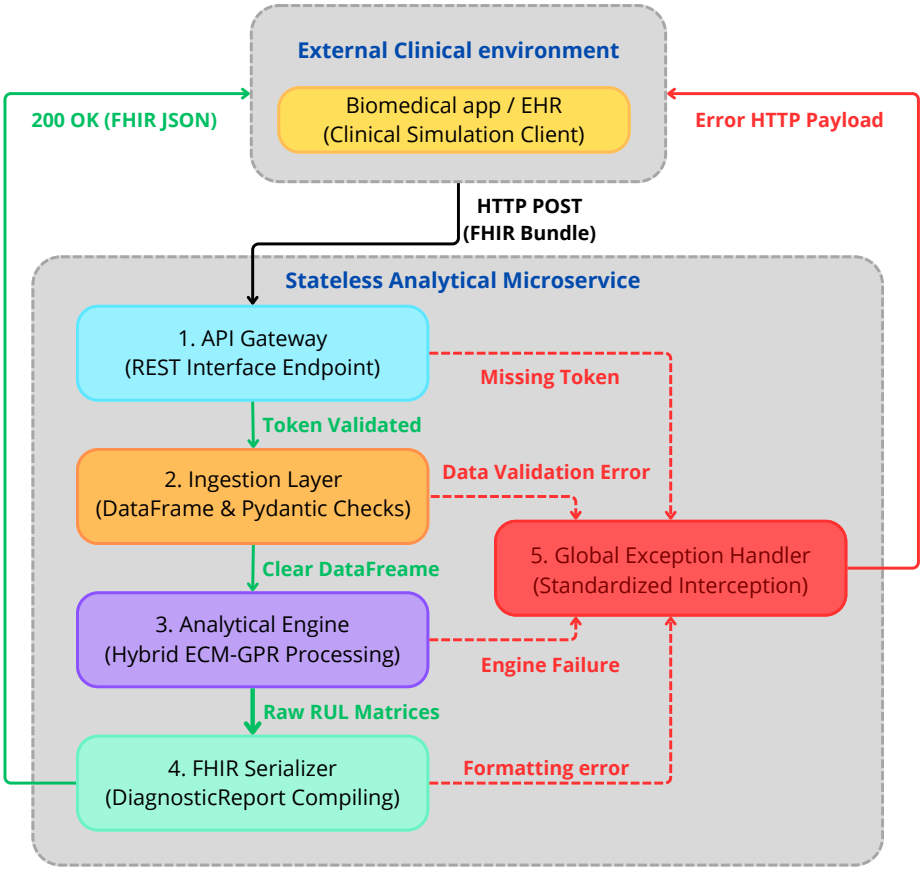


Figure 3.1: Logical layered architecture of the microservice, displaying the request-response lifecycle from the external clinical client through the validation boundaries and the analytical engine, including the global exception handler.

Table 3.1: Functional Layer Mapping and Operational Responsibilities Derived from Design Requirements.

Logical Component	Operational Responsibility
API Gateway	Exposes network entry points, processes inbound communication streams, and intercepts requests for token-based authentication.
Ingestion Layer	Extracts inbound JSON structures, instantiates tabular internal data representations, and performs strict datatype and logical boundary validation.
Analytical Engine	Coordinates the core mathematical execution loops, including historical re-simulation, localized residual extraction, and forward-projecting prognostics.
FHIR Serializer	Translates internal raw array outputs into standardized, semantically compliant clinical observation and diagnostic report formats.
Exception Handler	Intercepts runtime parsing or execution faults across all layers and translates anomalies into standardized, descriptive error responses.

Table 3.2: Physical Codebase Workspace Initialization and Associated Target Layers.

Directory Path	Foundational Code Files	Target Functional Layer
`routers/`	`battery.py`, `deps.py`	API Gateway (Layer 1)
`schemas/`	`fhir_ingress.py`	Ingestion Layer (Layer 2)
`services/`	`battery_service.py`, `hybrid_engine.py`	Analytical Engine (Layer 3)
`serializers/`	`fhir.py`	FHIR Serializer (Layer 4)
`utils/`	`service_result.py`, `app_exceptions.py`	Exception Handler (Layer 5)
(Root Folder)	`main.py`, `MSA-requirements.txt`	Microservice Initialization

1. **Container Build Specifications (`Dockerfile`):** A script was authored inside the `MSA/` directory to define the build steps for the backend engine container. This file specified a standardized baseline Python image runtime, established an internal absolute working directory hierarchy (`/app`), copied the package files, executed sequential pip installation commands from the dependency manifest, and configured the automated startup entry point via an ASGI server process.
2. **Multi-Node Network Orchestration (`docker-compose.yml`):** A composition engine manifest was implemented at the project root directory to organize the isolated components. To maintain service independence, the main backend container application (`msa_service`) was bounded by mapping its build context exclusively to the relative path `./MSA` and restricting outside incoming data packets to port 8000. Communication paths were restricted programmatically by deploying the service behind a private software-defined bridge network named `thesis_net`.
3. **External Infrastructure Simulation Node (`clinical_app`):** To test the core software pipeline, an independent companion container named `clinical_app` was added to the orchestration map. This service node was deployed to simulate the hosting clinical infrastructure and act as a client application. It was configured to operate inside the same virtual `thesis_net` subnet, allowing it to execute automated HTTP mock request dispatches to the isolated microservice endpoint without requiring exposure to a live corporate or hospital data network.

3.2.2 Standardized Clinical Ingress and Algorithmic Orchestration Lifecycle

This subsection chronicles the synchronous execution lifecycle of a clinical simulation request as it traverses the containerized microservice infrastructure. The development focused on constructing an asymmetric data exchange pipeline capable of consuming standardized healthcare records, mapping raw observation vectors to the analytical engine dependencies, and returning semantically compliant outputs.

To achieve this workflow, a deliberate selection of external software libraries was compiled within the dependency manifest. Each software package was introduced to manage an explicit operational requirement across the lifecycle layers. The core libraries utilized and their defined roles inside the backend ecosystem are summarized in Table 3.3.

The operational execution lifecycle of an on-demand simulation request is structured into three continuous software phases:

Standardized Semantic Input and Parameter Extraction: The request lifecycle initiates at the outer boundary when the API Gateway endpoint exposed in `routers/battery.py` intercepts an inbound network request containing a JSON payload. The gateway utilizes a `Pydantic` root data model (`FHIRTelemetryBundle`) to parse the multi-tiered JSON hierarchy natively. The inbound transaction type

Table 3.3: Core Software Libraries and Target Operational Roles Within the Microservice Network.

Library Name	Core Package Role	Specific Operational Application
`FastAPI`	Routing and Gateway	Exposes asynchronous REST endpoints, manages URI paths, and handles path dependency injection loops.
`Pydantic`	Semantic Validation	Enforces data type constraints and provides structural data-parsing models for JSON objects.
`pandas`	Matrix Structuring	Aligns unstructured, extracted parameters chronologically into tabular multi-dimensional arrays.
`numpy`	Array Operations	Accelerates continuous matrix array scaling and executes numerical transformations during the estimation process.
`joblib`	Model Serialization	Deserializes pre-trained global population kernel coefficient files directly into active container memory maps.

is defined to conform explicitly to an HL7 FHIR R4 `Bundle` model composed of a collection of nested `Observation` profiles.

Once the structural format is validated at the border, the router forwards the model tree to a specialized translation helper module, `schemas/fhir_ingress.py`. This module executes a sequential scanning routine across the nested observation arrays, filtering and isolating distinct telemetric parameter coordinates based on standardized ISO/IEEE 11073 medical device communication (MDC) nomenclature. Specifically, the parsing module extracts the target clinical parameters using their associated diagnostic MDC codes:

- Battery capacity percentage tracking metrics are identified via code `MDC_IDC_MSMT_BATTERY_STATUS`.
- Secondary lead path network impedance values are extracted via code `MDC_IDC_MSMT_LEAD_IMPEDANCE`.
- Pacing stimulus output current amplitudes are identified via code `MDC_IDC_SET_PACING_AMPLITUDE`.
- Stimulus pulse width tracking envelopes are isolated via code `MDC_IDC_SET_PACING_PULSE_WIDTH`.
- Monitored device pacing signal pulse rates are targeted via code `MDC_IDC_SET_PACING_RATE`.

Following coordinate extraction, the helper maps the parameters alongside their associated `effectiveDateTime` markers, flattening the multi-tiered resource arrays chronologically into a standardized internal data format.

Algorithmic Dependency Integration and Execution: Upon successful extraction, the data structure is passed directly into the core service layer within `services/battery_service.py`. This block bridges the network infrastructure with the analytical framework. Upon initial initialization of the container space, the service uses the `joblib` library to load the serialized population model coefficient file (`R0605_MAE00435.joblib`) directly into active container memory.

The service module arranges the extracted chronological parameter fields into structured `pandas` DataFrames and evaluates macro-scale indicator functions to calculate continuous load parameters. The structured numeric matrices are then handed off to the execution coordinator module (`HybridOrchestrator`) implemented inside `services/hybrid_engine.py`. This class orchestrates the chronological execution of the physical priors and the data-driven error correctors, utilizing `numpy` array blocks to perform continuous matrix scaling, step-wise time slicing, and state updates across the historical simulation loop without incurring data precision loss during runtime.

Interoperable Outbound Response Serialization: Once the execution core completes the forward-projecting calculation loops, the resulting multi-dimensional arrays—containing the reconstructed state-of-charge curves, remaining useful life horizons, and corresponding statistical variance indices—are returned to the application interface layer. The raw calculating variables are routed to the serialization handler inside `serializers/fhir.py`.

The serialization module maps the numerical vectors into an outbound HL7 FHIR R4 JSON structure. The generator's unique hardware identifier is bound as the primary subject identifier, while the multi-dimensional prediction timelines are nested as compliant component elements within a standardized `DiagnosticReport` profile. This semantically aligned JSON document is sent back through the gateway router as a successful HTTP response payload, rendering the internal calculations immediately readable by upstream clinical interface nodes. The hierarchical relationship and structural mappings of the inbound FHIR Telemetry Bundle and the outbound FHIR DiagnosticReport are visualized in Figure 3.2.

3.2.3 Robustness Engineering and Automated Verification Pipeline

The final stage of the microservice architecture development focused on implementing a defensive, centralized error-handling perimeter and configuring an automated, localized testing loop. Operating within a digital medical ecosystem requires predictable fault isolation to ensure that runtime anomalies do not cause cascading failures or unhandled server crashes across clinical network segments [4], [22]. This section documents the configuration of the global exception boundary and describes how the isolated simulation container was utilized to programmatically verify system stability.

Centralized Exception Boundary Implementation: To eliminate uncoordinated error checking across individual endpoints, a centralized exception interception framework was designed directly into the application core loop within `main.py`.

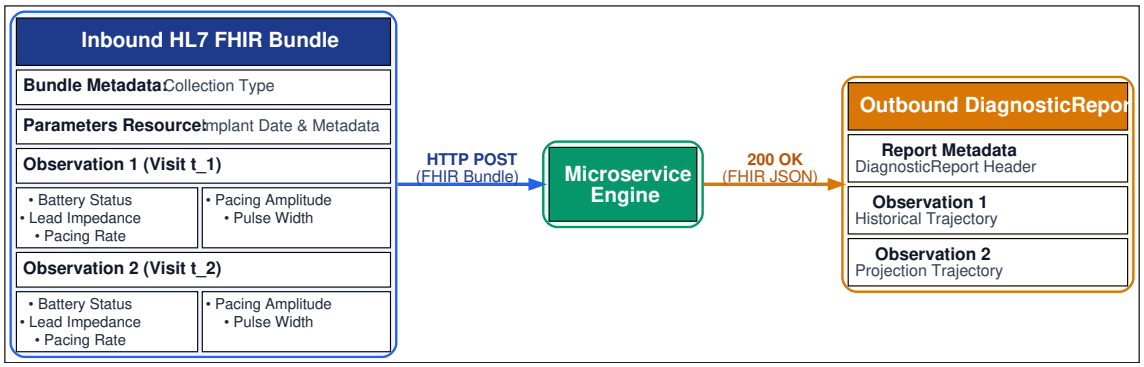


Figure 3.2: HL7 FHIR document topology and schema mapping across the microservice boundary. The client submits a composite FHIR `Bundle` containing longitudinal device observations and metadata; upon analytical processing, the microservice serializes a standardized FHIR `DiagnosticReport` containing historical and forecasted RUL trajectories.

This pattern acts as a uniform defensive perimeter that intercepts faults across all integration layers and normalizes them before they reach the network interface. The boundary logic catches and processes two distinct classes of runtime anomalies:

- **Framework-Level Schema Trapping:** FastAPI's native validation exceptions (`RequestValidationError`) are programmatically bound to a single global exception hook. When an incoming JSON packet violates core schema constraints, the framework intercepts the raw formatting failure at the border, halts the request lifecycle, and maps the event to a structured HTTP validation error context payload.
- **Application-Specific Fault Isolation:** Internal algorithmic, mathematical, and environment exceptions are inherited from a core structured abstract class (`AppExceptionCase`) and routed through the same global handler. If an internal calculation anomaly occurs deep within the regression core—or if the serialized coefficient file fails to initialize—the microservice isolates the active thread execution loop. The centralized handler maps the internal failure state to an explicit HTTP status code paired with a transparent diagnostic payload, allowing external applications to immediately identify the error signature.

Automated Local Anomaly Injection Testing: The resilience of the unified error boundary was verified using the localized multi-container testing infrastructure configured during the preparation phase. Rather than relying on manual validation checks, the external companion container (`clinical_app`) acted as an automated, programmatic client simulator operating within the isolated software-defined bridge network (`thesis_net`).

An automated validation script suite (`run_clinical_validation.py`) was developed to execute continuous negative testing cycles by purposefully injecting faulty requests across three distinct system boundaries:

1. **Authorization Boundary Injection:** The simulator dispatched HTTP requests with missing, malformed, or expired cryptographic bearer credentials

to verify that the API Gateway layer dependency (`routers/deps.py`) successfully aborted unauthenticated connections prior to running downstream parsing logic.

2. **Semantic Ingress Schema Injection:** Payloads containing corrupted JSON strings, omitted mandatory parameters, or incorrect HL7 FHIR structural paths were programmatically sent to the endpoint to confirm that the `fhir_ingress.py` translation engine executed fast-fail data trapping at the border.
3. **Algorithmic Boundary Stressing:** Intentionally truncated historical vectors and out-of-bounds telemetry values were passed through the pipeline to verify that the core analytical engine maintained numerical stability or isolated calculation exceptions cleanly through the global boundary handler without triggering a container crash.

The test automation suite logged that in each failure case, the microservice successfully prevented container-level system crashes, isolated thread-level anomalies, and maintained continuous service availability while delivering a valid diagnostic error payload back to the client simulator.

3.3 Validation Strategies

To satisfy the safety-critical constraints of active implantable medical devices (IMDs) and ensure the absolute reliability of the hybrid Equivalent Circuit Model - Gaussian Process Regression (ECM-GPR) microservice (`msa_service`), a unified, automated, multi-axis verification and validation framework was developed. The validation suite operates inside the clinical simulator container (`clinical_app`), executing isolated end-to-end integration tests over a private, software-defined bridge network (`thesis_net`).

The validation strategy is structured across dedicated functional dimensions overseen by a master CLI entry point (`run_suite.py`) which routes execution dynamically to specific orchestration modules depending on the selected action mode: `clinical`, `synthetic`, `stress`, `benchmark`, or `retro-eval`.

3.3.1 Microservice Verification and Latency Benchmarking

Direct verification is managed by orchestrator modules within the validation suite, which communicate with the microservice over the private virtual bridge network (`thesis_net`). This phase evaluates the service's functional correctness, input boundary protection, computational latency, and compliance with data exchange standards.

Functional Verification (Clinical and Synthetic Modes): The core pipeline for a single asset is verified by `orchestrator_request.py` to confirm mathematical and data-flow correctness. In `synthetic` mode, the module generates a clean, deterministic mock timeline of 16 telemetry readings to test base state propagation. In `clinical` mode, it extracts raw database logs matching a user-specified

pacemaker serial number. In both execution paths, the orchestrator packages observations chronologically into a standardized FHIR Collection Bundle, executes strict schema validation, transmits the payload to the gateway portal, validates the outbound DiagnosticReport schema, extracts the resulting state-space vectors, computes trajectory metrics, and exports a trajectory visualization plot.

Robustness and Anomaly Isolation (Stress Mode): Adversarial and negative testing is performed by `orchestrator_stress.py` using payload mutators to verify the exception handling and request validation of the microservice at the network, ingestion, and gateway boundaries. The testing suite programmatically injects data anomalies and authorization issues into standard FHIR bundles across seven scenarios:

- **Physical Parameter Boundaries:** Ingress values are set to physically impossible levels—specifically overriding the battery state of charge (SoC = 150%), pacing output currents ($I_{\text{load}} = -2.0\,\text{mA}$), or lead impedances ($Z_{\text{lead}} = -50.0\,\Omega$)—to verify parameter bounds checks.
- **Chronological Timeline Chaos:** Timestamps of telemetry entries are scrambled by swapping initial and final dates (t_0 and t_k) to test the robustness of the chronological state propagation and step-wise integration loops.
- **Envelope Integrity Breaches:** Required structural nodes, such as the hardware identifier envelope (`identifier`), are removed from the JSON payload to verify border block behavior.
- **Algorithmic Constant Omission:** Algorithmic constants, including the implant date timestamp (t_{implant}), are omitted to verify the fallback configurations of the engine.
- **Structural Type Sabotage:** Incompatible data types (e.g., text strings instead of floating-point numbers) are injected into quantity fields to test structural parser constraints.
- **Omission of Authorization Credentials:** The HTTP Authorization header containing the bearer token is omitted to verify gateway perimeter blocking.
- **Malformed or Incorrect Credentials:** An invalid signature token is transmitted to verify gateway dependency validation and credential culling.

The orchestrator captures the response codes and classifies the security posture. A status of `HTTP 400 Bad Request`, `HTTP 401 Unauthorized`, or `HTTP 422 Unprocessable Entity` is classified as a successful boundary block, whereas a status of `HTTP 500 Internal Server Error` indicates an unhandled exception or container crash, registering as a validation failure.

Computational Scaling and Latency Benchmarking (Benchmark Mode): To measure latency scaling, `orchestrator_benchmark.py` dispatches batch telemetry payloads across five observation sizes: $N \in \{1, 50, 100, 500, 1000\}$ entries. The benchmark isolates execution delays across two network topologies:

- **Local Docker Mesh:** Request routing occurs directly over the private software-defined bridge network (`thesis_net`) to isolate backend computational latency.
- **Online Edge Tunnel:** Request routing is directed through a public secure edge tunnel (`ngrok`) to evaluate latency overhead introduced by public networks.

For each size bracket, a warm-up request is executed to measure model cold-start latency, followed by $M = 30$ consecutive runs to capture steady-state latency profiles. High-precision monotonic clocks measure roundtrip times, which are processed to extract the mean, standard deviation, standard error, and the 95^{th} percentile.

Semantic Conformity Auditing: Bidirectional compliance with HL7 FHIR R4 schemas is validated at the client interface using the `fhir.resources` library. Payloads are checked before transmission to ensure observations and parameter structures conform to the FHIR Bundle model. Telemetry metrics are mapped using standardized ISO/IEEE 11073 medical device communication (MDC) codes and Unified Code for Units of Measure (UCUM) representations (see Table 3.4). Outbound responses are validated to ensure the estimated battery status, historical resimulation, and future projections are serialized correctly within a FHIR DiagnosticReport resource.

Table 3.4: ISO/IEEE 11073 (MDC) and UCUM Telemetry Semantic Mappings Used in FHIR Validation

Clinical Metric	IEEE 11073 MDC Code	MDC Display Name	UCUM Unit
Battery Capacity(SoC)	`MDC_IDC_MSMT_BATTERY_STATUS`	Battery Status	`%`
Lead Impedance (Z)	`MDC_IDC_MSMT_LEAD_IMPEDANCE`	Lead Impedance	`Ohm`
Pacing Output Current	`MDC_IDC_SET_PACING_AMPLITUDE`	Pacing Amplitude	`mA`
Pacing Pulse Width	`MDC_IDC_SET_PACING_PULSE_WIDTH`	Pulse Width	`us`
Pacing Signal Frequency	`MDC_IDC_SET_PACING_RATE`	Pacing Rate	`Hz`

3.3.2 Retrospective Cohort Validation

A retrospective cohort validation pipeline is implemented in `orchestrator_retro_validation.py` (`retro-eval` mode) to assess the predictive accuracy of the hybrid model on unseen patient histories.

Cohort Definition and Filtering: To prevent data leakage, patient records used during GPR training are identified via `gpr_training_data_final.csv` and excluded from the validation set. Unseen devices are selected from the clinical database

if they contain at least 30 visits ($N \geq 30$) and reached a minimum reported telemetry battery level of $\leq 10.0\%$ SoC, indicating traversal of the discharge curve knee. This process yields a validation cohort of $n = 87$ patient devices. **Data Splitting and Sanitization:** The chronological telemetry records of each candidate device are reconstructed. Transient 0.0% battery capacity readings followed by later positive values are culled as sensor anomalies, since the primary Li/CF_x battery chemistry cannot recover capacity. The sanitized history is then split chronologically:

- **Input History:** Observations where the battery level is $\geq 20.0\%$ SoC are compiled into a FHIR Bundle and sent to the microservice to personalize the hybrid estimator.
- **Test Set:** Subsequent observations where the battery level falls below 20.0% SoC are withheld and used as ground-truth.

Interpolation and Error Metrics: The microservice outputs prognostic projections at uniform 30-day intervals, whereas clinical visits occur irregularly. To compare the projected capacity with real test measurements, the projection curve is linearly interpolated at the exact elapsed days of each clinical visit. For a visit at t_k days since the split reference time, the predicted state of charge $\widehat{\mathrm{SoC}}(t_k)$ is:

$$\widehat{\mathrm{SoC}}(t_k) = \mathrm{SoC}_{\mathrm{proj}}(t_a) + \frac{t_k - t_a}{t_b - t_a} \left(\mathrm{SoC}_{\mathrm{proj}}(t_b) - \mathrm{SoC}_{\mathrm{proj}}(t_a)\right) \tag{3.10}$$

where t_a and t_b represent the nearest 30-day forecast intervals enclosing t_k ($t_a \leq t_k \leq t_b$).

Using the interpolated values, the framework evaluates the overall Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R^2) metrics. Accuracy is analyzed globally and stratified across the transition zone (20% to 10% SoC) and the critical zone ($< 10\%$ SoC). The Remaining Useful Life (RUL) error is defined as the temporal difference in days between the predicted and actual dates where the battery crossed the critical 10% SoC threshold.

4

Results

This chapter presents the empirical findings from the multi-axis validation framework. The evaluation is divided into a retrospective cohort validation across a patient cohort, computational scaling and latency benchmarks, robustness and exception validation, individual estimation trajectory case studies, and a comparative ablation study of the architectural configurations.

4.1 Retrospective Cohort Validation

To evaluate the generalization capability and clinical safety profile of the personalized hybrid state-of-charge (SoC) estimation model across a broad patient population, a retrospective cohort validation was performed. The validation cohort comprised telemetry records from $n = 87$ active implantable pacemakers. The dataset represents long-term, irregular real-world clinical logs with a median input history length of 123.0 clinic visits per patient, and a median test evaluation depth of 14.0 visits. The quantitative metrics are summarized in Table 4.1.

Table 4.1: Retrospective Cohort Validation Performance Summary ($n = 87$ patient devices)

Metric Category	**Mean**	**Median**
Overall Error Profiles		
State-of-Charge MAE (%)	6.6324	5.0556
State-of-Charge RMSE (%)	6.9155	5.4691
Stratified Capacity Tracking Errors		
Stratified 20% to 10% SoC MAE (%)	7.3644	7.4170
Stratified Below 10% SoC MAE (%)	6.1544	5.0300
Remaining Useful Life (RUL) Error (10% Threshold)		
Uncleaned Dataset Error (days)	297.9	54.7
Uncleaned Dataset Error (months)	9.79	1.80
Cleaned Dataset Error (days)[1]	73.0	54.2
Cleaned Dataset Error (months)[1]	2.40	1.78

The cohort-wide mean absolute error (MAE) is 6.6324% SoC with a median error of 5.0556% SoC, while the root mean squared error (RMSE) is 6.9155% SoC (mean)

and 5.4691% SoC (median). Under stratified analysis, tracking error is slightly higher when the battery is in the 20% to 10% SoC range (mean MAE of 7.3644%) compared to the critical zone below 10% SoC (mean MAE of 6.1544%).

The uncleaned Remaining Useful Life (RUL) prediction error at the 10% critical battery threshold has a mean of 297.9 days (9.79 months) and a median of 54.7 days (1.80 months), skewed by a maximum error of $12,219.5$ days. This extreme error is traced to telemetry date corruptions in 2 patient devices. Upon removing these two corrupted outliers, the cleaned RUL prediction error drops to a mean of 73.0 days (2.40 months) and a median of 54.2 days (1.78 months), with a maximum observed error of 616.8 days.

To illustrate the variability of the estimation model across different patient profiles, Figure 4.1 provides a side-by-side comparison of a representative high-performing fit and a representative poor-performing fit.

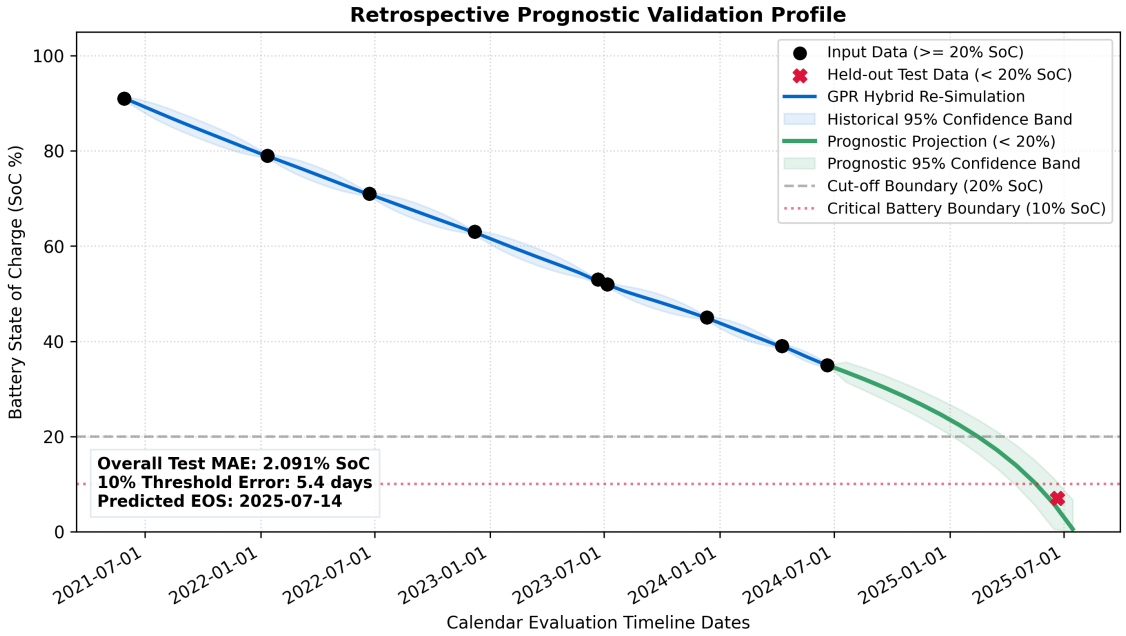
4.2 Computational Scaling and Latency Benchmarks

To evaluate the operational suitability of deploying the hybrid state observer in a cloud-native clinical infrastructure, a computational latency benchmark was performed. The execution times of the microservice were evaluated across direct container-to-container routing (Local Docker Mesh) and secure public cloud tunnel routing (Online ngrok Tunnel). Telemetry history payloads were scaled across five observation sizes: $N \in \{1, 50, 100, 500, 1000\}$ points, with $M = 30$ request runs per size. The experimental results are compiled in Table 4.2.

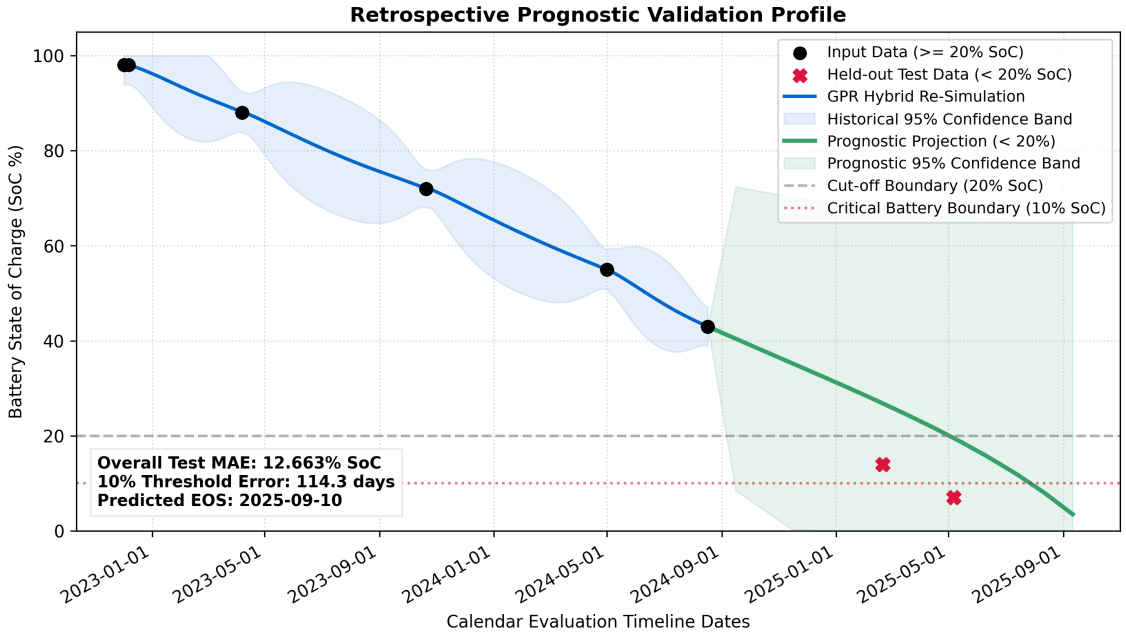
Table 4.2: Computational Latency and Network Overhead Metrics ($M = 30$ request runs per payload size)

Environment	Payload Size (N)	Mean (s)	Cold-Start (s)	Std Dev (s)	SEM (s)	p95 (s)
Local Docker	1	0.3822	0.4406	0.0242	0.0044	0.4174
	50	0.2767	0.2907	0.0127	0.0023	0.2981
	100	0.3520	0.3421	0.0147	0.0027	0.3754
	500	1.0949	1.1070	0.0641	0.0117	1.1993
	1000	2.4107	2.2925	0.2387	0.0436	2.7552
Online ngrok	1	0.5337	0.6592	0.0350	0.0064	0.5945
	50	0.4661	0.4976	0.0140	0.0026	0.4930
	100	0.5795	0.5981	0.0179	0.0033	0.6111
	500	1.4360	1.4186	0.0673	0.0123	1.5639
	1000	2.8509	2.8162	0.0995	0.0182	3.0117

In direct local container communication, the steady-state mean processing latency ranges from 0.4365 seconds at $N = 1$ to 2.5511 seconds at $N = 1000$ observation



(a) Representative High-Performing Fit



(b) Representative Poor-Performing Fit

Figure 4.1: Comparison of cohort estimation profiles showing representative best and worst absolute error tracks against clinical ground truth.

points. For the public cloud edge route, the latency increases, ranging from 0.5851 seconds ($N = 1$) to 3.1584 seconds ($N = 1000$). The average network overhead added by the public ngrok Edge Tunnel across all test brackets is +0.247 seconds.

Cold-start latency, measured during connection and model initialization on the first request of each size, is consistently higher than steady-state latency. On the local docker mesh, the cold-start latency is 0.6545 seconds at $N = 1$ (1.50× steady-state mean), and reaches 2.7245 seconds at $N = 1000$ points. Over the public ngrok tunnel, the cold start ranges from 0.6981 seconds ($N = 1$) to 3.1575 seconds ($N = 1000$).

The computational scaling and public edge tunnel overhead are visualized in the comparative latency plot in Figure 4.2.

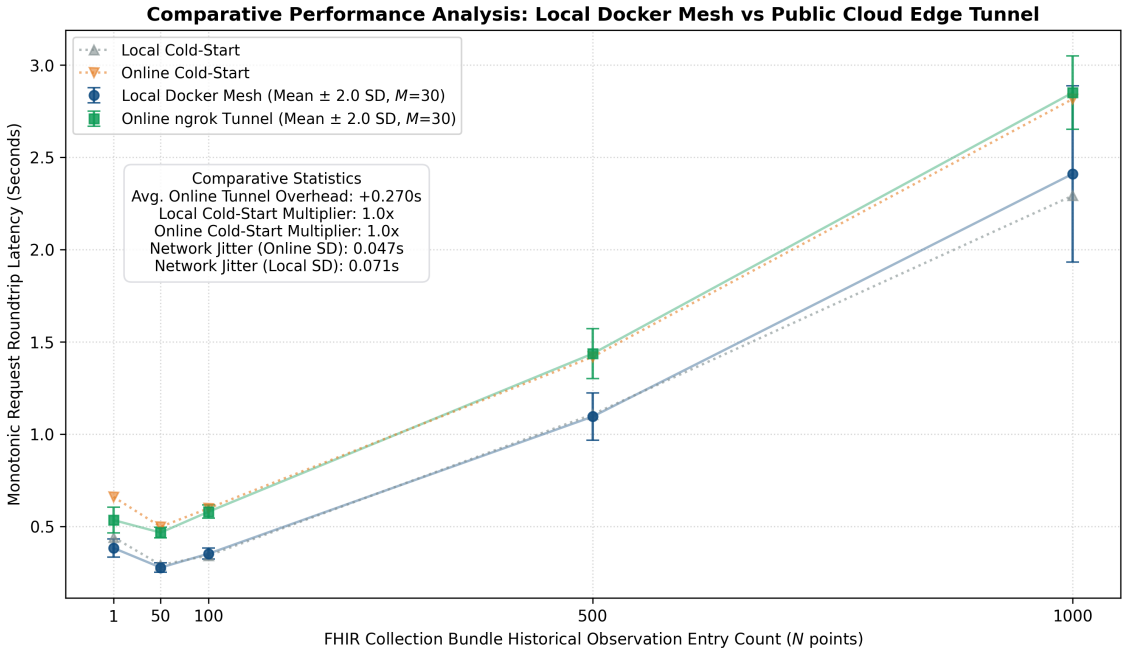


Figure 4.2: Comparative performance analysis overlay plotting direct local Docker mesh against public ngrok tunnel scaling trends, with separate steady-state error bounds ($\pm 1\sigma$), cold-start latencies, and summary statistics.

4.3 Robustness and Exception Validation

To evaluate the system's resilience under abnormal operational conditions and verify the performance of the Centralized Exception Boundary and API Gateway dependencies, adversarial stress testing was performed using the validation suite. The testing engine mutated valid HL7 FHIR bundles and authentication headers to inject common physical, chronological, structural, and security anomalies. The results of the seven distinct test scenarios (TC-01 through TC-07) are summarized in Table 4.3.

The evaluation demonstrated that the microservice successfully caught and isolated 100% of the injected anomalies. Under schema-level failures (TC-03 and TC-05),

Table 4.3: Adversarial Stress Testing and System Exception Handling Results

Targeted Ingress Mutation	HTTP Status	Validation Result Classification
Out-of-Bounds Metrics (SoC = 150%, Current = −2.0 mA)	422	Handled via Analytical Domain Exception Boundary
Chronological Timeline Chaos (Scrambled dates)	422	Blocked via Chronological Consistency Validation
Missing Structural Identifiers (Omitted envelope ID)	422	Blocked at Boundary via Pydantic Schema Check
Missing Algorithmic Constants (Omitted implant date)	422	Handled via Default Fallback Constant Enforcement
Type-Cast Sabotage (String in float field)	400	Blocked at Boundary via Pydantic Schema Check
Missing Authorization Token (No Auth header)	401	Blocked at Perimeter via Gateway Authentication
Malformed/Incorrect Credentials (Bad signature token)	401	Blocked at Perimeter via Gateway Authentication

the Pydantic parser rejected the payloads at the network border, returning status codes of `HTTP 422 Unprocessable Entity` or `HTTP 400 Bad Request` and preventing malformed parameters from reaching downstream mathematical routines. For domain-level violations (TC-01, TC-02, and TC-04), the internal exception handling layer intercepted the errors and returned clean, descriptive validation messages with `HTTP 422` status codes. For authentication breaches (TC-06 and TC-07), the gateway router stub aborted the connection prior to model parser execution, returning `HTTP 401 Unauthorized`. Crucially, no unhandled exceptions (`HTTP 500 Internal Server Error`) or container-level crashes were recorded across the entire testing cycle, verifying the reliability and security of the centralized exception boundary in safety-critical clinical environments.

4.4 Estimation Trajectory Case Studies

To verify the end-to-end integration and FHIR compliance of the analytical pipeline, representative patient diagnostic logs were executed through the simulation client. The microservice successfully validated the inbound FHIR Observation bundles, executed the hybrid observer personalization loops, and returned compliant HL7 FHIR DiagnosticReport resources.

Figures 4.3-4.11 display representative clinical trajectory reconstructions for patient devices. The plots show the model's ability to ingest sparse, irregular voltage telemetry, personalizing the physical prior using historical data, and projecting the battery depletion curve alongside uncertainty boundaries toward the 10% end-of-service (EOS) limit.

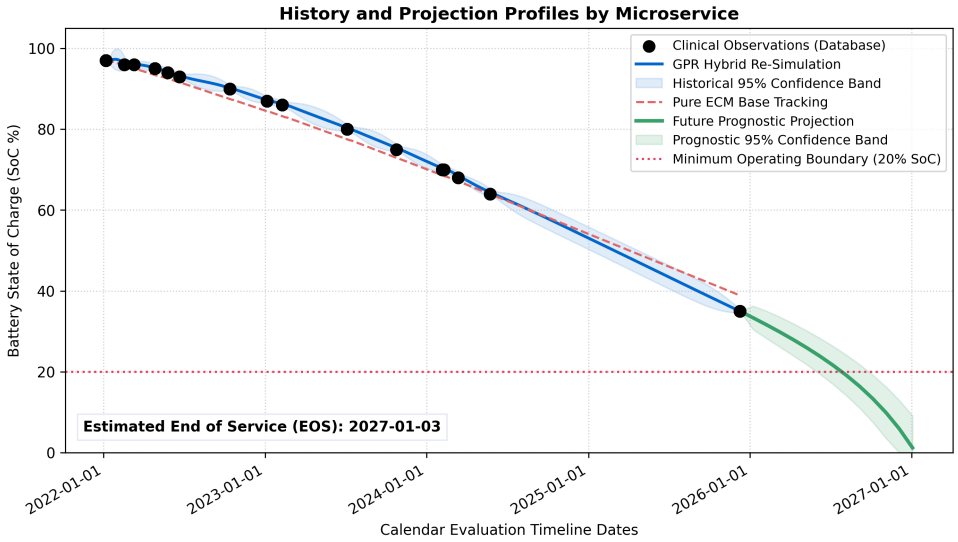


Figure 4.3: Reconstructed clinical telemetry profile for Clinical Trajectory Case A. The model personalizes the physical state bounds to match individual patient device trends.

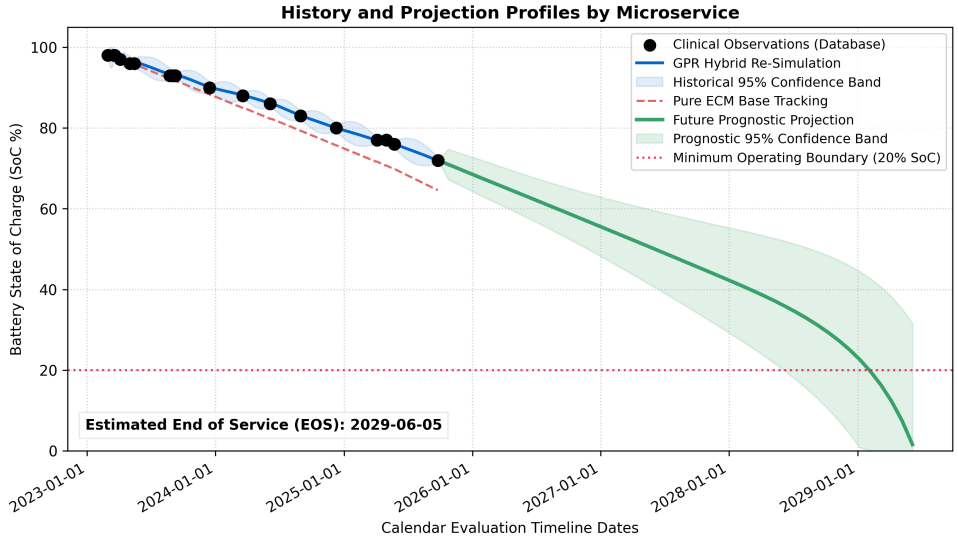


Figure 4.4: Reconstructed clinical telemetry profile for Clinical Trajectory Case B. The model projects battery depletion towards the end-of-service threshold.

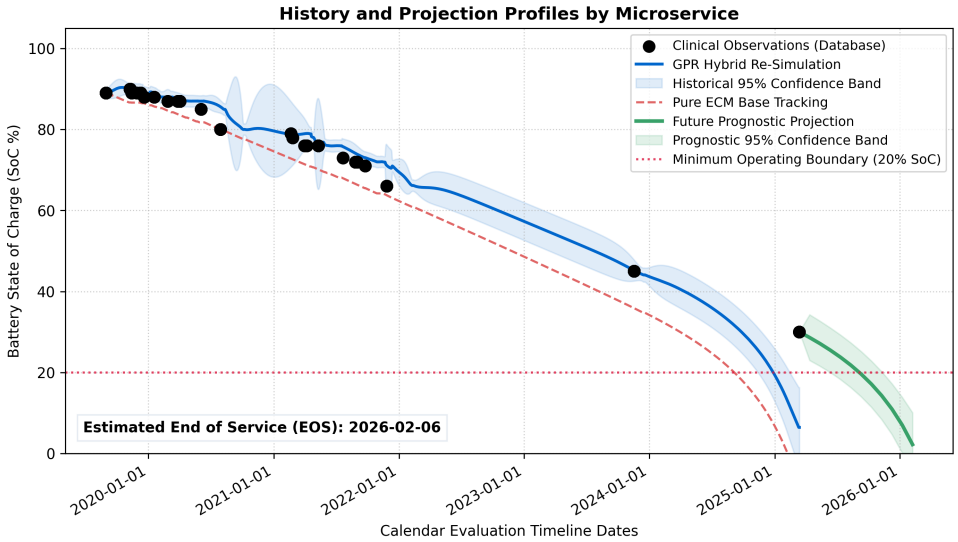


Figure 4.5: Reconstructed clinical telemetry profile for Clinical Trajectory Case C. The model projects battery depletion towards the end-of-service threshold.

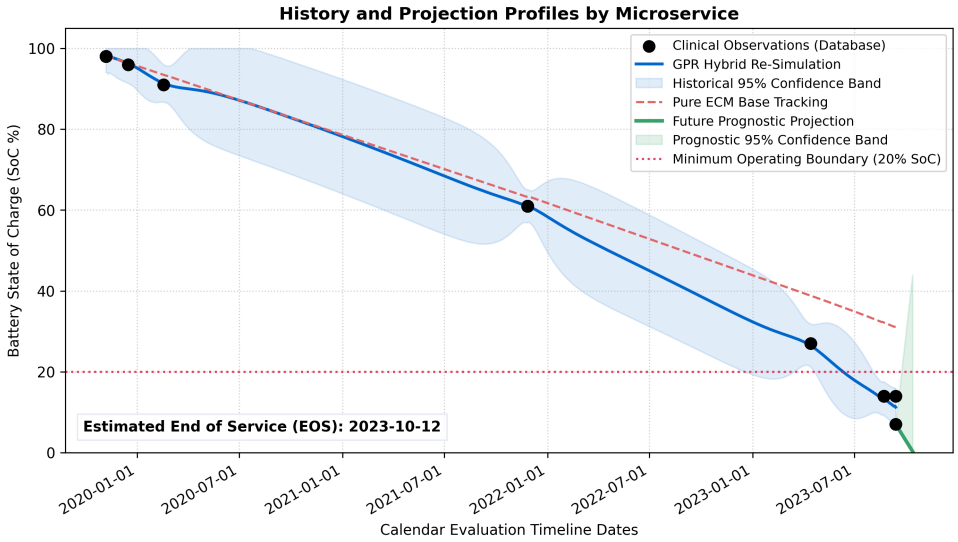


Figure 4.6: Reconstructed clinical telemetry profile for Clinical Trajectory Case D. The model projects battery depletion towards the end-of-service threshold.

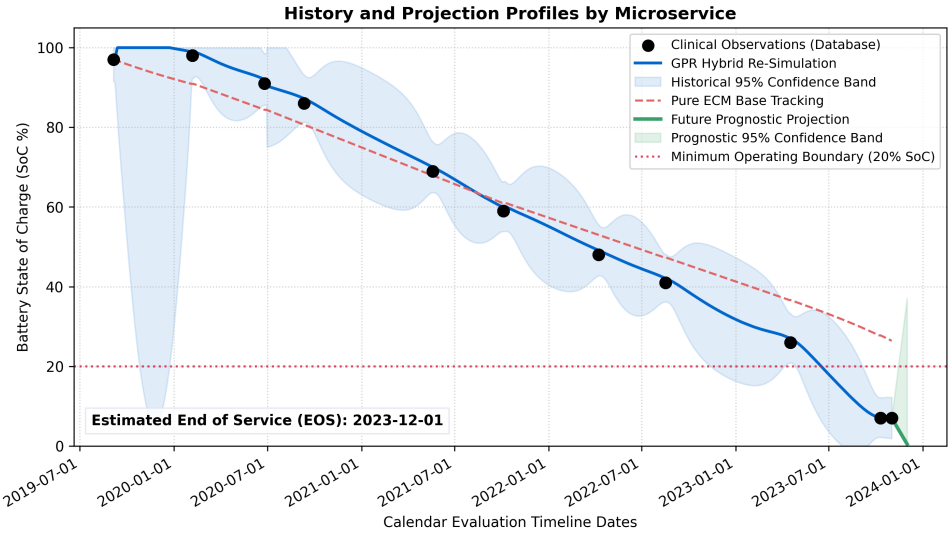


Figure 4.7: Reconstructed clinical telemetry profile for Clinical Trajectory Case E. The model projects battery depletion towards the end-of-service threshold.

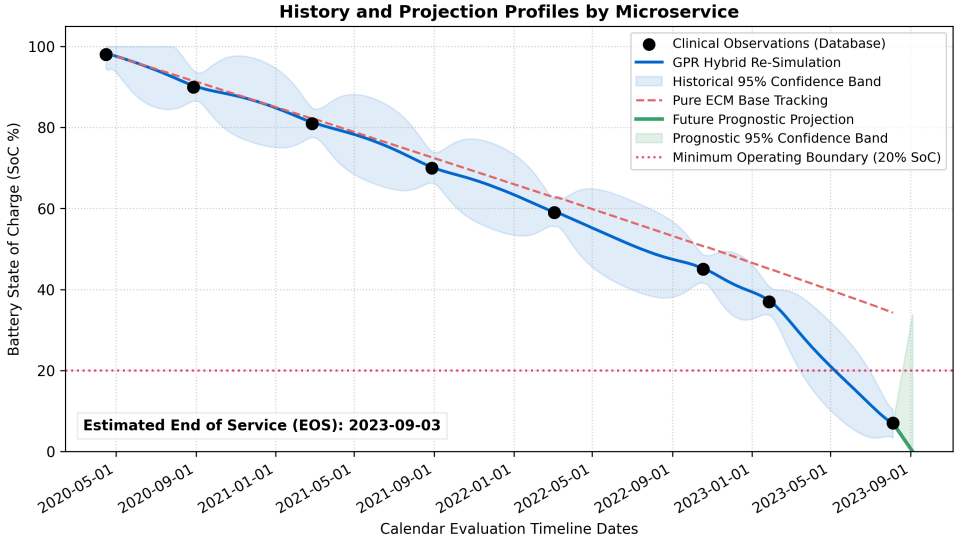


Figure 4.8: Reconstructed clinical telemetry profile for Clinical Trajectory Case F. The model projects battery depletion towards the end-of-service threshold.

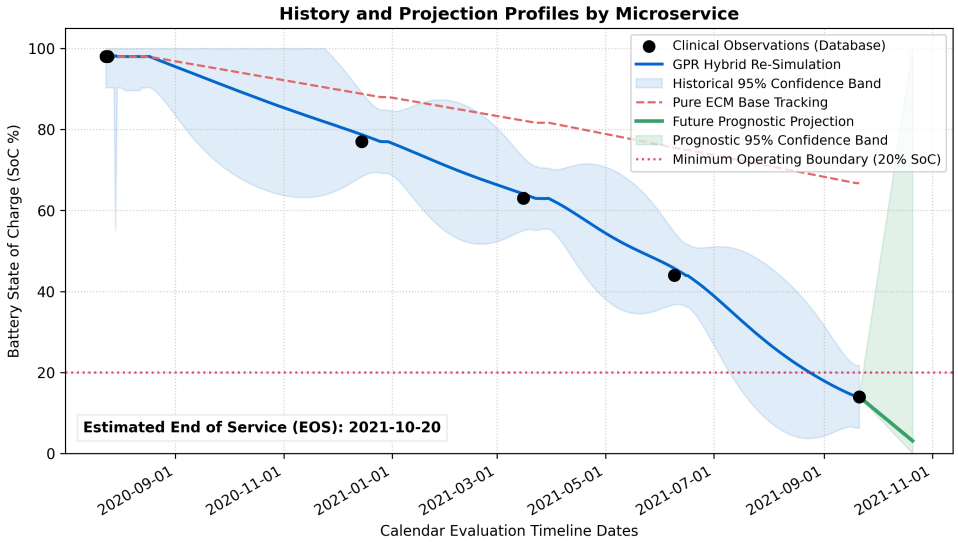


Figure 4.9: Reconstructed clinical telemetry profile for Clinical Trajectory Case G. The model projects battery depletion towards the end-of-service threshold.

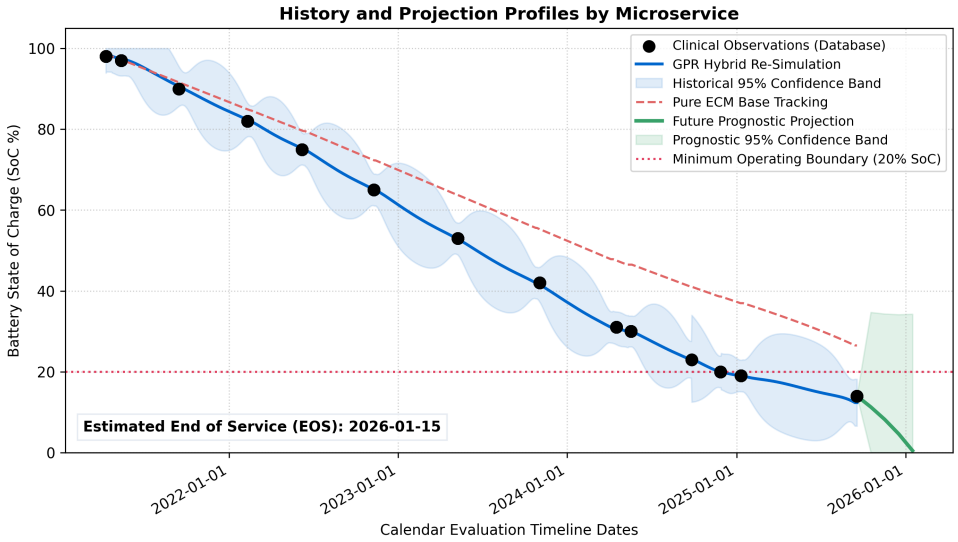


Figure 4.10: Reconstructed clinical telemetry profile for Clinical Trajectory Case H. The model projects battery depletion towards the end-of-service threshold.

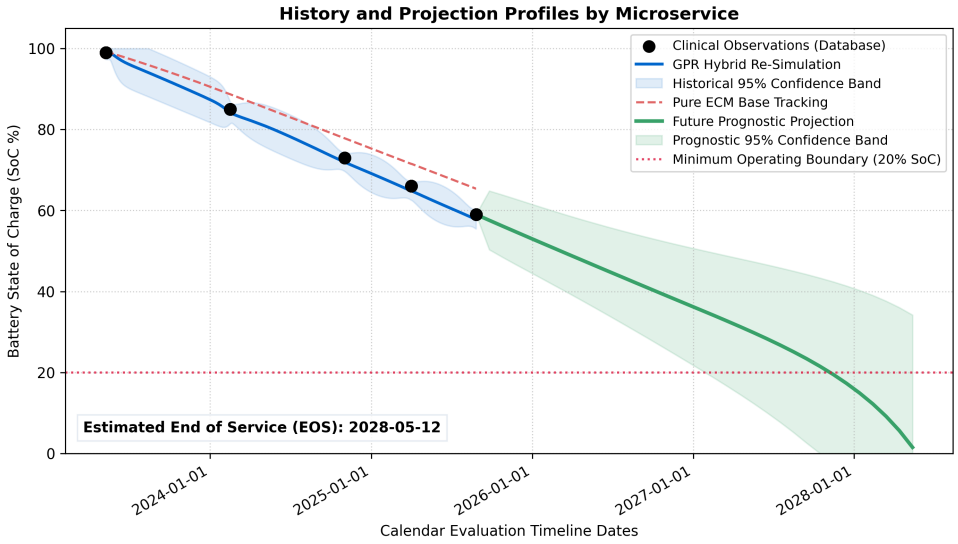


Figure 4.11: Reconstructed clinical telemetry profile for Clinical Trajectory Case I. The model projects battery depletion towards the end-of-service threshold.

4.5 System Performance and Ablation Analysis

The evaluation establishes a performance baseline by comparing the different architectural configurations. The predictive capacity, interval calibration fidelity, and computational overhead metrics recorded for each system permutation are detailed chronologically below.

The complete system performance profiles generated across the four structural configurations—encompassing the mechanistic physical baseline, the uncoupled data-driven estimator, the unconstrained hybrid framework, and the fully optimized proposed serial hybrid state observer—are consolidated in Table 4.4.

Table 4.4: Quantitative Performance Metrics Isolated Across Ablated Architectural Profiles

Model Configuration Profile	MAE (%) ↓	RMSE (%) ↓	PICP (Nominal: 0.95)	NLL ↓	Latency (ms) ↓
1. Pure Mechanistic (ECM Only)	4.611	6.421	N/A	N/A	**160.7**
2. Pure Data-Driven (Direct GPR)	6.370	11.320	Uncalibrated	N/A	N/A
3. Unconstrained Hybrid (No Guardrails)	1.400	5.766	1.00	-3.159	1156.0
4. Proposed Bounded Serial Hybrid	**1.346**	**5.654**	**0.98**	**-3.009**	1046.6

Profile 1: Pure Mechanistic Observer The implementation of the standalone second-order equivalent circuit model (ECM) operating without data-driven error corrections yielded a Mean Absolute Error (MAE) of 4.611% and a Root Mean Squared Error (RMSE) of 6.421% across the telemetry timelines. Prediction Interval Coverage Probability (PICP) and Negative Log-Likelihood (NLL) are not applicable (N/A) due to the deterministic formulation of the state equations. The structural compute footprint recorded a mean request processing latency of 160.7 ms.

Profile 2: Pure Data-Driven Baseline Assessment The standalone Gaussian Process Regressor mapping the uncoupled three-dimensional feature space

($\mathbf{X} = [I_{avg}, Z_{lead}, \text{Age}_{days}]$) directly to absolute capacity values scored an MAE of 6.370% paired with an RMSE of 11.320%. Due to spatial data-density dependency and the omission of physical bounding arrays, the predictive interval distribution exhibits a non-calibrated output state, making the calculation of a physical NLL value mathematically invalid.

Profile 3: Unconstrained Hybrid Framework Integrating the dual-layer serial framework (ECM $\rightarrow$ GPR) with the local device personalization offset (μ_{device}) but removing the custom mathematical guardrails—specifically disconnecting the cathode exhaustion pessimism multiplier and eliminating structural monotonicity clamps—produced an MAE of 1.400% alongside an RMSE of 5.766%. The empirical uncertainty metrics registered a PICP of 1.00 accompanied by a posterior evaluation score of -3.159 NLL. The matrix conditioning overhead increased the request rendering time to an average of 1156.0 ms.

Profile 4: Proposed Bounded Serial Hybrid Architecture The activation of the complete proposed serial hybrid architecture featuring integrated state-dependent parabolic boundaries, boundary constraints, and structural monotonicity clamps registered the lowest point tracking errors across the validation data vector, yielding an MAE of 1.346% and an RMSE of 5.654%. The corresponding interval coverage probability compressed from the unconstrained boundary down to a calibrated PICP of 0.98, yielding a posterior log-density score of -3.009 NLL. Mean runtime execution latency optimized to 1046.6 ms per clinical history request sequence.

5

Discussion

This chapter provides a critical evaluation and contextual interpretation of the empirical findings presented in Chapter 4. Within the broader medical technology landscape, healthcare systems and device manufacturers are navigating an inflection point: device telemetry is expanding rapidly, yet clinical software often relies on rigid, monolithic architectures and conservative, static battery look-up tables. By demonstrating how containerized, interoperable microservices can decouple complex Bayesian estimation from clinical databases, this work illustrates a scalable pathway for hospitals and manufacturers to modernize legacy device fleets without compromising patient safety. Building upon this high-level operational perspective, the subsequent sections bridge the gap between observed numerical performance and the electrochemical, clinical, and software architectural mechanisms that govern the hybrid system.

5.1 Battery Estimation and Diagnostics

The performance of the hybrid state observer must be evaluated in the context of the electrochemical properties of implantable medical device (IMD) batteries and the practical constraints of clinical telemetry.

5.1.1 Hybrid Modeling Rationale

The performance gap between the decoupled baselines and the proposed serial hybrid architecture (MAE = 1.346%) stems directly from how the models handle the weak observability of the primary Li/CF_x flat voltage plateau. A pure data-driven model (Profile 2) lacks a conservation-of-charge anchor. Because the battery's terminal voltage remains virtually flat for approximately 85% to 90% of its active lifecycle, raw telemetry signals do not provide sufficient gradients ($\mathrm{d}U/\mathrm{d}Q \approx 0$) for a standalone GPR to map terminal voltage directly to an absolute State of Charge (SoC). This lack of physical structure explains the elevated baseline errors (MAE = 6.37%, RMSE = 11.32%) observed during testing. Conversely, while a pure mechanistic prior (Profile 1) tracks charge depletion continuously via coulomb counting, it lacks the capacity to adapt to individual parameter drift and patient-specific load profiles over decadal clinical horizons, resulting in cumulative tracking drift (MAE = 4.611%).

The serial combination of a second-order (2-RC) Equivalent Circuit Model (ECM) and a non-parametric Bayesian corrector resolves these individual limitations through a systematic division of labor. The deterministic state-space equations govern state propagation during long unobserved periods, enforcing thermodynamic monotonicity. Concurrently, the GPR conditions and corrects the residual errors (ϵ_k) exclusively when sparse telemetry snapshots become available during clinical visits.

Alternative sequential architectures, such as replacing the GPR layer with a deep Recurrent Neural Network (RNN) or a Long Short-Term Memory (LSTM) network, could prove problematic with this clinical data reality. LSTMs require high-frequency, uniformly sampled sequences to maintain internal recurrent weights and prevent gradient issues [15]. Forcing a sequential neural network to train on highly irregular clinical visit logs spanning months or years introduces severe gradient instability and requires aggressive, non-physical interpolation data-filling strategies. Similarly, utilizing an online tracking filter such as an Extended Kalman Filter (EKF) to dynamically update physical parameters fails because the time intervals between clinical updates (Δt_{gap}) are too large. Over hundreds of days without a measurement checkpoint, the unguided state covariance (P_k) inside an EKF expands exponentially, leading to severe numerical divergence and filter instability at the next clinical touchpoint.

To circumvent the limitations of training models on sparse clinical data, a domain-transfer paradigm utilizing transductive transfer learning could be considered [27]. Under this framework, SOH estimation models (such as multiple linear regression or random vector functional link networks) are trained on laboratory cycle and calendar aging data, and subsequently adapted to field operation (target domain) by utilizing Kernel Mean Matching (KMM) to correct for covariate shift without requiring direct capacity labels in the target domain. While this approach effectively leverages laboratory datasets to manage field data discrepancies, it introduces distinct challenges when applied to active implants. The optimization of the KMM weights requires comparing the feature distributions of the source and target domains globally. In a clinical database where patient telemetry is retrieved on-demand and on an individual basis, compiling a representative target distribution to compute KMM weights is operationally difficult. Furthermore, the adapted models lack the native, calibrated uncertainty quantification provided by Gaussian Process Regression. Knowing the prediction confidence (uncertainty) is as clinically vital as the point estimate itself to establish safety-critical replacement margins.

5.1.2 Kernel and Parameter Adaptation

A critical modeling decision within the GPR corrector layer is the selection of the covariance kernel. Across all evaluated hybrid configurations, a Matérn 5/2 kernel was utilized to define the prior over functions. This kernel demonstrated superior tracking and generalization performance compared to the more commonly applied Squared Exponential (RBF) kernel. The RBF kernel, being infinitely differentiable, generates extremely smooth sample paths. While this smoothness is theoretically appealing, it introduces a severe over-smoothing effect at critical battery transition

phases, such as the sudden voltage knee encountered during cathode exhaustion or local transient behaviors.

To evaluate whether a more responsive covariance structure could improve tracking under high local volatility (e.g., sensor noise or load current fluctuations), a Matérn 1/2 (exponential) kernel was tested. Although a Matérn 1/2 (exponential) kernel, which is continuous but non-differentiable, can theoretically capture highly volatile, rough local changes, it proved overly sensitive to telemetry noise. This sensitivity led to non-physical, jagged fluctuations in the posterior mean and degraded the model's global forecasting capability. Consequently, the Matérn 5/2 kernel—which is twice-differentiable—represents an optimal mathematical compromise. It enforces sufficient smoothness to capture the long-term degradation trend while retaining the necessary flexibility to adapt to localized non-linearities and electrochemical transitions without over-smoothing.

A key methodological challenge in this research was the inability to perform destructive physical battery characterization tests—such as Electrochemical Impedance Spectroscopy (EIS) or laboratory-controlled pulse discharge profiles—on the clinical devices. Consequently, the second-order ECM parameters $(R_0, R_1, C_1, R_2, C_2)$ had to be initialized using normative, literature-derived constants rather than cell-specific laboratory measurements. In a conventional tracking system, this introduces a severe systematic parameter mismatch, as individual variations in internal resistance and capacitance are not physically accounted for.

The evaluation results demonstrate that the proposed hybrid architecture is highly robust against this physical parameter uncertainty. By routing the literature-derived ECM as a prior, the framework establishes a chemically plausible baseline tracking shape. The GPR corrector layer is then tasked with modeling the residual error $(\epsilon_k = U_{meas,k} - U_{ecm,k})$. In this configuration, the primary role of the GPR shifts from learning the complete degradation trajectory to acting as an adaptive compensator for the imperfect physical prior. The data-driven layer successfully absorbs the systematic error introduced by the generalized literature constants, personalizing the tracking model to the specific impedance and capacity characteristics of the patient's device without requiring custom laboratory calibration.

5.1.3 Observability and Telemetry Dynamics

The empirical results compiled in Table 4.1 present a counter-intuitive finding: the mean tracking error in the 20% to 10% SoC range (MAE = 7.3644%) is higher than the tracking error observed in the critical zone below 10% SoC (MAE = 6.1544%). Intuitively, error should accumulate and drift over time, making estimation errors larger as the prediction moves further away from the initialization state.

This paradox is explained by the non-linear relationship of battery observability:

1. **plateau Phase (20–10%):** In this range, the primary cell chemistry is still operating on its flat discharge plateau. The voltage gradient is extremely weak $(\mathrm{d}U/\mathrm{d}SoC \approx 0)$. In this region, measurement noise, temperature fluctuations, and telemetry calibration errors overwhelm the true chemical signal. Because

changes in SoC produce negligible changes in voltage, the observability of the internal state is minimal, limiting the capacity of the GPR corrector to perform precise updates.

2. **Knee Phase (Below 10%):** Once the cell drops below 10% SoC, it exits the plateau and enters the steep chemical "knee." Here, the voltage gradient drops rapidly and monotonically ($\mathrm{d}U/\mathrm{d}SoC \ll 0$). This sharp gradient increases the observability of the cell state. A small change in remaining capacity results in a distinct, measurable voltage drop that exceeds the telemetry noise floor. The mathematical prior and the GPR corrector utilize this strong physical signal to self-correct tracking errors, resulting in lower empirical tracking deviations in the critical zone.

While the electrochemical literature establishes that primary $\mathrm{Li/CF}_x$ cells are subject to a pronounced "voltage delay" transient after long idle periods due to anode passivation, the real-world clinical telemetry dataset analyzed in this project did not display this behavior. The empirical voltage snapshots did not exhibit the characteristic transient voltage dips and subsequent recoveries that typically follow zero-current storage.

There are two primary engineering explanations for this divergence from laboratory-established chemical behavior:

1. **Continuous Quiescent Current Draw:** Active IMDs are never under true zero-current conditions. Internal circuitry, pacing engines, and timing circuits maintain a continuous quiescent current draw in the microampere (μA) range. This continuous, low-level discharge keeps the lithium anode active and may prevent the growth of the thick, insulating lithium fluoride passivation layer that typically forms during complete zero-current storage.
2. **Device-Level Data Filtering:** The internal telemetry hardware or the microcontroller of the clinical programmer may perform hardware-level low-pass filtering or averaging on the voltage sensors before committing the value to the clinical log. This smoothing mechanism would eliminate high-frequency transient dips, recording only the stabilized terminal voltage.

From a modeling perspective, the absence of voltage delay in the dataset simplified the state estimation task. Because the GPR corrector was insulated from transient, passivation-induced voltage drops, it did not need to partition its learning capacity to separate transient chemical anomalies from long-term capacity fade.

5.2 Safety and Clinical Risk

5.2.1 Thermodynamic Constraints

Because GPR is a non-parametric, probabilistic regressor, its raw predictions are governed strictly by spatial data density rather than physical conservation laws. Under noisy telemetry conditions—such as sensor noise or vertical calibration anomalies during clinical visits—an unconstrained GPR model can output posterior mean predictions (μ_{gpr}) that fluctuate upward. This mathematically translates to the battery

physically *gaining* capacity over time, which is electrochemically impossible for a non-rechargeable, primary cell.

To prevent these non-physical state transitions, the integration of algorithmic guardrails (Equations 3.8 and 3.9) is a safety-critical necessity. By nesting the hybrid state estimate inside a hard floor-and-ceiling envelope ($[0.0, 1.0]$) paired with a backward-looking historical minimizer, the framework programmatically enforces thermodynamic monotonicity ($SoC(t_k) \leq SoC(t_{k-1})$). This structural configuration constrains the statistical corrector to operate strictly within physical boundaries. The resulting bounded hybrid observer successfully filters out high-frequency telemetry noise and sensor anomalies, maintaining numerical and chemical stability even under highly sparse or corrupted clinical ingress streams.

The physical utility of these thermodynamic constraints is further highlighted when analyzing the interval calibration of the models. For the unconstrained hybrid framework (Profile 3), the Prediction Interval Coverage Probability (PICP) reached a value of 1.00. While a coverage of 100% might superficially appear superior to the 0.98 achieved by the proposed bounded model (Profile 4), in the context of statistical calibration, a PICP of 1.00 relative to a nominal target of 0.95 indicates a state of severe underconfidence and over-conservatism. Without physical boundaries, the unconstrained model expands its predictive variance (σ^2_{gpr}) excessively in regions of sparse telemetry, yielding prediction intervals that are too wide to provide actionable clinical utility. Conversely, by constraining the estimation within the floor-and-ceiling envelope and enforcing a non-increasing trajectory, the proposed bounded serial hybrid (Profile 4) compresses the posterior variance to a physically plausible range. This results in a calibrated PICP of 0.98, closer to the nominal 0.95 threshold, offering a tighter and more informative boundary for remaining useful life predictions.

5.2.2 Clinical Risk Analysis

Deploying a state-estimation engine in safety-critical medical environments demands an evaluation of how estimation errors affect clinical outcomes. The clinical risks associated with forecasting errors are highly asymmetric:

- **Overestimation (False Optimism / Delayed Replacement):** If the model overestimates the remaining capacity (underestimating tracking error), it may project a delayed ERI date. This introduces the risk of the device running out of battery before surgical replacement can be scheduled, causing sudden loss of pacing therapy and life-threatening events for device-dependent patients.
- **Underestimation (False Pessimism / Premature Replacement):** If the model underestimates capacity, it may project an early ERI. This leads to premature surgical intervention to replace the IMD. Every invasive surgical procedure carries acute risks, including pocket infection, hardware complications, and vascular damage.

This risk asymmetry highlights why point-predictions alone are insufficient for clinical decision-making. The calibrated uncertainty boundaries provided by the GPR

corrector (achieving a PICP of 0.98) provide a robust safety buffer, enabling clinicians to schedule surgical replacements based on conservative lower bounds rather than optimistic averages.

Furthermore, because the evaluation metrics employed in this study (namely the overall cohort MAE and RMSE) aggregate errors symmetrically, they do not distinguish between optimistic and pessimistic estimation trajectories. Consequently, there is currently no quantitative basis to evaluate the model's performance on either side of this risk divide, which represents a critical area for future research. In clinical practice, the consequences of these error directions are profoundly unequal. A pessimistic underestimation may result in an unnecessarily early battery replacement surgery, carrying with it the usual risks of performing such an invasive procedure, as well as the associated healthcare costs. Conversely, an optimistic overestimation carries far more severe risks, as it may result in the device running out of battery before surgical replacement is scheduled. This would cause the patient to go without potentially life-critical therapy for some time before the condition is identified and a replacement surgery is planned and performed. Incorporating asymmetric loss functions during model training or evaluation could serve as a valuable extension to mathematically penalize optimistic errors more heavily.

5.2.3 Trajectory and Data Boundaries

The database filtering criteria adopted during cohort selection—specifically the minimum threshold of 30 clinical entries and the 10% SoC evaluation threshold—were determined based on statistical and clinical necessities:

- **Data Density for GPR Calibration:** Non-parametric Bayesian models rely on sufficient training observations to optimize their hyperparameters (length-scales and signal variances) via Maximum Marginal Likelihood Estimation. If a device diagnostic history contains fewer than 30 data points, the GPR prior calibration is highly sensitive to telemetry noise, resulting in poor regression performance and uncalibrated prediction intervals. The 30-entry minimum ensures mathematical stability.
- **10% Elective Replacement Indicator (ERI) Boundary:** In clinical practice, the 10% SoC threshold represents the critical milestone for elective replacement. The primary focus of the Remaining Useful Life (RUL) calculation must be centered on the accuracy of predicting when the device crosses this 10% ERI limit. Beyond this threshold, battery voltage declines rapidly, leaving limited operational reserve before reaching the End of Service (EOS) limit.

A detailed review of the individual trajectory reconstructions (Figures 4.3–4.11) reveals important insights into the behavior of the hybrid estimator under diverse real-world telemetry conditions, highlighting both stability characteristics and structural challenges:

- **Mean Stability vs. Prediction Interval Inflation:** In several examples (specifically Cases D, E, F, and G), the future prognostic uncertainty boundaries expand significantly as the forecast horizon extends. Crucially, however,

the posterior mean prediction remains highly stable and continues to follow a chemically realistic decay path without displaying non-physical upward drift. While this stability is encouraging, the rapid expansion of the uncertainty envelope explains the high overall cohort PICP values (0.98 and 1.00). Under sparse telemetry, the prediction intervals can become wide enough to envelop the true observations by default, even when the mean prediction deviates from the ground truth. This behavior is clearly visible in the cohort's representative poor fit (Figure 4.1b), where the mean estimate fails to track the test points but remains covered by the wide uncertainty envelope.

- **Diagnostic Value of Historical Trajectory Auditing:** In Case C, the historical resimulation trajectory drops abnormally to near-zero before snapping back up when anchored by the final observed clinical measurement. From a future-forecasting perspective, the prediction from the last checkpoint looks reasonable; however, this anomaly demonstrates the diagnostic value of simulating and returning the complete historical trajectory. Visualizing the historical path exposes local GPR over-corrections and numerical edge-case behaviors that would remain hidden if the microservice only returned future projections.
- **Non-Linear Slope Changes and the 30–40% SoC Knee:** Many of the device histories exhibit an abrupt increase in the rate of battery discharge around the 30%–40% SoC range. This slope change—which may stem from cell-level electrochemical processes or clinical therapy setting adjustments—disturbs the linear trend, creating an early capacity knee. When the telemetry aligns in a more consistent, straight line (such as Cases A and B), the uncertainty bounds and mean predictions are tighter and more accurate.
- **Data Density Constraints (Comparison of Case B and Case I):** The impact of telemetry density is highlighted by comparing Case B and Case I. Both cases have histories that end at relatively high battery capacities ($\approx 60\%$ SoC). However, because Case B features a much higher density of historical observations, its future projection uncertainty remains tightly constrained. In contrast, Case I has very few observations, causing its future uncertainty boundaries to expand rapidly due to the lack of local conditioning points.

5.3 Future Considerations

5.3.1 Software Architecture and Constraints

A fundamental design decision of this thesis was to deploy the battery estimation engine as an external cloud-native microservice rather than executing the code natively on the IMD's internal microcontroller. This choice is dictated by the physical constraints of implantable hardware:

- **Offline Architecture:** Active IMDs are closed, safety-critical systems with no direct connection to the internet or external wireless networks. Introducing active network radios directly on the implant would accelerate battery depletion and expose the safety-critical pacing firmware to security risks.

- **Clinical Query Workflow:** In practice, telemetry data is retrieved only when the device user visits a clinic. An external programmer wand uses near-field inductive coupling or short-range RF to query the IMD, extracting its operational parameters and saving them to a local clinical terminal. Once extracted, the clinical programmer transmits these records to a secure server where calculations are executed.
- **Mathematical Complexity:** The hybrid engine's GPR corrector relies on matrix inversion operations that scale as $\mathcal{O}(N^3)$, where N is the number of historical observation points. Performing floating-point matrix inversions and storing historical arrays exceeds the memory capacity and processor speeds of low-power IMD microcontrollers. Offloading the computation to an external microservice protects the implant's resources while enabling advanced statistical calculations.

The choice of a decoupled Microservice Architecture (MSA) using Python and FastAPI was evaluated against alternative software patterns:

- **Monolithic Deployment:** A monolithic architecture combines the telemetry frontend and the mathematical engine into a single application. While this simplifies deployment, it requires the web hosting framework to load large numerical libraries (pandas, scikit-learn, joblib). Under major concurrent request volumes, the CPU-intensive GPR regression loops could starve the frontend threads of CPU and memory, causing interface timeouts. Separating the engine into a microservice isolates the compute-heavy mathematical processes, ensuring the client frontend remains responsive.
- **Service-Oriented Architecture (SOA) with Enterprise Service Bus (ESB):** SOA patterns rely on central service buses to orchestrate communication. This model introduces communication latency and data transformation overhead. For a specialized microservice that only requires point-to-point verification of specific device telemetry files, an ESB adds unnecessary complexity. Furthermore, SOA requires the service to be put in a much larger context/system, while a microservice has much better possibility to be developed in an isolated manner.
- **FastAPI Decoupling Rationale:** Decoupling the network layer (`routers/battery.py`) from the internal execution layer (`services/hybrid_engine.py`) ensures that the mathematical code remains protocol-agnostic. Shifting the public communication channel from REST/HTTP to high-speed gRPC or AMQP message queues can be executed without modifying the underlying electrochemical state equations.

5.3.2 Interoperability and Middleware

Standardizing medical data across fragmented hospital networks requires strict adherence to interoperability protocols. Building the microservice interfaces upon the HL7 FHIR standard represents an active engineering design decision.

While custom JSON or flat CSV payloads would simplify parsing, they would fail to

integrate with modern Electronic Health Record (EHR) networks. Utilizing FHIR `Bundle` and `Observation` resources containing standardized IEEE 11073 codes guarantees semantic consistency. This approach isolates the core mathematical engine from data formatting dependencies. However, because many clinical environments still utilize legacy systems (such as pipe-delimited HL7 v2.x messages), deployment requires a middleware integration engine (e.g., Mirth Connect) to translate legacy inputs into FHIR observation schemas before sending them to the microservice endpoint.

5.3.3 Production Scaling and Security

Because GPR regression is computationally intensive, the service is vulnerable to Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks. An attacker could flood the endpoint with high-volume requests containing large historical datasets ($N > 1000$), causing CPU starvation and blocking clinical access.

To mitigate this risk, the deployment architecture must integrate rate-limiting controls at the network boundary. Exposing the API through a reverse proxy (such as NGINX or Envoy) or an API Gateway (such as Kong) allows enforcing request quotas per API key or client IP address. Furthermore, since the service should only be accessible by authorized clinical servers and programmer terminals, network security should enforce strict IP whitelisting alongside Mutual TLS (mTLS) to encrypt and authenticate all server-to-server communication, preventing unauthorized execution.

FastAPI's asynchronous ASGI framework manages I/O operations without blocking the event loop. However, GPR matrix operations are CPU-bound and block execution threads during calculation. Running heavy workloads directly on the main event loop degrades performance under concurrent request loads.

To achieve production-grade concurrency, the architecture must implement a decoupled execution queue:

- **Task Queues:** The HTTP router should accept the inbound payload, validate the FHIR schema using Pydantic, generate a unique task ID, and offload the calculation task to a Celery distributed task queue backed by a Redis message broker. The microservice then returns a `202 Accepted` response to the client.
- **Worker Scaling:** Dedicated Celery worker containers, running on separate virtual machines or Kubernetes nodes, pull tasks from the queue and execute the mathematical engine. Computational capacity can scale dynamically by adding worker nodes (Horizontal Pod Autoscaler) based on CPU utilization and queue length.

Operating within clinical environments requires compliance with patient privacy laws, such as the Health Insurance Portability and Accountability Act (HIPAA) and the General Data Protection Regulation (GDPR). Although the core analytical microservice was designed to be stateless (omitting a persistent database to minimize security risks), logging is necessary to audit system health and diagnose operational issues.

To resolve the conflict between logging requirements and patient privacy, the logging layer must enforce strict data minimization:

- **Metadata-Only Logging:** The application logging framework must intercept and sanitize all outgoing write streams. Logs are restricted to operational metadata, including request timestamps, token hashes, model versions, processing latency, and standardized error codes.
- **PII/PHI Exclusion:** Raw patient health information (PHI) and personally identifiable information (PII)—including specific voltage tables, clinical visit dates, device serial numbers, patient ages, and raw FHIR payloads—must be excluded from persistent log files. This configuration allows technicians to audit application performance and trace software errors without storing sensitive clinical data.

6

Conclusion

This thesis presented the design, implementation, and empirical validation of a decoupled microservice backend hosting a hybrid Equivalent Circuit Model–Gaussian Process Regression (ECM–GPR) framework to estimate Remaining Useful Life (RUL) for primary Li/CFx cells in active implantable medical devices. By isolating heavy non-linear state estimation from clinical databases and resource-constrained edge hardware, the architecture resolves tensions between computational scalability, security, and clinical interoperability.

Regarding the first research question, an asynchronous, stateless FastAPI microservice was engineered and containerized with Docker. This decoupled infrastructure isolates the $\mathcal{O}(N^3)$ complexity of Gaussian Process Regression, preventing resource starvation during high-concurrency bursts. Latency benchmarks verified sub-second response times (mean $< 0.4\,\mathrm{s}$ for histories up to 100 visits), demonstrating the horizontal scalability required for modern digital health backends.

Regarding the second research question, the serial hybrid modeling framework significantly enhanced tracking fidelity under severe observability constraints. Combining a second-order ECM physical prior with a non-parametric GPR corrector achieved an MAE of 1.346% and RMSE of 5.654% across $n = 87$ active implants, outperforming mechanistic (4.611%) and data-driven (6.370%) baselines. Programmatic thermodynamic guardrails eliminated non-physical capacity recoveries, guaranteeing strictly monotonic degradation trajectories ($\mathrm{SoC}(t_k) \leq \mathrm{SoC}(t_{k-1})$).

Regarding the third research question, the validation strategy combined automated adversarial stress-testing with HL7 FHIR semantic standardization mapped to IEEE 11073 nomenclatures. The centralized exception handler contained 100% of injected network and payload anomalies without container crashes or unhandled 500 errors, confirming operational resilience under clinical edge failures.

In summary, this research provides an interoperable blueprint for embedding predictive intelligence into digital health infrastructures. While regulatory reviews and clinical trials remain essential prior to deployment, this work demonstrates how physics-informed statistical learning and microservice design advance implantable device lifetime management and patient safety.

7

References

[1] World Health Organization, *Global Strategy on Digital Health 2020–2025*. Geneva: World Health Organization, 2021, Licence: CC BY-NC-SA 3.0 IGO. [Online]. Available: https://iris.who.int/handle/10665/344249.

[2] A. Dang, "Real-world evidence: A primer," *Pharmaceutical Medicine*, vol. 37, no. 1, pp. 25–36, 2023. DOI: 10.1007/s40290-022-00456-6.

[3] A. Bahmani et al., "A scalable, secure, and interoperable platform for deep data-driven health management," *Nature Communications*, vol. 12, no. 5757, pp. 1–15, 2021. DOI: 10.1038/s41467-021-26040-1.

[4] A. Warrier, "Real-time ai integration architectures for hipaa-compliant healthcare data interoperability," *International Journal of Emerging Trends in Computer Science and Information Technology*, vol. WorldCompAI-25, no. 2025, pp. 74–81, 2025. DOI: 10.63282/WCAI25-128.

[5] European Parliament and Council of the European Union, "Regulation (EU) 2025/327 of 11 february 2025 on the European Health Data Space and amending Directive 2011/24/EU and Regulation (EU) 2024/2847," *Official Journal of the European Union*, vol. L, no. 2025/327, 2025. [Online]. Available: http://data.europa.eu/eli/reg/2025/327/oj.

[6] S. Mishra, "Architecting scalable ai-enabled enterprise systems: Lessons from healthcare application integration," *International Journal of AI, BigData, Computational and Management Studies*, vol. ICAIDSCT26, no. 2026, pp. 141–145, 2026. DOI: 10.63282/3050-9416.ICAIDSCT26-116.

[7] R. Hussein, L. Scherdel, F. Nicolet, and F. Martin-Sanchez, "Towards the European Health Data Space (EHDS) ecosystem: A survey research on future health data scenarios," *International Journal of Medical Informatics*, vol. 170, p. 104 949, 2023. DOI: 10.1016/j.ijmedinf.2022.104949.

[8] D. H. Vu, "Technique of receiving data from medical devices to create electronic medical records database," in *Soft Computing Applications and Techniques in Healthcare*, 1st, CRC Press, 2020, p. 30. DOI: 10.1201/9781003003496-10.

[9] H. Calderón-Gómez et al., "Evaluating service-oriented and microservice architecture patterns to deploy ehealth applications in cloud computing environment," *Applied Sciences*, vol. 11, no. 10, p. 4350, 2021. DOI: 10.3390/app11104350.

[10] M. Harshith et al., "Federated microservices architecture with blockchain for privacy-preserving and scalable healthcare analytics," *Scientific Reports*, vol. 16, no. 9023, pp. 1–20, 2026. DOI: 10.1038/s41598-026-39837-1.

[11] M. M. Islam, S. Nooruddin, F. Karray, and G. Muhammad, "Internet of things: Device capabilities, architectures, protocols, and smart applications in healthcare domain," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 3611–3641, 2023. DOI: 10.1109/JIOT.2022.3228795.

[12] A. Kliem, A. Boelke, A. Grohnert, and N. Traeder, "A reconfigurable middleware for on-demand integration of medical devices," *IRBM*, vol. 37, no. 4, pp. 198–209, 2016. DOI: 10.1016/j.irbm.2016.05.003.

[13] J. Meng, G. Luo, M. Ricco, M. Swierczynski, D.-I. Stroe, and R. Teodorescu, "Overview of lithium-ion battery modeling methods for state-of-charge estimation in electrical vehicles," *Applied Sciences*, vol. 8, no. 5, 2018. DOI: 10.3390/app8050659.

[14] J. Meng, G. Luo, and F. Gao, "Lithium polymer battery state-of-charge estimation based on adaptive unscented kalman filter and support vector machine," *IEEE Transactions on Power Electronics*, vol. 31, pp. 1–1, Jan. 2015. DOI: 10.1109/TPEL.2015.2439578.

[15] N. Jantharamin, "Battery modeling based on artificial neural network for battery control and management," *2018 21st International Conference on Electrical Machines and Systems (ICEMS)*, pp. 2111–2114, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:54440872.

[16] S. S. Zhang, D. Foster, J. Wolfenstine, and J. Read, "Electrochemical characteristic and discharge mechanism of a primary li/cfx cell," *Journal of Power Sources*, vol. 187, no. 1, pp. 233–237, 2009. DOI: 10.1016/j.jpowsour.2008.10.076.

[17] V. Krishnamurthy, "On the discharge mechanism in Li-CFx batteries," Mar. 2023. DOI: 10.1184/R1/22259371.v1.

[18] A. Warrier, "Ipaas solutions for healthcare enterprise integration: Cloud-native integration platforms for multi-system orchestration," *International Journal of Leading Research Publication*, vol. 3, no. 1, pp. 1–9, 2022. DOI: IJLRP22011770.

[19] S. Abedian, E. Yesakov, S. Ostrovskiy, and R. Hussein, "Streamlining wearable data integration for EHDS: A case study on advancing healthcare interoperability using Garmin devices and FHIR," *Frontiers in Digital Health*, vol. 7, p. 1 636 775, 2025. DOI: 10.3389/fdgth.2025.1636775.

[20] M. Kasparick, S. Schlichting, F. Golatowski, and D. Timmermann, "New ieee 11073 standards for interoperable, networked point-of-care medical devices," in *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2015, pp. 1721–1724. DOI: 10.1109/EMBC.2015.7318709.

[21] M. Kasparick, S. Schlichting, F. Golatowski, and D. Timmermann, "Medical dpws: New ieee 11073 standard for safe and interoperable medical device communication," in *2015 IEEE Conference on Standards for Communications and Networking (CSCN)*, 2015, pp. 212–217. DOI: 10.1109/CSCN.2015.7390446.

[22] M. Kasparick et al., "Or.net: A service-oriented architecture for safe and dynamic medical device interoperability," *Biomedical Engineering / Biomedizinische Technik*, vol. 63, no. 1, pp. 57–68, 2018. DOI: 10.1515/bmt-2017-0020.

[23] European Parliament and Council of the European Union, "Regulation (EU) 2017/745 of 5 april 2017 on medical devices, amending Directive 2001/83/EC, Regulation (EC) no 178/2002 and Regulation (EC) no 1223/2009 and repealing Council Directives 90/385/EEC and 93/42/EEC," *Official Journal of the European Union*, vol. L, no. 117, pp. 1–175, 2017. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2017/745/oj.

[24] European Parliament and Council of the European Union, "Regulation (EU) 2024/1689 of 13 june 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act) and amending certain legislative acts," *Official Journal of the European Union*, vol. L, 2024. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2024/1689/oj.

[25] U. Krishnamoorthy, P. Lakshmipathy, M. Ramya, and H. H. Fayek, "Navigating the future of healthcare with innovations and challenges in implantable battery technology for biomedical devices," *Discover Applied Sciences*, vol. 6, 2024. DOI: 10.1007/s42452-024-06278-2.

[26] L. Ungurean, G. Cârstoiu, M. V. Micea, and V. Groza, "Battery state of health estimation: A structured review of models, methods and commercial devices," *International Journal of Energy Research*, vol. 41, no. 2, pp. 151–181, 2017. DOI: 10.1002/er.3598.

[27] S. B. Vilsen and D.-I. Stroe, "Transfer learning for adapting battery state-of-health estimation from laboratory to field operation," *IEEE Access*, vol. 10, pp. 26 514–26 528, 2022. DOI: 10.1109/ACCESS.2022.3156657.

[28] J. Yi, X. Zhou, J. Zhang, and Z. Li, "A hybrid method for soc estimation of power battery," *Journal of Control Science and Engineering*, vol. 2021, pp. 1–8, Nov. 2021. DOI: 10.1155/2021/6758679.

[29] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press, 2006. [Online]. Available: https://gaussianprocess.org/gpml/.

[30] M. Liu, G. Chowdhary, B. Castra da Silva, S.-Y. Liu, and J. P. How, "Gaussian processes for learning and control: A tutorial with examples," *IEEE Control Systems*, vol. 38, no. 5, pp. 53–86, 2018. DOI: 10.1109/MCS.2018.2851010.

[31] J. Wang, "An intuitive tutorial to gaussian process regression," *Computing in Science & Engineering*, vol. 26, no. 1, pp. 86–103, 2024. DOI: 10.1109/MCSE.2023.3342149.

[32] T. Beckers, "An introduction to gaussian process models," *arXiv preprint arXiv:2102.05497*, 2021, Accessed 02-04-2026. DOI: 10.48550/arXiv.2102.05497.

[33] R. R. Richardson, M. A. Osborne, and D. A. Howey, "Gaussian process regression for forecasting battery state of health," *Journal of Power Sources*, vol. 357, pp. 209–219, 2017. DOI: 10.1016/j.jpowsour.2017.05.004.

[34] S. Greenbank, "Data-driven battery state of health diagnostics and prognostics," Ph.D. dissertation, University of Oxford, 2022.

[35] J. Liu and Z. Chen, "Remaining useful life prediction of lithium-ion batteries based on health indicator and gaussian process regression model," *IEEE Access*, vol. 7, pp. 39 474–39 484, 2019. DOI: 10.1109/ACCESS.2019.2905740.

[36] Z. Sun, L. Zhong, X. Chen, and J. Guo, "Application of gaussian process regression model in industry," in *2022 2nd International Conference on Robotics, Automation and Artificial Intelligence (RAAI)*, 2022, pp. 221–225. DOI: 10.1109/RAAI56146.2022.10092999.

[37] Z. Chen, J. Fan, and K. Wang, "Multivariate gaussian processes: Definitions, examples and applications," *METRON*, vol. 81, pp. 181–191, 2023. DOI: 10.1007/s40300-023-00238-3.

[38] S. R. Gothi, "Event-driven microservices architecture for data center orchestration," *International Journal on Science and Technology*, vol. 16, no. 2, pp. –, 2025. DOI: 10.71097/IJSAT.v16.i2.3113.

[39] HL7 International, *Fast Healthcare Interoperability Resources (FHIR) Specification*, https://hl7.org/fhir, Accessed 03-06-2026, 2026.

[40] Epic Systems Corporation, *Epic open.epic FHIR API Specifications*, https://fhir.epic.com/Specifications, Accessed 03-06-2026, 2026.

[41] HL7 International, *HL7 FHIR Cross-Version Mapping Specification*, https://github.com/HL7/fhir-cross-version, Accessed 03-06-2026, 2025.

[42] N. Laranjeiro, M. Vieira, and H. Madeira, "A technique for deploying robust web services," *IEEE Transactions on Services Computing*, vol. 7, no. 1, pp. 68–81, 2014. DOI: 10.1109/TSC.2012.39.

[43] M. Levinson, J. Niestroy, and S. Al Manir, "Fairscape: A framework for fair and reproducible biomedical analytics.," *Neuroinform*, vol. 20, pp. 187–202, 2022. DOI: 10.1007/s12021-021-09529-4.

A

Software Implementation and Code Repository

The complete software implementation, containing the simulation engines, decoupled analytical microservice, and validation framework developed in this research, is hosted on GitHub.

Repository Access

The codebase, configuration manifests, and instructions for local deployment can be accessed at:
`https://github.com/DavidErikmats/Msc-Thesis`

Codebase Structure

The repository is organized into two primary sub-systems orchestrated by a containerized virtualization layer:

- **Decoupled Analytical Microservice (`MSA/`)**
  This directory contains the stateless FastAPI microservice that validation and routing layers use to expose the hybrid battery estimation engine to the clinical network.
  - `main.py`: Initializes the ASGI web application, configures API routing, and registers the global exception-handling hooks.
  - `routers/`: Exposes the HTTP endpoints (e.g., `battery.py`) and handles path dependency injection for token verification.
  - `schemas/`: Defines Pydantic validation schemas (e.g., `fhir_ingress.py`) that enforce strict data type constraints on incoming HL7 FHIR JSON bundles.
  - `serializers/`: Translates raw mathematical outputs into standardized HL7 FHIR R4 `DiagnosticReport` profiles.
  - `services/`: Exposes the core execution loop (`hybrid_engine.py`) and the underlying estimator models (`models/ECM.py` and `models/GPR.py`),

including the Matérn 5/2 covariance kernel and the discrete-time 2-RC state prior.

- `utils/`: Houses exception boundaries and wrapper types (`app_exceptions.py`, `service_result.py`) for uniform error reporting.

- **Offline Training and Simulation Suite (`estTool/`)**
  This folder contains the scripting environment used to train the Gaussian Process models offline, run unit tests, and simulate the hospital client infrastructure.
  - `services/`: Handles training routines (`gpr_train.py`) and hyperparameter optimization loops on historical datasets.
  - `simulator/`: Contains the clinical simulator script suites, including the validation entry points (`run_clinical_validation.py`, `run_benchmark.py`, and `run_robustness_tests.py`) that execute multi-axis testing profiles against the running microservice.
  - `tests/`: Houses unit tests (`test_hybrid.py`, `test_model_ecm.py`) that programmatically verify the mathematical stability of the ECM and GPR components and creates the specifc historical datasets that the model trains on.

- **Container Orchestration (Root Folder)**
  The root level contains files that manage virtual network topology and dependency isolation:
  - `docker-compose.yml`: Defines the multi-container topology, mapping the microservice (`msa_service`) and clinical client simulator (`clinical_app`) onto private, software-defined bridge networks.
  - `Dockerfile`: Contains container build instructions and dependency installations.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY