
# Symbolic AI in Critical Healthcare Setting:
## Towards Automated Governance of Spontaneous Breathing Trials for Streamlined Extubation

Master's Thesis

Computing in the Humanities

Faculty Information Systems and
Applied Computer Sciences

Otto-Friedrich-University Bamberg

Patrick Salvatore Stößel

October 6, 2025

Supervisor: Prof. Dr. Christoph Benzmüller
Supervisor: Prof. Dr. Andrea Vestrucci


URN: urn:nbn:de:bvb:473-irb-110690x
DOI: https://doi.org/10.20378/irb-110690

**Abstract**

This thesis presents a novel approach to clinical decision support in critical care by leveraging symbolic artificial intelligence (AI) for the formalization and automated verification of Spontaneous Breathing Trial (SBT) readiness assessments. It introduces a prototype system that transforms Intensive Care Unit (ICU) checklist logic into machine-verifiable higher-order logic (HOL) using Isabelle/HOL, ensuring correctness-by-construction, traceability, and semantic fidelity. The system was implemented in Java and supports modular, extensible workflows encompassing Health Level 7 (HL7)-compatible data ingestion, logic generation, and automated proof execution. A simulated environment was used to evaluate the prototype's performance across clinical and technical dimensions. Results demonstrate consistent and explainable classifications aligned with clinical expectations, and profiling indicates linear scalability with respect to patient count and checklist complexity. Qualitative analysis highlights the system's interpretability, workflow compatibility, and auditability. Limitations concerning real-world integration, proof traceability, and regulatory readiness are critically examined. The work establishes a reusable methodology for the formalization of protocol-based reasoning and outlines future directions including live deployment, integration with subsymbolic models, and regulatory certification. This research contributes a technically sound and ethically aligned foundation for the development of transparent, verifiable, and adaptable AI systems in healthcare.

# Contents

# List of Tables

# List of Figures

# List of Abbreviations

**ACL** Access Control Lists. 120

**ADT** Admissions, Discharges, and Transfers. 34

**AES** Advanced Encryption Standard. 55, 58, 67, 68, 113, 121, 126

**AI** Artificial Intelligence. iii, iv, 1, 2, 5–7, 9–22, 24, 26, 27, 29–31, 41, 50, 54, 108, 111, 117, 121–123, 125–128

**AISE** Chair of AI System Engineering. 2

**AMD** Advanced Micro Devices. 55, 97

**API** Application Programming Interface. 18, 55, 59

**ARDS** Acute Respiratory Distress Syndrome. 2, 28

**ASCII** American Standard Code for Information Interchange. 79

**AUC** Area under Curve. 20

**B3S** Industry-specific Security Standard. 4–6, 36–38, 113, 122, 126

**BSI** Federal Office for Information Security. 37

**CDSS** Clinical Decision Support System. iii, 5, 9, 13, 26–31, 34, 40, 49, 93, 121, 122, 128

**CDST** Clinical Decision Support Tool. 19, 107, 110

**$cmH_2O$** Centimetres of Water. 23, 79, 81, 94, 112

**$CO_2$** carbon dioxide. 19

**COVID-19** Coronavirus Disease. 25, 26

**CPU** Central Processing Unit. vii, 95–99, 101–103, 112, 113, 126

**DPIA** Data Protection Impact Assessment. 5, 37, 38, 113, 122, 126, 128

**EHR** Electronic Health Record. 19, 27, 34, 36, 107, 110

**EU** European Union. 36

**FHIR** Fast Healthcare Interoperability Resources. 35, 127

**$FiO_2$** fraction of inspired oxygen. 22, 23, 47, 74, 76, 79, 80, 94, 105, 112, 118

**FXML** FX Extended Markup Language. 55, 63, 64

**GB** Gigabyte. 55, 97

**GDPR** General Data Protection Regulation. 4–6, 36–38, 67, 69, 113, 122, 126

**GHz** Gigahertz. 97

**GT** Greater Than. 76

**GUI** Graphical User Interface. vii, 44, 69, 73, 91

**H2** Hypersonic 2. 55, 65, 67

**HAPI** HL7 Application Programming Interface. 55, 68

**HL7** Health-Level 7. iii, vii, 3, 5, 6, 9, 33–35, 44, 45, 49, 51, 55, 56, 59, 60, 65, 67–70, 90, 95–97, 99, 105, 112–115, 118, 126, 127

**HOL** Higher-Order Logic. iii, iv, vii, 2, 3, 6, 9–11, 15–21, 29–32, 35, 37, 39–43, 45–51, 53, 55, 61, 62, 70, 71, 73, 74, 76–79, 107–111, 125

**HTTP** Hypertext Transfer Protocol. 35

**I/O** Input/Output. 62, 73, 95, 98

**ICU** Intensive Care Unit. iii, 1, 2, 16–26, 29, 31–33, 35–37, 40, 41, 48–51, 80, 97, 105, 107, 108, 111, 113, 118, 122, 127

**ICU II** Interdisciplinary Intensive Care Unit. 2

**IEC** International Electrotechnical Commission. 123

**ISMS** Information Security Management System. 37, 38, 122

**ISO** International Organization for Standardization. 37, 123, 128

**IT** Information Technology. iii, 7, 9, 33, 35–37, 49, 54, 117, 118, 120, 122, 123

**IT-SIG** IT Security Act. 36, 37

**JDBC** Java Database Connectivity. 59

**JPA** Java Persistence API. 55

**JSON** JavaScript Object Notation. vii, 35, 58, 63, 64, 67, 82–84

**JVM** Java Virtual Machine. 101, 104, 112

**KRITIS** Critical Infrastructures. 4–6, 36–38, 113, 122, 126, 128

**LightGMB** Light Gradient Boosting Machine. 20

**LT** Lower Than. 76

**MB** Megabyte. 101, 103, 112

**MDR** Medical Device Regulation. 123, 128

**min** Minutes. 23

**ML** Machine Learning. 3, 15, 20, 27

**mmHg** Millimetres of Mercury. 23, 79, 80, 94, 112

**ms** Milliseconds. 97, 99, 103, 112

**MSH** Message Header Segment. 34

**MV** Mechanical Ventilation. 2, 3, 19, 21, 22, 25, 26

**OBX** Observation Segment. 69

**OWL** Web Ontology Language. 20

**$PaCO_2$** Partial Pressure of Carbon Dioxide. 20

**$PaO_2$** Partial Pressure of Arterial Oxygen. 20, 22, 47, 74, 76, 79, 80, 94, 105, 112

**PDMS** Patient Data Management System. 18, 28, 30, 32, 34, 35, 37, 41, 44–46, 49, 51, 56, 68, 69, 105, 117, 127

**PEEP** Positive End-Expiratory Pressure. 23, 79, 81, 94, 112, 118

**PID** Patient Identification Segment. 69

**POSIX** Portable Operating System Interface. 61

**PSS** Physical Symbol System. 11

**QRD** Query Definition Segment. 68

**QRY^A19** HL7 Query for Patient Demographics. 68

**RAM** Random Access Memory. 55, 103

**REST** Representational State Transfer. 35

**RIM** Reference Information Model. 34

**RSP^K21** HL7 General Acknowledgement Response. 68

**s** Seconds. 100, 103, 112

**SBT** Spontaneous Breathing Trial. iii, iv, vii, 1–5, 9, 15–26, 29, 31–33, 35, 37, 39, 40, 42–47, 50, 66, 88, 89, 91, 93–97, 106, 112, 114, 125

**SHACL** Shapes Constraint Language. 20, 107, 110

**SHAP** Shapley Additive Explanations. 20, 106, 110

**SLF4J** Simple Logging Facade for Java. 55

**SMT** Satisfiability Modulo Theory. 16

**SQL** Structured Query Language. 59

**SUS** System Usability Scale. 119

**TAI** Trustworthy AI. iii, 2, 12–15, 17, 18

**TCP** Transmission Control Protocol. 65

**TOM** Technical and Organizational Measures. 36, 38

**TrustKG** Trust Knowledge Graph. 20, 106–108, 110

**UI** User Interface. 5, 54–56, 60, 61, 63, 64, 72, 104–107, 109, 110, 120, 127

**VAP** Ventilator-Associated Pneumonia. 2, 3, 22, 24, 25

**WFSICCM** World Federation of Societies of Intensive and Critical Care Medicine. 22

**WSL** Windows-Subsystem Linux. 61, 62

**XAI** Explainable AI. iii, 2, 13–15, 18, 20

**XCSR** Extended Classifier System with Real-valued Inputs. 20, 106, 108, 110

**XML** Extensible Markup Language. 35

# Chapter 1

# Introduction

Artificial Intelligence (AI) is shaping contemporary healthcare. However, its deployment in safety-critical environments demands unparalleled transparency, reliability, and ethical accountability. Intensive Care Units (ICUs) exemplify this requirement, as clinicians must respond to rapidly changing physiological states and, therefore, cannot rely on opaque decision support tools. This thesis investigates how formal methods, realized through symbolic AI, can automate the governance of Spontaneous Breathing Trials (SBTs). An SBT is a clinical test used to assess a patient's ability to breathe with minimal or no ventilator support. It is a critical step in determining the readiness for extubation (Zein et al., 2016). By encoding protocol logic in machine-checked proofs, the work aims to streamline extubation workflows, minimize variability in practice, and ultimately safeguard patients.

In recent years, symbolic AI has demonstrated its utility not only in healthcare, but also in high-stakes domains requiring mathematical precision. For example, DeepMind has successfully employed symbolic reasoning to solve International Mathematical Olympiad problems at a silver medal level, underscoring the capabilities of formal systems in replicating structured human reasoning (DeepMind, 2022). Similarly, AI-driven analysis in oncology has identified complex cancer patterns at unprecedented speed, illustrating how algorithmic transparency can complement clinical intuition in diagnostic contexts (University of Gothenburg, 2024). These examples reinforce the relevance of explainable and verifiable AI systems in critical environments, such as the ICU.

The chapter unfolds as follows. Section 1.1 places the research within its technological and clinical context. Section 1.2 outlines the primary concern motivating this thesis, while Section 1.3 lists the guiding objectives. Section 1.4 clarifies the scope and limitations of the research, and Section 1.5 maps the structure of the subsequent chapters. Finally, Section 1.6 reflects on the use of generative AI tools during the writing process.

Collectively, these sections connect the broader demand for trustworthy governance in critical care with the technical potential of formal verification, thereby laying the foundation for the background, methodology, implementation, and evaluation that follow.

## 1.1 Context and Motivation

AI now permeates many clinical domains, yet its application in intensive care renews concerns about transparency, interpretability, and accountability. Surveys on trustworthy AI (TAI) and explainable AI (XAI) indicate that clinicians, regulators, and patients consider intelligible reasoning indispensable for responsible use in healthcare (Barredo Arrieta et al., 2020; Amann et al., 2020). These expectations are particularly acute in ICUs, where therapeutic decisions unfold within minutes and directly affect morbidity and mortality (Kelly et al., 2019).

Symbolic AI addresses these expectations by encoding domain knowledge in formally verified logic, rendering each inference step accessible and provably correct. This quality sets symbolic approaches apart from purely data-driven models whose internal representations remain opaque even when post-hoc visualizations are provided (Augusto, 2021). To generate and check such proofs, this thesis relies on Isabelle/Higher-Order Logic (HOL) as its verification backbone, as elaborated in Chapter 2.

#### Clinical Use Case: SBT

Within this technical framework, the study concentrates on SBTs, the standard method for determining whether mechanical ventilation (MV) can be safely discontinued. The timing of extubation is crucial. Therefore, the protocol offers an ideal test case for transparent, rigorous decision support. Development proceeds in collaboration with the Chair of AI System Engineering (AISE) at the University of Bamberg and the Interdisciplinary Intensive Care Unit (ICU II) at Hospital Bamberg, ensuring that each design choice reflects clinical realities.

The ICU II at Hospital Bamberg provides advanced care for patients with acute respiratory, cardiac, or multi-organ failure. It combines pulmonological, cardiological, and anesthesiological expertise under a unified treatment concept and supports both internal and external departments of the hospital. Its focus on interprofessional collaboration, protocol-guided weaning strategies, and comprehensive monitoring systems makes it a representative setting for evaluating AI-assisted extubation governance in real-world clinical practice (Sozialstiftung Bamberg, 2025).

By situating the research at the intersection of urgent clinical needs and verifiable symbolic reasoning, this section lays the groundwork for the problem definition that follows in Section 1.2.

## 1.2 Problem Defintion

MV is indispensable in intensive care, yet every additional day of ventilatory support heightens the risk of ventilator-associated pneumonia (VAP) and airway trauma (Belle et al., 2020). Removing the tube too early is equally hazardous, especially in complex conditions such as acute respiratory distress syndrome (ARDS), where extubation failure can also lead to airway trauma, VAP, dysphagia, and increased mortality (Stivi et al., 2024). Safe extubation therefore hinges on the precise and timely conduct of SBTs.

Although evidence-based checklists exist, bedside adherence remains uneven. Multicentre audits attribute these compliance gaps to staffing constraints, lim-

ited training, and intense workflow pressures (Ajay et al., 2022). Even after a successful trial, extubation may still be postponed. Clinicians might choose to re-evaluate the screening outcome or wait for senior confirmation. Such delays have been linked to avoidable morbidity and prolonged stays in intensive care (Belle et al., 2020). Each episode of VAP extends admission by more than three days and adds several thousand euros to costs, while MV itself raises daily expenditure by nearly 60% (Kaier et al., 2019, 2020).

#### Limitations of Current Decision-Support Systems

Available decision-support tools have not eliminated this variability. Commercial systems typically depend on machine-learning (ML) models whose internal reasoning cannot be proven correct for individual patients (Chen et al., 2019; Igarashi et al., 2022). Reviews conclude that such opacity undermines clinical trust, accountability, and regulatory acceptance in safety-critical care (Montani and Striani, 2019). Symbolic approaches promise verifiable correctness, yet current examples either target narrow device functions or operate offline without real-time patient data (Freitas et al., 2020). As of now, no known framework simultaneously encodes the complete SBT safety screen in HOL, supports live or simulated Health-Level 7 (HL7) input, and produces auditable, step-by-step readiness assessments for clinical review. The absence of a rigorously verified and clinically interpretable system perpetuates inconsistent extubation practice, limits organisational accountability, and hinders regulatory approval for advanced decision support in intensive care.

Designing an automated governance framework that translates the checklist into machine-checked logic, integrates smoothly with existing data flows, and delivers transparent bedside recommendations thus defines the central problem and directly leads to the objectives outlined in Section 1.3.

## 1.3 Objectives

This thesis advances five interrelated objectives that convert the conceptual groundwork laid in Sections 1.1 and 1.2 into a coherent programme of research and development. Each objective aligns with the thesis structure, so that every methodological and empirical chapter directly furthers the stated aims.

#### Objective 1: Formalisation in Isabelle/HOL

The first objective is to translate the SBT safety screen into HOL predicates expressed in Isabelle/HOL. The formalisation must retain the full clinical semantics of every checklist item, ensuring that nothing relevant to the extubation decision is lost during abstraction.

#### Objective 2: System Architecture and Prototype Development

The second objective is to develop a modular, Java-based prototype. The system must convert the formalised checklist into Isabelle theory files, execute batch-mode proof sessions, and return structured verification results to clinicians in near real time. Its architecture hinges on a code-generation pipeline, integrated session management, and HL7-compatible data ingestion to guarantee logical

soundness, clinical traceability, and a future-proof design that accommodates live patient-data systems.

#### Objective 3: Quantitative and Qualitative Evaluation

The third objective is to evaluate the system along quantitative and qualitative dimensions. Quantitative analysis will measure logical correctness, scalability, and computational resource usage across varying patient cohorts and checklist complexities. In parallel, qualitative assessment will examine the clarity of the generated explanations and judge how effectively the proof outputs support clinical interpretation.

#### Objective 4: Compliance with Data Protection and Infrastructure Standards

The fourth objective undertakes a normative analysis of how the prototype aligns with European data-protection and critical-infrastructure regulations, including General Data Protection Regulation (GDPR), Critical Infrastructures (KRITIS) (Kritische Infrastrukturen), and the Industry-specific Security Standard (B3S) catalogue (Branchenspezifischer Sicherheitsstandard). The focus is on showing how formal verification supports privacy by design, traceable audit trails, and accountable decision-making. Although full certification exceeds the scope of this thesis, its importance is noted as a direction for future work.

#### Objective 5: Methodological Generalisation and Guideline Formalisation

The fifth and final objective is to craft a reusable methodology for formalising additional intensive-care protocols and integrating symbolic with subsymbolic reasoning. By abstracting common checklist patterns into modular logical components, the thesis aims to foster adaptive "living guidelines" that can absorb new medical evidence without compromising formal safety guarantees.

Collectively, these objectives chart a clear path from theoretical formulation to technical implementation and empirical evaluation. They establish the criteria of verifiable correctness, clinical relevance, and ethical soundness, which the thesis will use to measure its success. Section 1.4 delineates the scope and practical limits within which these goals will be pursued.

## 1.4 Scope and Limitations

This thesis centres on automating the SBT safety screen through formally verified logic. The following delimitations define the project's scope and identify constraints that affect the validity and generalisability of its results.

#### Clinical Scope

The formal model is restricted to the safety screen and the immediate extubation decision. Subsequent stages of the weaning process, including post-extubation care and long-term ventilatory management, are explicitly excluded from this study.

#### Technical Setup

Patient parameters are supplied through a mocked HL7 v2 interface that emulates the message formats commonly used by commercial patient-data management systems. Although the system is designed to handle real HL7 traffic, it does not yet operate in a live clinical environment.

#### Validation and Testing Constraints

System validation relies on synthetic patient records. No usability testing or prospective clinical outcome studies involving bedside staff have been performed. Consequently, the results demonstrate technical feasibility rather than providing conclusive clinical validation.

#### Regulatory and Ethical Scope

The prototype incorporates privacy-by-design strategies and addresses selected aspects of GDPR, KRITIS, and the B3S regulatory catalogue. Full certification procedures, formal data protection impact assessments (DPIA), and rigorous user interface (UI) hardening remain outside the scope of this work and are designated for future research.

#### Performance and Evaluation Limitations

The evaluation focuses on logical correctness, scalability, and resource consumption under batch-processing conditions. Real-time throughput for deployment in large-scale intensive-care settings has not yet been assessed, and Isabelle proof latency currently constitutes the primary performance constraint.

These boundaries are reflected in the discussion presented in Chapter 6 and serve to guide future developments in areas such as live data integration, comprehensive clinical assessment, and formal compliance with regulatory standards. Section 1.5 outlines the organisation of the thesis and introduces the purpose of each chapter in the overall research narrative.

## 1.5 Thesis Structure

The thesis is structured into seven chapters that follow a logical sequence, progressing from conceptual foundations to system implementation and empirical evaluation. Each chapter builds upon the preceding one, collectively advancing the central argument that formally verified symbolic reasoning can offer transparent and accountable decision support for SBTs.

#### Chapter 2: Background and Related Work

Chapter 2 establishes the interdisciplinary foundation of the thesis. It surveys the core principles of symbolic AI, reviews clinical literature concerning SBTs, examines current clinical decision support systems (CDSS), and outlines relevant healthcare information standards. The discussion highlights the suitability of formal verification techniques for safety-critical applications in intensive care.

**Chapter 3: Methodology**

Chapter 3 presents the research design underpinning the project. It introduces the design-science paradigm adopted for this study, describes the modelling approach used to formalise the safety screen, and explains the rationale for selecting Isabelle/HOL and its supporting toolchain for conducting machine-checked formal proofs.

**Chapter 4: System Architecture and Implementation**

Chapter 4 details the architecture and operational logic of the implemented prototype. It describes the modular structure linking Java and Isabelle components, tracks the flow of data from HL7 message ingestion through to theorem prover execution, and provides a comprehensive walkthrough illustrating how checklist items are transformed into formally verified proofs.

**Chapter 5: Results and Evaluation**

Chapter 5 presents the empirical findings. It provides quantitative assessments of the system's logical correctness, computational scalability, and resource consumption across synthetic patient datasets designed to reflect clinical use cases.

**Chapter 6: Discussion and Limitations**

Chapter 6 interprets the evaluation results in relation to clinical, technical, and regulatory considerations. It explores integration barriers, addresses ethical implications, and evaluates the extent to which the current system design meets key requirements of GDPR, KRITIS, and the B3S catalogue.

**Chapter 7: Conclusion and Future Work**

The final chapter 7 synthesises the thesis's main contributions, revisits the original research objectives, and outlines future directions for extending the framework. This includes the formalisation of additional intensive-care protocols and the integration of hybrid symbolic-subsymbolic reasoning architectures.

Before proceeding to the substantive chapters, a brief note is warranted on the role of generative AI tools during the thesis writing process, particularly in supporting linguistic formulation and translation.

## 1.6 Use of AI in the Thesis Writing Process

In the context of this thesis, generative AI tools were employed in a supportive capacity during the drafting process. Language models developed by OpenAI, such as GPT-4o, were used to enhance linguistic formulation and improve clarity of expression, including the handling of occasional technical terminology. Additionally, DeepL served as a translation aid, particularly useful for ensuring accurate bilingual phrasing between German and English, as English is not the author's native language. At no point did these tools contribute to the development of original argumentation or theoretical content.

With these preliminary clarifications in place, the thesis now turns to the conceptual and interdisciplinary foundations that underpin the proposed approach, beginning with an exploration of symbolic AI, clinical protocols, and healthcare Information Technology (IT) standards.

# Chapter 2

# Background and Related Work

The development of reliable, explainable, and domain-specific AI in healthcare requires an interdisciplinary foundation that encompasses symbolic reasoning, clinical practice, healthcare IT standards, and decision support methodologies. This chapter establishes the necessary conceptual and technical background for the work presented in subsequent sections. It integrates core knowledge domains that underpin the proposed approach and delineates prior research that informs its design and implementation.

An introduction to symbolic AI and formal theorem proving lays the foundation for understanding the reasoning paradigms employed in this thesis. Section 2.1 presents a concise overview of symbolic AI and the principles of formal verification, with particular attention to the Isabelle/HOL proof assistant.

To contextualize the clinical relevance of the target use case, Section 2.2 describes SBTs in intensive care settings. This includes the clinical rationale, weaning procedures, and associated risks relevant to extubation decisions.

A broader perspective on computational decision support is provided in Section 2.3, which explores the integration of AI into CDSSs. The discussion includes both conventional systems and more recent approaches that seek to formalize clinical protocols.

Lastly, Section 2.4 outlines the essential standards governing data exchange and security in healthcare IT. This includes an overview of HL7-based communication as well as regulatory expectations related to data integrity and patient privacy.

Collectively, these sections offer a structured synthesis of the domains relevant to the formalization and automation of clinical reasoning, thereby forming the basis for the methodology and system implementation discussed in the following chapters.

## 2.1 Overview of Symbolic AI and Theorem Proving

The development of formal, interpretable, and clinically relevant AI systems requires a robust understanding of symbolic reasoning and its applications. Symbolic AI, rooted in logic and formal semantics, provides mechanisms to represent and reason over structured knowledge. These mechanisms are of particular importance in domains such as healthcare, where safety, accountability, and traceability are paramount. In contrast to purely data-driven paradigms, symbolic approaches offer transparency and verifiability, aligning with clinical demands for explainable and justifiable decision-making processes. This section introduces the foundational concepts and motivations for symbolic AI, especially in relation to formal verification, and situates its relevance within the broader landscape of AI in healthcare.

The section begins by outlining the core principles of symbolic AI and the formal characteristics that distinguish it from other AI methodologies. This is addressed in Subsection 2.1.1, which details the foundational logic, rule-based inference systems, and structured representations that define symbolic AI.

Following this, the Subsection 2.1.2 contrasts symbolic methods with subsymbolic approaches, such as deep learning, to highlight their respective strengths and limitations in clinical contexts. This comparative analysis also serves to motivate the integration of symbolic AI into healthcare applications.

Given the critical importance of transparency in medical decision-making, a dedicated Subsection 2.1.3 explores how symbolic AI contributes to the development of explainable and trustworthy systems in clinical environments.

The Subsection 2.1.4 then introduces Isabelle/HOL, a state-of-the-art interactive theorem prover that serves as the formal verification backbone for the system developed in this thesis. The corresponding subsection presents its foundational logic, key features, and practical relevance in verifying clinical safety protocols.

Finally, the Subsection 2.1.5 concludes with an overview of other symbolic and hybrid approaches in the domain, thereby positioning this thesis within the broader research landscape and identifying complementary and alternative methodologies.

This structured introduction lays the groundwork for understanding how formal methods and symbolic reasoning are leveraged in subsequent chapters to support the formalization and automation of clinical safety screening protocols.

### 2.1.1 Foundations and Key Features of Symbolic AI

Symbolic AI represents one of the earliest and most influential paradigms in the field of AI. Rooted in formal logic and cognitive science, symbolic AI seeks to emulate intelligent behavior by manipulating discrete symbols according to a set of well-defined rules. This approach is grounded in the foundational work of Allen Newell and Herbert A. Simon, who conceptualized intelligence as the processing of symbol structures within physical symbol systems (Augusto, 2021).

### Symbol Systems and Foundational Concepts

At the core of symbolic AI lies the notion of a symbol system, in which symbols serve as explicit carriers of meaning that can be composed, transformed, and interpreted to perform cognitive tasks. Newell and Simon defined a Physical Symbol System (PSS) as any system, human or machine, that generates intelligent behavior through the physical manipulation of symbol structures. This theoretical framework provided a foundation for viewing reasoning, problem-solving, and learning as computational processes instantiated through symbol operations (Augusto, 2021).

### Key Functional Components

Three primary features characterize symbolic AI: *representation*, *manipulation*, and *heuristic search*. First, knowledge in symbolic AI is represented through formal structures such as rules, frames, and semantic networks. These representations are human-readable and can be systematically interrogated and interpreted. Second, symbolic systems employ algorithms to manipulate these representations, enabling the execution of logical inference, rule chaining, and deduction. Such manipulations are used to model tasks including theorem proving and planning. Third, symbolic systems rely on heuristic search techniques to efficiently navigate large problem spaces, selecting promising paths based on domain knowledge or procedural strategies (Augusto, 2021).

### Advantages of Symbolic AI

One of the most significant advantages of symbolic AI is its explainability. Since reasoning steps are explicit and traceable, symbolic systems offer transparent justifications for their outputs, a feature of critical importance in domains requiring accountability such as healthcare. Moreover, symbolic AI supports rational action by enabling systems to act based on structured, contextualized knowledge. Finally, its universality allows it to model any process that can be described in symbolic terms, providing a flexible foundation for diverse applications (Augusto, 2021).

### Limitations and Challenges

However, symbolic AI also faces notable limitations. Its rigidity makes it ill-suited for tasks involving uncertainty, dynamic contexts, or sensory-rich environments. Symbolic systems typically require hand-crafted knowledge bases and struggle with scalability in real-time applications. Moreover, symbolic AI suffers from the grounding problem: the difficulty of linking abstract symbols to real-world referents without external interpretation mechanisms (Augusto, 2021).

### Relevance for Formal Verification and Clinical AI

Despite these limitations, symbolic AI remains essential for applications demanding high levels of interpretability and structured reasoning. In particular, it provides the logical foundation for formal verification tools such as Isabelle/HOL, which are instrumental in modeling and validating complex clinical

protocols, as will be elaborated in Subsection 2.1.4. The capacity of symbolic AI to represent medical knowledge in explicit and verifiable forms aligns directly with the overarching objectives of explainability and safety in clinical decision support systems, which are further discussed in Subsection 2.1.3.

In summary, symbolic AI offers a rigorous and interpretable framework for intelligent reasoning, grounded in formal representations and logical operations. These foundational characteristics make it an indispensable tool in the development of TAI systems, particularly in domains where transparency, precision, and verifiability are non-negotiable.

Having established the core principles and capabilities of symbolic reasoning, the discussion now turns to its counterpart: subsymbolic AI.

### 2.1.2 Comparison with Subsymbolic AI and Healthcare Relevance

While symbolic AI offers a robust and interpretable framework for representing and reasoning over structured knowledge, it exists alongside an alternative paradigm known as subsymbolic AI. This subsection examines the foundational differences between these two approaches, highlights their respective strengths and limitations, and elucidates their implications for applications in the healthcare domain.

#### Architecture and Capabilities of Subsymbolic AI

Subsymbolic AI, often associated with connectionist models such as neural networks and deep learning, departs fundamentally from the explicit representations that characterize symbolic systems. Rather than relying on human-readable symbols and rule-based reasoning, subsymbolic systems operate on distributed representations that learn correlations between inputs and outputs through data-driven statistical processes (Ilkou and Koutraki, 2020; Shapiro, 2003). These models are capable of adapting to large, heterogeneous datasets and are particularly well-suited for tasks involving perception, such as image recognition or natural language processing. Their architecture allows them to generalize from training data and to handle imprecise or noisy inputs with notable robustness (Ilkou and Koutraki, 2020; Shapiro, 2003).

#### Explainability and Clinical Implications

However, the opacity of subsymbolic systems remains a critical concern. Their decision-making processes are often difficult to interpret or verify, a limitation that has significant implications in high-stakes environments such as healthcare (Augusto, 2021; Shapiro, 2003). The absence of transparent inference chains hampers efforts to justify clinical recommendations, undermining trust and accountability in medical decision support. In contrast, symbolic AI systems offer full traceability and explainability by operating on well-defined logical structures. As such, they are particularly advantageous in domains where the rationale behind a decision must be accessible to human experts and subject to rigorous validation (Ilkou and Koutraki, 2020; Shapiro, 2003).

#### Adaptability and Knowledge Acquisition

The comparison between symbolic and subsymbolic AI also reveals a divergence in adaptability and scalability. Subsymbolic models excel in dynamic or sensory-rich environments due to their ability to learn from vast quantities of data without requiring predefined rules. Symbolic systems, by contrast, often struggle in such contexts due to their rigidity and dependence on manually curated knowledge bases (Ilkou and Koutraki, 2020; Augusto, 2021). This limitation is often referred to as the knowledge acquisition bottleneck, wherein the development and maintenance of extensive rule sets becomes a prohibitive task Shapiro (2003).

#### Symbolic AI in CDSSs

Despite these trade-offs, symbolic AI maintains a decisive edge in scenarios demanding formal guarantees, structured reasoning, and interpretability, requirements that are particularly stringent in the medical field. Healthcare applications frequently necessitate the use of transparent models that support accountability, verifiability, and ethical oversight. CDSSs, for example, benefit from symbolic representations that mirror the logical structure of medical guidelines and protocols, enabling precise and reproducible inferences (Augusto, 2021; Shapiro, 2003).

#### Hybrid Approaches and Integration Strategies

The increasing complexity of clinical data and protocols has prompted growing interest in hybrid methodologies that integrate symbolic and subsymbolic techniques. These approaches seek to harness the scalability and learning capabilities of subsymbolic AI while retaining the clarity and structure of symbolic reasoning (Ilkou and Koutraki, 2020). Such hybrid systems are particularly promising in healthcare, where they may allow for efficient data processing without sacrificing the interpretability essential for clinical use. Although standardization and generalization remain challenges, hybrid models such as Knowledge-Based Neural Networks and Logic Tensor Networks offer a pathway toward more adaptive yet transparent AI systems.

The contrast between symbolic and subsymbolic AI, therefore, is not merely technical. It carries profound implications for healthcare. Subsymbolic models offer powerful tools for learning from data, yet their limited explainability restricts deployment in safety-critical settings. Symbolic AI, grounded in formal logic and transparency, aligns more closely with the ethical and procedural demands of clinical reasoning. Understanding how these qualities shape trust and accountability leads naturally to the Subsection 2.1.3, which examines XAI and TAI frameworks in healthcare.

### 2.1.3 XAI and TAI in Healthcare

The integration of AI into clinical environments has introduced both unprecedented opportunities and substantial ethical, technical, and legal challenges. Among the most pressing concerns is the lack of explainability and transparency in advanced AI models, particularly those relying on subsymbolic architectures

such as deep learning. These challenges are especially pronounced in high-stakes domains such as healthcare, where decision-making must be not only accurate but also comprehensible and ethically sound. The concepts of XAI and TAI have emerged as foundational pillars for addressing these demands and facilitating the safe deployment of AI systems in clinical practice.

### Core Concepts of XAI

XAI encompasses a set of methodologies aimed at making AI model behavior understandable to human stakeholders without compromising predictive performance. Barredo Arrieta et al. (2020) emphasize that explainability is essential for ethical AI applications, particularly in domains like medicine, where opaque decision mechanisms can undermine user trust and accountability. They differentiate between related but distinct concepts such as *interpretability*, *comprehensibility*, and *transparency*. Interpretability refers to the ability to describe a model's function in human terms, while transparency denotes the inherent clarity of a model's structure, which may include simulatability or decomposability. Explainability, in this context, functions as an interface that conveys meaningful and accurate explanations of a system's reasoning to diverse audiences (Barredo Arrieta et al., 2020).

### Stakeholder Requirements and Use Cases

In healthcare, the importance of explainability extends beyond technological transparency. Markus et al. (2021) outline that explainability comprises both interpretability, which refers to the clarity and compactness of explanations, and fidelity, defined as the degree to which explanations truthfully represent the model's actual reasoning. These dual properties must be balanced depending on the stakeholder group and application scenario. For instance, clinicians may prioritize interpretability for practical usability, whereas regulatory bodies may require high fidelity for legal accountability. The same authors identify three central use cases of explainability in healthcare: *verifying the appropriate use of clinical features*, *managing interpersonal and institutional communication*, and *enabling knowledge discovery for research and guideline refinement* (Markus et al., 2021).

### Empirical Evidence and Practical Challenges

The significance of explainability is further supported by empirical insights into clinical practice. Kelly et al. (2019) underscore that AI interpretability remains in its infancy but is essential for trustworthy deployment in healthcare settings. They note that the opaque nature of high-performing models often precludes the kind of clinician-AI interaction necessary for informed decision-making. The risks include model bias, misclassification, and alert fatigue, phenomena that can lead to clinical errors if not properly understood and mitigated. To address this, both post-hoc explanation methods and inherently interpretable models are being explored, though each entails trade-offs between performance and clarity (Kelly et al., 2019).

### Foundations of TAI

TAI, by contrast, encompasses a broader framework that situates explainability within a constellation of ethical, legal, and technical requirements. According to Floridi (2019), TAI must fulfill seven essential conditions: *human agency and oversight*, *technical robustness and safety*, *privacy and data governance*, *transparency*, *diversity and fairness*, *societal well-being*, and *accountability*. These principles have been widely adopted in European guidelines and academic literature and serve as benchmarks for the ethical deployment of AI systems (Floridi, 2019). Reinhardt (2023) extends this analysis by cautioning that trustworthiness must not become an abstract ideal devoid of actionable criteria. Instead, it requires rigorous attention to sociotechnical contexts, regulatory compliance, and institutional accountability.

### Dimensions of Trust and Accountability

The multidimensional nature of trust in AI is further elaborated by Amann et al. (2020), who examine the implications of explainability from technological, legal, medical, and patient perspectives. Their analysis reveals that explainability is critical for securing informed consent, meeting certification standards, and clarifying liability in the event of adverse outcomes. From the medical viewpoint, explainability helps clinicians distinguish between accurate and spurious predictions, thereby fostering continuous learning and improving care quality. From the patient perspective, it enhances autonomy and participation in shared decision-making, aligning with core principles of biomedical ethics such as beneficence and justice (Amann et al., 2020).

The systematic review by Albahri et al. (2023) reinforces the necessity of a holistic approach to TAI in healthcare. The authors highlight key ethical and robustness challenges, such as data biases, incomplete information, and the opaqueness of decision-making processes. Their findings suggest that explainability is indispensable for building systems that are not only accurate but also ethical, secure, and aligned with human values. They also advocate for integrating fairness, accountability, and continuous validation as core requirements for trustworthy deployment in medical settings (Albahri et al., 2023).

### Hybrid Approaches and Technical Integration

Recent contributions have proposed technical strategies to reconcile the trade-offs between explainability and performance. One such approach is the hybrid integration of symbolic and subsymbolic AI. Vestrucci et al. (2024) demonstrate the feasibility of combining symbolic reasoning, exemplified by theorem proving in Isabelle/HOL, with ML techniques to enhance transparency and adaptability in clinical decision-making. Their case study on SBTs in intensive care illustrates how symbolic methods can enforce clinical rules while ensuring traceability and consistency. This hybrid model addresses a key concern identified by Vivek and Martin Lloyd (2021), namely, the need to embed ethical reasoning into AI systems by automating logic-based evaluation processes while maintaining human oversight.

XAI and TAI therefore constitute the cornerstone of responsible AI in healthcare. Explainability guarantees that decisions remain interpretable, justifiable,

and verifiable, fostering clinical acceptance and regulatory compliance, while the broader framework of trustworthiness embeds these qualities within ethical, legal, and societal imperatives. As AI-driven decision support becomes more prevalent, meeting both criteria will be vital to safeguard patient safety, promote fairness, and maintain public confidence. Subsection 2.1.4 turns to Isabelle/HOL, the theorem-proving environment that underlies the formal verification strategy of this thesis, to show how its features operationalise the requirements set out here.

### 2.1.4 Isabelle/HOL Foundations, Features, and Healthcare Applications

Isabelle/HOL represents one of the most mature and widely adopted environments for formal verification and theorem proving based on HOL. Its architecture, toolset, and logic framework make it particularly suitable for applications in safety-critical domains such as healthcare, where correctness, explainability, and adaptability are essential. Within the context of this thesis, Isabelle/HOL serves as the foundational framework for formalizing and verifying SBT checklists used in ICUs, integrating clinical protocols with machine-verifiable logic.

#### HOL Foundations

HOL serves as a powerful foundation for formal reasoning by extending second-order logic to allow quantification over predicates and functions of arbitrary order, thereby increasing its expressive power for modeling complex systems and reasoning tasks (Benzmüller and Andrews, 2024; Väänänen, 2024). This feature enables the formalization of intricate logical dependencies, such as those found in clinical checklists where patient parameters, decision thresholds, and safety rules are interdependent and dynamic. HOL structures variables using types and supports compound terms and formulas constructed through recursive definitions.

#### Core Features of Isabelle/HOL

Isabelle/HOL builds on this foundation by providing a structured and extensible environment for proof development. As described in the Isabelle System Manual, it includes tools for defining datatypes, inductive constructs, and recursive functions, which are essential for modeling modular and dynamic elements within medical checklists. The system's Isar proof language facilitates the construction of human-readable, machine-verifiable proofs, thereby enhancing interpretability and reducing verification errors (Wenzel, 2024).

#### Automation Tools and Reasoning Strategies

Isabelle/HOL combines interactive and automated reasoning within a cohesive framework. It supports both declarative and procedural proof styles, ensuring flexibility across a range of verification tasks (Boldo et al., 2016). Among its notable automation tools is `Sledgehammer`, which connects Isabelle to external first-order provers and Satisfiability Modulo Theory (SMT) solvers. `Quickcheck` and `Nitpick` aid in counterexample generation and model exploration, while

tools like `Auto`, `Blast`, and `Metis` enable internal heuristic and resolution-based proof search. Additionally, `simp` serves as a core technique that applies conditional, pattern-driven rewriting and contextual simplification of goals (Blanchette et al., 2011).

### Role of simp in This Thesis

The automation tool `simp` plays a central role in the verification process developed for this thesis. It facilitates structured and efficient reasoning by iteratively applying rewrite rules and refining proof goals. This method supports both manual and automatic invocation and proves particularly effective in domains with well-defined conditional logic, such as ICU checklists. By leveraging contextual information and pattern matching, `simp` allows for controlled expansion of definitions and logical transformations, preserving proof clarity while maintaining semantic correctness (Blanchette et al., 2011).

These automated features are further enhanced by Isabelle's modular infrastructure based on locales and type classes, which support reusable specifications and hierarchical theory development (Wenzel, 2024).

### Healthcare Applications and Clinical Relevance

The application of Isabelle/HOL to healthcare domains has been substantiated by multiple studies that highlight its potential for improving safety, reliability, and transparency in clinical workflows. Formal methods in general, and Isabelle/HOL in particular, offer mathematical frameworks for modeling, simulating, and verifying the behavior of clinical systems before deployment, thereby reducing design-phase errors and enhancing patient safety (Seidl, 2024). In the Protocure project, for instance, formal representations were used to uncover ambiguities and inconsistencies in medical guidelines, demonstrating the effectiveness of theorem provers in validating protocol correctness (Balser et al., 2004).

Medical checklists, such as the SBT readiness protocol, are prone to variation in interpretation and compliance, particularly in resource-constrained ICU environments (Ajay et al., 2022). Isabelle/HOL addresses these challenges by enabling a formal encoding of checklist logic, including readiness assessments, trial evaluations, and extubation criteria. This transformation from informal to formal specification follows a stepwise methodology: textual descriptions are first captured in intermediate representations and subsequently formalized in a HOL-based system such as Isabelle for automated validation and continuous improvement (Balser et al., 2004).

### Formalising ICU Checklists with Isabelle/HOL

Recent work by Vestrucci et al. (2024) illustrates the use of Isabelle/HOL in the automation of SBT procedures within the ICU. Their symbolic approach to TAI employs formal theorem proving to enforce ethical, technical, and clinical rules in a transparent and verifiable manner. In their implementation, clinical rules governing extubation readiness were encoded using Boolean expressions and structured functions within Isabelle/HOL. This enabled consistent, bias-free decision-making aligned with established medical guidelines.

A key advantage of this approach lies in its adaptability. By structuring checklists as modular components within Isabelle, updates to medical knowledge can be readily incorporated into the verification model. Additionally, the formal structure enhances traceability, allowing clinicians and developers to audit the logical reasoning behind extubation decisions, which is critical for maintaining clinical trust and regulatory compliance (Vestrucci et al., 2024).

This methodology also supports integration with external data sources, such as the Patient Data Management System (PDMS), through middleware and Application Programming Interfaces (APIs). The capacity to map real-time patient data onto formal checklist models enables automated decision support without compromising on explainability or safety constraints (Vestrucci et al., 2024; Seidl, 2024).

#### Safety-Critical Systems and Real-Time Constraints

Formal verification in Isabelle/HOL facilitates rigorous validation of safety-critical systems, akin to the verification of communication protocols in integrated clinical environments described by Bernardeschi et al. (2019). These environments require precise specification and validation of interactions among medical devices, clinician actions, and patient data. Similarly, the ICU extubation process necessitates strict adherence to safety conditions and real-time responsiveness, both of which can be modeled and verified within Isabelle/HOL.

Moreover, the modularity of HOL-based representations aligns with the structure of complex systems in clinical practice. By decomposing the SBT checklist into logical units, oxygenation checks, hemodynamic stability, neurological readiness, developers can verify individual components and their interactions systematically (Bernardeschi et al., 2019; Seidl, 2024).

#### Handling Uncertainty and Living Guidelines

As outlined by Oliveira et al. (2013), formal representations must accommodate incomplete and uncertain data, a common feature in clinical settings. Isabelle/HOL's expressiveness enables modeling of such scenarios, either through conservative approximations or explicit handling of unknown values. This makes the system not only reliable but also flexible enough to support the development of "living guidelines" that evolve with new evidence and clinical practices (Balser et al., 2004; Oliveira et al., 2013).

To conclude, Isabelle/HOL provides a rigorous framework for formalising medical checklists and protocols, especially in critical contexts such as ICU extubation. Its basis in HOL, combined with advanced proof automation and modular theory development, renders it well-suited for modelling, verifying, and evolving clinical decision logic. Among its automation techniques, the simplifier remains pivotal for goal refinement and logical rewriting and has been employed extensively throughout this thesis. By enabling transparent, traceable, and adaptable guideline formalisation, Isabelle/HOL directly advances the objectives of TAI and XAI in healthcare. The following Subsection 2.1.5 therefore turns to complementary symbolic and subsymbolic approaches, situating the Isabelle/HOL methodology within the broader landscape of AI techniques used for clinical decision support.

### 2.1.5 Related Approaches in Symbolic and Subsymbolic AI

The development of AI systems for critical care environments has drawn from both symbolic and subsymbolic paradigms. A review of recent approaches reveals not only their distinct contributions but also emerging hybrid methodologies that combine interpretability with data-driven adaptability. This section synthesizes representative symbolic, subsymbolic, and hybrid systems relevant to the modeling and formalization of SBT readiness and extubation decision-making.

#### Symbolic Approaches

Symbolic approaches remain foundational in applications requiring interpretability, safety, and formal verifiability. One prominent example is the GANESH system developed by Dojat et al. (1992), which encodes expert clinical knowledge about MV weaning into a knowledge-based rule system. GANESH operates in a closed-loop manner, continuously adjusting ventilatory support based on physiological parameters such as respiratory rate, tidal volume, and end-tidal carbon dioxide ($CO_2$). The system leverages 142 production rules to maintain patients within a zone of respiratory comfort and determine weaning readiness. Its rule-based structure and deterministic reasoning make it a model candidate for formalization using HOL frameworks such as Isabelle/HOL.

A more recent example of symbolic reasoning integrated into clinical decision support is presented by Cucchi et al. (2024). Their dynamic Clinical Decision Support Tool (CDST) is embedded within an electronic health record (EHR) and uses rule-based logic derived from clinical practice guidelines to evaluate ICU quality metrics, including SBT readiness and extubation decisions. The CDST demonstrated perfect accuracy in identifying both SBT readiness and successful trial completion, affirming the robustness of structured symbolic models in clinical workflows. Moreover, the logic of this system, based entirely on structured EHR data, is highly suitable for encoding as logical predicates and thresholds within formal verification environments such as Isabelle/HOL.

The importance of symbolic AI in safety-critical healthcare contexts is further reinforced by Freitas et al. (2020), who explore the application of formal verification techniques to medical device software. Through case studies involving intensive care technologies such as dialysis controllers and neurostimulators, they demonstrate how theorem proving in Isabelle/HOL can be used to validate system behavior against safety requirements. Their work contrasts correctness-by-construction, achievable through formal logic, with the more conventional correct-by-trial approach. Crucially, the paper underscores how formal verification supports dependability, traceability, and regulatory alignment, making it an indispensable methodology for ensuring the safe deployment of AI in medical environments (Freitas et al., 2020). The techniques they describe, though not specific to SBT evaluation, are directly transferable to the modeling and validation of structured ICU checklists.

#### Subsymbolic Approaches

While symbolic models excel in clarity and traceability, their rigidity and limited adaptability have motivated the integration of subsymbolic techniques. Subsym-

bolic models, typically based on ML, have demonstrated high performance in extubation outcome prediction but often lack the interpretability necessary for critical care deployment. Chen et al. (2019), for instance, utilized Light Gradient Boosting Machine (LightGMB) models trained on over 3,600 ICU patient records. The model achieved a high predictive performance, with an area under the curve (AUC) of 0.8130 and identified key predictors such as Partial Pressure of Arterial Oxygen ($PaO_2$), Partial Pressure of Carbon Dioxide ($PaCO_2$), and sedation duration. The inclusion of Shapley Additive Explanations (SHAP) provided some level of transparency but did not reach the structured rigor of symbolic reasoning (Chen et al., 2019).

Similarly, Igarashi et al. (2022) and Stivi et al. (2024) surveyed ML applications using neural networks and decision tree models to predict extubation success. These studies emphasized the potential of such models to improve clinical outcomes but also highlighted concerns regarding their opacity and generalizability. XAI methods, such as SHAP and model-specific transparency metrics, were incorporated to mitigate interpretability challenges, yet the lack of formal validation mechanisms limits their suitability for safety-critical decision support without supplementary safeguards (Igarashi et al., 2022; Stivi et al., 2024).

### Hybrid Approaches

In response to the limitations of both paradigms, hybrid systems have emerged that seek to combine the strengths of symbolic and subsymbolic AI. Fenske et al. (2024) propose a neuro-symbolic architecture for patient monitoring, treating it as a variant of human activity recognition. Their three-tiered system extracts sensor data through neural networks, interprets semantic states using symbolic logic, and infers high-level activities relevant to patient care. This layered structure offers a clear analogue to ICU protocols such as SBT readiness, where raw physiological inputs must be transformed into actionable clinical states. The incorporation of probabilistic logic frameworks, such as DeepProbLog, further enhances the ability of hybrid models to manage uncertainty while maintaining formal reasoning capabilities (Fenske et al., 2024).

A comparable hybrid methodology is employed by Huang et al. (2022) through their use of the Extended Classifier System with Real-valued Inputs (XCSR). This model combines symbolic rule structures with evolutionary learning mechanisms, generating interpretable decision rules from clinical and ventilator data collected during SBTs. With an AUC of 0.993, the model offers near-perfect predictive performance while preserving human-readable output. The identified rules, particularly those concerning respiratory rate and minute ventilation across timepoints, are not only clinically salient but also structurally suited for translation into formal logic representations in Isabelle/HOL.

Finally, Vidal et al. (2025) introduce Trust Knowledge Graph (TrustKG), a modular architecture integrating knowledge graphs with symbolic reasoning and subsymbolic models such as large language models and graph embeddings. Although their clinical focus lies outside the ICU, the framework's use of constraint validation, counterfactual reasoning, and explainable interfaces presents a transferable paradigm for ICU applications. TrustKG demonstrates how domain knowledge can be encoded and validated using Shapes Constraint Language (SHACL) and Web Ontology Language (OWL) ontologies while interacting with predictive models through transparent workflows. This approach

aligns closely with the objectives of this thesis, particularly regarding the use of Isabelle/HOL to formalize clinical logic and ensure correctness by construction.

The reviewed literature affirms the complementary roles of symbolic, subsymbolic, and hybrid approaches in modeling clinical decision-making. Symbolic systems offer verifiability and interpretability but require structured input and domain-specific rule sets. Subsymbolic models excel in pattern recognition and adaptability but fall short in traceability and safety assurance. Hybrid architectures seek to bridge this divide, offering robust performance while enabling formal verification and clinical accountability. The methods explored herein provide both methodological justification and technical scaffolding for the integration of formal symbolic logic into automated SBT readiness evaluation, as elaborated in the subsequent chapters.

Having outlined the methodological landscape of symbolic, subsymbolic, and hybrid AI approaches, in Section 2.2 the thesis now turns to the clinical context in which these methods are applied, specifically, the use of SBTs in critical care as a structured decision point for extubation readiness.

## 2.2 SBTs in Critical Care

SBTs represent a pivotal process in the ICU, serving as a clinical method for assessing whether a patient is ready to breathe independently without the aid of MV (Zein et al., 2016). This evaluation step is central to the broader weaning process, which encompasses the gradual withdrawal of ventilatory support and culminates in extubation. The success of SBTs has profound implications for patient outcomes, resource allocation, and overall ICU management. Due to their clinical significance, SBTs have become a focal point in the development of protocol-based decision support systems aimed at enhancing safety, efficiency, and consistency in critical care environments.

The rationale for focusing on SBTs within this thesis is twofold. First, SBTs embody a decision-making context that is both medically complex and time-sensitive, making them well-suited for formal modeling and verification. Second, they involve a clearly defined sequence of clinical criteria and protocol-based assessments that align structurally with symbolic reasoning frameworks. These features render SBTs an ideal use case for investigating the integration of formal logic into clinical decision-making, especially when aiming to ensure traceability, accountability, and adherence to best practices.

To contextualize this focus, Subsection 2.2.1 presents the clinical foundation of ICU operations and ventilator weaning. It introduces the structural characteristics of modern ICUs, the risks associated with prolonged MV, and the procedural stages involved in safe extubation. The discussion also covers challenges in protocol adherence, emphasizing the need for structured support systems.

Subsection 2.2.2 then examines the clinical effectiveness and economic implications of standardized weaning protocols. It draws on empirical evidence demonstrating improved patient outcomes and reduced ICU costs through protocolized weaning, particularly in resource-constrained settings. This section also outlines systemic barriers to implementation, highlighting the practical need for robust, automated, and context-sensitive decision support tools.

Together, these subsections provide the medical, procedural, and economic context necessary for the formalization efforts explored in the remainder of this thesis. They lay the groundwork for investigating how symbolic AI can enhance safety, reliability, and transparency in one of critical care's most consequential decision points: the readiness for extubation.

### 2.2.1 Clinical Background - ICU Structure, Ventilation Risks and Weaning Protocols

The ICU constitutes a highly specialized and organized environment for the management of critically ill patients. It is designed to deliver continuous monitoring, comprehensive physiological support, and specialized care during life-threatening episodes of organ dysfunction. According to the World Federation of Societies of Intensive and Critical Care Medicine (WFSICCM), an ICU encompasses a defined geographic area within a hospital, staffed by multidisciplinary teams and equipped with the technological capacity for advanced organ support, including MV (Marshall et al., 2017). The scope of ICU care frequently extends beyond its physical boundaries, interfacing with emergency departments, general wards, and follow-up clinics.

#### ICU Stratification and Function

ICUs are typically stratified into three levels based on the intensity of care delivered. Level 1 units provide basic monitoring and oxygen therapy, primarily for patients with mild organ dysfunction. Level 2 ICUs are capable of delivering invasive monitoring and short-term life support, while Level 3 units offer comprehensive, round-the-clock care with full-spectrum life support technologies and specialist personnel (Marshall et al., 2017). Tertiary-level ICUs, in particular, serve as regional centers for critical care delivery, research, and education, making them the principal setting for complex interventions such as prolonged MV and advanced weaning protocols.

#### Risks of MV and Extubation

MV constitutes a cornerstone of organ support in the ICU, yet it is associated with significant risks, particularly when prolonged unnecessarily. Extended intubation has been repeatedly linked to increased morbidity and mortality, arising from complications such as VAP, airway injury and diaphragmatic dysfunction (Belle et al., 2020). Furthermore, the decision to discontinue MV must carefully balance the risks of premature extubation, leading to re-intubation and associated complications, with those of delayed weaning, which can prolong ICU stay and increase healthcare costs (Liu et al., 2022).

#### Stages of the Weaning Process

The clinical process of weaning from MV generally unfolds in three stages: *readiness assessment*, *SBT*, and *extubation evaluation*. Readiness assessment determines whether patients meet physiological criteria such as stable hemodynamics, adequate oxygenation as measured by the ratio of $PaO_2$ to the fraction of inspired oxygen ($FiO_2$), with a threshold greater than 150 millimetres of mercury

(mmHg), minimal ventilatory support as indicated by a positive end-expiratory pressure (PEEP) of no more than 10 centimetres of water ($cmH_2O$) and an $FiO_2$ not exceeding 50%, as shown in Figure 2.1, and sufficient neurological alertness (Ajay et al., 2022). Patients meeting these thresholds proceed to an SBT, during which they are observed for 30 minutes (min) to 120min under minimal ventilatory support or spontaneous breathing modes. Successful tolerance of the SBT, indicated by stable respiratory and cardiovascular parameters, paves the way for the final extubation assessment.

<table>
<tr><td colspan="3">Patientenname:</td><td>Fall Nr:</td><td colspan="2">Datum:</td></tr>
<tr><td colspan="6">Spontaner Atemversuch (SBT) und Bewertung (SOP Nr:..... )</td></tr>
<tr><td colspan="6">SBT nicht durchgeführt aufgrund von:<br>TM-Weaning End of Life Care Langzeitbeatmung Sonstiges (bitte dokumentieren)</td></tr>
<tr><td colspan="6">CAVE: Bei prolongiertem Weaning wird der SBT individualisiert geplant und durchgeführt</td></tr>
<tr><td colspan="3">SBT Sicherheitsscreen</td><td>SBT-Zeit (30-120Min)</td><td>1:____</td><td>2:____</td></tr>
<tr><td>Spontane Atemversuche?</td><td>Ja</td><td>Nein</td><td>Falls RSBI (AF/Vt) nach 2 Minuten mehr als 110 Atemzüge/Min/L den Patienten 15 Minuten lang überwachen</td><td></td><td></td></tr>
<tr><td>$PaO_2/FiO_2$ größer 150 mmHg und $FiO_2$ weniger als 50%?</td><td>Ja</td><td>Nein</td><td>Basis HF<br>1)____2)____<br><br>Basis Blutdruck<br>1)____2)____</td><td></td><td></td></tr>
<tr><td>PEEP von 10cm $H_20$ oder weniger?</td><td>Ja</td><td>Nein</td><td>$SpO_2$ unter 88% für mehr als 2 Minuten</td><td></td><td></td></tr>
<tr><td>Norepinephrin 0.2 mcg/Kg/Min oder weniger (oder Äquivalent) und nicht mehr als 1 vasoaktives Medikament?</td><td>Ja</td><td>Nein</td><td>Atemfrequenz von mehr als 35 Atemzügen/ Minute und Anzeichen von Atemnot für mehr als 5 Minuten</td><td></td><td></td></tr>
<tr><td>Für Patienten mit akuter Hirnverletzung: Ist der neurologische Zustand des Patienten stabil, ohne Bedenken hinsichtlich eines erhöhten ICP (dazu gehört auch die $CO_2$-Kontrolle)?</td><td>Ja</td><td>Nein</td><td>Andere Gründe (bitte dokumentieren)</td><td></td><td></td></tr>
<tr><td colspan="3"></td><td>Patient hat SBT bestanden?</td><td>Ja Nein</td><td>Ja Nein</td></tr>
<tr><td>Wenn alle oben genannten Fragen 'Ja' sind, wird ein SBT gemäß SOP Nr:.... durchgeführt. Der Sicherheitsscreen kann bei Bedarf wiederholt werden</td><td></td><td></td><td>Wenn SBT bestanden aber Patient nicht extubiert, dokumentieren Sie bitte die Gründe</td><td></td><td></td></tr>
</table>

Figure 2.1: Clinical checklist for assessing readiness for a SBT of Hospital Bamberg

### Challenges in Protocol Adherence

Despite the existence of well-defined protocols, extubation practices often remain inconsistent, particularly in resource-limited settings. Studies have revealed substantial gaps in compliance with weaning protocols, attributed to factors such as inadequate training, insufficient awareness, and operational constraints (Ajay et al., 2022). In one such study, the introduction of a structured weaning protocol in a high-burden tertiary care ICU led to notable improve-

ments in compliance, readiness assessment, and extubation screening. While the overall duration of ventilation and ICU stay did not change significantly, the protocol implementation was associated with a decrease in re-intubation rates and VAP, underlining the clinical value of standardized practices (Ajay et al., 2022).

#### Timeliness and Quality Improvement Initiatives

Timeliness of extubation remains a critical determinant of patient outcomes. Delays in extubation following successful SBTs, even by a few hours, can increase the risk of complications and prolong ICU stays. Belle et al. (2020) emphasized this concern in a quality improvement initiative that implemented a screening tool to expedite extubation decisions. Although the tool improved readiness evaluations, systemic barriers such as pre-rounding demands and lack of targeted education hindered timely action. These findings underscore the need for not only clinical guidelines but also effective implementation strategies that address educational and logistical constraints.

#### AI-Based Support for Weaning Decisions

Advancements in AI have introduced new modalities for supporting ventilator weaning decisions. Liu et al. (2022) developed a two-stage AI system for predicting the optimal timing of weaning and extubation, leveraging electronic medical record data to improve consistency and reduce intubation durations. Their system demonstrated strong predictive performance and was well-received by clinical staff, highlighting the potential for computational approaches to augment protocol adherence and decision-making accuracy.

#### Role of the Safety Screen in Preventing Complications

A particularly critical component of the weaning process is the initial safety screen that precedes the SBT. This step determines whether a patient is stable enough to undergo spontaneous breathing evaluation and is intended to prevent premature progression toward extubation. Clinical consequences of bypassing or misapplying the safety screen include extubation failure, reintubation, and increased risk of infection and mortality (Quintard et al., 2017; Murselović et al., 2024). Despite its clinical significance, this screening process is not always applied with consistent rigor. The formalization and automation of the safety screen therefore offer a promising means of enhancing reliability, reducing human error, and supporting safe extubation practices in intensive care settings.

Having outlined the clinical rationale and procedural context of mechanical ventilation and weaning, in Subsection 2.2.2 the discussion now turns to empirical evidence on the effectiveness and economic implications of standardized weaning protocols in intensive care settings.

### 2.2.2 Effectiveness and Economic Impact of Protocol-Based Weaning

The clinical and economic outcomes of protocol-based weaning strategies have been the subject of extensive empirical investigation, particularly in response

to the rising demand for cost-effective and standardized care in ICUs. These strategies aim to reduce both the duration and complications of MV by systematizing the decision-making process through structured protocols and checklists. This subsection examines the effectiveness of such interventions and evaluates their economic implications based on recent empirical findings.

### Clinical Effectiveness of Weaning Protocols

Empirical evidence suggests that protocol-based weaning can enhance clinical safety and consistency in extubation practices, especially when applied in settings with significant resource constraints. In a quasi-experimental study conducted in a high-burden tertiary ICU in India, Ajay et al. (2022) implemented a structured weaning protocol encompassing readiness-to-wean assessments, SBTs, and pre-extubation screenings. Training medical personnel in the protocol significantly improved compliance across these stages. The study reported an increase in successful extubation rates from 50% to 66% and a reduction in VAP from 34% to 18%. Although the duration of MV decreased from 10 days to 7.3 days and ICU length of stay declined from 9.2 days to 8.2 days, these reductions did not reach statistical significance. Nevertheless, the downward trends underscore the potential for clinical improvement through standardized practice (Ajay et al., 2022).

### Economic Impact of Delayed Extubation and Protocolized Weaning

The effectiveness of protocolized weaning is closely linked to its capacity to reduce avoidable complications associated with prolonged MV. VAP, for instance, is a severe and common complication that extends ICU stay and increases mortality risk. Kaier et al. (2019) estimated that each episode of VAP adds an average of 3.52 additional ICU days, translating to an incremental financial burden of approximately €5,565 per patient in Germany and €5,822 in France. These costs are primarily driven by extended durations of MV rather than the ICU stay alone, highlighting the economic urgency of timely extubation.

From an economic perspective, the implementation of weaning protocols can be seen as a cost-containment strategy. MV significantly increases ICU costs due to elevated demands for personnel, equipment, and monitoring. According to Kaier et al. (2020), the presence of MV was associated with a 59% increase in daily ICU costs in Germany, raising the average daily expenditure from €999 to €1,590. A systematic review and meta-regression by the same authors further confirmed this trend, establishing a 25.8% average increase in daily ICU costs attributable to MV across multiple healthcare systems (Kaier et al., 2019). This financial burden is exacerbated in patients with prolonged intubation or ventilation-associated complications, thereby reinforcing the relevance of efficient weaning protocols.

### Financial Burden in Pandemic and High-Utilization Scenarios

The cost implications are even more pronounced in the context of pandemics or large-scale ICU utilization. Zwerwer et al. (2024), in a study of Coronavirus Disease (COVID-19) patients in a German university hospital, found that each additional day of MV in the ICU incurred a cost of €2,224.84. This figure is more than double the cost of a non-ventilated ICU day, which was estimated

at €927.45. Median hospitalization costs for mechanically ventilated COVID-19 ICU patients reached €21,536, compared to €5,887 for non-ventilated ICU patients. These findings not only reaffirm MV as a primary cost driver but also demonstrate how extended ventilation periods intensify the financial strain on healthcare systems.

#### Barriers to Protocol Adherence

Despite these clear advantages, full adherence to protocol-based weaning remains challenging, particularly in resource-limited environments. As reported by Ajay et al. (2022), certain protocol components, such as SBT monitoring, had persistently low adherence rates even after targeted training interventions. Barriers included limited availability of monitoring equipment, staffing constraints, and variable clinician engagement. These limitations underscore the need for implementation strategies that integrate not only evidence-based checklists but also educational initiatives and infrastructural support.

In conclusion, protocol-based weaning represents a clinically effective and economically prudent approach to managing MV in ICU settings. It reduces complication rates, enhances decision-making consistency, and curtails healthcare expenditures by shortening ventilation durations and preventing adverse events. While the degree of economic benefit may vary by context, the cumulative evidence supports the systematic integration of weaning protocols as a standard component of critical care. Their formalization and automation, as explored in subsequent chapters of this thesis, further enhance their applicability, especially in high-complexity and resource-constrained environments.

Building on the clinical and economic rationale for standardized weaning protocols, Section 2.3 examines how such structured approaches can be operationalized through CDSSs and enhanced with AI to improve consistency and transparency in critical care decision-making.

## 2.3 CDSSs and AI Integration

CDSSs assist healthcare professionals by offering structured guidance during diagnostic and therapeutic processes. As clinical protocols grow in complexity and data availability increases, CDSSs have become central to promoting consistent, safe, and efficient decision-making.

The integration of AI into CDSSs has expanded their capabilities, allowing for more dynamic and context-aware support. By incorporating formal logic and computational reasoning, modern systems aim not only to assist but also to explain their recommendations in transparent and verifiable ways.

Subsection 2.3.1 introduces the foundations and typical structures of conventional CDSSs. Subsection 2.3.2 explores how AI is applied within these systems to enhance reasoning and adaptability. Subsection 2.3.3 then discusses how clinical protocols, especially in intensive care, can be formalized for use in automated and trustworthy decision support.

### 2.3.1 Overview of CDSSs

CDSSs constitute a pivotal technological advancement in modern healthcare, aimed at enhancing clinical decision-making through the delivery of intelligently processed, patient-specific information at the point of care. Functioning as integral components of EHR systems, CDSSs are designed to provide the right information to the right individual at the right time and location, thereby augmenting clinical efficiency, accuracy, and safety (Kawamoto and Fial, 2016; Sutton et al., 2020).

#### Definition and Core Principles

Across the literature, CDSSs are consistently defined as computerized tools that assist clinicians in their decision-making by integrating clinical knowledge with patient data. As articulated by Berner and La Lande (2007), these systems influence clinician actions by delivering patient-specific assessments at critical moments during the care process. Similarly, Kawamoto and Fial (2016) emphasize the situational specificity of CDSSs, which provide filtered and actionable insights precisely when and where clinical decisions are made. Sutton et al. (2020) further characterize CDSSs as systems that embed targeted clinical knowledge into the care pathway, supporting medical decisions with structured reasoning and relevant data inputs.

#### Active vs. Passive Systems

A fundamental characteristic of CDSSs is their capacity to function as active or passive decision aids. Active systems proactively generate alerts, reminders, or recommendations, while passive systems require clinicians to initiate data retrieval or consult the system manually (Sutton et al., 2020). These functionalities are tailored to address varying clinical needs and decision-making contexts, from real-time critical care interventions to guideline-based protocol adherence.

#### Knowledge-Based vs. Non-Knowledge-Based CDSSs

CDSSs may be broadly categorized into knowledge-based and non-knowledge-based systems. Knowledge-based CDSSs operate through rule-based reasoning mechanisms, often structured as `IF-THEN` rules derived from clinical guidelines or expert consensus. These systems are characterized by transparency, as the reasoning process is explicitly encoded and interpretable by human users. Common applications include medication safety checks, diagnostic prompts, and guideline compliance monitoring (Berner and La Lande, 2007; Sutton et al., 2020).

In contrast, non-knowledge-based CDSSs rely on data-driven methodologies such as ML or neural networks. These systems identify patterns and generate outputs through statistical modeling rather than predefined rules. While they offer adaptability and scalability for large datasets, their lack of explainability poses significant challenges for trust and clinical adoption (Berner and La Lande, 2007; Sutton et al., 2020). The dichotomy between these approaches highlights a central tension in CDSS design: the trade-off between performance and interpretability.

### Types of CDSSs by Intervention Modality

Additional classifications based on intervention modality further distinguish CDSS types. Kawamoto and Fial (2016) identify several prominent categories, including point-of-care alerts and reminders, order facilitators (e.g., predefined order sets), relevant information displays, and expert systems providing diagnostic or therapeutic recommendations. Each type plays a unique role in enhancing different aspects of the clinical workflow, from reducing information overload to standardizing treatment protocols.

### Clinical Benefits and Use Cases

Extensive empirical evidence underscores the benefits of CDSSs in improving patient safety, clinical outcomes, and operational efficiency. By providing alerts for drug interactions or inappropriate dosing, CDSSs have been shown to significantly reduce adverse drug events (Sutton et al., 2020). Furthermore, these systems enhance adherence to clinical guidelines and protocols, supporting consistent and evidence-based decision-making (Berner and La Lande, 2007; Kawamoto and Fial, 2016). In intensive care contexts, CDSSs have demonstrated measurable benefits, such as improved survival rates in ARDS through the use of rule-based ventilation protocols (Kawamoto and Fial, 2016).

### Operational Advantages

Operationally, CDSSs streamline workflows by reducing redundancy and automating administrative functions. Order facilitators and data summarization tools have been particularly effective in minimizing unnecessary testing and enabling more complete and standardized clinical documentation (Kawamoto and Fial, 2016).

### Implementation Barriers

Despite their proven benefits, the widespread implementation of CDSSs continues to face significant barriers. A major technical challenge lies in achieving seamless integration with existing health information systems, particularly due to variability in data formats and the lack of interoperability standards (Berner and La Lande, 2007; Sutton et al., 2020). This challenge is especially pertinent in settings where CDSSs must interface with complex and heterogeneous systems such as PDMS.

Another persistent issue is alert fatigue, which arises when clinicians are exposed to excessive or non-specific system notifications. Studies have shown that up to 95% of CDSS alerts are overridden or ignored, diminishing their intended effectiveness and potentially undermining clinical trust in the system (Berner and La Lande, 2007; Sutton et al., 2020). Moreover, manual data entry requirements can disrupt clinical workflows, increasing the cognitive and administrative burden on clinicians.

Cultural and organizational factors also affect CDSS adoption. Resistance may stem from perceived intrusions on clinical autonomy, lack of transparency in decision logic, particularly in non-knowledge-based systems, or inadequate training in system use (Berner and La Lande, 2007).

#### Relevance to This Thesis

The foundational concepts of CDSSs are directly relevant to the formalization and automation of ICU protocols, as addressed in later sections of this thesis. Specifically, the knowledge-based approach to CDSSs aligns with the goal of encoding the SBT safety checklist in a structured, machine-verifiable format using Isabelle/HOL. This method supports explainability and formal validation, both of which are essential for safety-critical applications in critical care. Furthermore, the use of predefined templates and order sets, as exemplified by the "Order Facilitator" type of CDSS, provides a practical analogue for the structured automation of extubation decision pathways (Kawamoto and Fial, 2016).

CDSSs represent a mature and evolving domain within healthcare informatics. Their capacity to improve clinical outcomes, reduce costs, and promote standardized care is well documented, though significant implementation challenges persist. The integration of formal logic and symbolic reasoning into CDSS design, as proposed in this thesis, addresses these challenges by enhancing transparency, traceability, and clinical trust, attributes that are particularly vital in the context of ICU protocol automation. Against this backdrop, the following Subsection 2.3.2 explores how AI, and especially symbolic approaches, can further augment CDSSs by enabling dynamic, explainable, and verifiable decision-making in critical care environments.

### 2.3.2 AI-based Approaches in CDSS

AI has become an increasingly prominent element within CDSSs, augmenting their capabilities through the incorporation of advanced computational reasoning and dynamic pattern recognition. While a diverse range of AI methodologies has been employed, the particular requirements of clinical environments, such as safety, interpretability, and formal validation, have sustained continued interest in symbolic and knowledge-based approaches. This section examines contemporary AI-based approaches to CDSSs, foregrounding the distinctions between symbolic, subsymbolic, and hybrid methodologies, and situating them in relation to the goals of transparent, verifiable, and clinically accountable decision support.

#### Symbolic vs. Subsymbolic Paradigms

Methodologically, AI integration into CDSSs is typically structured around three paradigms: *symbolic* (knowledge-based), *subsymbolic* (data-driven), and *hybrid systems*, which combine elements of both. Symbolic AI relies on formally encoded clinical knowledge, such as rule sets, ontologies, and logical inference mechanisms. These models offer intrinsic explainability, modularity, and auditability, qualities of particular relevance in domains where accountability and safety are paramount. Subsymbolic models, by contrast, learn patterns from data using statistical techniques such as neural networks or ensemble learning. While these systems have demonstrated considerable predictive accuracy, especially in high-dimensional datasets, their opaque reasoning processes pose challenges for traceability and regulatory acceptance (Montani and Striani, 2019).

### Enduring Relevance of Symbolic AI

Despite growing enthusiasm for data-driven models, symbolic approaches retain a fundamental role in healthcare due to their alignment with the procedural and ethical demands of clinical reasoning. As noted by Montani and Striani (2019), symbolic AI enables the construction of systems whose behavior can be formally verified and systematically adapted without retraining. These properties render symbolic methods particularly suitable for the formalization of clinical protocols, as exemplified in this thesis by the use of the Isabelle/HOL theorem prover. SymbolicAI's capacity to represent medical knowledge in structured, interpretable formats not only facilitates explainability but also supports integration with external systems such as PDMS, enabling traceable and compliant automation.

### Rise of Hybrid Approaches

Nonetheless, recent literature indicates a prevailing dominance of data-driven systems in current CDSS research. Montani and Striani (2019) report that the majority of contemporary contributions adopt subsymbolic AI methods, primarily due to their scalability and adaptability. However, the authors also emphasize the increasing interest in hybrid architectures, which combine the pattern recognition capabilities of subsymbolic models with the transparency and formal grounding of symbolic systems. These hybrid approaches seek to mitigate the limitations of each paradigm by integrating data-driven insights with logical representations, offering a promising direction for developing robust yet interpretable decision support.

### Clinical and Operational Benefits

The practical benefits of AI in CDSSs are considerable. As outlined by Sunarti et al. (2021), AI technologies contribute to improved diagnostic precision, streamlined clinical workflows, cost reduction, and broader access to healthcare services. Systems such as Google's DeepMind Health exemplify the potential of AI-driven decision-making to augment clinician performance and reduce administrative burden. Importantly, symbolic systems offer a pathway to integrate these benefits while preserving rigorous validation and audit trails, an essential requirement in safety-critical domains such as intensive care.

### Ethical and Regulatory Challenges

However, the introduction of AI into clinical settings is accompanied by substantive ethical, technological, and regulatory challenges. Among the most salient concerns is the lack of explainability in many high-performance AI models. Esmaeilzadeh (2020) identifies trust as the most critical determinant of risk perception in AI-driven healthcare, emphasizing that ethical concerns, ranging from data privacy and algorithmic bias to diminished patient autonomy, must be addressed to secure widespread acceptance. Technological issues, particularly those pertaining to opaque functionality and poor communication design, further intensify these perceptions of risk. Symbolic systems, with their capacity for explicit representation and formal reasoning, offer a direct response to these challenges by making the logic of recommendations transparent and verifiable.

#### Human-Centered Design and Communication

The importance of human-centered design is also emphasized in the literature. Communication-related barriers, such as the perceived loss of interpersonal interaction and empathy in AI-mediated care, are highlighted as significant predictors of public hesitance. Esmaeilzadeh (2020) notes that integrating social cues into AI interfaces, such as through empathetic chatbots or assistive devices, may partially alleviate these concerns. Nevertheless, the underlying requirement remains the development of systems whose recommendations can be explained, challenged, and understood, a standard that symbolic AI meets more readily than black-box alternatives.

#### Patient Trust and Adoption

Patient acceptance further shapes the trajectory of AI in CDSSs. Although the general public increasingly recognizes the potential of AI technologies, adoption remains tempered by unfamiliarity and fear of loss of control. The findings of Esmaeilzadeh (2020) suggest that emphasizing the benefits of AI systems is insufficient without simultaneously addressing technological and ethical concerns. In this context, symbolic AI's traceability and normative rigor provide a meaningful foundation for fostering trust, especially when paired with regulatory oversight and stakeholder engagement.

#### Need for Standardization and Certification

To that end, regulatory frameworks play a pivotal role in ensuring the responsible deployment of AI-based CDSSs. As both Esmaeilzadeh (2020) and Sunarti et al. (2021) argue, the absence of standardized evaluation criteria and enforcement mechanisms exacerbates uncertainty and mistrust. Regulatory bodies must therefore implement clear guidelines and audit mechanisms, with formal models offering a reliable basis for certification and continuous validation. Symbolic , by enabling the formal specification and verification of clinical logic, directly supports these objectives and can serve as a normative infrastructure for accountable AI in healthcare.

While subsymbolic and hybrid models dominate the contemporary discourse on AI-based CDSSs, symbolic AI continues to play a vital role in applications where transparency, formal correctness, and ethical accountability are paramount. Its ability to encode clinical guidelines as verifiable logical structures, as demonstrated through the use of Isabelle/HOL in this thesis, provides a strong foundation for trustworthy automation. As efforts to integrate AI into healthcare systems advance, symbolic reasoning offers a principled and actionable path toward building reliable, explainable, and safety-oriented decision support. These capabilities are especially relevant in intensive care settings, where formalizing protocols such as the SBT checklist demands rigorous validation and seamless integration with real-time clinical workflows, topics explored in greater detail in Subsection 2.3.3.

### 2.3.3 Formalizing ICU Protocols and Guidelines

In intensive care settings, clinical protocols serve as the foundation for standardized, evidence-based care. However, their implementation is frequently chal-

lenged by ambiguity, incompleteness, and variability in interpretation. These issues are particularly consequential in safety-critical procedures such as SBTs, where extubation decisions must be both timely and precise. To address these challenges, the formalization of ICU protocols using logic-based methods offers a structured and verifiable approach to improving clinical consistency, safety, and traceability.

**Justification for Formalization in Critical Care**

Formal methods rely on mathematically grounded representations to model and verify system properties. When applied to clinical practice, these methods allow guidelines to be encoded into structured logical models, enabling rigorous validation of their internal consistency and compliance with safety requirements (Hommersom et al., 2008; Balser et al., 2004). This transformation is essential in environments such as the ICU, where complex clinical dependencies and real-time decision-making necessitate high levels of precision and accountability.

A recurring difficulty in guideline implementation stems from their traditional format: extensive, text-based documents that are difficult to apply uniformly, particularly under time pressure. The Protocure project addressed this issue by proposing a stepwise formalization process that begins with informal textual protocols and culminates in fully formal representations expressed in HOL (Balser et al., 2004). This method has proven effective in uncovering logical inconsistencies and ambiguities in clinical protocols, offering a valuable precedent for modeling extubation readiness checklists.

A comparable approach is presented by Pérez and Porres (2010), who employ model checking and model-driven development to enable non-experts to verify guidelines using structured statecharts. Their methodology underscores the importance of making formal tools accessible to multidisciplinary ICU teams, facilitating collaboration across clinical and technical domains. Similarly, Oliveira et al. (2013) demonstrate that decision models using extended logic programming can capture incomplete and uncertain information, conditions commonly encountered in ICU scenarios, thus ensuring that formalized guidelines remain applicable even under informational constraints.

The use of HOL, as implemented in theorem provers such as Isabelle/HOL, provides the expressive power necessary for modeling time-sensitive ICU protocols and their modular components. While the specific features and capabilities of Isabelle/HOL are addressed in Subsection 2.1.4, it is important to highlight its relevance in the context of ICU formalization. In particular, Isabelle/HOL allows extubation criteria, such as oxygenation thresholds and hemodynamic stability, to be encoded as verifiable logical conditions, ensuring that each protocol step is both transparent and traceable. As discussed in Subsection 2.3.2, this level of explainability is critical for gaining clinician trust and for aligning with ethical and regulatory standards.

Moreover, formal representations support the development of "living guidelines", protocols that can evolve with emerging medical knowledge while preserving logical integrity. This adaptability is essential in critical care, where treatment standards are continually updated in response to new evidence. Integrating formalized protocols with real-time data sources such as PDMS further enhances their utility, enabling dynamic evaluation of patient-specific conditions and automated compliance monitoring.

**Application in This Thesis: Formalizing the SBT Checklist**

In the context of this thesis, formalizing the SBT safety checklist exemplifies the practical application of these principles. By structuring checklist logic within a formal verification environment, the system ensures consistent interpretation of readiness criteria and supports automated decision-making aligned with clinical best practices. Rather than replacing clinical judgment, this formalization provides a robust foundation for computational decision support, reducing the likelihood of error and facilitating safe, transparent extubation workflows.

The formalization of ICU protocols addresses core challenges in interpreting and applying clinical guidelines, particularly under the high-stakes conditions of critical care. By enabling precise, verifiable, and adaptable representations of medical logic, formal methods provide a foundation for consistent decision-making and transparent automation. Within this framework, the SBT checklist serves as a practical example of how symbolic logic can enhance safety and accountability in extubation workflows. As these systems begin to interface more directly with real-time clinical data and broader hospital infrastructures, their effectiveness increasingly depends on seamless and secure integration, an aspect governed by standards in healthcare IT, which are examined in Section 2.4.

## 2.4 Standards in Healthcare IT

The reliable and secure exchange of medical information is fundamental to modern healthcare delivery. As clinical systems grow increasingly interconnected and digital infrastructures expand, standardized approaches to data representation, communication, and protection have become indispensable. Healthcare IT standards ensure that diverse systems, ranging from bedside monitoring devices to hospital information systems, can communicate effectively, preserve data integrity, and uphold the confidentiality of sensitive patient information.

In the context of this thesis, which addresses the formalization and automation of clinical decision protocols, adherence to healthcare IT standards plays a pivotal role. Standardized data exchange mechanisms enable the integration of clinical knowledge models with external patient data sources, thereby facilitating real-time decision support. Similarly, robust data security frameworks ensure that the deployment of such systems respects ethical, legal, and institutional requirements for privacy and safety. These aspects are particularly critical when interfacing with clinical environments such as ICUs, where timely access to accurate and protected data directly impacts patient outcomes.

The following sections explore two foundational domains of healthcare IT standards. Subsection 2.4.1 introduces widely adopted communication protocols and data interchange frameworks, with a focus on HL7 and its relevance for structured medical information exchange. Subsection 2.4.2 examines data security standards that govern the protection of health information throughout its lifecycle, from transmission to storage and access control.

Together, these sections provide the technical and regulatory backdrop necessary for understanding how clinical logic models can be integrated safely and effectively within healthcare IT infrastructures.

### 2.4.1 HL7 and Data Interchange Standards

The electronic exchange of clinical data across heterogeneous information systems is a foundational requirement for modern healthcare delivery. To enable such interoperability, healthcare environments rely on standardized communication protocols that ensure semantic and syntactic consistency. Among these, the HL7 family of standards has emerged as the most widely adopted framework for structuring and transmitting health information. HL7 facilitates the integration of patient data between disparate systems such as EHRs, PDMSs, and CDSSs, thereby playing a critical role in enabling real-time, informed, and coordinated clinical care.

#### Historical Context and HL7 v2

Originating in the late 1980s, HL7 was developed to provide a formalized framework for modeling and exchanging health-related data across systems (iNTERFACEWARE Inc., 2025). The initial versions of the standard focused on defining message formats for common hospital events, particularly Admissions, Discharges, and Transfers (ADT). These efforts culminated in HL7 v2, which was first widely adopted in 1998 and remains the dominant messaging protocol in clinical environments today (Oluwalade, 2024).

HL7 v2 provides syntactic interoperability by employing structured, event-driven messages composed of delimited segments. Each message is initiated in response to a trigger event and is constructed using a set of predefined delimiters (e.g., |, ^, ~, &). Segments are identified using three-letter codes, such as the Message Header Segment (MSH), and fields within these segments convey clinical and administrative information including patient demographics, lab results, vital signs, and medication orders (Oluwalade, 2024; iNTERFACEWARE Inc., 2025). Its backward compatibility across versions has facilitated long-term adoption by allowing legacy systems to interact with modern applications without extensive reconfiguration.

Despite its technical efficacy and widespread use, HL7 v2 is not without limitations. Its inherent flexibility, intended to support diverse clinical workflows, has also led to inconsistent implementations and semantic ambiguities. These challenges often necessitate custom extensions, such as "Z-segments," to address site-specific needs, which can further hinder interoperability and reuse (iNTERFACEWARE Inc., 2025).

#### HL7 v3 and the Shift to Semantic Rigor

To address these semantic deficiencies, HL7 v3 was introduced. This iteration aimed to establish a more rigorous semantic framework based on the Reference Information Model (RIM), an object-oriented structure representing core healthcare concepts such as "Act," "Role," and "Entity" (Oluwalade, 2024)(Oluwalade, 2024). While v3 provided greater expressiveness and formalization, it was criticized for its complexity and lack of backward compatibility. These factors contributed to its limited adoption, particularly in the United States, where existing HL7 v2 infrastructures remained deeply entrenched (iNTERFACEWARE Inc, 2025).

### Fast Healthcare Interoperability Resources (FHIR): A Modern, Web-Based Standard

The shortcomings of HL7 v3 catalyzed the development of the more flexible and modern standard FHIR. Initiated in 2011 and released in 2014, FHIR integrates the strengths of HL7 v2 and v3 while aligning with contemporary web technologies. It is built on a Representational State Transfer (REST) architecture and supports data serialization in human-readable formats such as JavaScript Object Notation (JSON) and Extensible Markup Language (XML). FHIR structures clinical data into modular "resources", such as Patient, Observation, and Encounter, that can be accessed, updated, and queried through standard Hypertext Transfer Protocol (HTTP) methods (Oluwalade, 2024; iNTERFACEWARE Inc., 2025). This design simplifies implementation and promotes scalability.

Nevertheless, the transition to FHIR poses its own challenges, particularly in environments dominated by HL7 v2-based infrastructure. Bridging these systems requires careful mapping and validation to ensure semantic integrity and consistent data interpretation.

In practical terms, HL7 standards serve as the backbone for a wide range of clinical workflows. Common use cases include the transmission of laboratory results, vital signs, medication prescriptions, diagnostic reports, and admission or discharge notifications. In ICUs, where real-time data accuracy is paramount, HL7 v2 is often employed to facilitate continuous communication between bedside devices, PDMS platforms, and external applications (Oluwalade, 2024).

### Application in This Thesis

Within the scope of this thesis, HL7 v2 serves as the primary mechanism for simulating patient data exchange from the PDMS to the Java-based system that supports the formalization of SBT readiness checklists. Although a mocked interface is used during development, the design preserves compatibility with real-world HL7 v2 data formats to ensure future integration feasibility. The structured exchange of patient parameters, such as oxygenation ratios, ventilator settings, and hemodynamic indicators, is critical for the real-time evaluation of extubation readiness within the Isabelle/HOL formal verification framework.

The HL7 suite of standards provides the essential infrastructure for healthcare data interoperability. While HL7 v2 remains dominant due to its widespread adoption and backward compatibility, FHIR introduces a modernized, web-based framework that facilitates scalable and flexible integration. A clear understanding of these standards and their respective strengths and limitations is crucial for designing clinical systems that depend on structured, reliable, and actionable data. As this thesis aims to formalize and automate ICU decision protocols, HL7 enables logical models to interact meaningfully with clinical environments by ensuring access to consistent and semantically aligned patient data. This capacity for integration, however, must be balanced with stringent safeguards to protect sensitive health information, requirements that are addressed through healthcare data security standards, explored in the following Subsection 2.4.2.

### 2.4.2 Data Security Standards

The secure handling of patient data represents a foundational requirement in the design and deployment of clinical systems, particularly those interfacing with sensitive environments such as ICUs. This section delineates the regulatory frameworks, technical safeguards, and institutional responsibilities that govern the protection of healthcare information, with an emphasis on the European GDPR and German critical infrastructure requirements, KRITIS and B3S. These standards not only establish the legal parameters for data use and processing but also shape the ethical and technical constraints within which systems such as the one formalized in this thesis must operate.

#### Core Principles of the GDPR in Healthcare

Enforced since May 2018, the GDPR provides a unified legal structure for the protection of personal data within the European Union. In healthcare contexts, it applies to a broad spectrum of sensitive information, including biometric identifiers, EHRs, and treatment data. The GDPR emphasizes the principles of data minimization, accuracy, and integrity, requiring healthcare providers and system developers to collect and process only the data necessary for defined purposes (Dougherty, 2025).

A central requirement under the GDPR is the acquisition of explicit patient consent prior to any form of data processing or communication, unless narrowly defined exceptions apply (of Radiology, 2017). Furthermore, patients retain the right to access their personal data, request corrections, obtain data portability in machine-readable formats, and, in certain contexts, invoke the right to erasure (Dougherty, 2025). These rights impose critical design constraints on any system handling patient data, particularly in applications such as extubation readiness evaluation where real-time access to clinical parameters is essential.

The GDPR also mandates the implementation of Technical and Organizational Measuress (TOMs) to protect patient data. Recommended techniques include *anonymisation*, *pseudonymisation*, *encryption*, *access controls*, and *audit trails* (of Radiology, 2017; Dougherty, 2025). Pseudonymisation, in particular, is relevant to research-oriented applications where data identifiability must be suppressed without entirely severing the link to clinical context.

In instances where healthcare data are processed for secondary purposes, such as research or public health planning, Article 89 of the GDPR permits conditional exemptions from certain consent and purpose limitations. These exemptions, however, require additional safeguards and the prevention of misuse by third parties (of Radiology, 2017).

Given the federal nature of healthcare governance within the European Union (EU), the GDPR accommodates national derogations through which member states may define additional regulations. In Germany, this flexibility manifests in the integration of GDPR principles within sector-specific laws and frameworks, such as KRITIS and the IT Security Act (IT-SIG) (IT-Sicherheitsgesetz).

#### KRITIS and B3S: German-Specific Extensions

In the German context, hospitals serving more than 30,000 patients annually are classified as KRITIS and are subject to comprehensive cybersecurity obligations

under the Federal Office for Information Security (BSI) (Bundesamt für Sicherheit in der Informationstechnik). These obligations are operationalized through the B3S, which outlines 168 domain-specific controls for healthcare institutions (ForeNova Technologies, 2024). Since 2022, these standards have been extended to all hospitals, irrespective of size (Czeschik, 2022).

The B3S mandates the protection of both core medical processes (e.g., diagnostics, therapy, discharge) and technical support systems (e.g., patient information systems, alarm infrastructure). Hospitals must implement an Information Security Management System (ISMS) aligned with International Organization for Standardization (ISO) 27001, maintain round-the-clock security readiness, perform risk assessments, and develop emergency response plans (ForeNova Technologies, 2024; Czeschik, 2022). These measures are reinforced by IT-SIG 2.0, a cybersecurity law introduced in 2021, which imposes strict breach notification requirements and penalties for non-compliance (ForeNova Technologies, 2024).

Technical safeguards emphasized in both KRITIS and GDPR-aligned frameworks include encryption, pseudonymisation, access control, audit logging, and intrusion detection systems. These are designed to uphold the confidentiality, integrity, and availability of patient data while supporting traceability in system operations (ForeNova Technologies, 2024). Hospitals are also encouraged to perform DPIAs for high-risk data processing activities, a requirement directly applicable to systems integrating real-time ICU data (Dougherty, 2025).

### Practical Challenges in Secure System Implementation

Implementing these regulatory and technical standards presents several challenges. Pseudonymisation and anonymisation, while essential for protecting patient identity, introduce trade-offs in data utility and traceability. For example, while anonymised data cannot be re-identified, it may lose clinical relevance in longitudinal analyses. Conversely, pseudonymised data retains linkability but demands secure key management to prevent unauthorized re-identification.

Another persistent challenge is ensuring data security across systems and jurisdictions, particularly when data are transmitted internationally. Cross-border data transfers must adhere to GDPR conditions, often necessitating Standard Contractual Clauses and adequacy assessments of third-country protections (Dougherty, 2025). In the ICU context, where interoperability with external systems may be required, these considerations become critical.

Furthermore, maintaining system auditability and breach responsiveness in high-complexity environments like critical care poses operational difficulties. Healthcare institutions must balance the urgency of clinical workflows with the legal imperative of complete traceability and prompt incident response.

### Application in This Thesis

This thesis formalizes the SBT checklist within a symbolic verification framework using Isabelle/HOL. While real-world integration with PDMS is a design consideration, the current implementation relies on mocked interfaces to simulate clinical data. Although patient identifiers are used within the prototype application to facilitate development and testing, efforts have been made to

limit data exposure to clinically relevant parameters and to maintain a clear separation between identifying information and formalized logic structures.

The GDPR's emphasis on data minimization and explicit consent is reflected in the focus on clinically necessary parameters for evaluating extubation readiness. While pseudonymisation is assumed in principle during the mocked data ingestion process, the prototype implementation currently retains patient identifiers for development purposes. This pragmatic choice supports functionality testing but also highlights the need for stricter data handling measures in future clinical deployments to fully align with the expectations of GDPR and B3S standards for secondary data processing.

Looking forward, the deployment of this system in real clinical environments would necessitate compliance with the full set of GDPR, B3S, and KRITIS requirements. This includes the formal documentation of processing activities, execution of DPIAs, implementation of TOMs, and conformance with hospital-specific ISMS protocols. These considerations are elaborated further in Chapter 4 and Chapter 6, particularly in relation to the architectural limitations and ethical implications of simulating real-world data exchange.

Data security standards in healthcare are shaped by intersecting legal, technical, and ethical imperatives. The GDPR provides a foundational regulatory framework for the protection of patient information, while German-specific extensions through KRITIS and B3S introduce sectoral obligations that address the heightened risks of clinical environments. These frameworks not only impose comprehensive responsibilities on institutions and developers but also delineate the ethical boundaries within which clinical systems must be designed, tested, and eventually deployed. Within the context of this thesis, such standards inform the permissible scope of data use and guide the simulation of clinical workflows in a manner that balances practical prototyping with future compliance expectations.

With this regulatory and infrastructural foundation established, the Chapter 3 turns to the methodological architecture underpinning the system developed in this work. It outlines the design strategies, formal modeling principles, and technical implementation choices that enable the safe and verifiable automation of protocol-based clinical reasoning.

# Chapter 3

# Methodology

This chapter presents the methodological foundation for the formalization and automation of protocol-based clinical reasoning as developed in this thesis. Building on the theoretical and clinical context outlined in the previous chapters, it delineates the design principles, system logic, and technical choices that underlie the proposed approach. The methodology is motivated by the need for transparent, reliable, and verifiable decision support systems in intensive care, particularly in the context of SBT readiness assessments.

The formalization of clinical checklists within a symbolic framework necessitates a clear and rigorous research strategy. The system described in this thesis is not merely a technical artifact but a structured representation of medical logic designed to support safety-critical decision-making. As such, the methodology integrates formal modeling, iterative design, and explainability-focused development to ensure both correctness and clinical relevance. These aims guide the system's structure, its integration with clinical data, and its automated verification processes.

The chapter begins by outlining the overarching research paradigm and development strategy employed in this work. This includes the rationale for adopting formal methods, the iterative modeling process, and the key goals of verifiability, automation, and explainability, which are discussed in Section 3.1.

Following this, the operational structure of the system is introduced. The high-level workflow, from checklist definition to formal proof, is broken down into logical modules, each responsible for a distinct transformation of data or logic. This conceptual architecture is presented in Section 3.2.

The methodology further rests on specific principles for translating clinical logic into formal representations. These principles include design choices related to modeling paradigms, rule validity, and the semantics of checklist conditions. These formalization strategies are addressed in Section 3.3.

Technical implementation decisions are equally essential to the method. The selection of Isabelle/HOL for verification, Java for system integration, and the use of batch-mode automation all reflect concrete requirements of the clinical and computational context. These decisions and their justifications are elaborated in Section 3.4.

Finally, the methodology is designed to support reuse and extensibility across various clinical checklists and use cases. This includes architectural strategies for modularity, robustness, and adaptability, which are detailed in Section 3.5.

Together, these sections establish a comprehensive methodological framework that enables the formalization, validation, and practical deployment of checklist-based clinical logic. The approach is grounded in formal reasoning but is shaped by real-world clinical constraints, ensuring both theoretical rigor and translational applicability.

## 3.1 Research Design and Development Approach

The methodological orientation of this thesis is grounded in the principles of constructive research and the design science paradigm. Rather than pursuing empirical generalization or hypothesis testing, the focus lies in the development and formalization of a domain-specific artifact: a symbolic verification system tailored to model ICU checklist logic. This approach reflects a research-through-prototyping methodology, where the primary objective is the realization and demonstration of a functional system that embodies clinical reasoning in a verifiable and explainable form.

#### Research Aim and Methodological Objectives

The central research aim is to establish the feasibility of employing symbolic reasoning for the formal verification of critical care decision protocols, specifically those governing the SBT safety screen. By doing so, the thesis seeks to contribute a methodological and technical foundation for future CDSSs that integrate symbolic logic as a core feature.

- **Correctness-by-construction**, by ensuring that all clinical decisions are derived from automatically verified logical models.
- **Elimination of interpretative ambiguity**, through unambiguous mappings from clinical checklist phrases to formalised higher-order predicates.
- **Traceability**, by maintaining a transparent and auditable trail from clinical data ingestion to formal proof outcomes.
- **Modularity and reusability**, to allow for the seamless extension of the system beyond the SBT safety screen, facilitating reuse across diverse clinical protocols.

These objectives guide both the conceptual framework and implementation strategy of the prototype system. Their realisation is critically assessed in Section 5.5, where the alignment between the implemented artifact and the broader methodological goals is evaluated.

#### Development Strategy and Domain Involvement

The development process was conducted iteratively, combining system modeling in Java with formal verification using Isabelle/HOL. This dual implementation strategy was informed by practical constraints and conceptual goals: Java offered the flexibility and interoperability required to model clinical logic and interface with simulated healthcare data, while Isabelle/HOL provided the formal rigor necessary to validate protocol correctness. System requirements were

shaped through a series of informal, domain-specific consultations with intensive care personnel at Hospital Bamberg. These exchanges, while unstructured, yielded critical insights into checklist semantics, clinical expectations, and interpretative ambiguities. No formal interviews or clinical trials were conducted. Instead, the design process was refined through ongoing prototyping and domain-expert feedback.

#### Validation Approach and Evaluation Logic

To evaluate the system, a series of internal validations were performed. These involved running the prototype on simulated patient data and inspecting the generated Isabelle theories for consistency, correctness, and adherence to clinical logic. Although no formal empirical testing was conducted, this manual review provided iterative feedback for improving the model architecture and rule encoding. In particular, correctness was assessed through logical soundness of the generated formal proofs, while clinical interpretability was verified by examining the traceability of each decision path within the formal representation.

#### Technical Toolchain: Isabelle/HOL and Java

The technical selection of Isabelle/HOL was motivated by its robust support for HOL, its maturity within safety-critical verification domains, and its demonstrated applicability in healthcare formalization efforts. Java was chosen to structure domain logic, manage data input and transformation, and support future integration with hospital information systems via standard communication protocols. Together, these tools facilitated a seamless translation from clinical checklist definitions to machine-verifiable logical models.

#### System Limitations and Current Status

The current status of the system is that of a proof-of-concept. While it successfully demonstrates the symbolic formalization and automated verification of ICU checklist logic, it remains constrained by several developmental limitations. The system operates exclusively on mocked data inputs, without integration into live PDMS. Its graphical user interface is minimal and has not undergone usability testing or clinician validation. Furthermore, the implementation does not yet address full data privacy compliance or standard interfacing requirements, both of which are discussed in Chapter 6. Consequently, while the prototype establishes conceptual feasibility, it is not yet suitable for clinical deployment.

#### Translational Outlook and Broader Relevance

Despite these limitations, the translational value of the system lies in its demonstration of explainable, logically grounded decision support for critical care applications. It provides a structured foundation for subsequent hybrid AI systems that combine symbolic transparency with data-driven adaptability. The design and implementation strategies adopted in this work align with evolving standards for ethical, interpretable, and technically robust clinical AI, as outlined in Chapter 2. The methodological insights gained here not only inform future enhancements to the prototype but also contribute to the broader discourse on trustworthy automation in healthcare.

With the overarching research strategy and development rationale in place, Section 3.2 details the system's internal structure by mapping its conceptual workflow and the logical architecture that transforms clinical input into formally verified outputs.

## 3.2 Conceptual Workflow and System Logic

The formalization and verification system developed in this thesis is founded on a modular, traceable, and explainable workflow that operationalizes clinical checklist logic, particularly the SBT readiness assessment, into machine-verifiable proofs using the Isabelle/HOL theorem prover.

Figure 3.1 illustrates the system's conceptual architecture, mapping the sequence from clinical input to formally verified output. This workflow decomposes the overall process into distinct, logically cohesive modules that collectively support correctness-by-construction as well as traceability across the entire verification lifecycle.

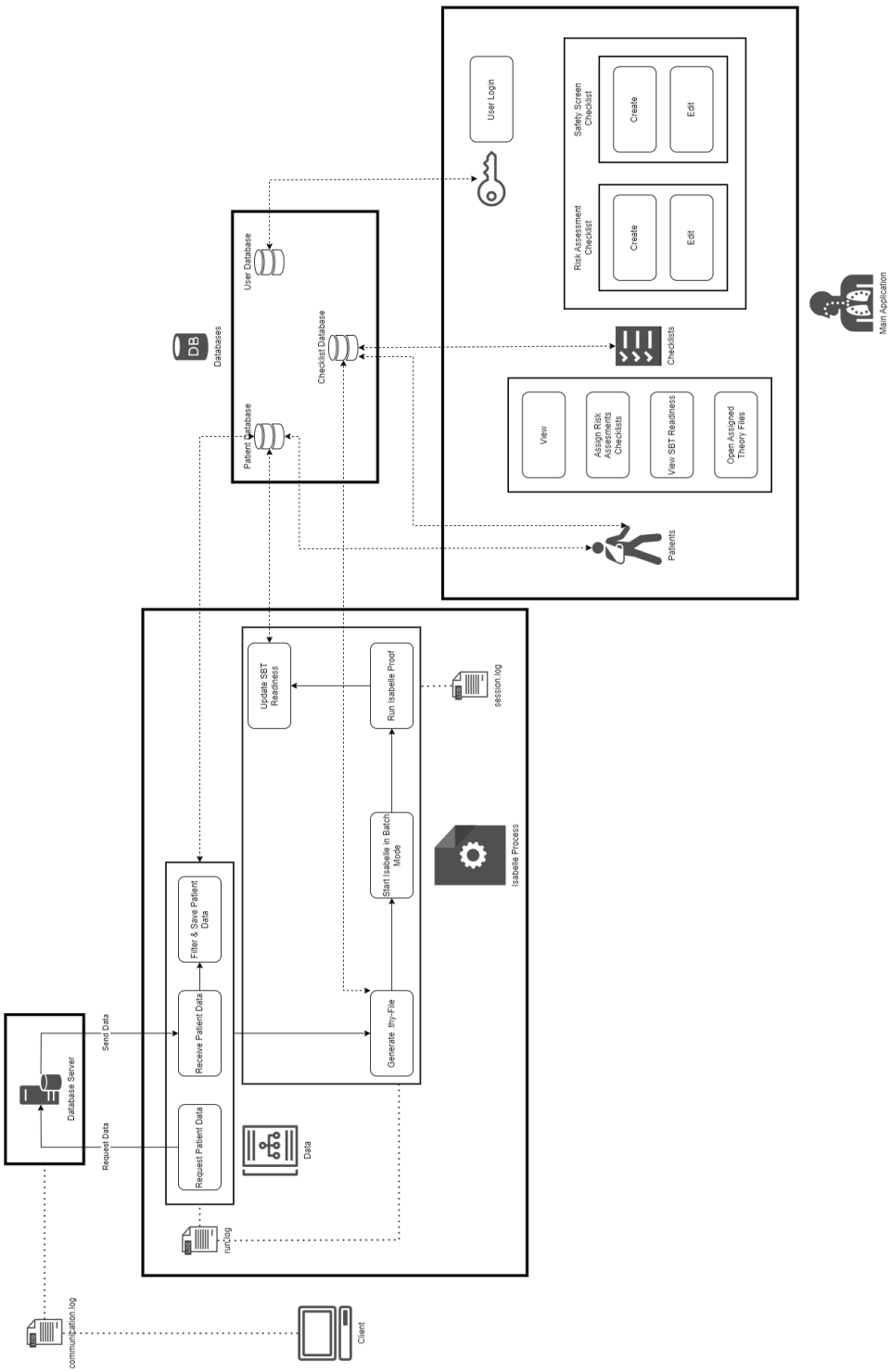


Figure 3.1: System architecture for formalizing and verifying SBT readiness using Isabelle/HOL

### Modular System Architecture

At the highest level, the system consists of four primary modules, each encapsulating a specific functional domain and interacting through well-defined interfaces:

1. **Client Module:**
   This module serves as the interactive and computational front end of the system. It comprises two key subcomponents:
   - **Data Module** – Responsible for requesting patient data from external sources, receiving and filtering incoming data streams, and persisting relevant patient information. This component ensures that only clinically pertinent parameters are retained for subsequent evaluation.
   - **Isabelle Process Module** – Manages the generation of Isabelle `.thy` theory files from the structured checklist logic and patient data. It subsequently initiates Isabelle in batch mode to execute automated proofs, and finally processes the verification outcomes to update SBT readiness assessments.
2. **Database Server Module:**
   Implements a mocked PDMS simulating real-world clinical data sources. It provides structured patient data through HL7-based communication, enabling development and testing in a controlled environment without dependence on live hospital systems.
3. **Databases Module:**
   Encompasses all persistent storage systems utilized by the application, including the Patient Database, Checklist Database, and User Database. It ensures durable, consistent, and secure storage of clinical checklists, patient records, user credentials, and other domain-specific data necessary for system operation.
4. **Main Application Module:**
   Serving as the core operational hub, this module integrates:
   - **Patients Module** – Allows users to view patient information, assign risk assessment checklists, monitor SBT readiness statuses, and access assigned Isabelle theory files for inspection.
   - **Checklists Module** – Enables the creation, editing, and management of Risk Assessment Checklists and the SBT Safety Screen.
   - **User and Session Management** – Provides secure and role-based access control to system features through user authentication and session management functionalities.

### Checklist-Based Workflow Execution

The system workflow commences with the interactive definition and configuration of checklists via a graphical user interface (GUI) by authorized users. The checklists consist of the comprehensive Risk Assessment Checklist, within which the SBT Safety Screen is an integral subset. These checklists are composed of

structured logical conditions that encode clinical decision rules derived from established medical protocols, such as those illustrated by the real-world example from Hospital Bamberg (see Figure 2.1).

Each checklist condition is specified as a combination of parameter thresholds, logical operators, and rule predicates.

Once defined, these checklists can be assigned to individual patients within the system, contextualizing evaluation to patient-specific data profiles. Throughout the day, patient data are continuously ingested from the mocked PDMS via HL7 message simulation. The Data Module filters this incoming data, extracting parameters relevant to checklist conditions and discarding extraneous information.

### Transformation Pipeline and Verification Stages

The core transformation pipeline integrates these modules to bridge the clinical and formal verification domains. It consists of four key stages:

- **Checklist Construction and Management:** Facilitated by the Checklists Module, this stage supports the definition of complex logical constructs, including simple and derived conditions, parameter thresholds, and Boolean operators, and manages checklist-to-patient assignments.

- **Patient Data Handling:** The Data Module interfaces with the mocked PDMS, retrieving and filtering patient information before storing it within the Patient Database for subsequent formal analysis.

- **Formalization and Code Generation:** This stage converts clinical checklist logic and patient parameters into Isabelle/HOL `.thy` theory files. Utilizing a domain-specific templating system, it defines formal predicates, typed constants, and reusable logical rules within a HOL framework. This translation preserves semantic alignment between clinical intent and formal verification requirements.

- **Automated Proof Execution and Result Integration:** The Isabelle Process Module executes the generated theory files in Isabelle's automated batch mode, performing rigorous proof checking. It then parses the output logs to determine compliance results, updating the patient's SBT readiness status accordingly.

This modular and layered architecture ensures that clinical semantics remain tightly coupled with their formal representations, thereby supporting correctness-by-construction. Simultaneously, traceability is preserved across all transformation steps through comprehensive logging mechanisms (`run.log`, `communication.log`, and `session.log`), which capture detailed records of data flow, proof execution, and verification outcomes. These logs enable auditing, debugging, and compliance monitoring.

### Foundational Design Principles

The system's design is guided by four foundational principles:

- **Explainability and Transparency:** Each transformation and verification step is explicitly documented and accessible, ensuring that clinical decisions can be traced back to their formal origins.
- **Modularity:** Logical separation of checklist construction, patient data handling, formalization, and verification facilitates maintainability and extensibility.
- **Reusability:** Clinical checklists and data templates are structured to allow reuse across diverse clinical scenarios and evolving guidelines.
- **Formal Abstraction Layers:** Clinical semantics are decoupled from technical implementation details, enabling consistent reasoning across heterogeneous clinical contexts.

Currently, the system operates within a controlled, simulated environment, employing mocked patient data and PDMS interfaces to enable robust testing and development. Nonetheless, the architecture is explicitly designed with future integration into live PDMS systems and real-world hospital infrastructure in mind.

To ensure that the system's verification process faithfully reflects clinical logic, the Section 3.3 defines the modeling principles and formal abstraction strategies used to translate checklist-based medical reasoning into a machine-verifiable format.

## 3.3 Principles for Translating Clinical Logic into Formal Models

The development of a formal model for checklist-based decision support requires the establishment of design principles that ensure clarity, modularity, and verifiability. This section outlines the criteria and rationale guiding the transformation of SBT safety screen logic into a machine-verifiable structure, using HOL as the formal substrate. The goal is not only to enable automated reasoning but to ensure that the formal model preserves the semantic intent and clinical integrity of the underlying checklist.

The formal Isabelle/HOL theory files produced by this system are not authored manually. Rather, they are automatically generated by a domain-specific code generator developed as part of the implementation. Although no specific theory file is analyzed in this section, the modeling principles discussed here directly inform the structure and semantics of the generated output. The generator itself, along with representative `.thy` files, is introduced in Chapter 4.

#### Dimensions of Clinical Correctness

In the clinical context, correctness is understood as more than syntactic validity within a formal system. It encompasses the accurate representation of medical rules, their traceable application to patient data, and the verifiable satisfaction of clinical safety criteria. The modeling process therefore emphasizes three dimensions of correctness: *logical consistency*, *traceability of individual rules*, and *alignment with clinically defined thresholds for readiness evaluation*.

Each clinical rule, such as assessing whether the $FiO_2$ is below a given percentage or whether the $PaO_2/FiO_2$ ratio exceeds a critical threshold, is translated into a formally defined condition. These conditions can then be evaluated independently or in combination, supporting both granular verification and integrated checklist assessments.

HOL provides the expressive power needed to model clinical decision-making, especially where conditions involve nested rules, parameterized thresholds, or combinations of criteria. Its support for typed structures and abstract functions enables the definition of clinical models that are both precise and adaptable. These features are particularly relevant for capturing real-world protocols, which often involve interdependent criteria and domain-specific thresholds.

**Layered Model Structure and Logical Predicates**

The formal model adopts a layered strategy. At the base, clinical data are represented in a structured format, grouping relevant parameters, such as oxygenation ratios or ventilator settings, into a coherent record. Thresholds are likewise encapsulated in a general structure that includes both the numerical value and the comparison operator required for evaluation.

Building upon this, each rule is expressed as a logical predicate that maps patient data to a Boolean outcome. For instance, a rule might verify whether a particular respiratory value satisfies the corresponding threshold condition. These predicates are defined modularly, supporting individual validation and systematic composition into more complex logic, such as multi-criteria checklist fulfillment.

By decoupling threshold values from their logical application, the model promotes flexibility and reuse. It also supports centralized updates to clinical parameters, ensuring that adjustments to medical guidelines can be reflected across the formal structure without structural redesign.

A key aspect of the design is the abstraction of threshold comparisons into a generalized mechanism. Instead of embedding fixed inequality relations into each rule, the system uses a uniform representation that pairs threshold values with abstract operators (e.g., less than, greater than or equal to). A central evaluation function interprets these comparisons, enabling consistent application of clinical rules across different parameters.

This generalization has two important consequences. First, it ensures that each rule can be expressed declaratively, in terms of data and logic, without conflating the two. Second, it introduces an explicit control structure for handling missing or invalid data, supporting realistic clinical scenarios where information may be incomplete or unavailable.

To enhance both verification and clinical transparency, each validation rule is defined as an independent logical unit. This modularity allows for the identification of specific conditions that fail during checklist evaluation, a critical requirement for traceable decision support. Composite predicates then combine these base rules into higher-level logic corresponding to full checklist conditions, such as SBT readiness.

This structured layering, from individual thresholds to composite predicates and full checklist rules, supports both proof decomposition and clinical auditability. Each layer can be tested, reasoned about, and extended without disturbing the broader model structure.

#### Hybrid Modeling: Balancing Simplicity and Formal Expressiveness

The decision to use a hybrid approach, combining structured data records with discrete logical predicates, reflects a balance between simplicity and formal expressiveness. Pure Boolean logic, though easy to implement, lacks the structural clarity and extensibility needed for realistic checklist modeling. Conversely, a purely record-based approach may introduce unnecessary complexity for basic validation tasks. By combining both perspectives, the model ensures that clinical logic remains accessible while also supporting rigorous formal analysis.

Having established the formal modeling principles that underpin the system's logic, the Section 3.4 outlines the rationale behind the selection and integration of tools, specifically Isabelle/HOL and Java, that enable the technical realization of the verification pipeline.

## 3.4 Tool and Integration Rationale

The system architecture developed in this thesis is built upon a deliberate selection of tools that align with the demands of formal correctness, traceability, and clinical integration. Isabelle/HOL was chosen as the formal verification engine due to its support for HOL and its proven efficacy in safety-critical domains. Java was employed as the implementation environment for checklist modeling and system orchestration. Batch-mode execution of verification tasks ensures repeatability and automation, aligning with the requirements of scalable, trustworthy decision support. This section outlines the rationale behind each of these decisions and integrates supporting evidence from relevant literature.

#### Isabelle/HOL for Formal Clinical Verification

Isabelle/HOL provides a rigorous logical framework suitable for modeling the complex interdependencies found in ICU protocols. As explained by Väänänen (2024), HOL extends second-order logic by allowing quantification over predicates and functions of arbitrary order, thereby enabling the expression of rich clinical dependencies. This expressiveness is essential for formalizing checklist rules involving derived parameters, logical thresholds, and structured data abstraction.

Blanchette et al. (2011) emphasize Isabelle's combination of human-readable syntax and automated reasoning tools. These include Sledgehammer, Quickcheck, and the simplifier, the latter of which plays a central role in the system developed here. The simplifier performs goal refinement through recursive rewriting, enabling structured verification of complex logical conditions and promoting traceability through step-wise evaluation.

Boldo et al. (2016) further highlight Isabelle's ability to combine interactive and automated proof construction, supporting correctness-by-construction in safety-critical domains. The system's integration of declarative and procedural proof styles, modular locales, and theory reuse makes it particularly appropriate for modeling checklists that are both reusable and adaptable across patient cases.

The use of Isabelle/HOL for verifying ICU-specific logic is further supported by Lloyd (2017), who demonstrates its applicability to rule-based clinical rea-

soning, and by Vivek and Martin Lloyd (2021), who argue for formal logic frameworks as necessary tools for embedding ethical transparency in automated medical systems.

A clinically grounded example is provided by Vestrucci et al. (2024), who apply Isabelle/HOL to formalize extubation readiness assessments in intensive care. Their study demonstrates how symbolic reasoning can support the encoding of ethical, clinical, and technical requirements into formally verifiable logic structures. Although their work does not explicitly invoke batch verification pipelines, it illustrates how structured formal models can automate extubation logic while maintaining traceability and alignment with ICU standards. This aligns directly with the methodology and system goals outlined in this thesis.

#### Java for System Integration and Code Generation

Java was selected as the primary implementation language due to its platform independence, object-oriented structure, and widespread use in clinical software environments. It provides an effective means to model patient data, parameter thresholds, and condition logic in modular and extensible ways. Moreover, Java supports file-based template generation, automated invocation of external tools, and integration with standardized protocols such as HL7, positioning the system for eventual deployment in healthcare IT environments.

The Java-based code generation module compiles clinical checklist logic into Isabelle-compatible `.thy` files. These files include formal predicates, typed constants, and parameterized rules. Once generated, they are passed into Isabelle for proof execution. This separation of modeling (Java) and verification (Isabelle) ensures system modularity, logical soundness, and technical scalability.

Java's ecosystem also supports logging, session management, and simulated data ingestion, all of which are used in the current implementation. While the system currently interfaces with a mocked PDMS, its architecture was designed to maintain compatibility with real-world HL7 v2 messages, thereby facilitating future clinical integration.

#### Automation through Batch-Mode Verification

A key goal of the system is to enable automation of the checklist verification process. In a clinical context, this supports repeatable, non-interactive, and unbiased evaluation of patient readiness based on formalized logic. Isabelle's batch-mode functionality, documented extensively in the Isabelle System Manual Wenzel (2024), supports this by enabling headless proof sessions that verify generated theory files and return structured log outputs.

The use of batch mode eliminates the need for manual user interaction and enables the system to process large volumes of checklist evaluations continuously. Logs such as `session.log` and `application.log` provide auditability, which is critical in healthcare settings for legal traceability and quality assurance.

#### Synthesis: A Toolchain for Ethical and Scalable CDSSs

Together, the tool and integration decisions constitute a coherent infrastructure tailored to the demands of clinical checklist verification. Isabelle/HOL contributes logical soundness, symbolic expressiveness, and modular formalization. Java provides structural flexibility, integration readiness, and an extensible

foundation for future development. Batch-mode execution ensures automation, reproducibility, and traceable verification. As a whole, this toolchain reflects not only technical adequacy but also normative alignment with explainable, ethically grounded AI in healthcare.

These components interact through a well-defined pipeline described in Section 3.2. Each stage, from patient data ingestion to theory generation and verification, preserves semantic consistency and allows modular expansion. This architecture supports the broader aim of transforming ICU checklists from informal, interpretive tools into formal, validated representations that can guide clinical decisions with precision and transparency.

With the foundational toolchain and integration strategy in place, the Section 3.5 examines how the system's architecture supports both automation and reusability, key requirements for extending the formal verification approach across evolving clinical protocols and diverse ICU workflows.

## 3.5 Automation and Reusability Considerations

The methodological framework developed in this thesis is intentionally designed to support not only the automation of formal checklist evaluation but also its reuse across diverse clinical contexts. This design imperative is rooted in the structural and procedural characteristics of critical care protocols, where clinical logic often follows repeated patterns of conditional assessment, parameter thresholding, and Boolean decision-making. By generalizing these patterns into formal abstractions, the system accommodates both the initial SBT Safety Screen and subsequent components of the full SBT Risk Assessment Checklist, as illustrated in the clinical reference material from Hospital Bamberg (see Figure 2.1).

As described in Section 3.2, the core architecture consists of modular components that separate checklist construction, data handling, logic formalization, and automated verification. This modularity serves as the primary mechanism for achieving reusability. Each component operates independently on abstract representations of clinical logic and patient data, enabling the system to be extended without modification to the underlying verification engine. For instance, while the current implementation focuses on the SBT Safety Screen (left side of the checklist), the same data structures and verification procedures apply to the SBT assessment (right side), since both are composed of discrete evaluative conditions or "questions" that conform to the same formal schema.

#### Automation Through Formal Verification Pipeline

Automation is achieved through the batch-mode operation of Isabelle/HOL, as described in Section 3.4. By generating theory files programmatically from structured checklist logic and patient data, the system avoids manual proof construction and ensures repeatable, unbiased evaluation. Each checklist condition is transformed into a verifiable predicate, and the collection of such predicates is composed into higher-order expressions that represent full checklist logic. This transformation pipeline is consistent regardless of the specific checklist phase, which further facilitates reuse and scaling. The formal abstraction of thresh-

olds and logical operators, as introduced in Section 3.3, enables the same core evaluation logic to apply across varied parameters and comparator types.

### Generalization and Abstraction for Extensibility

The generalization of threshold evaluation is a particularly significant feature in terms of extensibility. Clinical parameters, whether oxygenation indices or neurological assessments, are represented by typed values paired with abstract comparison operators. The logic governing their evaluation is centralized and uniformly interpreted, allowing new conditions to be added through configuration rather than code revision. This abstraction supports the progressive formalization of other clinical protocols that share similar logical semantics, including sedation scales, readiness-to-wean assessments, or infection risk evaluations.

In terms of explainability, each checklist condition remains independently traceable throughout the verification process. This is enabled by the structural separation between predicate definitions and their composition, as discussed in Section 3.3. Because the system preserves a one-to-one mapping between clinical conditions and formal predicates, clinicians and developers can identify which specific conditions fail during evaluation and examine the corresponding data and logical operations. This level of transparency supports both clinical auditability and regulatory alignment, two essential requirements for trustworthy deployment in healthcare environments.

### Compatibility with Real-World Clinical Data

Moreover, the system's current data ingestion model, though based on simulated HL7 inputs, has been structured to preserve compatibility with real-world PDMS interfaces. Patient data are retrieved, filtered, and transformed into structured records that correspond directly to formalized parameters. This consistent internal representation further enhances reusability, as it decouples the checklist logic from the data source, permitting the system to be adapted to various hospital environments and patient populations without altering its formal logic layer.

Finally, the use of Isabelle/HOL as the verification substrate reinforces the robustness and maintainability of the system. Its support for modular theory development, locale-based scoping, and higher-order abstractions ensures that new checklist types or decision criteria can be integrated without compromising the consistency or correctness of existing models. The simplifier mechanism, central to the proof automation strategy, enables efficient validation of expanded rule sets, making the system suitable for long-term evolution as clinical guidelines are updated.

In summary, the system presented in this thesis is not only an automated formalization of a specific ICU protocol but a reusable framework for symbolic reasoning across clinical checklists. Its modular architecture, abstract threshold logic, and explainability-focused design collectively support extensibility to broader decision pathways within intensive care.

Building on the methodological framework for automation and reusability, Chapter 4 translates these principles into concrete implementation by detailing the system architecture, module interactions, and execution pipeline that bring the formalized clinical logic into operational form.

# Chapter 4

# System Architecture and Implementation

This chapter presents the technical foundation upon which the prototype system for checklist-based clinical reasoning was built. Following the conceptual methodology established in the previous chapter, it now shifts focus to the realization of a functioning software pipeline that connects formal logic with clinical structures. The system developed here serves as a practical demonstration of how symbolic reasoning, patient data transformation, and automated verification can be orchestrated into a coherent, modular framework. Each subcomponent contributes to a pipeline that translates clinical intent into machine-verifiable statements, culminating in the execution of formal proofs that determine checklist compliance.

The system architecture reflects the core principles defined in the methodology: *modularity*, *traceability*, *reusability*, and *explainability*. Each module is designed to perform a specific role within the transformation pipeline, and the interfaces between components are structured to preserve logical integrity and semantic clarity. In contrast to isolated implementations of symbolic reasoning, this system emphasizes end-to-end integration, from clinical checklist design to formal verification output. This integration is essential for demonstrating not only conceptual feasibility but also operational cohesion across software layers.

Section 4.1 provides a high-level overview of the system architecture and its main components. It describes how the system is organized into discrete modules, and explains the responsibilities and interactions of each.

The functional sequence of data movement through the system is elaborated in Section 4.2, which traces the full runtime operation from checklist construction and patient data ingestion to the generation of formal theory files and their automated verification.

Section 4.3 focuses specifically on the integration of Isabelle/HOL, the theorem prover at the core of the verification process. This section outlines how theory files are generated and passed to Isabelle in batch mode, how proof results are captured, and how these outputs are returned to the user interface for interpretation.

The logic and implementation of the code generation layer are detailed in Section 4.4. This part explains the template system that transforms check-

list definitions and patient parameters into valid Isabelle code, highlighting its support for parameter injection, logical structures, and reusable predicates.

To conclude the chapter, Section 4.5 offers a comprehensive walkthrough of a single use case. It follows a concrete checklist instance as it progresses through the system, from UI definition to formal verification, and presents selected outputs to illustrate the functionality of each component in practice.

Together, these sections provide a complete account of the system's technical structure and operational logic. They demonstrate how symbolic AI can be instantiated in a real-world software architecture to support the automated and explainable evaluation of clinical decision rules.

## 4.1 System Overview and Component Architecture

The architectural foundation of the system is defined in Section 3.2, where the conceptual workflow is described as a sequence of logically distinct but interconnected modules. These include the *Client Module*, *Database Server Module*, *Databases Module*, and *Main Application Module*. Each module fulfills a dedicated role in the end-to-end transformation of clinical checklists into formally verified decision support outputs.

This chapter presents the technical realization of that conceptual architecture. First the technologies, libraries, and runtime environments used to implement the system are introduced. Then a structured overview of the Java application is provided, detailing how the conceptual modules defined in Section 3.2 are mapped to concrete package structures within the implementation. This mapping ensures that the software design maintains traceability to the methodological principles of modularity, explainability, and verifiability established in the preceding chapter.

### Technology Stack and Implementation Tools

The technical implementation of the presented system relies on a modern and robust set of software technologies and platforms. The choice of technologies is aimed at ensuring reliability, maintainability, and future extensibility. This section outlines the core technologies, programming environment, and supporting tools utilized in the development and execution of the application.

The application is developed entirely in Java, targeting version 23 of the Java language standard. Java 23, a recent release in the Java platform evolution, was selected to benefit from enhanced performance, language features, and long-term maintainability. The use of Java further allows the application to run cross-platform and integrate efficiently with both modern libraries and legacy hospital IT environments.

Project builds and dependency management are handled via Apache Maven, a widely used project management tool for Java applications. The Maven project configuration (`pom.xml`) specifies strict compatibility with Java 23, ensuring that all source and target code is compiled against this version. The application structure also incorporates standard Maven practices to organize source files, resources, and testing modules for optimal project clarity.

The user interface is implemented with JavaFX 23.0.2, leveraging both the javafx-controls and javafx-fxml modules. JavaFX is chosen for its advanced capabilities in building responsive and modern graphical interfaces, as well as its seamless integration with Java-based backend logic. The use of FX Extended Markup Language (FXML) facilitates clear separation between UI layout and application logic, supporting maintainable and modular design.

Jackson is employed for efficient and reliable serialization and deserialization of data, primarily using the jackson-databind and jackson-module-parameter-names modules version 2.18.2. Jackson enables the system to handle structured data interchange, configuration, and persistence operations in a flexible and type-safe manner.

The system employs an embedded, lightweight, and highly reliable relational database engine using Hypersonic 2 (H2) Database version 2.3.232. This enables local storage of patient records, checklists, and user data without the need for external database infrastructure. The application leverages Java Persistence API (JPA) version 2.2 for object-relational mapping, supporting transactional integrity and type-safe queries. Liquibase version 4.31.0 is integrated for automated schema management and database migrations. Liquibase profiles enable isolated management of the distinct databases for patients, checklists, and users, each configured with Advanced Encryption Standard (AES) encryption for at-rest data protection.

To enable interoperability with external clinical systems, particularly for patient data acquisition, the application integrates the HL7 Application Programming Interface (HAPI) library (hapi-base 2.5.1). The HAPI library provides tools for parsing, constructing, and validating HL7 messages, supporting the seamless integration of the system with hospital infrastructure.

Robust logging capabilities are achieved through Simple Logging Facade for Java (SLF4J) API version 2.0.16 combined with the Logback backend (logback-classic 1.5.17). This enables configurable and high-performance logging, providing detailed traceability and support for debugging during both development and production use.

Development and primary testing were performed on Microsoft Windows 11 Home (Build 26100), running on a 64-bit Advanced Micro Devices (AMD)-based system with 32 Gigabyte (GB) of Random Access Memory (RAM).

Automated theorem proving and formal verification tasks are carried out via Isabelle/HOL (Isabelle2024). While a newer version of Isabelle (Isabelle2025) have since been released, all integrations and verification components in the present work are implemented and validated against the 2024 release to ensure consistency and reproducibility.

The selection of these technologies reflects a deliberate effort to build a system that is reliable, secure, and well-suited for the clinical and technical requirements of intensive care data governance. The modular and open-source nature of the chosen libraries and tools ensures that the system can be extended and adapted as requirements evolve.

### Java Application Architecture and Package Mapping

The Java implementation of the system mirrors the modular structure introduced in Section 3.2. Each conceptual module is represented by a distinct set of packages, structured under the root namespace `com.example.sbt_safetyscreen_auto`. The top-level and second-level packages reflect the logical boundaries of the system's operational workflow:

- **Client Module:** Handles both data acquisition and formal verification. It is implemented across the `hl7application` and `isabelleapplication` packages. The `hl7application` package covers HL7-based message exchange, including message construction, parsing, and simulated server communication. The `isabelleapplication` package encapsulates the end-to-end verification pipeline, including code generation, theory execution, session file management, and cleanup routines.

- **Database Server Module:** Realized through the combined use of the `hl7application.mockserver` and `servermanager` packages. These packages jointly simulate the behavior of a PDMS, provide synthetic patient data, and manage server lifecycles for integration testing.

- **Databases Module:** Implemented via the `database` and `model` packages. The `database` package includes subpackages for encrypted repository access, schema migration and initialization, and direct data access logic. The `model` package contains the domain abstractions for checklists and patient-related entities, providing a semantically rich and structured representation of clinical data.

- **Main Application Module:** Corresponds to the `mainapplication` and `initializer` packages. The `mainapplication` package manages application logic such as user interface controllers, validation routines, checklist text generation, and navigation services. The `initializer` package governs startup orchestration, including configuration loading, database setup, core service coordination, and UI bootstrapping.

This alignment between conceptual modules and software components ensures that the system remains consistent in both design and execution. Each package is dedicated to a well-defined architectural responsibility, and the modular organization facilitates independent testing, extensibility, and clinical traceability. The remainder of this section presents the full package structure and descriptions, structured according to this mapping.

Below (see Figure 4.1) is an overview of the primary packages in the codebase, shown with their first and second-level subpackages to illustrate the system's modular organization:

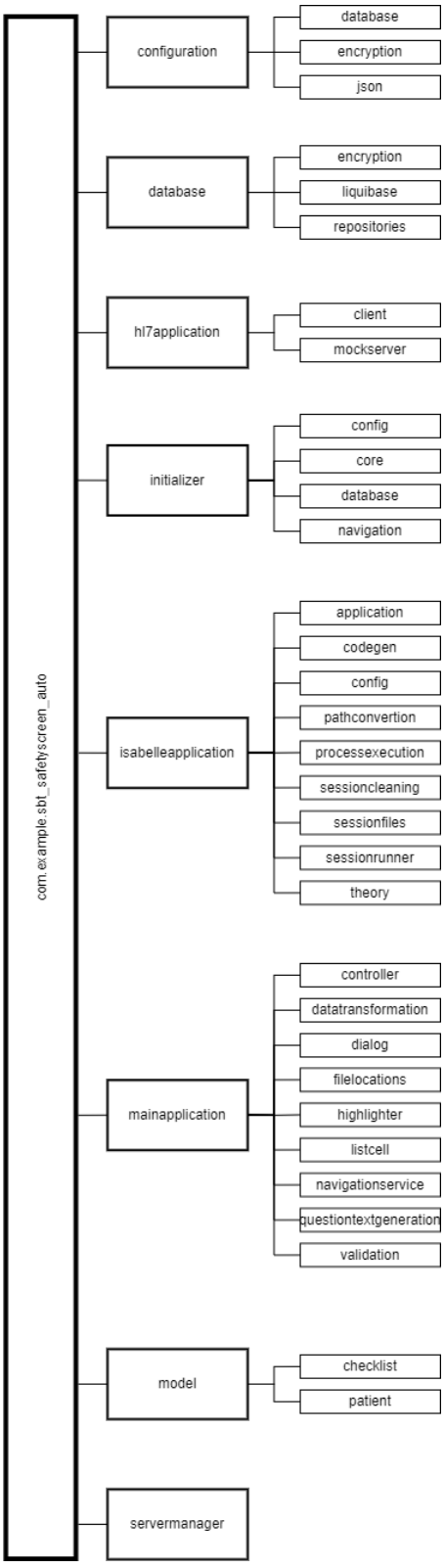


Figure 4.1: Internal package structure of the developed application

### Configuration and Application Setup

The `configuration` package (see Table 4.1) provides the foundation for all configuration management across the application. It centralizes access to global properties and settings using dedicated classes such as `AppConfiguration` and utility loaders like `PropertyLoader`. This structure ensures consistent configuration loading and access throughout the system.

The `configuration.database` subpackage focuses on database-specific configuration. It defines how database properties are loaded and managed and provides factories for generating data source objects. This clear separation helps maintain a robust and flexible approach to database integration, making it easier to adjust or replace database backends.

Within `configuration.encryption`, all encryption-related settings are managed. This includes not only holding encryption parameters but also implementing loaders for secure retrieval of encryption configuration from external sources. This setup helps ensure that sensitive data remains protected and that encryption practices are configurable and adaptable.

The `configuration.json` subpackage encapsulates settings for JSON processing, including the creation and management of object mappers needed for serialization and deserialization. By centralizing JSON configuration, the system guarantees consistent data interchange formats and makes future adjustments to JSON handling straightforward.

| Package | Short Description |
|---|---|
| configuration | Central configuration logic for the whole application. |
| configuration.database | Database connection and configuration management. |
| configuration.encryption | Handles encryption settings and configuration loading. |
| configuration.json | Configures JSON serialization/deserialization. |

Table 4.1: `configuration` package and subpackages

### Database Access and Persistence

The `database` package (see Table 4.2) serves as the central layer for all database-related operations within the application. It contains core service classes such as `UserService`, `PatientService`, and `ChecklistService`, which provide high-level data management, business logic, and application-specific data workflows. Utility components like `H2DatabaseShutdown` ensure reliable and clean shutdowns of embedded databases.

The `database.encryption` subpackage is responsible for ensuring that all sensitive data managed by the application is protected through encryption. It provides a robust AES-based encryption service (`AesEncryptionService`), supports secure initialization vector generation (`DefaultIVGenerator`), and handles base64 encoding/decoding. Specialized encrypted repositories (e.g., `EncryptedUserRepository`, `EncryptedPatientRepository`, `EncryptedCheck-`

listRepository) transparently encrypt and decrypt records, ensuring compliance with security and privacy requirements.

The `database.liquibase` subpackage provides mechanisms for managing and migrating the database schema using Liquibase. This includes automatic application of schema updates through LiquibaseSchemaUpdater, as well as factories for Liquibase and database objects, allowing seamless upgrades and environment setup. This approach ensures the application's schema stays consistent, versioned, and easy to evolve across different deployment scenarios.

Finally, the `database.repositories` subpackage implements the repository pattern for all persistent entities in the system. Java Database Connectivity (JDBC) provides the underlying API for executing Structured Query Language (SQL) statements and managing database connections in Java. epositories such as JdbcUserRepository, JdbcPatientRepository, and JdbcChecklistRepository handle direct database interactions for users, patients, and checklists respectively. The subpackage also includes tools like ChecklistParameterExtractor for extracting metadata or supporting complex queries. This separation of repository logic enables clean, testable, and easily extendable access to the underlying database for each main domain object.

| **Package** | **Short Description** |
|---|---|
| database | Core services and utilities for data access and application persistence. |
| database.encryption | Encryption logic for secure data storage and retrieval. |
| database.liquibase | Database schema migration and versioning management using Liquibase. |
| database.repositories | Repository pattern for all persistent entities. |

Table 4.2: `database` package and subpackages

### HL7 Communication and Mock Server Infrastructure

The `hl7application` package (see Table 4.3) is responsible for all HL7-based client-server communication in the application, providing a modular foundation for health data exchange. It coordinates startup and management of both the HL7 client and a mock HL7 server for test scenarios.

The `hl7application.client` subpackage implements all client-side HL7 logic, including configuration, message building, parsing of incoming and outgoing messages, and patient data processing. Key classes such as HL7Client, HL7ClientProcessor, and various builder and parser classes enable the client to communicate with an HL7 server, construct and interpret messages, and update patient records accordingly. The subpackage is highly modular, allowing easy extension for different HL7 message types and workflows.

The `hl7application.mockserver` subpackage provides a fully functional mock HL7 server for testing. This server listens for incoming HL7 messages, processes them with configurable response builders, and simulates the data flows expected in real hospital environments. Key classes such as HL7ServerApplica-

tion and HL7ConnectionHandler handle network communication and protocol framing, while classes like MultiPatientDataResponseBuilder and MockPatientRepository generate and supply synthetic patient data.

| Package | Short Description |
|---|---|
| hl7application | Main entry point for HL7 communication, coordination, and mock server setup. |
| hl7application.client | Handles HL7 client messaging, message building, parsing, and processing logic. |
| hl7application.mockserver | Provides a mock HL7 server for test data exchange. |

Table 4.3: `hl7application` package and subpackages

### Application Initialization Workflow

The `initializer` package (see Table 4.4) acts as the top-level entry point for the application's startup and setup logic. It manages the initialization of the UI, domain logic, encryption service, and the primary application configuration. Core classes such as UIInitializer, DomainInitializer, EncryptionInitializer, and AppInitializer collectively ensure that all necessary components are correctly constructed and wired together before the application becomes operational.

The `initializer.config` subpackage handles the loading and aggregation of application settings, such as database and encryption configurations. ConfigInitializer reads these settings from external sources (e.g., properties files), while ConfigContext acts as a central data holder for these configuration objects. This ensures a unified, type-safe way to access application settings throughout the lifecycle.

The `initializer.core` subpackage brings together the configuration, database setup, encryption logic, and domain service initialization. CoreInitializer orchestrates the entire initialization sequence, producing a CoreContext that contains all core subsystems ready for use. This structure centralizes critical startup processes and reduces coupling between major subsystems.

The `initializer.database` subpackage is responsible for establishing and configuring the database connections for all application domains, including schema updates using Liquibase. DataBaseInitializer creates and configures data sources, while DataBaseContext holds references to all initialized database connections for easy access by the rest of the application.

The `initializer.navigation` subpackage configures the application's navigation and controller logic, especially for JavaFX user interface flows. NavigationInitializer creates controller factories, sets up condition builder components, and ties them to the UI. NavigationContext holds the navigation service and controller factory, making these elements available for UI initialization.

| Package | Short Description |
|---|---|
| initializer | Entry point for overall app setup, UI, domain, and encryption. |
| initializer.config | Loads and provides database and encryption configuration. |
| initializer.core | Coordinates core app components (config, database, encryption, domain). |
| initializer.database | Initializes database connections and schema. |
| initializer.navigation | Sets up application navigation and controller logic. |

Table 4.4: `initializer` package and subpackages

### Isabelle/HOL Integration and Verification Pipeline

The `isabelleapplication` package (see Table 4.5) is responsible for all aspects of integrating Isabelle/HOL theorem proving with the Java application. It acts as a bridge, allowing clinical checklist logic and patient data to be automatically formalized, verified, and processed through Isabelle sessions. Core entry points include `IsabelleService` and its associated factory, which bundle together all necessary services for theory file generation and proof automation.

The `isabelleapplication.application` subpackage contains the central logic that orchestrates the execution flow within Isabelle from the application's perspective. Here, `IsabelleApplication` encapsulates the main operations that connect Java-based logic with formal verification routines.

The `isabelleapplication.codegen` subpackage is dedicated to generating Isabelle/HOL theory files from Java data structures. It provides extensive support for composing theory files, building up start and end blocks, defining condition logic, thresholds, patient sections, and formatting theory content according to reusable templates. The core generator is `ThyRenderer`, supported by many helper and section-specific generator classes.

The `isabelleapplication.config` subpackage manages the loading, validation, and storage of Isabelle-specific configuration settings, such as script paths and session directories. These are encapsulated in classes like `IsabelleConfig` and are loaded via robust configuration loaders.

The `isabelleapplication.pathconvertion` subpackage provides services to convert Windows file paths into Unix-like formats suitable for Cygwin or Windows-Subsystem Linux (WSL), which is necessary for running Isabelle/HOL in cross-platform environments. Cygwin is a compatibility layer that allows Unix-based applications to run on Windows by providing a Portable Operating System Interface (POSIX) environment. POSIX is a family of standards that define compatibility between Unix-like operating systems at the system interface level.

The `processexecution` subpackage abstracts the execution of external processes, including process creation, command-line building, output reading, and result handling. It includes services for executing shell or batch commands and for streaming process outputs into the Java environment.

The `isabelleapplication.sessioncleaning` subpackage ensures the secure deletion and cleanup of session files and directories after proofs are completed. It implements secure wiping mechanisms for sensitive data and automates the removal of temporary directories.

The `isabelleapplication.sessionfiles` subpackage is tasked with all file operations related to Isabelle sessions, such as creating directories, writing theory files, and handling session-specific metadata like `ROOT` files. It ensures the correct organization and storage of files required by Isabelle's build system.

The `isabelleapplication.sessionrunner` subpackage coordinates the preparation and execution of Isabelle sessions. It builds commands, invokes proof sessions, collects results, and connects these processes with session cleaning and file management. It also maps proof results back to the patient data domain.

The `isabelleapplication.theory` subpackage encapsulates the logic for managing theory files, including their creation, naming, and lifecycle. It provides services to generate theories for specific patients and risk assessments, enabling automated, patient-specific formal verification.

| **Package** | **Short Description** |
|---|---|
| isabelleapplication | Integrates Isabelle/HOL with the Java application. |
| isabelleapplication.application | Main application logic for Isabelle integration. |
| isabelleapplication.codegen | Generates Isabelle/HOL theory files and templates. |
| isabelleapplication.config | Loads and holds configuration for Isabelle execution. |
| isabelleapplication.pathconvertion | Converts file paths for compatibility (e.g., Cygwin, WSL). |
| isabelleapplication.processexecution | Executes external processes and handles results/streams. |
| isabelleapplication.sessioncleaning | Cleans up and securely wipes session files/directories. |
| isabelleapplication.sessionfiles | Manages file I/O for Isabelle sessions (theories, ROOT, etc.). |
| isabelleapplication.sessionrunner | Orchestrates session execution and command building for Isabelle. |
| isabelleapplication.theory | Encapsulates theory files and provides theory-related services. |

Table 4.5: `isabelleapplication` package and subpackages

### JavaFX Frontend and User Interaction

The `mainapplication` package (see Table 4.6) contains the main application entry point (`MainApplication`) and is responsible for initializing, launching, and gracefully shutting down the JavaFX application. It orchestrates the high-

level setup, such as loading configurations and initializing core components. By centralizing these functions, it ensures a robust and controlled lifecycle for the entire client application.

The `mainapplication.controller` subpackage holds all JavaFX controllers and their provider classes, managing user interfaces and handling user interactions. Each controller is dedicated to a specific view or workflow, such as creating checklists, managing patients, handling authentication, or responding to menu actions. The use of providers and a central `ControllerFactory` enables flexible dependency injection and modular UI design, promoting testability and maintainability.

The `mainapplication.datatransformation` subpackage is focused on serialization and deserialization, particularly for converting Java model objects to and from JSON. It includes implementations using the Jackson library, allowing seamless persistence and retrieval of application data. By abstracting (de)serialization logic, this subpackage ensures that data can be efficiently exchanged, saved, and restored across application components.

The `mainapplication.dialog` subpackage contains all logic related to user dialogs, notifications, warnings, and modal prompts. It standardizes dialog presentation across the application, ensuring consistent user feedback and interaction styles. By abstracting dialog behavior into services, it simplifies UI logic in controllers and supports custom styling and button actions.

The `mainapplication.filelocations` subpackage acts as a central registry for all path constants used throughout the application, including FXML views and configuration property files. By keeping these references in one place, it facilitates easy updates and reduces the risk of broken links due to path changes. This improves maintainability, readability, and ensures consistency in resource loading.

The `mainapplication.highligther` subpackage focuses on form validation and user feedback through error highlighting. Utilities in this package manage the visual marking of erroneous fields and clearing of error states, using both field-level and form-wide mechanisms. This enhances the usability and safety of data entry, making validation robust and user-friendly.

The `mainapplication.listcell` subpackage provides reusable JavaFX list cell logic for displaying complex items like patients and questions in lists. It leverages a builder and delegate pattern to ensure cells are modular, easily customizable, and consistent across different lists in the application. This approach promotes code reuse and uniformity in list-based UI elements.

The `mainapplication.navigationservice` subpackage abstracts navigation tasks, including opening modal dialogs and managing scene transitions. It centralizes how different parts of the UI interact and move between screens, providing a decoupled, testable way to manage navigation. Such an architecture simplifies controller logic and ensures a uniform user experience for modal and non-modal flows.

The `mainapplication.questiontextgeneration` subpackage handles the transformation of formal logical conditions into human-readable text for display in the UI. It typically uses the visitor pattern to traverse logical condition trees and generate text on demand. This supports explainability and transparency by allowing users to see and verify the logic behind checklist items.

The `mainapplication.validation` subpackage contains logic for checking the correctness of user inputs and business logic, providing structured error

reporting. Validators enforce constraints for single and combined conditions and other domain entities, ensuring only valid data enters the system. By structuring validation and errors here, the application maintains data integrity and provides clear, actionable feedback to users.

| **Package** | **Short Description** |
|---|---|
| mainapplication | Main entry point and application orchestration. Initializes, launches, and shuts down the app. |
| mainapplication.controller | JavaFX controllers for UI logic and user interaction, including their providers. |
| mainapplication.datatransformation | JSON (de)serialization utilities for converting model objects. |
| mainapplication.dialog | Dialog and prompt services for user feedback and modal interaction. |
| mainapplication.filelocations | Central location for all FXML and property file path constants. |
| mainapplication.highlighter | Error field highlighting and validation feedback for UI forms. |
| mainapplication.listcell | Reusable JavaFX list cell renderers for lists of patients and questions. |
| mainapplication.navigationservice | Central navigation and modal dialog management across the application. |
| mainapplication.questiontextgeneration | Generates readable text from logical condition objects for display. |
| mainapplication.validation | Validation logic and error data structures for form and business rule checking. |

Table 4.6: `mainapplication` package and subpackages

### Domain Model: Patients and Clinical Checklists

The `model` package (see Table 4.7) is the root for all domain-related types in the application. It provides foundational abstractions for both clinical and patient-centric data, ensuring that all core concepts are centrally organized and easy to reference throughout the system. This package acts as the primary source of truth for application state, supporting clear separation from infrastructure and presentation concerns.

The `model.checklist` subpackage contains all structures and domain types related to clinical checklists. It defines both high-level checklists (such as RiskAssessmentChecklist, and SafetyScreen) and the internal structure of these checklists, including safety screen components, questions, and logical conditions.

This package enables the encoding of clinical protocols in a structured, formalized, and extensible way, facilitating their automatic processing and verification.

The `model.patient` subpackage represents all patient-related domain entities required by the application. It models patients, their associated health parameters, and collections of these parameters over time (HealthParameterBatch). By maintaining a clear separation between patient data and checklist logic, this package allows for precise linkage of real-world patient states to clinical processes, as well as efficient integration with analytics and verification routines.

| Package | Short Description |
|---|---|
| model | Root domain models for clinical data and patients. |
| model.checklist | Clinical checklist structures, logic, and related types. |
| model.patient | Patient data models: patient, health parameters, and parameter batches. |

Table 4.7: `model` package and subpackages

### Server Lifecycle Management

The `servermanager` package (see Table4.8) is responsible for orchestrating the lifecycle of all server-side infrastructure components within the application. It provides abstractions and concrete classes for starting, stopping, and coordinating both the H2 database Transmission Control Protocol (TCP) server and a mock HL7 server, enabling robust integration testing and modular deployment. The ServerManager class serves as the main entry point, ensuring all configured servers are correctly initialized and gracefully shut down as a coordinated group.

Within this package, H2ServerComponent encapsulates the setup and management of the embedded H2 database server, allowing the application to expose a database endpoint for external connections or integration. Similarly, the HL7ServerComponent wraps the mock HL7 server logic, supporting the emulation of HL7-compliant communication for system testing without the need for a real hospital system backend. ServerLifecycleManager brings these components together, managing their startup sequence and clean shutdown.

| Package | Short Description |
|---|---|
| servermanager | Manages startup, shutdown, and orchestration of backend server components. |

Table 4.8: `servermanager` package

While the previous section detailed the modular and technological composition of the system, the subsequent Section 4.2 shifts focus to the dynamic

runtime processes that govern data flow, checklist execution, and verification, thereby illustrating how these components operate together during clinical decision support.

## 4.2 Data Flow and System Operation

This section traces the complete runtime pathway that links clinical artefacts, simulated patient data, and automated theorem proving. Whereas Section 4.1 delineated the static module structure, the present discussion follows the dynamic sequence that begins when a clinician defines a checklist and culminates in an updated SBT readiness flag for each patient. The description proceeds in seven subsections that correspond to the phases in Figure 4.2.

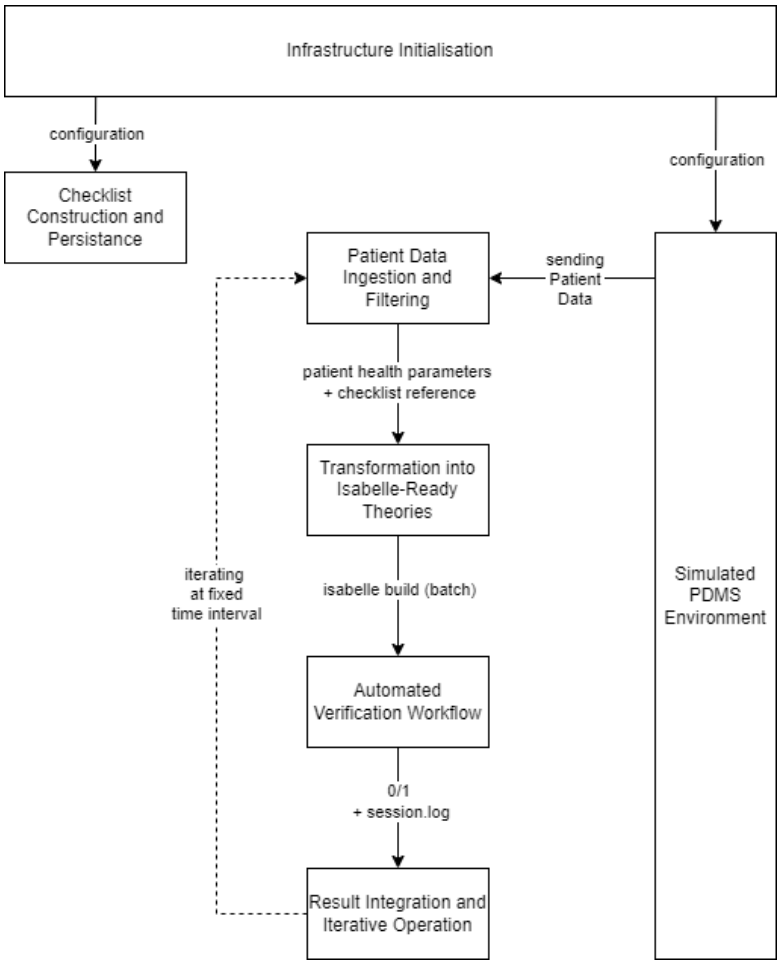


Figure 4.2: Runtime data flow

### Infrastructure Initialisation

At application launch the SERVERMANAGER class reads the Isabelle configuration file, sets a session-wide logging context, and instantiates two server compo-

nents: an embedded H2 database (H2ServerComponent) and a mocked HL7 v2 server (HL7ServerComponent). Both components remain resident for the entire programme lifecycle, thereby eliminating start-up latency for each verification cycle. The class hands control to a ServerLifecycleManager, which blocks on standard input and orchestrates graceful shutdown. This bootstrap sequence guarantees that database encryption keys and HL7 port bindings are available before the client attempts to connect.

### Checklist Construction and Persistence

Authorised users create Safety Screen and Risk Assessment checklists through the graphical interface. The checklist is serialised to JSON and passed as a string to an AesEncryptionService. The service applies an AES cipher with a configurable key length, transformation string, and randomly generated initialisation vector. The encrypted payload is stored within the Checklist Database, ensuring confidentiality at rest and satisfying the principle of data minimization mandated by the GDPR. The same process applies when a Risk Assessment Checklist is designated as the default checklist, an essential prerequisite for headless verification.

The concise excerpts (see Figure 4.3 and Figure 4.4) illustrate the persisted artefacts (without AES encryption).

```
{
  "name" : "checklist_name",
  "safetyScreenQuestions" : [ {
    "questionText" : "question_text",
    "conditions" : [ {
      "conditionType" : "condition_type",
      "parameter" : {
        "parameterType" : "parameter_type",
        "parameterName" : "parameter_name",
        "parameterId" : "parameter_id"
      },
      "threshold" : {
        "type" : "threshold_type",
        "value" : "threshold_value",
        "metric" : {
          "unit" : "metric_unit"
        }
      },
      "comparisonOperator" : "comparison_operator"
    } ]
  } ]
}
```

Figure 4.3: Stored Safety Screen Checklist Template

```
{
  "safetyScreen" : { ... },
  "isDefault" : "true/false"
}
```

Figure 4.4: Stored Risk Assessment Checklist Template

### Patient Data Ingestion and Filtering

The Client Module initiates a socket connection to the mocked PDMS at fixed intervals through `ServiceStarter`. An `HL7ClientProcessor` composes a HL7 Query for Patient Demographics (QRY^A19), including a Query Definition Segment (QRD) to specify the query parameters, and transmits it via an `HL7Client`. The server responds with a HL7 General Acknowledgement Response (RSP^K21), which is parsed into Java objects using the HAPI library. This enables automated and cyclic polling of structured patient data.

Within the processor, each incoming patient is assigned either the existing checklist reference or the default one. A parameter filter retains only those observation identifiers that are required by the referenced checklist. The filtered parameters and the original HL7 timestamp are written to the Patient Database under AES encryption. This step realises syntactic and semantic validation: syntactic correctness is ensured by HAPI parsing, whereas semantic completeness is enforced by comparing received parameter identifiers with the checklist-specific whitelist.

The concise excerpt (see Figure 4.5) illustrates the persisted artefacts (without AES encryption):

```
{
  "patientId" : "patient_id",
  "healthParameterBatches" : [ {
    "timestamp" : "yyyyMMdd-HHmmss",
    "parameters" : [ {
      "id" : "param_id",
      "name" : "param_name",
      "value" : numeric_value,
      "unit" : "unit
    } ],
    "associatedTheory" : {
      "name" : "theory_file_name",
      "content" : "theory content"
    },
    "sbtReady" : "READY/NOT_READY/NOT_AVAILABLE" (describes whether a patient in the
HealthParameterBatch under consideration is ready for SBT)
  } ],
  "checklistReference" : "reference_risk_assessment_checklist",
  "sbtReady" : "READY/NOT_READY/NOT_AVAILABLE" (describes the current SBT status of the
patient)
}
```

Figure 4.5: Stored Patient Template

### Transformation into Isabelle-Ready Theories

`IsabelleApplication` retrieves all patients from the database and forwards them to `PatientTheoryFileProcessingService`. For every patient the service loads the referenced Risk Assessment Checklist, invokes `TheoryFileGenerator`, and produces a distinct Isabelle theory file whose name combines the patient identifier and checklist. The fine-grained template that governs imports, record declarations, predicate definitions, local lemmas, and proof scripts is presented in Section 4.4 and Section 4.5.

File-system utilities place the theory inside a unique session directory that also contains a minimal `ROOT` file, allowing batch compilation without interfering with other patients.

### Automated Verification Workflow

The SessionProcessingService iterates over all generated theory files and delegates their execution to the TheorySessionRunner. This runner first converts the respective Windows paths into Cygwin-compatible format. It then assembles and executes the batch command as described in Section 4.3. Throughout the build process, all output is continuously streamed into the corresponding `session.log` file. Upon completion, the runner classifies the exit code according to the mapping provided in Table 4.9 of Section 4.3. Based on this classification, the appropriate readiness flag is written back to the associated patient record, thereby making the verification result accessible within the GUI.

### Result Integration and Iterative Operation

ServiceStarter schedules the entire sequence, from data polling and theory generation to proof execution on a fixed-delay executor. After each cycle the scheduler waits for a certain time interval, maintaining continuous operation throughout the clinical day. Because the server and client processes are long-lived, session directories accumulate only transiently. SessionCleaner removes all files except `session.log`, thereby reducing disk usage and aiding GDPR-compliant retention policies.

### Simulated PDMS Environment

The mock PDMS emits HL7 v2 observation messages with realistic temporal spacing and parameter distributions. Each message contains a single Patient Identification Segment (PID) and multiple Observation Segments (OBXs) (see Figure 4.6):

```
MSH|^~\&|MockServer|Receiver||CLIENT|20250518095743||RSP^K21|1|P|2.3
PID|||1001||
OBX|a10|NM|pao2||89.31|mmHg||F
OBX|b10|NM|fio2||0.69|||F
...
```

Figure 4.6: HL7 Message Example

The mock server supplies deterministic yet clinically plausible values, which facilitates repeatable evaluation while avoiding personal data disclosure. Section 6.1 discusses the implications of this simulation strategy for clinical integration.

This runtime narrative demonstrates how the modular architecture introduced in Section 4.1 realises a closed verification loop that begins with encrypted checklist definitions, passes through HL7-based data acquisition, and concludes with formally verified readiness assessments, all without manual intervention. The following Section 4.3 delves into the programmatic interaction between the Java layer and Isabelle, detailing session management, command construction, and error-handling strategies.

## 4.3 Isabelle/HOL Integration and Verification Pipeline

A principal objective of this thesis is the automated formal verification of clinical checklist logic with Isabelle /HOL. This section provides a detailed, step-by-step account of the technical integration between the Java application and Isabelle's proof engine. It follows the complete programmatic path from theory file generation to session execution and result propagation, while emphasising the design choices that guarantee reliable, unattended verification in batch mode.

### Batch-Mode Execution Strategy

Isabelle is executed exclusively through its batch-processing interface, which is designed for large-scale, non-interactive verification (Teege, 2024; Wenzel, 2024). The Java application launches the prover in a fixed time interval by means of a scheduled executor in ServiceStarter. Each cycle performs the following operations: First, a timestamped directory `run-<yyyyMMdd-HHmmss>` is created. This directory contains one subfolder per patient theory, for example Patient1002SafetyScreenss1. The structure guarantees file isolation, prevents name collisions, and permits future parallelisation. Second, TheoryFileGenerator renders checklist logic, patient data, derived thresholds, and local lemmas into a `.thy` file. SessionFileManager then places a minimal `ROOT` file into the session directory so that Isabelle's build tool can discover the new session. The precise internal arrangement of imports, record declarations, predicates, lemmas, and proof scripts is determined by the template system that Section 4.4 presents in detail. Third, IsabelleCommandBuilder assembles the command

```
isabelle build -v -o quick_and_dirty=false -o show_sorts=true
-D <sessionDir>
```

and passes it to ProcessExecutor, which in turn delegates to a Java ProcessBuilder. An explanation of the meaning of the parameters can be found in Section 4.5. PathConversionService translates Windows paths to Cygwin format, so the pipeline executes consistently across platforms. Verbose mode captures all proof messages for later inspection. Fourth, while Isabelle can build many sessions in parallel and reuse ancestor heap images, the current one-session-per-patient configuration runs sequentially. Nonetheless, the directory layout remains compatible with later parallel execution.

The interactions among the scheduler thread, HL7 client, Isabelle application, and external isabelle build process are summarized in Figure 4.7, which presents a high-level sequence diagram of the four-step cycle.

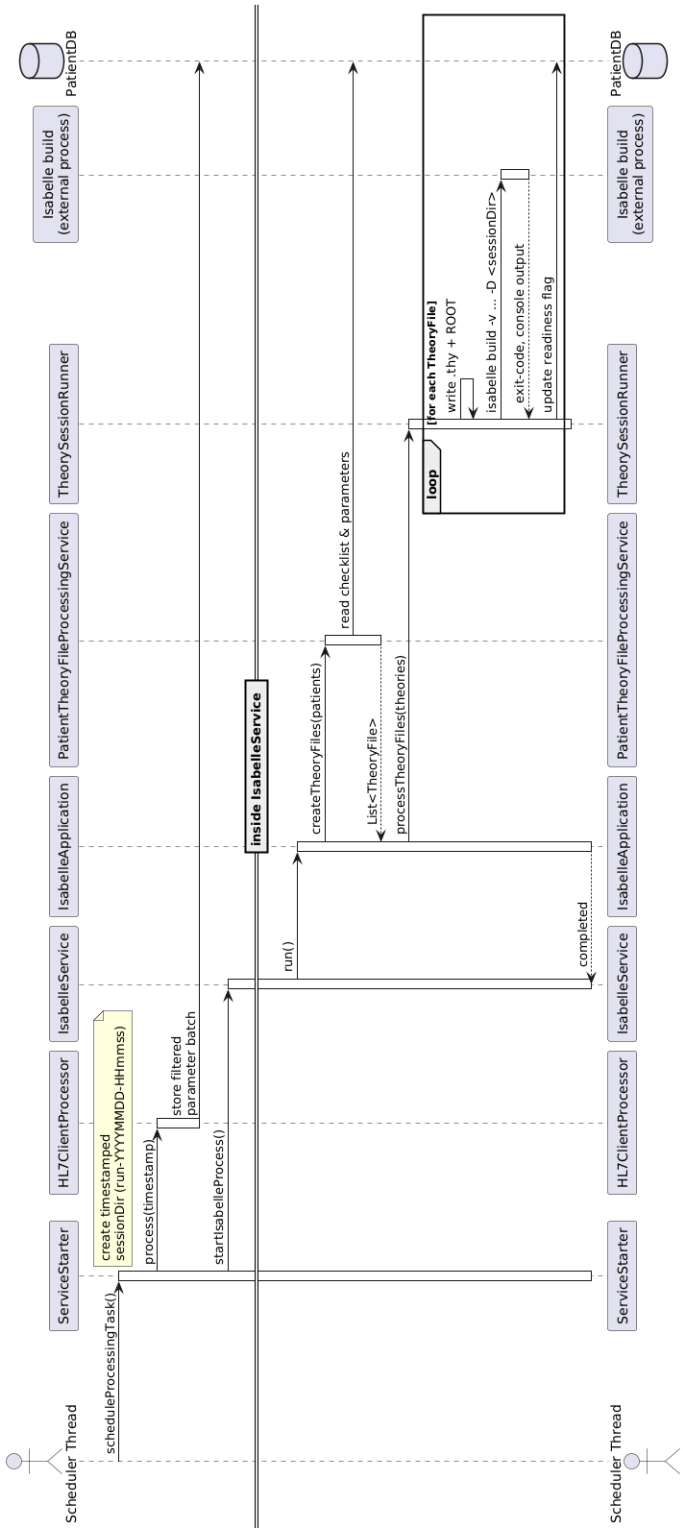


Figure 4.7: High-Level Sequence Diagram of the Batch-Mode Verification Cycle

The batch-mode approach removes the need for manual intervention, provides deterministic behaviour, and satisfies auditability requirements in clinical settings.

### File System Layout and Traceability

Transitioning to the structure of the file system, the organization of session directories and theory files supports maintainability and incremental builds. Each verification run creates its own timestamped directory. All artifacts, including the patient-specific theory file and the `ROOT` file, are written there, processed, and then securely overwritten and deleted. Only `session.log` is retained for audit purposes. For every patient the system generates exactly one theory file per cycle, named `<PatientId><ChecklistName>` (e.g. Patient1002SafetyScreenss1). The file identifier is stored alongside the patient record in the database and surfaced in the graphical interface. From that interface the user can view the transient `.thy` file and open it in Isabelle/jEdit for interactive inspection. This design gives users full traceability, from a patient entry in the UI to the exact theory file used for verification, without leaving residual clinical data on disk.

### Result Interpretation and Application Flags

To manage the outcome of each build process, the system employs structured output capture and interpretation. ProcessExecutor captures the standard output of the Isabelle build process using a dedicated StreamReader. Upon termination, the TheorySessionRunner interprets the exit code as outlined in Table 4.9. Exit code `0` indicates successful verification of all lemmas and triggers the `SBT_READY` flag. A return code of `1` denotes that at least one lemma has failed, leading to the `NOT_READY` status. Any other exit code results in the `NOT_AVAILABLE` flag, typically reflecting internal build errors, proof timeouts, or incomplete termination of the Isabelle process.

| Exit Code | Interpretation | Application Flag |
|---|---|---|
| 0 | All lemmas passed | `SBT_READY` |
| 1 | At least one lemma failed | `NOT_READY` |
| Other | Unexpected error | `NOT_AVAILABLE` |

Table 4.9: Exit codes and associated application flags

The resulting flag is written to the patient record and becomes visible in the UI. The complete console transcript is stored in `session.log`, which provides an immutable audit trail. The option `show_sorts=true` enriches the log with type-inference details and therefore supports diagnosis when proofs fail.

### Tactic Selection and Proof Script Design

As for proof tactics, the pipeline relies exclusively on the simplification tactic `simp`. The decision is grounded in several factors that the dedicated strategy note enumerates. `simp` targets rewriting and simplification, which is exactly what the arithmetic and Boolean expressions in the checklist logic require. Its

behaviour is highly predictable, a property that is essential for unattended batch execution. It outperforms more general tactics such as auto by avoiding superfluous reasoning steps, and it avoids the configuration overhead associated with external tools like sledgehammer and nitpick. Minimal, transparent proof scripts promote maintainability and auditability, in accordance with recommendations for clinical decision support (Blanchette et al., 2011). Consequently, the template system embeds a single line by simp for every local lemma, thereby maximizing efficiency without compromising correctness.

#### Limitations of Batch Mode and Diagnostic Safeguards

Nevertheless, the use of batch mode introduces several well-known challenges. Diagnostic feedback is less detailed than in the interactive environment, and adaptive tools such as sledgehammer are unavailable. The pipeline incorporates several safeguards to address these limitations. I/O errors in `SessionFileManager` and runtime exceptions in the scheduler are caught and logged. Such failures map to the flag `NOT_AVAILABLE`, ensuring that a single fault does not block subsequent cycles. For deeper analysis a user can view the transient `.thy` file from the GUI and open it in Isabelle/jEdit, where richer state views permit step-by-step investigation of proof failures. Although batch mode restricts available proof tools and offers reduced feedback, these measures, together with verbose logs, make problem diagnosis tractable.

#### Security Measures and Future Enhancements

Security considerations are also central to the design. Each run operates in a separate directory, which simplifies permission management and confines potential breaches. `DefaultFileWiper` overwrites theory and `ROOT` files before deletion, thereby limiting the exposure period of sensitive data. Currently, Java and Isabelle run under the same operating-system account. A future migration to a dedicated Isabelle account will enable finer-grained control. Encryption of databases and broader regulatory issues are examined in Section 6.1.

Taken together, the integration of Isabelle/HOL through a purpose-built batch-mode pipeline enables continuous, machine-checked verification of patient-specific checklist models. Deterministic proof tactics, modular session directories, secure file handling, and structured result mapping together create a dependable foundation. Section 4.4 describes the code-generation templates that produce the theory files executed here, and Section 4.5 offers an end-to-end demonstration of the entire workflow.

## 4.4 Code Generator and Template System

A central requirement for automated and trustworthy clinical decision support is the programmatic translation of high-level checklist logic and patient data into formally verifiable code. This section details the structure, responsibilities, and operational principles of the code generator and template system developed for generating Isabelle/HOL theory files from the application's domain models. Building upon the conceptual workflow and implementation pipeline described

in Sections 3.2 and Section 4.3, this subsystem operationalizes the transformation of clinical artefacts into rigorous, machine-checked formal models.

### Architecture and Component Responsibilities

At the architectural level, the code generator operates as a dedicated component within the `isabelleapplication.codegen` package, tightly integrated with the broader Java application architecture. Its primary responsibility is to translate instances of the SafetyScreen checklist and associated patient data into syntactically valid and semantically faithful Isabelle/HOL theory files. These files encapsulate all threshold logic, condition definitions, and supporting constructs required for automated proof execution, as introduced in Section 4.3. The generation process is coordinated through a layered system of classes, each entrusted with distinct aspects of theory file assembly. ThyRenderer serves as the entry point for template-based generation, orchestrating the composition of theory sections start, body, and end via delegation to specialized generators. Section-specific generators (TheoryStartGenerator, TheoryBodyGenerator, TheoryEndGenerator) handle the incremental construction of their respective theory file segments, corresponding to the session header, imports, datatypes, body logic, and the closing marker. Supporting generators such as ThresholdGenerator, PatientGenerator, and ConditionDefinitionGenerator manage the injection of domain-specific parameters and logic constructs into reusable code templates. Each sub-generator queries collector classes that traverse all conditions in the referenced safety screen. These collectors ensure, for instance, that derived parameters such as $PaO_2/FiO_2$ are included only once, even if referenced in multiple conditions.

This modular design reflects the overall system architecture's emphasis on explainability, maintainability, and extensibility. The overall orchestration of the code generation process is illustrated in Figure 4.8. The diagram depicts the interaction between the central generator components, highlighting the sequential composition of the theory file from modular sections and its subsequent serialization to disk.

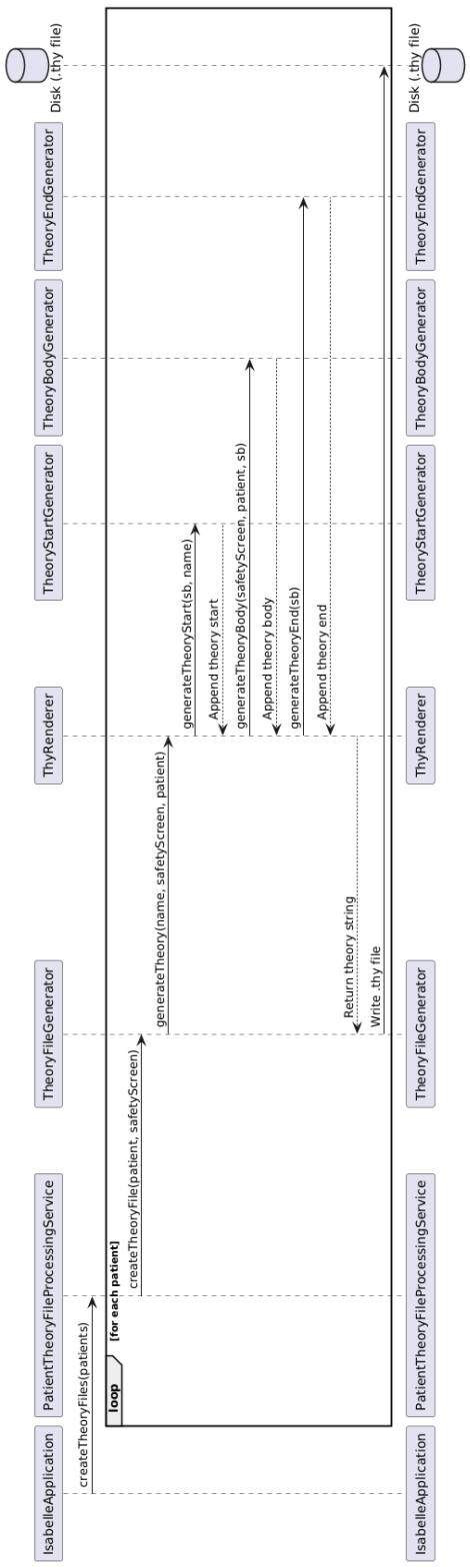


Figure 4.8: High-Level Diagram of Code Generation Process

### Generation Goals and Template Abstraction

The generator has two principal goals: first, to enable the translation of arbitrary checklist instances into syntactically correct Isabelle theories without manual intervention and second, to preserve a one-to-one correspondence between domain semantics and formal definitions, thereby enabling the traceability requirements identified in Chapter 3.

To realize these goals, the generator relies on a robust template system that abstracts away Isabelle/HOL boilerplate while exposing well-defined placeholders for the injection of checklist logic, parameter values, and proof scaffolding. Templates are provided through the `TemplateProvider` interface, with a default implementation that supplies all necessary code fragments for theory construction as Java multi-line strings for human readability. Placeholders are marked with ordinary format specifiers and are resolved exclusively inside the Java process, thereby avoiding runtime string manipulation within Isabelle and keeping the trusted computing base small.

Parameter injection follows a precise and deterministic process. First, clinical parameters are normalized by `ParameterHelper`, which converts display names into lower-case identifiers that are valid in Isabelle. Second, structural information, such as record fields or threshold assignments, is collected in deterministic insertion order, guaranteeing stable file diffs across runs. Third, the generator substitutes the collected strings into their templates, producing fully expanded sections that can be streamed directly to disk. The same strategy governs the insertion of numerical thresholds and comparison operators. For example, a template line such as

```
%s_threshold = ⟦ th_value = %s, th_operator = %s ⟧
```

receives three arguments: the field name, the threshold value, and the operator token (Lower Than (LT), Greater Than (GT), and so forth). This approach decouples textual layout from semantic data and supports centralized changes to the presentation layer.

### Theory File Structure and Logical Semantics

In terms of output, the generated `.thy` files adhere to a standardized structure. First, the theory header introduces the theory, imports required Isabelle modules, and opens the proof context. This is followed by datatype and record definitions, which declare a generalized comparison operator (cmp_op), records for thresholds (threshold_with_op, thresholds), and the patient data structure (patient). Next, functions and default instances are defined, including the central threshold application function (apply_threshold) and default records for thresholds and patient data. Subsequently, each checklist condition is expressed as a dedicated logical predicate (e.g., valid_$PaO_2$_DIVIDE_$FiO_2$), including both single and combined (Boolean) forms. These predicates are then aggregated into higher-order checklist fulfillment definitions (e.g., checklist_fulfilled). Finally, the file includes proof scaffolding that declares one lemma per predicate, verifying correctness against the default patient instance, and concludes with an end marker that closes the theory context.

The following excerpt (see Figure 4.9) summarizes the overall structure:

```
theory <PatientChecklistName>
  imports Main "HOL.Real"
begin

datatype cmp_op = LT | LE | EQ | GT | GE

record threshold_with_op = ...
fun apply_threshold :: "threshold_with_op ⇒ real ⇒ bool" where ...

record thresholds = ...

definition default_thresholds :: thresholds where ...

record patient = ...

definition default_patient :: patient where ...

definition <condition_name> :: "patient ⇒ bool" where ...
declare <condition_name>_def [simp]

definition <combined_condition_name> :: "patient ⇒ bool" where ...
declare <combined_condition_name>_def [simp]

definition checklist_fulfilled :: "patient ⇒ bool" where ...
declare checklist_fulfilled_def [simp]

lemma test_<condition_name>_pass: "<condition_name> default_patient" by (simp)
...
lemma test_checklist_fulfilled_pass: "checklist_fulfilled default_patient" by (simp)

end
```

Figure 4.9: Isabelle/HOL Theory File Template

Each section of the theory file is populated through parameter injection. The code generator programmatically fills template placeholders with the following: Threshold parameters are injected as record fields and assignment statements in the thresholds record and its defaults, reflecting the structure and values defined in the checklist. Patient parameters are translated into record fields and default assignments within the patient record, sourced from current patient data. Checklist conditions are recursively traversed. For each condition, the system generates either a single-parameter predicate or a composite Boolean function, as determined by its logical structure. For each condition and the checklist as a whole, the generator produces a corresponding lemma, asserting that the default patient instance satisfies the associated predicate. The result is a fully populated, Isabelle/HOL-compliant theory file, ready for batch-mode verification.

### Formalizing Complex Conditions and Naming Strategy

To support a wide variety of logical forms, the generator system addresses the formalization of heterogeneous clinical logic through a layered abstraction. Single conditions are rendered as predicates comparing patient parameters to threshold values via the `apply_threshold` function. The abstraction over comparison operators and encapsulation of threshold logic in threshold_with_op supports easy extension and generalization. These predicates are always generated with the signature `patient` $\Rightarrow$ `bool`, allowing higher-level constructs and lemmas to be derived mechanically. Combined conditions are composed using logical conjunction ($\wedge$) and disjunction ($\vee$), with each logical operator mapped to the appropriate Isabelle syntax and semantics. CombinedConditionDefinitionGenerator constructs conjunctions or disjunctions over previously generated predicate names, and both generators rely on ConditionDefinitionNameCollector to guarantee unique identifiers. The collector appends a fixed suffix `p` when the patient argument is curried, which prevents accidental clashes with threshold field names and aligns naming conventions across simple and combined forms. Record-based fields further facilitate modularity, enabling future extension as new parameters or thresholds are introduced. This strategy preserves clinical traceability and supports stepwise verification. Each logical unit, whether a base condition or a compound predicate, corresponds directly to an interpretable clinical rule, and each is accompanied by an explicit test lemma for proof granularity.

### Extensibility and Reusability Across Checklists

The template system is also designed to maximize reusability and extensibility. All code blocks are parameterized and decoupled from hard-coded clinical specifics. Reusable templates allow boilerplate code, from type definitions to lemma declarations, to be defined once and populated dynamically for any checklist or patient instance. Template methods (e.g., *getThresholdRecord*(), *getPatientRecord*(), *getSingleCondition*(), *getCombinedCondition*() encapsulate canonical Isabelle/HOL idioms, ensuring consistency and minimizing duplication. Institutions that require alternative coding guidelines can supply a bespoke implementation of TemplateProvider without altering the generation logic. Extensibility is supported through the modular decomposition of condition generation, which allows new checklist logic to be introduced by extending the domain model, with no requirement for template redesign. New parameters, conditions, or logical forms are automatically integrated through the generator's traversal and code emission logic. Additionally, the abstract threshold representation, where the record threshold_with_op pairs a value with a comparison operator, can be extended by adding new operator variants to the cmp_op datatype and updating the pattern match in `apply_threshold`, without affecting downstream templates. This architectural decision supports long-term maintainability and enables adaptation to evolving clinical guidelines, integration of additional checklists, and extension to new domains (Section 3.5).

### Alignment with Methodological Requirements

Finally, the template system and code generator are aligned with the modeling and verification principles established in Chapter 3. They operationalize the

requirements of explainability, traceability, and correctness-by-construction by ensuring that every predicate and lemma in the generated file maps directly to a discrete clinical rule or checklist item, supporting fine-grained error localization and auditability. Typed records for thresholds and patients encapsulate all relevant data, supporting modular, scalable model growth. The use of a uniform comparison operator type (cmp_op) and the encapsulation of threshold application logic in a single function (`apply_threshold`) permit declarative and robust rule expression. All predicates are generated with a uniform type signature, supporting mechanical derivation of higher-level constructs, including automated lemmas. All generated definitions are marked as simplification rules (`[simp]`), supporting automated, unattended proof execution, and the generation of dedicated lemmas ensures granular error localization. Lastly, all files use plain American Standard Code for Information Interchange (ASCII)-compatible Isabelle syntax, maximizing interoperability and minimizing encoding issues during automated batch-mode processing.

The code generator and template system constitute the technical heart of the system's formalization pipeline, bridging the gap between domain-specific clinical logic and the requirements of automated, batch-mode formal verification. Through modular design, parameterized templates, and a principled approach to condition composition, this subsystem ensures that all generated Isabelle/HOL theory files are correct-by-construction, transparent, and ready for integration into clinical workflows. This foundation enables the safe, repeatable, and extensible formalization of critical care checklists and supports the methodological goals articulated in preceding chapters.

The following Section 4.5 demonstrates the practical execution of this generation process through a real-world use case, illustrating how clinical checklists and patient data are transformed into formal proofs within the complete verification pipeline.

## 4.5 Implementation Walkthrough - From Checklist to Proof

This section presents a comprehensive, end-to-end walkthrough of the verification pipeline as outlined in Sections 4.1 through 4.4. While previous sections have addressed general artifact formats and procedural steps, the current focus is on a single real-world use case. The description references data formats, architectural components, and the overall process flow of the system. An evaluation of correctness, clinical relevance, or system performance is not performed at this stage. These aspects are addressed in Chapter 5.

### Checklist Selection and Question Setup

The demonstration centers on the Safety Screen checklist `ss1`, which is based on the protocol established at Bamberg Hospital (see Figure 2.1). For this illustrative scenario, two questions are included:

> "$PaO_2/FiO_2$ größer 150mmHg und $FiO_2$ weniger als 50%?" and
>
> "PEEP von 10cm$H_2O$ oder weniger?".

These questions reflect clinically relevant decision criteria and showcase both derived and simple parameter types typically used in ICU safety screens. Two synthetic patient identifiers, `1001` and `1002`, are used throughout. These records are generated for demonstration purposes only and do not contain any personal data.

Checklist construction begins with the question "$PaO_2/FiO_2$ größer 150mmHg und $FiO_2$ weniger als 50%?", which is created through the graphical interface (see Figure 4.10). Within this interface (see Figure 4.11), users can add questions to a checklist or save the checklist itself.

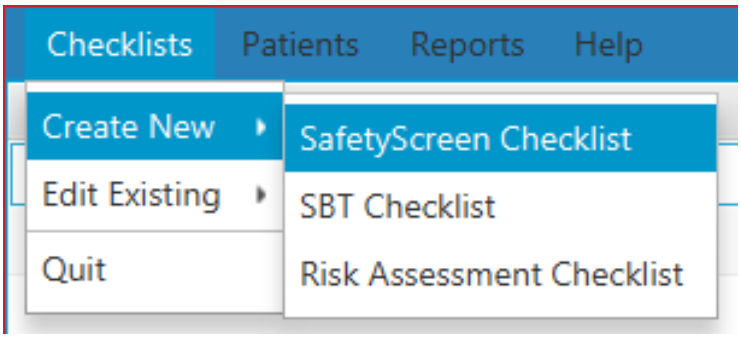


Figure 4.10: Create New SafetyScreen Checklist Tab

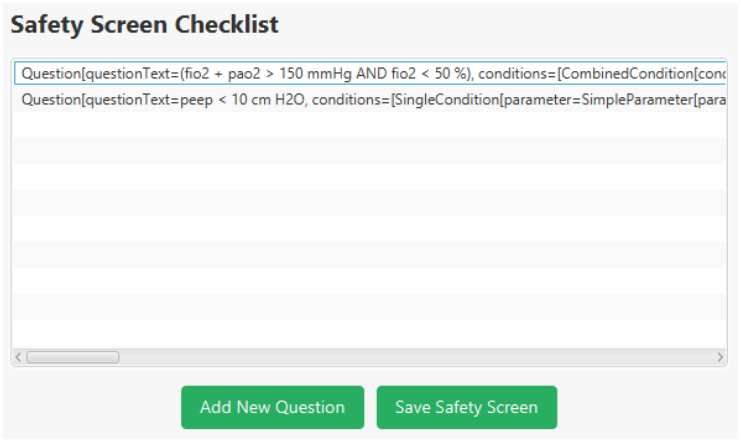


Figure 4.11: Safety Screen Checklist - Add Question and Save Safety Screen

### Interactive Question Configuration

When a new question is added, a configuration window (see Figure 4.12) allows for the interactive definition of logical conditions. The first question is modeled as a combined condition using the logical `AND` operator. It consists of a derived parameter on the left and a simple parameter on the right.

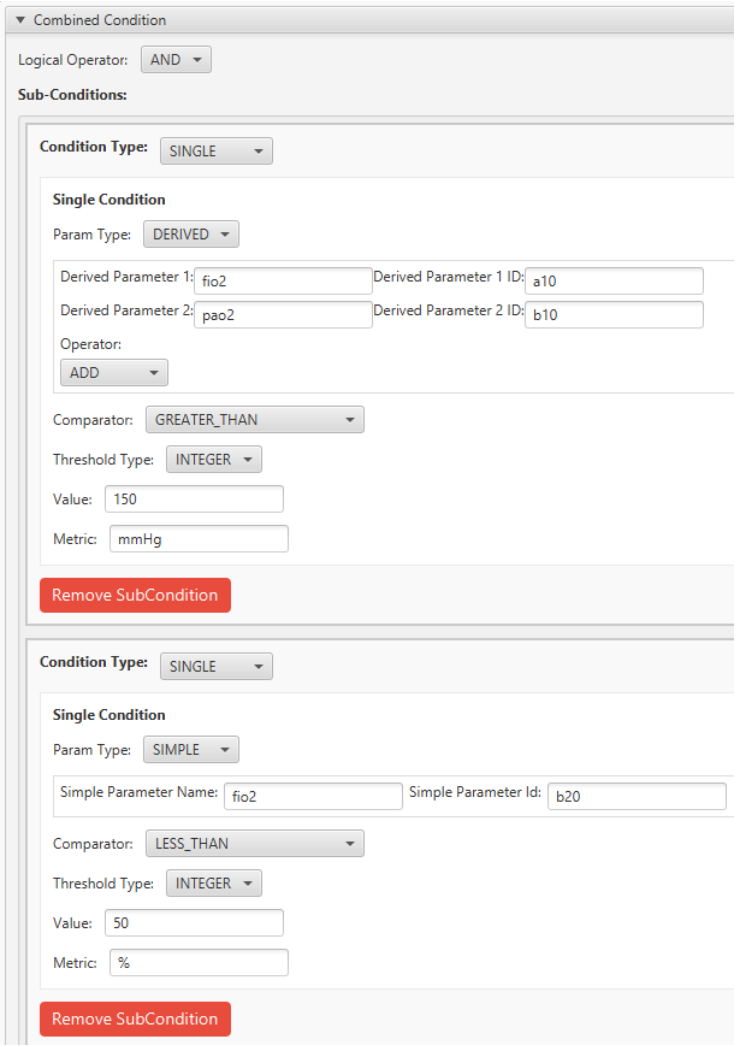


Figure 4.12: Condition Configuration Window

The second question, “PEEP von 10cm$H_2O$ oder weniger?”, is defined as a single condition using a simple parameter.

### Checklist Serialization and Embedding in Risk Assessment

Once completed, the entire checklist is saved in JSON format. Its structure is illustrated in Figure 4.13.


```
{
  "name": "ss1",
  "safetyScreenQuestions": [
    {
      "questionText": "(pao2 / fio2 > 150 mmHg AND fio2 < 50 %)",
      "conditions": [
        {
          "conditionType": "combined",
          "conditions": [
            {
              "conditionType": "single",
              "parameter": {
                "parameterType": "derived",
                "firstParameter": "pao2",
                "firstParameterId": "a10",
                "secondParameter": "fio2",
                "secondParameterId": "b10",
                "derivationOperator": "DIVIDE"
              },
              "threshold": {
                "value": 150,
                "metric": {"unit": "mmHg"}
              },
              "comparisonOperator": "GREATER_THAN"
            },
            {
              "conditionType": "single",
              "parameter": {
                "parameterType": "simple",
                "parameterName": "fio2",
                "parameterId": "b20"
              },
              "threshold": {
                "value": 50,
                "metric": {"unit": "%"}
              },
              "comparisonOperator": "LESS_THAN"
            }
          ],
          "logicalOperator": "AND"
        }
      ]
    },
    {
      "questionText": "peep <= 10 cmH2O",
      "conditions": [
        {
          "conditionType": "single",
          "parameter": {
            "parameterType": "simple",
            "parameterName": "peep",
            "parameterId": "c10"
          },
          "threshold": {
            "value": 10,
            "metric": {"unit": "cmH2O"}
          },
          "comparisonOperator": "LESS_THAN_OR_EQUAL"
        }
      ]
    }
  ]
}
```


Figure 4.13: Structure of Safety Screen Checklist in JSON format

This Safety Screen checklist is embedded into the broader Risk Assessment Checklist named `ra1` and set as the system default. This ensures that it is automatically applied to all patients unless a specific alternative is assigned (see Figure 4.14).

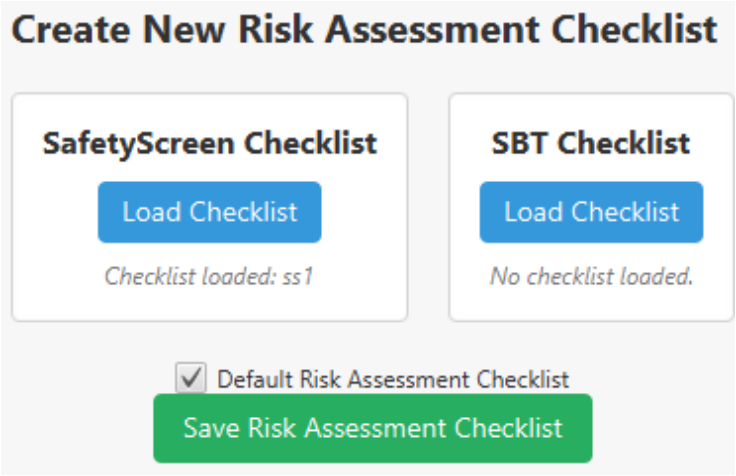


Figure 4.14: Create New Risk Assessment Checklist Window

### Patient Ingestion and Batch Preparation

Patients 1001 (see Figure 4.15) and 1002 (see Figure 4.16) are processed according to the ingestion procedure described in Section 4.2. Their data include all parameters referenced by the checklist `ss1`. Parameters not used in the checklist are filtered out prior to storage.


```
{
  "patientId": "1001",
  "healthParameterBatches": [
    {
      "timestamp": "20250524-180624",
      "parameters": [
        {"id": "a10", "name": "pao2", "value": 103.0, "unit": "mmHg"},
        {"id": "b10", "name": "fio2", "value": 0.69, "unit": "mmHg"},
        {"id": "b20", "name": "fio2", "value": 68.9, "unit": "%"},
        {"id": "c10", "name": "peep", "value": 10.71, "unit": "cmH2O"}
      ],
      "sbtReady": "NOT_READY"
    }
  ],
  "checklistReference": "ra1",
  "sbtReady": "NOT_READY"
}
```


Figure 4.15: Structure of Patient 1001 stored in JSON format

```
{
  "patientId": "1002",
  "healthParameterBatches": [
   {
    "timestamp": "20250524-180624",
    "parameters": [
     {"id": "a10", "name": "pao2", "value": 139.78, "unit": "mmHg"},
     {"id": "b10", "name": "fio2", "value": 0.4, "unit": "mmHg"},
     {"id": "b20", "name": "fio2", "value": 40.0, "unit": "%"},
     {"id": "c10", "name": "peep", "value": 8.54, "unit": "cmH2O"}
    ],
    "sbtReady": "READY"
   }
  ],
  "checklistReference": "ra1",
  "sbtReady": "READY"
 }
```


Figure 4.16: Structure of Patient 1002 stored in JSON format

The graphical user interface displays the batch data (see Figure 4.17) for each patient accordingly.

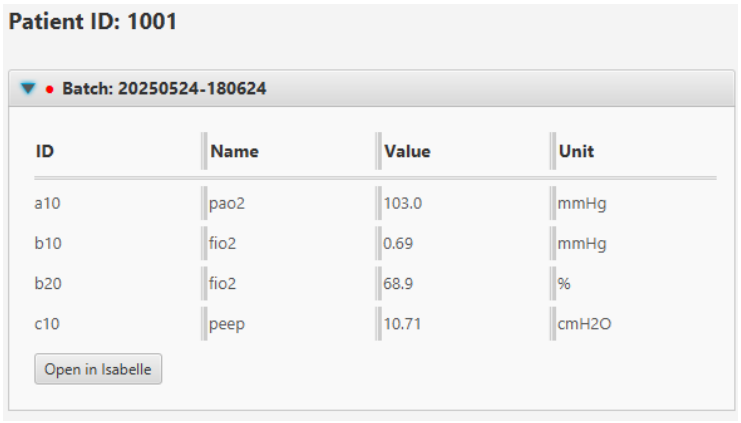


Figure 4.17: Health parameter batch of patient 1001

Patient 1002 is shown similarly in the interface, but marked in green for visual distinction.

### Theory Generation and Formal Verification in Isabelle

When the scheduler triggers the verification process, the Isabelle Process Module generates a theory file for each patient–checklist combination. These theory files follow the structure introduced in Section 4.4. Each checklist question is mapped to a predicate and accompanied by a lemma that evaluates the predicate against the current patient instance. The generated theory file for patient 1001 is shown in Figure 4.18.

```
theory Patient1001SafetyScreenss1
  imports Main "HOL.Real"
begin


datatype cmp_op = LT | LE | EQ | GT | GE


record threshold_with_op =
  th_value    :: real
  th_operator :: cmp_op


fun apply_threshold :: "threshold_with_op ⇒ real ⇒ bool" where
  "apply_threshold t x = (
    if x = -1.0 then False
    else case th_operator t of
        LT ⇒ x < th_value t
      | LE ⇒ x ≤ th_value t
      | EQ ⇒ x = th_value t
      | GT ⇒ x > th_value t
      | GE ⇒ x ≥ th_value t
  )"
record thresholds =
  pao2_DIVIDE_fio2_threshold :: threshold_with_op
  fio2_threshold :: threshold_with_op
  peep_threshold :: threshold_with_op


definition default_thresholds :: thresholds where
  "default_thresholds = ⦇
    pao2_DIVIDE_fio2_threshold = ⦇ th_value = 150.0, th_operator = GT ⦈,
fio2_threshold = ⦇ th_value = 50.0, th_operator = LT ⦈,
peep_threshold = ⦇ th_value = 10.0, th_operator = LE ⦈
  ⦈"
declare default_thresholds_def [simp]


record patient =
  pao2_DIVIDE_fio2 :: real
  fio2 :: real
  peep :: real


definition default_patient :: patient where
  "default_patient = ⦇
    pao2_DIVIDE_fio2 = 103.0/0.69,
fio2 = 68.9,
peep = 10.71
  ⦈"
declare default_patient_def [simp]


(*(pao2 / fio2 > 150 mmHg AND fio2 < 50 %)*)
definition valid_pao2_DIVIDE_fio2 :: "patient ⇒ bool" where
  "valid_pao2_DIVIDE_fio2 p ≡ apply_threshold (pao2_DIVIDE_fio2_threshold default_thresholds)
(pao2_DIVIDE_fio2 p)"
declare valid_pao2_DIVIDE_fio2_def [simp]


definition valid_fio2 :: "patient ⇒ bool" where
  "valid_fio2 p ≡ apply_threshold (fio2_threshold default_thresholds) (fio2 p)"
declare valid_fio2_def [simp]


definition valid_pao2_DIVIDE_fio2_and_fio2 :: "patient ⇒ bool" where
  "valid_pao2_DIVIDE_fio2_and_fio2 p ≡ valid_pao2_DIVIDE_fio2 p ∧valid_fio2 p"
declare valid_pao2_DIVIDE_fio2_and_fio2_def [simp]


(*peep <= 10 cmH2O*)
definition valid_peep :: "patient ⇒ bool" where
  "valid_peep p ≡ apply_threshold (peep_threshold default_thresholds) (peep p)"
declare valid_peep_def [simp]


definition checklist_fulfilled :: "patient ⇒ bool" where
  "checklist_fulfilled p ≡ valid_pao2_DIVIDE_fio2_and_fio2 p ∧valid_pao2_DIVIDE_fio2 p ∧valid_fio2 p
∧valid_peep p"
declare checklist_fulfilled_def [simp]


lemma test_valid_pao2_DIVIDE_fio2_and_fio2_pass:
  "valid_pao2_DIVIDE_fio2_and_fio2 default_patient"
  by (simp)


lemma test_valid_pao2_DIVIDE_fio2_pass:
  "valid_pao2_DIVIDE_fio2 default_patient"
  by (simp)


lemma test_valid_fio2_pass:
  "valid_fio2 default_patient"
  by (simp)


lemma test_valid_peep_pass:
  "valid_peep default_patient"
  by (simp)


lemma test_checklist_fulfilled_pass:
  "checklist_fulfilled default_patient"
  by (simp)


end
```

Figure 4.18: Full Theory File for Patient 1001

This theory file can be opened directly in Isabelle/jEdit using the "Open in Isabelle" button (see Figure 4.17) within the patient batch view. In the case of patient 1001, some lemmas fail, indicating that certain checklist conditions are not met. This is shown in Figure 4.19

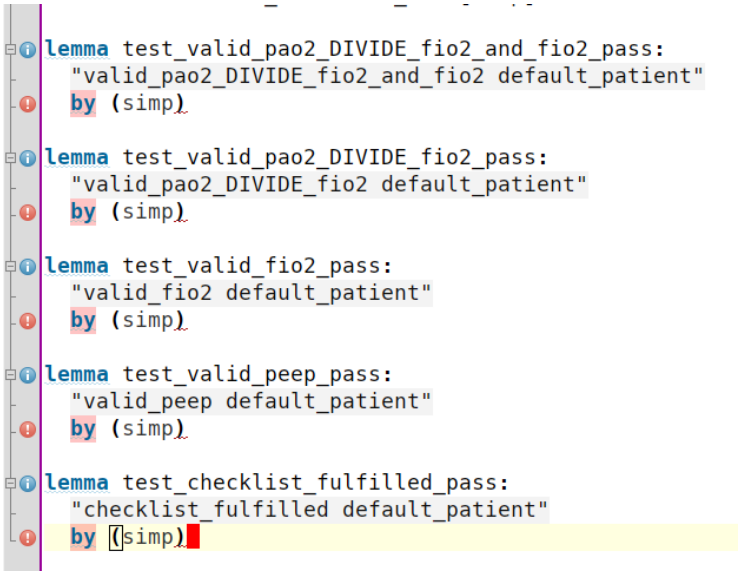


Figure 4.19: Failed Lemmas for Patient 1001 in Isabelle/jEdit

The file for patient 1002 follows the same structure, differing only in the parameter values used during evaluation.

### Batch Execution and Session Management

As outlined in Section 4.3, the execution of theory files is performed in batch mode using the session runner. The session files are organized into a directory structure:

- Superordinate folder: `run-20250524-180624`
  - Patient 1001 folder: `Patient1001SafetyScreenss1`
  - Patient 1002 folder: `Patient1002SafetyScreenss1`

The execution of each session is initiated via a custom Isabelle build command that incorporates specific options to ensure detailed and precise output. An example command is shown below:

```
[.../Isabelle2024/contrib/cygwin/bin/bash.exe, -l, -c,
.../Isabelle2024/bin/isabelle build -v -o quick_and_dirty=false
-o show_sorts=true -D /cygdrive/.../run-20250524-180624
/Patient1001SafetyScreenss1]
```

This command launches Isabelle in batch mode using the Cygwin environment, invoking the `isabelle build` tool with the following flags:

- `-v` enables verbose output, which is helpful for debugging and traceability.
- `-o quick_and_dirty=false` ensures that all proofs are fully checked rather than skipped for speed.
- `-o show_sorts=true` includes sort annotations in the output, which supports formal traceability of type constraints.
- The `-D` flag specifies the session directory containing the theory files for the current patient.

Each patient's verification results are recorded in a dedicated `session.log` file within their respective session directory (see Figure 4.20 and Figure 4.21). The log file provides a detailed trace of the execution process, including invoked commands, build options, session start and end times, as well as success or failure indicators. In the case of a failure, the log pinpoints the specific theory line (e.g., via `At command "by"`), which aids in diagnosing the proof error. Conversely, a successful session is explicitly marked with a message confirming that the theory was proved successfully.

```
2025-05-24 18:06:34.492 INFO  c.e.s.i.s.TheorySessionRunner - ------------------------------------------------------------
2025-05-24 18:06:34.494 INFO  c.e.s.i.s.TheorySessionRunner - Running session for theory:
Patient1002SafetyScreenss1
2025-05-24 18:06:34.494 INFO  c.e.s.i.s.TheorySessionRunner - Session directory (local): ...\run-
20250524-180624\Patient1002SafetyScreenss1
2025-05-24 18:06:34.494 INFO  c.e.s.i.s.TheorySessionRunner - Session directory (cygwin):
/cygdrive/.../run-20250524-180624/Patient1002SafetyScreenss1
2025-05-24 18:06:34.494 INFO  c.e.s.i.s.TheorySessionRunner - Executing command:
[.../Isabelle2024/contrib/cygwin/bin/bash.exe, -l, -c, .../Isabelle2024/bin/isabelle build -v -o
quick_and_dirty=false -o show_sorts=true -D/cygdrive/.../run-20250524-
180624/Patient1002SafetyScreenss1]
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - Started at Sat May 24 18:06:37 GMT+2
2025 (polyml-5.9.1_x86_64_32-windows on WINDOWS-8FHG1NL)
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - ISABELLE_TOOL_JAVA_OPTIONS="-
Djava.awt.headless=true -Xms512m -Xmx4g -Xss16m"
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - ISABELLE_BUILD_OPTIONS=""
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner -
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - ML_PLATFORM="x86_64_32-
windows"
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner -
ML_HOME="/cygdrive/...i/Isabelle2024/contrib/polyml-5.9.1/x86_64_32-windows"
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - ML_SYSTEM="polyml-5.9.1"
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - ML_OPTIONS="--minheap 500"
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner -
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - Session Pure/Pure
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - Session Misc/Tools
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - Session HOL/HOL (main)
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - Session
Unsorted/Patient1002SafetyScreenss1
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - Running Patient1002SafetyScreenss1
...
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - Patient1002SafetyScreenss1: theory
Patient1002SafetyScreenss1.Patient1002SafetyScreenss1
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - Timing Patient1002SafetyScreenss1 (6
threads, 1.326s elapsed time, 1.547s cpu time, 0.000s GC time, factor 1.17)
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - Finished Patient1002SafetyScreenss1
(0:00:02 elapsed time)
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner -
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - Finished at Sat May 24 18:06:43
GMT+2 2025
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - 0:00:06 elapsed time
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - Session for theory
Patient1002SafetyScreenss1 finished with exit code 0
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - Theory Patient1002SafetyScreenss1
proved successfully!
2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - ------------------------------------------------------------


2025-05-24 18:06:43.349 INFO  c.e.s.i.s.TheorySessionRunner - Local session log written to: ...\run-
20250524-180624\Patient1002SafetyScreenss1\session.log
```

Figure 4.20: session.log with succeeded SBT verification for Patient 1002

```
2025-05-24 18:06:25.883 INFO  c.e.s.i.s.TheorySessionRunner - ----------
2025-05-24 18:06:25.884 INFO  c.e.s.i.s.TheorySessionRunner - Running session for theory:
Patient1001SafetyScreenss1
2025-05-24 18:06:25.884 INFO  c.e.s.i.s.TheorySessionRunner - Session directory (local): ...\run-
20250524-180624\Patient1001SafetyScreenss1
2025-05-24 18:06:25.886 INFO  c.e.s.i.s.TheorySessionRunner - Session directory (cygwin):
/cygdrive/.../run-20250524-180624/Patient1001SafetyScreenss1
2025-05-24 18:06:25.886 INFO  c.e.s.i.s.TheorySessionRunner - Executing command:
[.../Isabelle2024/contrib/cygwin/bin/bash.exe, -l, -c, .../Isabelle2024/bin/isabelle build -v -o
quick_and_dirty=false -o show_sorts=true -D/cygdrive/.../run-20250524-
180624/Patient1001SafetyScreenss1]
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner - Started at Sat May 24 18:06:28 GMT+2
2025 (polyml-5.9.1_x86_64_32-windows on WINDOWS-8FHG1NL)
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner - ISABELLE_TOOL_JAVA_OPTIONS="-
Djava.awt.headless=true -Xms512m -Xmx4g -Xss16m"
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner - ISABELLE_BUILD_OPTIONS=""
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner -
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner - ML_PLATFORM="x86_64_32-
windows"
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner -
ML_HOME="/cygdrive/.../Isabelle2024/contrib/polyml-5.9.1/x86_64_32-windows"
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner - ML_SYSTEM="polyml-5.9.1"
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner - ML_OPTIONS="--minheap 500"
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner -
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner - Session Pure/Pure
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner - Session Misc/Tools
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner - Session HOL/HOL (main)
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner - Session
Unsorted/Patient1001SafetyScreenss1
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner - Running Patient1001SafetyScreenss1
...
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner - Patient1001SafetyScreenss1: theory
Patient1001SafetyScreenss1.Patient1001SafetyScreenss1
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner - Patient1001SafetyScreenss1 FAILED
(see also "isabelle build_log -H Error Patient1001SafetyScreenss1")
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner - *** Failed to finish proof (line 87 of
"~/IsabelleFiles/run-20250524-180624/Patient1001SafetyScreenss1/Patient1001SafetyScreenss1.thy"):
2025-05-24 18:06:34.473 INFO  c.e.s.i.s.TheorySessionRunner - *** goal (1 subgoal):
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - ***  1. False
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - *** At command "by" (line 87 of
"~/IsabelleFiles/run-20250524-180624/Patient1001SafetyScreenss1/Patient1001SafetyScreenss1.thy")
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - *** Failed to finish proof (line 71 of
"~/IsabelleFiles/run-20250524-180624/Patient1001SafetyScreenss1/Patient1001SafetyScreenss1.thy"):
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - *** goal (1 subgoal):
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - ***  1. False
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - *** At command "by" (line 71 of
"~/IsabelleFiles/run-20250524-180624/Patient1001SafetyScreenss1/Patient1001SafetyScreenss1.thy")
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - *** Failed to finish proof (line 75 of
"~/IsabelleFiles/run-20250524-180624/Patient1001SafetyScreenss1/Patient1001SafetyScreenss1.thy"):
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - *** goal (1 subgoal):
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - ***  1. False
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - *** At command "by" (line 75 of
"~/IsabelleFiles/run-20250524-180624/Patient1001SafetyScreenss1/Patient1001SafetyScreenss1.thy")
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - *** Failed to finish proof (line 83 of
"~/IsabelleFiles/run-20250524-180624/Patient1001SafetyScreenss1/Patient1001SafetyScreenss1.thy"):
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - *** goal (1 subgoal):
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - ***  1. False
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - *** At command "by" (line 83 of
"~/IsabelleFiles/run-20250524-180624/Patient1001SafetyScreenss1/Patient1001SafetyScreenss1.thy")
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - *** Failed to finish proof (line 79 of
"~/IsabelleFiles/run-20250524-180624/Patient1001SafetyScreenss1/Patient1001SafetyScreenss1.thy"):
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - *** goal (1 subgoal):
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - ***  1. False
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - *** At command "by" (line 79 of
"~/IsabelleFiles/run-20250524-180624/Patient1001SafetyScreenss1/Patient1001SafetyScreenss1.thy")
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - Unfinished session(s):
Patient1001SafetyScreenss1
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner -
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - Finished at Sat May 24 18:06:34
GMT+2 2025
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - 0:00:05 elapsed time
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - Session for theory
Patient1001SafetyScreenss1 finished with exit code 1
2025-05-24 18:06:34.474 ERROR c.e.s.i.s.TheorySessionRunner - There were errors in the proof for
theory Patient1001SafetyScreenss1.
2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - ----------


2025-05-24 18:06:34.474 INFO  c.e.s.i.s.TheorySessionRunner - Local session log written to: ...\run-
20250524-180624\Patient1001SafetyScreenss1\session.log
```

Figure 4.21: session.log with failed SBT verification for Patient 1001

### System Logs and Diagnostic Outputs

Runtime and communication logs further document the entire process. The `communication.log` (see Figure 4.22) includes events such as system startup, HL7 message handling, and shutdown operations.

The `run.log` (see Figure 4.23) captures file system events, HL7 polling activities, patient save operations, theory generation steps, and verification results.

```
...
2025-05-24 18:06:18.709 INFO  global - H2 TCP Server started on port 9092.
2025-05-24 18:06:18.718 INFO  global - HL7 Mock Server started on port 2575.
2025-05-24 18:06:18.719 INFO  global - Press ENTER to stop all servers...
2025-05-24 18:06:18.719 INFO  global - HL7 Mock Server listening on port 2575
2025-05-24 18:06:22.593 INFO  global - Starting core initialization
2025-05-24 18:06:22.602 INFO  global - Loading application configuration
2025-05-24 18:06:22.611 INFO  global - Initializing databases and applying schema updates
2025-05-24 18:06:24.070 INFO  global - Initializing repositories and repository services
2025-05-24 18:06:24.076 INFO  global - Booting Client Listening Application.
2025-05-24 18:06:24.100 INFO  global - Established Checklist DB: User=sa URL=jdbc:h2:tcp://localhost:9092/~/checklists;CIPHER=AES
Established Patient  DB: User=sa
URL=jdbc:h2:tcp://localhost:9092/~/patients;CIPHER=AES;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE;AUTO_RECONNECT=TRUE;WRITE_DELAY=0
Established User     DB: User=sa URL=jdbc:h2:~/users;CIPHER=AES


2025-05-24 18:06:24.412 INFO  global - Received HL7 message from /127.0.0.1:61618:
MSH|^~\&|MyApp|MyFacility|ExternalSystem|Facility|20250524-180624||QRY^A19|1|P|2.3
QRD|20250524-180624|R||1|||^RD123||
2025-05-24 18:06:24.430 WARN  global - Connection closed before start-of-message from /127.0.0.1:61617
2025-05-24 18:06:49.170 INFO  global - Client stopping, shutting down scheduler...
2025-05-24 18:06:49.171 INFO  global - Shutting down H2 databases...
2025-05-24 18:06:49.244 INFO  global - H2 database shutdown successfully for DataSource: ds0: url=jdbc:h2:tcp://localhost:9092/~/checklists;CIPHER=AES
user=sa
2025-05-24 18:06:49.291 INFO  global - H2 database shutdown successfully for DataSource: ds1:
url=jdbc:h2:tcp://localhost:9092/~/patients;CIPHER=AES;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE;AUTO_RECONNECT=TRUE;WRITE_DELAY=0
user=sa
2025-05-24 18:06:49.369 INFO  global - H2 database shutdown successfully for DataSource: ds2: url=jdbc:h2:~/users;CIPHER=AES user=sa
...
```

Figure 4.22: communication.log

```
2025-05-24 18:06:24.103 INFO  c.e.s.h.client.ServiceStarter - Creating session directory:
C:\Users\patri\IsabelleFiles\run-20250524-180624
2025-05-24 18:06:24.104 INFO  c.e.s.h.client.ServiceStarter - Starting HL7 processing...
2025-05-24 18:06:24.408 INFO  c.e.s.h.client.HL7ClientProcessor - Polling server for all patients + all
parameters
2025-05-24 18:06:24.409 INFO  c.e.s.h.client.HL7Client - Client started listening on Port 2575 and
Server localhost.
2025-05-24 18:06:24.410 INFO  c.e.s.h.client.HL7Client - Connecting to localhost:2575
2025-05-24 18:06:24.411 INFO  c.e.s.h.client.HL7Client - Sending HL7 message
2025-05-24 18:06:24.411 INFO  c.e.s.h.client.HL7Client - Awaiting response
2025-05-24 18:06:24.618 INFO  c.e.s.h.client.HL7ClientProcessor - Saved new patient 1001 → kept
parameters: [a10, b10, b20, c10]
2025-05-24 18:06:24.779 INFO  c.e.s.h.client.HL7ClientProcessor - Saved new patient 1002 → kept
parameters: [a10, b10, b20, c10]
2025-05-24 18:06:25.432 INFO  c.e.s.i.a.IsabelleApplication - Start creating Isabelle/HOL Theory files.
2025-05-24 18:06:25.504 INFO  c.e.s.i.t.s.PatientTheoryFileProcessingService - Creating Isabelle/HOL
theory file for patient 1001.
2025-05-24 18:06:25.504 INFO  c.e.s.i.theory.TheoryFileGenerator - Generating Theory file:
Patient1001SafetyScreenss1.
2025-05-24 18:06:25.516 INFO  c.e.s.i.t.s.PatientTheoryFileProcessingService - Finished creating
Isabelle/HOL theory file for patient 1001.
2025-05-24 18:06:25.589 INFO  c.e.s.i.t.s.PatientTheoryFileProcessingService - Creating Isabelle/HOL
theory file for patient 1002.
2025-05-24 18:06:25.589 INFO  c.e.s.i.theory.TheoryFileGenerator - Generating Theory file:
Patient1002SafetyScreenss1.
2025-05-24 18:06:25.590 INFO  c.e.s.i.t.s.PatientTheoryFileProcessingService - Finished creating
Isabelle/HOL theory file for patient 1002.
2025-05-24 18:06:25.881 INFO  c.e.s.i.a.IsabelleApplication - Finished creating Isabelle/HOL Theory
files.
2025-05-24 18:06:25.881 INFO  c.e.s.i.a.IsabelleApplication - Processing Isabelle/HOL Theory files.
2025-05-24 18:06:25.881 INFO  c.e.s.i.t.s.SessionProcessingService - Validating Theory file for patient
1001.
2025-05-24 18:06:34.476 INFO  c.e.s.i.t.s.SessionProcessingService - Updating SBT Readiness for
patient 1001.
2025-05-24 18:06:34.489 INFO  c.e.s.i.t.s.SessionProcessingService - Validating Theory file for patient
1002.
2025-05-24 18:06:43.351 INFO  c.e.s.i.t.s.SessionProcessingService - Updating SBT Readiness for
patient 1002.
...
```

Figure 4.23: run.log

### SBT Outcome Display and Visual Feedback

Finally, the verification result is displayed in the graphical user interface (see Figure 4.24), indicating the outcome of the SBT readiness evaluation for each patient.

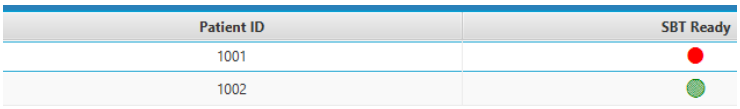


Figure 4.24: SBT verification result in the GUI

This walkthrough demonstrates the coordinated execution of all system modules as outlined in Sections 4.1 to 4.4. From checklist construction and patient data ingestion to theory generation and formal verification, all tasks are executed in a closed, automated loop. A critical evaluation of the approach, regarding formal correctness, clinical interpretability, and operational relevance, will follow in Chapter 5.

# Chapter 5

# Results and Evaluation

This chapter presents the results of the implemented system and evaluates its performance, clinical relevance, and comparative positioning. While the previous chapters established the architectural and methodological foundations, the focus here shifts to examining how effectively the system fulfills its intended objectives in practice. The evaluation encompasses both technical and domain-specific dimensions, reflecting the dual emphasis on formal verification and clinical applicability.

To begin, the use of the Safety Screen checklist in a real-world context is examined to assess the fidelity of its formal representation and the alignment of verification results with clinical expectations. This analysis is presented in Section 5.1, which explores the practical deployment and interpretability of the formalized logic conditions.

In Section 5.2, the system's performance is measured along quantitative dimensions, including execution time for theory file generation and proof processing. This section also considers the robustness of the implementation, particularly in handling edge cases and malformed input.

Beyond performance metrics, Section 5.3 offers a qualitative assessment of the system's utility, with particular attention to explainability and integration potential. This includes a reflection on the system's capacity to support clinical decision-making in a manner that is transparent and auditable.

To contextualize the findings within the broader research landscape, Section 5.4 compares the developed approach with existing symbolic and subsymbolic methodologies in CDSSs. The discussion highlights distinctive features of the current implementation and identifies remaining challenges.

Finally, Section 5.5 consolidates the key results of the chapter. It summarizes the system's effectiveness across technical and clinical dimensions and prepares the ground for the subsequent discussion of limitations, ethical considerations, and opportunities for future enhancement in Chapter 6.

## 5.1 Use Case Evaluation – SBT Safety Screen in Practice

This section presents the first empirical validation of the developed prototype through its application in a controlled yet clinically plausible scenario. The

evaluation focuses on a reduced version of the Bamberg extubation checklist (see Figure 2.1), referred to as `ss1`. This subset consists of two questions that assess oxygenation and ventilatory support. Although derived from the broader SBT protocol, `ss1` does not represent the full scope of the original checklist. The checklist was introduced in Section 4.5. All verification procedures were executed using the automated pipeline described in Chapter 4. Throughout the evaluation, no manual interventions or corrections were applied at any stage.

The central question addressed in this context is whether the prototype can accurately formalize and automatically verify the two-question version of the Bamberg SBT Safety Screen with complete logical fidelity. To investigate this, two synthetic patient records were constructed as described in Section 4.5, specifically illustrated in Figure 4.15 and Figure 4.16. Patient 1001 was designed to violate all specified threshold conditions, whereas Patient 1002 was configured to meet all clinical criteria as defined by `ss1`.

The evaluation proceeded through a structured process in which the checklist logic and patient parameters were formally translated into Isabelle 2024 theory files. These theory files were then processed in batch mode using automated proof checking. The computed results were subsequently compared with the expected clinical judgments based on guideline-conform criteria.

### Verification Results and Interpretation

The results of the automated verification are summarized in the following table:

| Patient | Lemmas Passed | Total Lemmas | Isabelle Exit Code | Computed SBT Flag |
|---|---|---|---|---|
| 1001 | 0 | 5 | 1 (at least one proof fails) | NOT READY |
| 1002 | 5 | 5 | 0 (all proofs succeed) | READY |

Table 5.1: Automated proof results for the Safety Screen `ss1`

The full Isabelle theory file and accompanying build logs for Patient 1001 are provided in Section 4.5. The observed outcomes correspond to clinically plausible interpretations. Patient 1001 failed the verification due to three distinct violations. The $PaO_2/FiO_2$ ratio of 103 divided by 0.69 results in 149.28, which falls below the required threshold of 150mmHg. The $FiO_2$ value of 68.9% exceeds the allowed upper limit of 50%. Furthermore, the PEEP value of 10.71cm$H_2O$ surpasses the defined threshold of 10cm$H_2O$. Each of these deviations would, in standard clinical practice, lead to the postponement of a SBT. By contrast, Patient 1002 fulfilled all relevant criteria, which led to the successful verification of all lemmas and a computed classification of `"READY"`. These results confirm that the prototype's reasoning mechanisms are aligned with current clinical decision-making norms and weaning guidelines.

### Interface-Level Explainability and User Access

In addition to producing accurate outcomes, the system provides features that support explainability for clinical users. Figure 4.17 displays the health parame-

ter batch associated with Patient 1001. It includes a clearly visible readiness flag indicating the verification outcome, as well as a tabular presentation of all physiological values that contributed to the classification. The interface also allows users to open the corresponding Isabelle theory file directly in jEdit by clicking the `"Open in Isabelle"` button, thereby facilitating in-depth examination of the underlying formalization. Figure 4.24 further illustrates the overall classification results for multiple patients as presented in the main application view. In this interface, readiness flags are visualized using colour-coded indicators, providing an immediate summary of each patient's SBT status.

Although this validation demonstrates the functional accuracy of the prototype, it is important to acknowledge certain limitations. The evaluation was conducted using a simulated HL7 data feed and involved only two synthetic patient cases. Moreover, the study did not include any usability assessments with clinical practitioners. As a result, the generalisability of the findings is limited in scope. Broader evaluation efforts, including performance analysis and qualitative feedback, are presented in the subsequent chapters. The next section, Section 5.2, focuses on system performance metrics under varying operational conditions.

## 5.2 Performance Evaluation and Metrics

This section presents a systematic evaluation of the prototype system's performance using three key metrics captured via the IntelliJ Ultimate Profiler: Central Processing Unit (CPU) time, total time, and memory allocations. These metrics were selected to capture different dimensions of computational efficiency and resource utilization, which are particularly relevant in clinical decision support contexts where responsiveness and predictability are essential.

CPU time measures the cumulative time spent by the processor on application code execution, exclusive of I/O and thread scheduling. Total time, as captured by the profiler, reflects the wall-clock duration of Java-level execution from start to finish of the batch run, including thread synchronization and background processing but excluding time spent in external processes. Memory allocations track the volume of heap memory reserved during execution and provide insight into garbage collection pressure and memory footprint.

While these profiler-based metrics offer valuable insights into internal performance characteristics, they do not cover the full runtime impact of interacting with external systems such as the Isabelle proof engine. Therefore, an additional measurement of Isabelle execution time was conducted separately using Java-based timing instrumentation around the relevant invocation point.

### Definition of Evaluation Objectives

To evaluate the system's technical viability under realistic usage conditions, a targeted set of performance-oriented subobjectives (see Table 5.2) was derived from the broader thesis objectives defined in Section 1.3. These subobjectives further specify the third overarching objective, which seeks to quantitatively assess the prototype's correctness, scalability, and resource efficiency. Each subobjective corresponds to a distinct research focus, enabling a systematic investigation of the system's behavior across relevant computational dimensions.

| **Evaluation Focus** | **Performance-Oriented Subobjective** |
|---|---|
| Processing Time (CPU) | **S1**: How much processor time does each processing stage consume per run? |
| End-to-End Latency | **S2**: What is the end-to-end latency from HL7 ingestion to a verified result? |
| Memory Consumption | **S3**: How many heap bytes are allocated during theory generation and proof checking, and which methods dominate these allocations? |
| Scalability: Patient Load | **S4**: How do CPU time, total time, and allocation volume change as patient count scales up? |
| Scalability: Checklist Size | **S5**: How do these metrics respond to increasing checklist complexity (i.e., number of items)? |

Table 5.2: Performance-oriented subobjectives

These subobjectives serve to identify potential performance bottlenecks, latency constraints, and memory limitations that could affect the system's integration into routine clinical workflows. By doing so, they contribute directly to validating the feasibility of symbolic reasoning in a production-grade, safety-critical environment.

All measurements were conducted across nine configurations outlined in Table 5.3:

| **ID** | **Patients** | **Checklist Items (SBT)** |
|---|---|---|
| A1 | 6 | 2 |
| A2 | 6 | 6 |
| A3 | 6 | 10 |
| B1 | 18 | 2 |
| B2 | 18 | 6 |
| B3 | 18 | 10 |
| C1 | 30 | 2 |
| C2 | 30 | 6 |
| C3 | 30 | 10 |

Table 5.3: Measurement configurations

Each configuration represents a distinct profiler run that simulates typical clinical workloads by varying patient volume and checklist length. These configurations help isolate scaling effects across both axes and provide a comprehensive empirical foundation for the subsequent analyses.

### System Configuration and Profiling Setup

Experiments were executed on a Windows 11 Home workstation with the following specifications:

- **System Model:** Gigabyte B550M DS3H
- **Processor:** AMD Ryzen (AMD64 Family 25 Model 33 Stepping 0), 8 cores @ 3.7Gigahertz (GHz)
- **Installed RAM:** 32GB (17GB available during profiling)
- **Operating System:** Microsoft Windows 11 Home, Version 10.0.26100 Build 26100
- **Java Version:** Java23
- **Virtual Memory:** 34.8GB max, 16.4GB available during runtime
- **Isabelle Version:** 2024

The following processing stages were included: HL7 patient retrieval, saving to database, theory file generation, Isabelle proof startup via `isabelle build`, and updating the SBT-Readiness. The *`ServiceStarter.main()`* method, responsible for bootstrapping, was not examinated in detail. Each configuration was measured as an isolated run with a fresh patient database.

Profiling was conducted using IntelliJ IDEA Ultimate with async-profiler 3.0 (`event=wall,interval=10Milliseconds (ms),jfrsync=profile`). Here, `event=wall` configures the profiler to sample based on wall-clock time, capturing both active computation and periods of waiting or blocking. Although all samples are collected at regular elapsed-time intervals (every 10ms, as set by `interval=10ms`), IntelliJ's profiler interface allows post-hoc filtering and visualization by CPU time or total time. This enables both processor-centric and end-to-end perspectives to be derived from the same profiling session. The `jfrsync=profile` option ensures alignment between async-profiler output and Java Flight Recorder events, improving temporal accuracy and traceability. Methods with a contribution of less than 1% were excluded to focus on significant execution paths. The evaluation is structured around a $3 \times 3$ factorial design: three patient counts (6, 18, 30) combined with three checklist sizes (2, 6, 10 questions), yielding nine distinct configurations. These values were selected to reflect clinically plausible ranges for both patient throughput and checklist complexity. The patient counts represent realistic ICU loads across low, moderate, and high usage scenarios, while the checklist sizes correspond to typical variations observed between simplified, standard, and extended safety screens. Each configuration simulates the typical workload a production system would handle when executing the prototype's batch-processing loop. Although only a single run per configuration was performed, this reflects a realistic clinical workload of approximately 8 to 10 batches per day.

### Results: Execution Time (S1 and S2)

As described before CPU time in this analysis reflects the proportion of sampled execution intervals during which the application performed active computations. It excludes periods of thread idling, blocking, or I/O latency. Figure 5.1 visualizes this metric as a percentage for each of the nine configurations listed in Table 5.3. Active-CPU time ranged from 46% to 59%. The lowest value (46.7%) occurred at the largest batch (Configuration C2), while the highest (59.3%) arose for the smallest batch (Configuration A1).

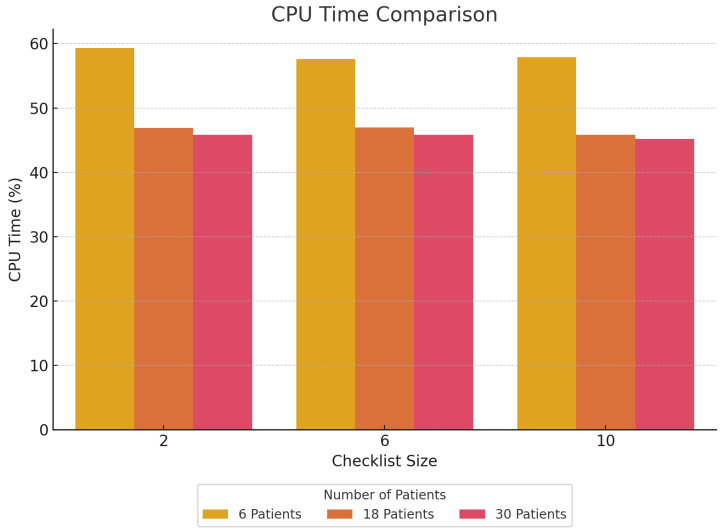


Figure 5.1: CPU Time Comparison

Across configurations, the distribution of CPU activity was shared among several key components. Significant contributors included `MDCRunnable.run()`, `HL7ClientProcessor.process()`, and Isabelle-related routines such as `startIsabelleProcess()` and `SessionProcessingService.processTheoryFiles()`. The relative influence of these components varied slightly depending on the workload, but no single method consistently dominated CPU time across all configurations. This distribution highlights a balanced architecture, in which workload-dependent paths share the computational burden.

In conclusion, `S1` is answered by the distribution shown in Figure 5.1, which demonstrates that CPU time is shared across multiple application components with consistent behavior under scaling.

Figure 5.2 presents the corresponding total time in ms for each configuration. This metric captures the wall-clock duration required to process an entire batch, from the ingestion of HL7 messages to the final update of readiness status. As expected, total time increased approximately linearly with the number of patients. For instance, configuration A2 required 347ms, while configuration C2 required 1655ms. Checklist size had a smaller, though visible, impact. At 30 patients, the increase from C1 to C3 led to an additional 140ms of runtime.

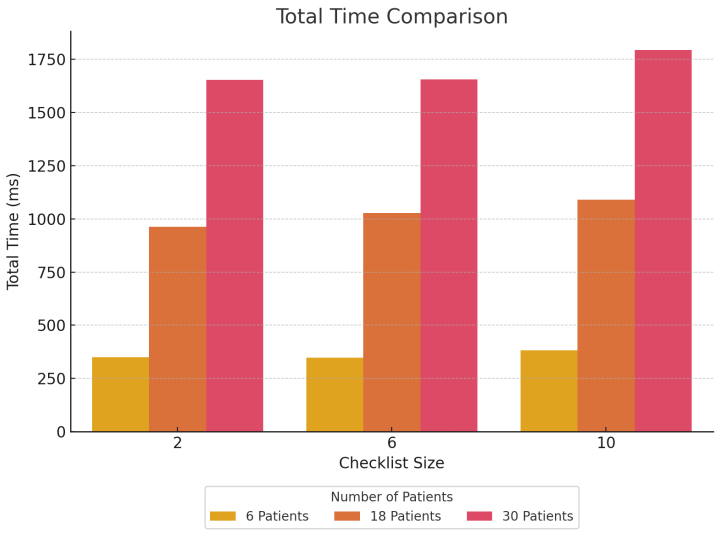


Figure 5.2: Total Time Comparison

While the IntelliJ profiler captures CPU-bound activity and heap allocation within the Java process, it does not account for time spent inside external processes such as Isabelle. This limitation is especially relevant because `isabelle build` is invoked in batch mode for each patient, and its runtime contributes significantly to total latency perceived by end users. To complement the profiling results, wall-clock execution time was measured explicitly around the Isabelle invocation using Java's *`System.nanoTime()`* instrumentation.

The following Figure 5.3 presents total Isabelle execution time across all nine configurations from Table 5.3. Each value represents the cumulative time in Seconds (s) spent inside the Isabelle process during one batch run:

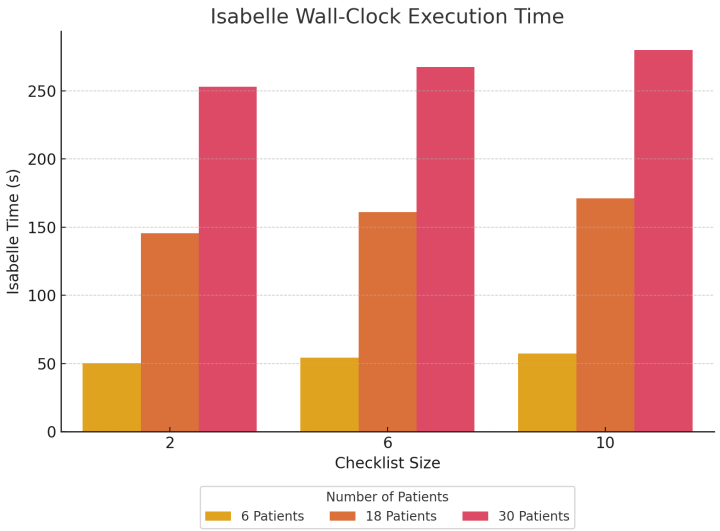


Figure 5.3: Isabelle Wall-Clock Execution Time

The results reveal a clear linear relationship between execution time and both patient count and checklist complexity. For configurations involving six patients, Isabelle execution time ranged from 50s to 57s. At 18 patients, the total duration increased to approximately 145s to 171s. In the configurations with 30 patients, total execution time further expanded, ranging from 253s to 280s.

These values provide an important complement to the IntelliJ-derived total time metrics. Whereas profiler-based total time excludes external process duration, the wall-clock measurements confirm that Isabelle invocation dominates the end-to-end latency experienced by the application. The additional insight justifies future optimization efforts around process parallelization, file preparation, and session startup overhead.

`S2` is only fully answered when combining the internal latency from the profiler (see Figure 5.2) with the external Isabelle timing (see Figure 5.3). The measurements confirm that Isabelle execution constitutes the primary bottleneck in end-to-end processing, highlighting the importance of treating subprocess duration as a central factor in performance evaluation.

### Results: Memory Allocation (S3)

Memory allocations were measured as the cumulative heap memory reserved during execution. Figure 5.4 shows these values in Megabyte (MB) across the nine configurations from Table 5.3. The y-axis indicates the peak allocation per batch.

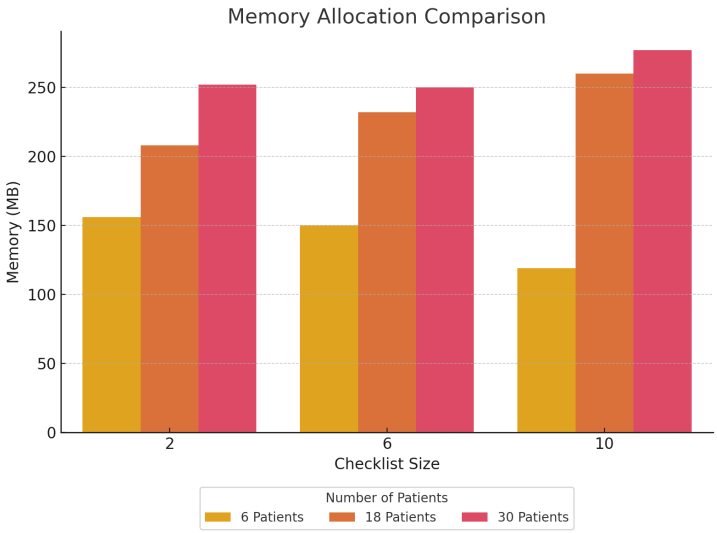


Figure 5.4: Memory Allocation Comparion

Memory usage increased consistently with patient count. For a fixed checklist size of six questions, heap allocation rose from 150MB (Configuration A2) to 250MB (Configuration C2) as patient count increased from 6 to 30. The influence of checklist size was less pronounced, although still measurable. At 18 patients, memory allocation rose from 208MB (Configuration B1) to 260MB (Configuration B3) as the number of questions increased from 2 to 10.

The most resource-intensive routines included *`HL7ClientProcessor.process()`*, checklist deserialization, and batch-level coordination. Isabelle-specific methods accounted for approximately 8% to 10% of overall memory allocations, and their contribution remained proportionally stable across all conditions.

`S3` is addressed by the data in Figure 5.4, which confirms that memory use scales linearly with patient volume and proportionally with checklist size. No configurations approached critical Java Virtual Machine (JVM) heap thresholds, and no garbage collection effects were observed.

### Results: Scalability Across Patients and Checklists (S4 and S5)

Scalability was assessed using linear regressions on the three measured metrics (CPU time, total time, memory allocation) across two scaling dimensions: patient count and checklist size. Regressions were fitted to the configuration groups A2 to C2 (scaling patient count) and B1 to B3 (scaling checklist size), as defined in Table 5.3. Each regression produced a slope and an $R^2$ value. The slope quantifies the rate of change per unit increase, while $R^2$ expresses the

strength of the linear relationship, with values closer to 1.0 indicating highly consistent behavior.

The scaling by patient count is summarised in Table 5.4. The results show a decline in CPU time per patient due to amortised fixed costs, while total time and memory usage increase linearly with patient volume.

| **Metric** | **Slope** | $\mathbf{R^2}$ | **Interpretation** |
|---|---|---|---|
| CPU time (%) | −0.45 per patient | 0.77 | Slight decline per patient due to amortization of fixed startup costs. |
| Total time (ms) | +54.5 per patient | 0.99 | Strong linear growth; latency increases predictably with batch size. |
| Memory (MB) | +2.0 per patient | 0.98 | Heap usage scales proportionally with patient volume, remaining within limits. |

Table 5.4: System performance scaling by patient count (checklist size fixed; configurations A2, B2, C2)

The scaling by checklist size is presented in Table 5.5. Here, the prototype exhibited only marginal increases in resource consumption, indicating efficiency in handling more complex logical structures.

| **Metric** | **Slope** | $\mathbf{R^2}$ | **Interpretation** |
|---|---|---|---|
| CPU time (%) | +0.16 per question | 0.81 | Negligible rise; checklist logic is light. |
| Total time (ms) | +14 per question | 0.99 | Runtime increases modestly as checklist logic expands. |
| Memory (MB) | +2.1 per question | 0.89 | Heap usage rises proportionally with checklist complexity. |

Table 5.5: System performance scaling by checklist size (patient count fixed at 18; configurations B1, B2, B3)

All regression models exhibited high $\mathbf{R^2}$ values, confirming that each metric behaves in a stable and predictable manner under both scaling dimensions. Importantly, the prototype demonstrated linear or sub-linear scaling throughout, with no evidence of performance degradation or nonlinear thresholds.

`S4` is answered through the patient scaling regressions, which confirm that increases in batch size lead to proportionally predictable increases in latency and memory usage without impairing system efficiency.

S5 is addressed through checklist scaling results, which demonstrate that additional checklist items introduce only modest overhead, validating the prototype's suitability for more complex clinical logic.

### Evaluation Limitations and Validity

While the results presented in this chapter indicate promising performance characteristics, they are subject to several potential threats to validity. These limitations are important to acknowledge in order to contextualize the findings and avoid overgeneralization.

First, internal validity may be affected by the nature of the profiling setup. The IntelliJ profiler samples method execution with a granularity of 10ms, which may overlook very short-lived method calls or introduce sampling bias. Additionally, each configuration was executed only once. Although this aligns with the intended one-run-per-configuration evaluation model, it does not account for run-to-run variability or transient system behavior. Furthermore, the process of manually copying call tree data from the IntelliJ profiler into structured tables introduces a non-negligible risk of human transcription error.

Second, external validity is constrained by the specific hardware and software environment in which the experiments were conducted. The tests were run on a workstation with a fixed CPU, RAM, and operating system, and used a specific version of the IntelliJ Ultimate Profiler. As such, performance metrics may vary on different machines, particularly in terms of total time and garbage collection behavior. The results should therefore not be interpreted as universally representative of all deployment environments.

Third, construct validity is challenged by the profiler's exclusion of subprocess timing. Although Isabelle proofs constitute a central part of the application's functionality, their execution occurs in a separate process. This necessitated an auxiliary timing mechanism, which was implemented using `System.nanoTime()` in the Java wrapper. While accurate at the ms level, this does not provide insight into internal Isabelle behavior or CPU utilization during proofs.

Taken together, these factors do not undermine the conclusions drawn from the evaluation but highlight the need for careful interpretation and future replication under broader conditions.

The performance evaluation revealed that the system operates efficiently and predictably across varying workloads. In the median configuration, representing 18 patients and a checklist of six questions, the prototype achieved a CPU usage peak of 45.8%, a memory allocation of approximately 232MB, and a total runtime of around 1028ms. The corresponding Isabelle proof phase required approximately 161s, representing the majority of user-visible latency. This measurement reinforces the importance of capturing subprocess timing to ensure a holistic understanding of performance. These values reflect a balanced distribution of compute and memory resources, without any signs of saturation or instability.

Across all configurations, CPU time remained relatively stable, even as checklist complexity increased. This indicates that core computational workloads, particularly those associated with theory generation and proof checking, scale modestly and do not incur significant additional cost as more checklist items are introduced. Total runtime, by contrast, scaled more directly with patient count, rising linearly as additional patients were processed. This pattern

aligns with expectations for a batch-oriented system where latency is primarily affected by the volume of input data.

Memory allocations followed a similarly linear trend, particularly with increasing patient numbers. The system's heap usage remained well within the configured limits of the JVM, and no evidence of garbage collection overhead or performance degradation was observed. Even at the upper bound of the test matrix, 30 patients and 10 questions, the system maintained acceptable latency and resource usage.

Overall, the findings support the conclusion that the prototype is performant and scalable for real-world use. Its execution characteristics remain within tolerable boundaries across both scaling dimensions. This confirms its readiness for further clinical integration, pending additional qualitative validation and robustness analysis, which are addressed in Section 5.3.

## 5.3 Qualitative Analysis and Clinical Implications

This section provides a structured qualitative assessment of the prototype system by evaluating its clarity, clinical relevance, and traceability. While the preceding section focused on performance metrics, the aim here is to explore how the output is perceived and interpreted by its intended users. The evaluation follows three guiding perspectives: the explainability of the system's results, its compatibility with established clinical workflows, and the extent to which it offers trust and auditability. These dimensions collectively reflect the prototype's maturity in terms of practical usability and adoption potential.

#### Explainability and User Transparency

The UI of the prototype system offers immediate interpretability of the verification outcome, as described in Section 4.3. Each patient entry is marked with a distinct flag: green indicates readiness for extubation, red signals non-readiness, and yellow signifies that the system was unable to reach a decision. This color-coded summary enables clinicians to assess a patient's status at a glance. In the detailed patient view, each verification run is documented, showing the outcomes of all individual checks. These results are coupled with access to the corresponding Isabelle proof, which can be launched directly from the interface in interactive mode.

The terminology used throughout the interface is deliberately aligned with the vocabulary employed at Hospital Bamberg. The checklist design similarly avoids technical jargon, favoring easily understood language.

This layered approach to explanation allows for flexibility in user interaction. Clinicians may rely solely on the summary verdict for routine assessments or delve into the formal logic if a deeper review is warranted. The presence of detailed proof links provides a robust transparency mechanism, offering traceability without requiring deep specialized knowledge for every user.

While most interface elements are accessible, some ambiguities persist regarding certain verification terms and proof representations. These can be resolved with minimal effort by integrating a contextual glossary and basic on-

boarding material. Such additions would further reduce the learning curve and facilitate broader clinical adoption.

### Clinical Workflow Compatibility and Usability

The prototype has been designed to support the clinical decisions involved in assessing readiness for extubation, as detailed in Subsection 2.2.1. Its operation follows the data path established in Section 4.2, beginning with real-time parameter acquisition from monitors, followed by HL7-based ingestion into the PDMS, subsequent processing through the verifier, and checklist-based formalization. This automated flow reflects the typical rhythm of intensive care decision-making and adheres to familiar clinical practices.

As mentioned in Section 4.2, the Isabelle proofs are generated automatically in the background throughout the day. This means that by the time clinicians start their morning routines or prepare for patient rounds, all results are already available in the system. Because the verification runs are performed in advance, the prototype supports clinical decision-making without causing delays or interfering with established workflows. This asynchronous design ensures that assessments can be made quickly, using information that is both up to date and formally validated.

The system also alleviates cognitive load by automating the evaluation of complex criteria. For instance, the system handles the $PaO_2$ to $FiO_2$ ratio calculations that would otherwise require manual derivation. By delegating this logic to the formal verifier, clinicians are freed from performing repetitive computations, thereby reducing error potential and mental fatigue.

A typical user journey may involve a clinician accessing the patient tab, identifying Patient 1001, and observing a red flag. Upon opening the detailed view, the user sees that one criterion, $PaO_2/FiO_2$ ratio below the acceptable threshold, has failed. The clinician may then explore the associated rationale and, if desired, inspect the formal proof either through the user interface or by reviewing the `session.log`, which records the outcome of the Isabelle verification. This scenario illustrates how the prototype offers both a high-level recommendation and multiple paths for comprehensive verification.

Overall, the prototype fits well within the ICU's operational tempo. While the current version provides core functionality, its workflow integration would benefit from enhancements such as push notifications for verdict changes and clearer timestamp annotations.

### Trust, Auditability, and Future Readiness

To ensure transparency and trustworthiness, the system records each processing step in distinct artifacts. These include the communication.log, which captures incoming HL7 messages. The `run.log`, which documents the checklist verification pipeline. The `session.log`, which logs Isabelle's proof outcomes. And lastly the `.thy` file, which contains the full formal proof and can be viewed via a button in the UI.

The integrity of these artifacts is reinforced by the inclusion of timestamps. This design permits retrospective auditing and deters tampering, supporting clinical traceability and accountability.

Currently, the system relies on manual checks to validate input data quality. While this step is sufficient for prototype evaluation, future versions should incorporate automated test suites and input validation procedures to detect and resolve inconsistencies proactively. This enhancement is outlined as a priority in Chapter 6.

In its current state, the system provides a reliable and auditable chain of reasoning for each extubation recommendation. The combination of human-readable summaries, formal proof links, and complete log trails enables clinical staff and supervisors to assess both the outcome and the reasoning process. Before real-world deployment, however, the introduction of automated validation tools and regulatory-compliant documentation will be essential to establish full clinical and institutional trust.

The prototype demonstrates that qualitative properties such as explainability, clinical compatibility, and auditability can be achieved through symbolic verification. Its output is understandable, actionable, and traceable. However, the absence of automated data validation and refined UI features limits its readiness for deployment. These shortcomings are addressed in Chapter 6, which outlines the remaining limitations and the roadmap for clinical evaluation and regulatory alignment.

While the preceding section assessed the prototype's clinical usability and trust-enabling features in isolation, the following Section 5.4 situates these findings within the broader landscape of existing decision support systems, offering a structured comparison across symbolic, subsymbolic, and hybrid paradigms.

## 5.4 Comparison with Existing Approaches

This section evaluates the prototype developed in this thesis relative to prior systems for supporting extubation decisions and SBTs. It critically contrasts symbolic, subsymbolic, and hybrid approaches based on four central criteria: explainability, verifiability, clinical integration, and adaptability. The structure mirrors previous chapters and avoids repeating taxonomies introduced in Subsection 2.1.5.

#### Explainability and Transparency of Reasoning

The Isabelle-based system developed in this thesis provides an explicitly structured decision process, rooted in formalized clinical logic. Each criterion in the SBT checklist is encoded as a logical predicate, with outcomes derived through stepwise verification. The generated output, consisting of a flag verdict, satisfied/unsatisfied conditions, and a link to the full proof, offers immediate interpretability to users (see Section 5.3).

By contrast, machine learning approaches such as those by Chen et al. (2019) and Igarashi et al. (2022) rely on high-dimensional feature spaces and non-transparent decision mechanisms. Although these models offer high predictive performance, they require supplementary tools such as SHAP to approximate their inner workings. These approximations do not equate to formal logic and are not embedded in the decision pipeline.

Hybrid systems like XCSR by Huang et al. (2022) and TrustKG by Vidal et al. (2025) attempt to balance explainability with data-driven insights. XCSR,

in particular, produces human-readable rules that can resemble symbolic logic. However, these rules are derived through stochastic processes, and their consistency across patient populations is not guaranteed. In contrast, the system presented in this thesis ensures deterministic logic paths for every patient evaluation.

### Verifiability and Formal Assurance

The system introduced here is formally verifiable by construction. Isabelle/HOL proofs provide machine-checked guarantees that the checklist logic has been correctly applied to each patient's data. This capability positions the system within the domain of certifiable medical software, an area often highlighted but rarely achieved by subsymbolic or hybrid systems.

Legacy symbolic systems such as GANESH by Dojat et al. (1992) and the dynamic EHR-CDST by Cucchi et al. (2024) also employ deterministic rule logic but lack formal proof integration. Their rule correctness must be asserted manually or validated through retrospective studies. In contrast, the Isabelle approach enables continuous, per-instance formal validation, enhancing both regulatory readiness and clinical confidence.

Freitas et al. (2020) underscore the importance of theorem proving in medical contexts, noting that verification tasks should be embedded throughout the development lifecycle. The system developed in this thesis operationalizes that philosophy by embedding formal logic not only at design-time but also at runtime.

### Clinical Integration and Usability

Workflow integration is central to clinical acceptance. The present system offers a patient-centric UI where users can review checklist outcomes, explore proof steps, and verify individual conditions. The use of traffic-light visual indicators facilitates rapid interpretation while retaining a path to full technical traceability.

By comparison, symbolic systems like that of Cucchi et al. (2024) are fully embedded within EHRs and offer high clinical usability but operate only at the level of metric flags, without deeper proof or logic access. Subsymbolic models often remain external to clinical interfaces, with integration challenges stemming from model opacity and clinician distrust.

Hybrid frameworks such as TrustKG propose improved communication mechanisms through tools like InterpretME and SHACL validation (Vidal et al., 2025). While promising, these systems remain largely conceptual and are not yet deployed in ICU workflows.

### Adaptability to Evolving Data and Guidelines

A frequently cited limitation of symbolic systems lies in their perceived inflexibility when adapting to new clinical knowledge or evolving data patterns. Traditional rule-based architectures, such as GANESH by Dojat et al. (1992) or the EHR-embedded CDST by Cucchi et al. (2024), implement static decision logic. Updates to their rules require expert intervention, manual encoding, and system-wide revalidation.

The system developed in this thesis overcomes this rigidity through a user-configurable design. Rather than being bound to a single, hardcoded checklist, it enables clinicians or developers to dynamically compile custom checklists based on current institutional guidelines or clinical requirements. These configurations are then automatically translated into Isabelle/HOL theory files through a fully automated code generation pipeline. No manual rewriting of logical rules is necessary. The formal verification pipeline processes these inputs into machine-checkable proof artifacts. This architecture introduces a novel form of flexibility within symbolic AI that has not been realized in previous systems.

Nevertheless, this adaptability operates within the constraints of a predefined logical framework. While users may vary checklist composition, thresholds, and structural patterns, the symbolic backbone does not autonomously evolve in response to empirical data. In contrast, subsymbolic models, such as those developed by Chen et al. (2019) and Igarashi et al. (2022), continuously learn from large and heterogeneous patient datasets. These models adapt their internal representations to reflect emergent correlations or clinical drift, but do so at the expense of transparency and reproducibility. Their decisions are shaped by model weights and latent structures that defy direct audit or verification.

Hybrid models attempt to combine these complementary strengths. For example, the XCSR system by Huang et al. (2022) evolves interpretable rules via genetic reinforcement, and TrustKG by Vidal et al. (2025) employs constraint-aware knowledge graphs to manage knowledge evolution. These systems propose frameworks in which symbolic structures guide reasoning while subsymbolic modules adapt to empirical variance. While not yet fully implemented in ICU environments, they present promising templates for augmenting systems like the one developed here.

While the symbolic system presented in this thesis does not engage in autonomous learning, it introduces an important and underexplored mode of configurability: structured symbolic adaptability through user-defined, formally verifiable logic generation. Future extensions may integrate empirical feedback mechanisms or machine-assisted rule evolution, without compromising the core guarantees of explainability and correctness-by-construction. This vision is further elaborated in Section 6.2.

**Summary Comparison of Paradigm Characteristics**

The system developed in this thesis occupies a unique position. It is rooted in symbolic AI, yet introduces a notable form of flexibility by allowing users to construct and compile custom clinical checklists. These are automatically translated into formally verifiable Isabelle/HOL theories, which distinguishes the system from earlier symbolic tools that rely on fixed, manually hardcoded rules.

In contrast, subsymbolic systems provide high adaptability and are designed to discover patterns in large datasets, but their decision-making processes are opaque and difficult to audit or certify. Hybrid systems attempt to integrate both paradigms, though most existing examples remain at the conceptual or research prototype stage and lack deployment in safety-critical workflows.

To facilitate direct comparison, Table 5.7 summarizes how each paradigm aligns with four evaluation dimensions. The qualitative ratings are based on the criteria described below (Table 5.6):

| **Criterion** | **Level** | **Label** | **Interpretation** |
|---|---|---|---|
| Explainability | High | Transparent | Decision process is directly interpretable, traceable, and deterministic (e.g., symbolic rules, formal proof artifacts). |
| Explainability | Medium | Partial Insight | Some form of explanation is available, but lacks formal structure or full traceability. |
| Explainability | Low | Opaque | Decision logic is opaque or inaccessible; explanations are approximate or absent. |
| Verifiability | High | Machine-Checked | System provides machine-checkable guarantees (e.g., Isabelle/HOL proofs). |
| Verifiability | Medium | Expert-Curated | Relies on expert review or internal consistency, but not formally verified. |
| Verifiability | Low | Unverified | No structured verification; correctness is inferred from statistical performance. |
| Clinical Fit | High | Integrated | Fully embedded into clinical workflows and used in live environments. |
| Clinical Fit | Medium | Usable Prototype | Usable UI or integration exists, but not yet widely deployed. |
| Clinical Fit | Low | Conceptual | Remains at a conceptual, algorithmic, or research-only stage. |
| Adaptability | High | Self-Updating | Continuously learns or evolves with new data; supports real-time model updates. |
| Adaptability | Medium | Dynamically Configurable | Allows dynamic configuration or rule selection within a symbolic framework. |
| Adaptability | Low | Static | Fixed logic; requires manual modification to incorporate changes. |

Table 5.6: Qualitative scale definitions for key evaluation criteria

| **Criterion** | **This Thesis (Symbolic, Dynamic)** | **Symbolic (Dojat et al., 1992; Cucchi et al., 2024)** | **Subsymbolic (Chen et al., 2019; Igarashi et al., 2022)** | **Hybrid (Huang et al., 2022; Vidal et al., 2025; Fenske et al., 2024)** |
|---|---|---|---|---|
| **Explainability** | **High**: fully traceable checklist logic, formal proof output, user-visible proof artifacts | **High**: explicit static rules, human-readable, but without integrated proof output | **Low-Medium**: opaque model logic, partial explanation via SHAP or feature importance tools | **Medium–High**: rules or knowledge graphs offer partial traceability and intuitive structure |
| **Verifiability** | **High**: automatically generated Isabelle/HOL theories with machine-checked correctness | **Medium**: expert-curated rules; no formal guarantees; validated through clinical outcomes | **Low**: statistical validation only; no formal model correctness | **Medium**: partial constraint checking (e.g., SHACL); limited formal validation |
| **Clinical Fit** | **Low–Medium**: functioning prototype UI, no real-world deployment or clinical evaluation | **High**: embedded in live clinical systems, e.g., GANESH controlling ventilators, CDST in EHR | **Low–Medium**: clinical relevance shown in retrospective datasets; limited workflow integration | **Medium**: conceptual or pilot-stage integration; usability promising but not yet realized |
| **Adaptability** | **Medium**: user-defined checklists dynamically compiled to symbolic logic without needing rule re-coding | **Low**: fixed logic; updates require manual modification of rules and system revalidation | **High**: models adapt to new data patterns automatically; support ongoing training | **Medium–High**: hybrid rule evolution (XCSR), constraint-aware learning (TrustKG), symbolic embeddings |

Table 5.7: Comparative positioning of decision support paradigms

This structured comparison highlights the distinctive profile of the system developed in this thesis. It demonstrates how symbolic AI, when enhanced with a user-configurable checklist compiler and a formal verification backend, can offer a compelling alternative to black-box models in safety-critical domains. Future work may further enhance adaptability by incorporating machine learning-based rule suggestion or knowledge drift detection, provided these can be reconciled with the system's formal integrity. These possibilities are discussed in Section 6.2.

The prototype presented in this thesis contributes a unique instance of verifiable, explainable, and semi-integrated decision support in the domain of ICU weaning. While symbolic AI prioritizes safety and transparency, its limited adaptability must be acknowledged. In contrast, the flexibility of subsymbolic and hybrid approaches makes them promising for augmentation, not replacement.

The challenge ahead lies in merging the best elements of both worlds: retaining the assurance of Isabelle/HOL while embedding hooks for data-driven insight. This hybridization, if properly constrained and validated, could yield ICU systems that are both intelligent and trustworthy.

Having positioned the prototype within the broader landscape of symbolic, subsymbolic, and hybrid systems, in Section 5.5 the thesis now consolidates the empirical and analytical results of Chapter 5 to assess how effectively the proposed solution fulfils its technical, clinical, and methodological objectives.

## 5.5 Summary of Findings

This section synthesizes the empirical and analytical results presented throughout Chapter 5 to evaluate the viability of the proposed system. The findings are reviewed across multiple dimensions, technical fidelity, performance efficiency, clinical interpretability, and methodological alignment. Together, they offer a structured assessment of how effectively the prototype fulfills the thesis objectives and methodological principles, while laying the groundwork for the discussion of limitations and future directions in Chapter 6.

### Consolidated Empirical Findings

The evaluation framework in Chapter 5 was deliberately multifaceted, comprising four complementary axes to establish a robust evidentiary foundation. Each axis contributes to a comprehensive understanding of system performance and its potential applicability in critical care environments:

- **Use-case fidelity.** Section 5.1 demonstrated that the prototype formalization represents each SBT Safety-Screen criterion with exact semantic equivalence. The two synthetic test patients produced diametrically opposed proof outcomes: every lemma failed for Patient 1001 because the $PaO_2/FiO_2$ ratio was below 150mmHg, the fraction of inspired oxygen exceeded 50%, and the PEEP surpassed 10cm$H_2O$. All lemmas succeeded for Patient 1002, whose parameters lay inside guideline thresholds. The resulting `READY` or `NOT_READY` classifications mirror standard bedside decisions, confirming that failed proofs are clinically coherent—that is, each logical contradiction corresponds to a physiologically unsafe condition that would indeed postpone an SBT in practice.

- **Quantitative performance.** Nine workload configurations (three patient cohorts × three checklist sizes) were evaluated in Section 5.2. Peak active-CPU utilization reached 59%, JVM heap usage stayed below 280MB, and the Java-level end-to-end latency never exceeded 1.8s (≈ 1793ms), even for the largest batch of 30 patients and 10 questions. Separately-timed Isabelle runs scaled almost linearly, from ≈51s to 57s at six patients through ≈145s to 171s at 18 patients to ≈253s to 280s at 30 patients—confirming that proof construction remains the dominant contributor to user-visible delay.

- **Clinical interpretability.** The qualitative study in Section 5.3 showed that clinicians can trace every failed predicate from the colour-coded overview to the exact theory line in the `.thy` file, consult the associated physiological value, and inspect the full Isabelle proof trail if desired. Structured logs (`communication.log`, `run.log`, `session.log`) facilitate retrospective audits and strengthen institutional accountability.

- **Research positioning.** The comparative analysis in Section 5.4 established that no extant system combines higher-order formal proofs with HL7 data ingestion and user-configurable checklist logic. Subsymbolic models offer superior predictive accuracy but lack verifiability, whereas legacy rule engines provide transparency without machine-checked correctness. The present work occupies the intersection by delivering both auditable proofs and configurable workflow integration.

Together, these strands confirm that the artefact is technically sound, clinically plausible, and methodologically novel.

### Fulfilment of Thesis Objectives

Table 5.8 consolidates the alignment between the thesis objectives defined in Section 1.3 and the extended evidence base generated in the evaluation sections. For clarity and transparency, each objective is explicitly linked to supporting findings and assessed based on fulfilment.

| **Objective** | **Status** | **Label** | **Evidence Base** |
|---|---|---|---|
| Formalisation | Achieved | Semantically Sound | All checklist items can be encoded as higher-order predicates with one-to-one mappings to their textual counterparts. In the test scenario every successful proof aligned with a clinically acceptable parameter constellation, and every failed proof corresponded to at least one guideline violation, thereby demonstrating **semantic completeness** as well as **clinically coherent failures**. |
| Engineering | Largely Achieved | Layered, Secure | A multilayer architecture in Java orchestrates HL7 ingestion, theory generation, proof invocation, and user-interface rendering. Profiling shows linear resource growth across patient counts and checklist sizes, while encryption and immutable log storage fulfil basic security requirements. Real-time HL7 streaming remains simulated. |
| Evaluation | Achieved | Dual Evidence | Quantitative metrics cover CPU time, wall-clock latency, memory allocation, and proof duration for nine configurations; qualitative feedback assesses usability, explainability, and auditability. This dual evidence confirms both computational viability and clinical relevance. |
| Ethical–Regulatory | Partially Achieved | Basic Compliance | The prototype employs AES-encrypted storage, and retains tamper-evident logs, thereby supporting basic GDPR compliance. However, several additional requirements defined by GDPR and KRITIS remain unaddressed, and formal certifications such as a DPIA or B3S audit lie outside the scope of this thesis. |
| Methodological | Achieved | Template-Based Generalisability | The checklist-to-theory pipeline is template-driven and parameter-agnostic, allowing rapid adaptation to additional ICU protocols. Section 3.5 describes guidelines for reusing logical components across heterogeneous decision paths. |

Table 5.8: Fulfilment of the research objectives in terms of design, implementation, and evaluation

### Alignment with Methodological Goals

In addition to the fulfilment of concrete objectives, the system was conceived under broader methodological aspirations rooted in the design-science paradigm discuessed in Section 3.1. The following analysis evaluates how well these guiding principles were realised in practice:

- **Correctness-by-construction.** Central to the system's credibility is the guarantee that only formally verified outputs reach clinical visibility. This is realized through a transformation pipeline in which Isabelle theory files are automatically generated from checklist logic and submitted to machine-checked proof before any readiness classification is displayed. This mechanism ensures that extubation flags are not inferred from unchecked or manually written logic, but rather emerge from semantically verified conditions.

- **Elimination of interpretative ambiguity.** Clinical guidelines are often subject to differing interpretations due to linguistic vagueness or inconsistent documentation. By translating each checklist item into a uniquely defined higher-order predicate, the system removes this ambiguity. Clinicians reviewing the system's decisions can directly inspect the corresponding logical structures, ensuring clarity in both meaning and implementation. This transparency serves as a counterpoint to the opacity typical of subsymbolic models.

- **Traceability.** The system architecture incorporates comprehensive logging mechanisms that capture HL7 ingestion, theory generation, proof outcomes, and system-level messages. These logs, together with stored Isabelle proof artifacts, allow for full retrospective audits. This level of traceability strengthens institutional accountability and offers clinicians and evaluators a complete epistemic trail from input data to decision outcome.

- **Modularity and reusability.** The system exhibits strong modularity through its layered design: HL7 message processing, logic construction, theorem proving, and presentation are implemented as separable components. This allows not only easier maintenance and debugging but also adaptation to future clinical use cases beyond the SBT safety screen. Section 3.5 outlines how the underlying logic templates can be reused across different protocols, reinforcing methodological generalisability.

Taken together, these elements confirm that the artifact is not only technically and clinically effective, but also methodologically consistent with the rigorous demands of design science. The deliberate alignment between high-level methodological aims and implementation details reflects the project's dual commitment to conceptual soundness and applied relevance.

While the findings substantiate the conceptual and technical premises of this work, several limitations constrain immediate clinical deployment. Dependence on synthetic data, sequential proof execution, and the absence of bedside usability trials each introduce risks that must be addressed. Chapter 6 will examine these constraints, propose architectural optimizations such as parallel

proof workers and live HL7 integration, and outline a roadmap toward regulatory certification and prospective clinical evaluation.

# Chapter 6

# Discussion and Limitations

While the preceding chapter established the empirical viability and methodological strengths of the developed system, a critical analysis of its constraints remains essential. This chapter reflects on the broader implications of the prototype's design, implementation, and evaluation, focusing on the practical, technical, and institutional boundaries that shape its current applicability. The aim is not only to identify limitations but also to outline viable strategies for addressing them in future iterations. The discussion is structured around three dimensions: the challenges of clinical integration and data access (Section 6.1), architectural and diagnostic limitations within the technical implementation (Section 6.2), and the regulatory, ethical, and organizational factors that influence adoption in real-world healthcare environments (Section 6.3).

These reflections are aligned with the thesis's overarching objectives outlined in Section 1.3. In particular, they interrogate the extent to which the formalization and engineering objectives have been met under realistic conditions, evaluate the barriers to realizing the ethical-regulatory objective in clinical settings, and assess the foundational strengths and current limitations of the proposed methodology for broader reuse. This chapter also reinforces the design-science methodology adopted in Section 3.1, which emphasizes the iterative development and critical assessment of domain-specific artifacts as a route to advancing knowledge in applied AI and healthcare IT.

## 6.1 Clinical Integration and Data Challenges

The implementation and evaluation of the proposed system were conducted in a simulated environment due to restricted access to live hospital infrastructure. While this approach facilitated controlled experimentation and reproducibility, it also imposed limitations on the system's clinical integration potential. This section outlines the key challenges encountered in the data acquisition and clinical embedding of the prototype, and proposes corresponding mitigation strategies for future work.

#### Lack of Access to Live PDMS Infrastructure

A critical limitation was the absence of direct access to the PDMS at the clinical site. Without the ability to interface with the actual PDMS, it was not possible

to verify data flows, naming conventions, or field formats in situ. As a result, the prototype had to rely on a synthetically generated HL7 data stream, which was constructed to mirror expected message structures but could not reflect institution-specific idiosyncrasies.

This constraint introduced uncertainty regarding the semantic accuracy of parameter identifiers, units, and their hierarchical associations. For example, parameters such as $FiO_2$ or PEEP must be consistently named and correctly typed to allow valid mapping against formal checklist entries. The use of a mock HL7 server provided structural completeness but lacked the empirical fidelity required for robust clinical validation.

To overcome this limitation, future deployments should aim to secure access to anonymized production snapshots or participate in test environments managed by clinical IT departments. This would enable empirical verification of integration correctness and ensure that the system aligns with the operational semantics of real-world hospital databases.

#### Synthetic Data vs. Real-World Variability

Although the mock server was engineered to emit temporally spaced, clinically plausible HL7 messages, it inherently lacked the variability and edge-case conditions present in actual intensive care data. For instance, missing parameters, atypical value ranges, or contradictory timestamps, common in ICU environments, were not represented in the synthetic test cases. Consequently, the prototype's robustness under pathological or degraded input conditions remains only partially tested.

This discrepancy may lead to an overestimation of system resilience and decision reliability. Furthermore, performance metrics derived from the clean synthetic data stream may not reflect real-time delays or load-induced failures that arise under operational conditions.

To address this gap, a hybrid testing approach is advisable. In addition to continued use of deterministic synthetic data for regression testing, the system should be evaluated against anonymized edge-case logs or retrospective ICU datasets. Such datasets would enable stress-testing of the verification pipeline and facilitate the calibration of fault-handling mechanisms, particularly in the data transformation and logic generation stages.

#### Unassessed Clinical Usability and Stakeholder Feedback

While the technical pipeline from HL7 ingestion to formal proof execution was shown to operate end-to-end, the system's usability from a clinical and institutional perspective remains unassessed. No formal usability testing, heuristic evaluation, or cognitive walkthroughs were conducted with practicing ICUclinicians, hospital IT personnel, or data protection officers. As such, it remains uncertain whether the interface adequately supports clinical workflows, whether the explanation outputs are interpretable, or whether the system imposes undue cognitive or procedural overhead.

Given the high-stakes nature of decision-making in critical care, any symbolic reasoning tool must present its results in a manner that is both verifiable and immediately comprehensible to its end users. This applies not only to ICU physicians but also to IT staff who oversee integration and maintenance, and

to administrative stakeholders responsible for compliance audits. The current interface allows for inspection of condition logic and checklist configuration, but its effectiveness in conveying proof results and their clinical or operational implications has not been empirically validated.

Future work should therefore include structured usability studies involving diverse stakeholder groups. These should assess interface clarity, time-to-insight when reviewing proof results, perceived system reliability, and integration friction within existing workflows. Established methods such as the System Usability Scale (SUS), think-aloud protocols, and task-based walkthroughs could be used to evaluate comprehension of proof failures and the ease of locating problematic checklist items. The integration of feedback from these user groups will be essential to improving both the technical usability and institutional acceptability of the system.

While these integration-related challenges highlight the need for stronger clinical alignment and data realism, several technical limitations within the system architecture itself also warrant critical attention, as explored in Section 6.2.

## 6.2 Technical Limitations and Future Improvements

The developed system satisfies its core functional goals but also reveals several architectural and diagnostic limitations. These concern the performance overhead introduced by formal verification, the restricted traceability of failed proof attempts, the need for stricter file-level security, and the manual nature of debugging procedures. Each limitation is discussed in turn, with a focus on opportunities for improvement.

#### Performance Bottlenecks in Isabelle Execution

A major contributor to end-to-end latency is the Isabelle proof phase. Although the generation of theory files and preliminary transformations are efficiently handled by the Java subsystem, the invocation of Isabelle in batch mode introduces a significant temporal overhead. Measurements reported in Section 5.2 demonstrate that Isabelle execution accounts for the majority of total runtime, particularly as patient count and checklist complexity increase.

This bottleneck limits system scalability and responsiveness. While acceptable for a batch-processing architecture, the cumulative delay could be problematic in larger deployments or real-time monitoring scenarios. To address this, two complementary strategies are proposed. First, session parallelization is introduced, allowing proofs for multiple patients to be verified concurrently in separate Isabelle instances. Second, session startup time is reduced by reusing preloaded heap images or employing incremental build configurations. These modifications would preserve formal rigour while improving throughput.

#### Insufficient Proof Traceability and Diagnostic Feedback

Although session logs provide concise indicators of proof outcomes, including proof names, file locations, and line numbers where failures occur, they do not offer sufficient insight into the underlying causes of those failures. In the event

of an unsuccessful proof, the log typically concludes with a contradiction such as `"1. False"`, without further exposition of which intermediate predicate, condition, or parameter value triggered the failure.

For technically experienced users or developers, the `.thy` files can be opened interactively in Isabelle, where full proof traces and term states are available. This interactive mode allows detailed debugging and model inspection but requires a high degree of familiarity with Isabelle's internal logic and tooling.

From a clinical or operational perspective, however, such manual inspection is impractical. Clinicians, IT administrators, and data protection staff typically lack the expertise or bandwidth to interpret formal proof structures or error messages derived from theorem prover logs. The absence of high-level summaries or contextual diagnostics thus limits transparency for these non-technical stakeholders.

To resolve this, future implementations could include an optional diagnostic mode that extracts human-readable summaries of proof failures by cross-referencing failed goals with checklist logic and instantiated patient values. These summaries could be presented in the UI with links that allow users to trace the failure back to specific conditions or parameter ranges. For IT staff responsible for integration and compliance, this would also support faster debugging and better operational oversight.

### File Security and Access Control Limitations

The current system executes both the Java application and the Isabelle proof engine under a shared operating system user account. This configuration simplifies deployment but limits the granularity of file-level access controls. Specifically, sensitive intermediate artifacts such as `.thy` and `ROOT` files are accessible to any process operating under the same user context until they are securely deleted. While secure wiping is performed after each session, the temporal exposure of these files constitutes a potential vulnerability, particularly in regulated clinical environments.

A more robust approach would involve isolating the Isabelle execution context via a dedicated system account. This would enable the application of restrictive Access Control Listss (ACLs) that grant read access only to the verification engine, while denying access to other users or processes. Such separation would also facilitate compliance with regulatory expectations for minimal privilege and auditability. Although this model introduces additional configuration complexity, it offers a more secure foundation for clinical deployment.

### Manual Debugging and Developer Tooling

The current prototype offers only limited infrastructure for developers to efficiently diagnose and resolve proof failures. When a verification task fails, the system records a contradiction in the session log and specifies the line of failure within the corresponding `.thy` file. However, it does not provide automated mappings between this failure and the checklist condition or patient parameter responsible for the contradiction.

As a result, identifying the root cause of a failed proof requires manual inspection of Isabelle theory files. Developers must locate the relevant lemma, infer which checklist entry it corresponds to, and trace the substituted val-

ues from the input data. This process can be time-consuming, especially in checklists with derived parameters or nested logical conditions. The absence of structured diagnostic tooling increases the likelihood of misinterpretation and prolongs the resolution of integration or modeling issues.

Although this workflow is strictly internal and does not affect end-users or clinicians directly, its implications for system maintainability and reliability are significant. Debugging inefficiencies can lead to slower iteration cycles and increased difficulty in extending the system to new clinical checklists or patient models.

To improve the developer experience and ensure robust lifecycle support, future implementations should include dedicated tooling that cross-references failed lemmas with their origin in the checklist configuration and displays the associated parameter values. Such tooling would enhance traceability for developers, reduce reliance on manual file inspection, and indirectly improve the reliability and auditability of the system as a whole.

While these technical improvements enhance the system's performance and maintainability, real-world deployment also requires compliance with regulatory standards and acceptance by institutional stakeholders, as discussed in the following Section 6.3.

## 6.3 Regulatory and Adoption Considerations in Healthcare AI

Beyond technical and clinical integration challenges, the deployment of AI-based CDSSs in healthcare environments requires careful alignment with regulatory, legal, and institutional standards. The proposed prototype system introduces a novel symbolic verification pipeline, yet its full operationalization in clinical practice is constrained by current gaps in compliance, certification, and trust-building mechanisms. This section addresses key regulatory and adoption considerations, and outlines concrete steps to strengthen the system's readiness for institutional uptake.

#### Balancing Data Protection and Traceability

The system implements several privacy-preserving measures in line with European data protection requirements. These include AES-encrypted storage of patient records and the secure deletion of Isabelle `.thy` files and related session artifacts upon completion of each proof run. These mechanisms minimize data persistence and ensure that sensitive artifacts are not retained beyond the verification window.

However, this design introduces a trade-off between data protection and traceability. In particular, once `.thy` and `ROOT` files are deleted, post-hoc inspection of the underlying proof logic is no longer possible through system logs alone. This could limit retrospective validation or audit capabilities in contexts where formal accountability is required.

Importantly, this limitation is partially mitigated by the system's support for interactive proof review. During the verification session and prior to deletion, authorized users can open the `.thy` files in Isabelle's interactive environment.

This enables clinicians or domain experts with sufficient access rights to inspect the proof steps, explore the logical structure of the checklist, and interpret the reason for a failed verification. While this access is transient, it provides a valuable temporal window for transparency and review.

To further enhance auditability without compromising patient confidentiality, future versions of the system could support configurable artifact retention. Selected proof sessions could be archived in encrypted form with strict access controls, allowing for institutional audits while still satisfying data minimization principles. Alternatively, anonymized proof logs or structured proof summaries could be generated for long-term retention, supporting downstream compliance without exposing sensitive clinical data.

### GDPR, KRITIS, and Sectoral Compliance Obligations

Although the prototype is designed with privacy-by-design principles in mind, it does not yet fully conform to all applicable regulatory frameworks. In the European context, systems handling patient data must demonstrate compliance with the GDPR, as well as sector-specific requirements under the KRITIS regulation and the B3S catalogue for critical infrastructure. These frameworks impose stringent obligations on data processing, access control, breach mitigation, and documentation.

At present, the system does not include a documented DPIA, nor has it been evaluated against a formal ISMS. As such, it cannot yet be considered deployment-ready in regulated healthcare environments.

To address these gaps, future development phases should incorporate structured compliance planning. This includes performing a DPIA in collaboration not only with data protection officers but also with legal experts and institutional stakeholders involved in compliance governance. Additionally, clear data governance roles should be defined, and technical safeguards must be aligned with healthcare standards. Adherence to these frameworks is not only a legal prerequisite but also a foundation for institutional trust.

### Institutional Trust and Adoption Challenges

Beyond regulatory compliance, the adoption of AI-driven CDSSs also hinges on institutional acceptance by clinicians, administrators, and IT departments. While symbolic verification provides strong guarantees of correctness and transparency, its practical value must be demonstrable to non-technical stakeholders.

At present, the system has not been subjected to real-world clinical evaluation, and no structured onboarding materials or training interfaces are in place. Consequently, it is unclear how clinicians will perceive the system's interpretability, reliability, or integration burden. As with many AI systems, trust is not earned solely through formal correctness but through repeated exposure, ease of use, and alignment with clinical priorities.

To promote adoption, pilot deployments in controlled ICU settings should be accompanied by embedded feedback loops, training sessions, and clear documentation of the system's rationale and limitations. Transparent presentation of verification logic, combined with clinician-tailored explanations of proof outcomes, will be essential for fostering trust and facilitating sustained use. In parallel, onboarding materials should be developed for other key stakeholders,

including hospital IT personnel responsible for system integration, and data protection officers tasked with evaluating compliance and data flows.

#### Medical Certification and Legal Accountability

Finally, the current prototype has not been subjected to any formal certification pathway under the European Medical Device Regulation (MDR) or related standards such as International Electrotechnical Commission (IEC) 62304 for software lifecycle processes. Without such certification, the system cannot be considered a validated clinical device and remains ineligible for operational deployment within regulated healthcare systems.

Achieving certification requires extensive documentation, traceable development workflows, rigorous validation procedures, and alignment with recognized risk management frameworks. While this thesis does not aim to deliver a certifiable product, it provides a foundation on which such a process could be initiated.

Future steps toward certification should include consultation with notified bodies, gap analyses against MDR requirements, and the establishment of a quality management system compliant with ISO 13485. These steps are nontrivial but essential if the symbolic verification pipeline is to transition from a research prototype to a clinically deployable tool.

With the technical, regulatory, and institutional challenges outlined, the final Chapter 7 consolidates the thesis contributions and reflects on future opportunities to advance the system's clinical applicability and methodological impact.

# Chapter 7

# Conclusion and Future Work

This final chapter concludes the thesis by consolidating its core contributions and outlining promising directions for future development. The research set out to demonstrate that symbolic AI, grounded in HOL and formal verification, can offer a transparent and clinically coherent approach to automating decision protocols in critical care. Through the design, implementation, and evaluation of a prototype system tailored to the SBT, the work has delivered both a functioning artifact and a reusable methodological framework. The sections below first summarize the main contributions before turning to potential extensions and refinements that could enhance the system's clinical, technical, and regulatory scope.

## 7.1 Summary of Contributions

While Section 5.5 provided a detailed mapping between each thesis objective and its supporting empirical evidence, the following synthesis consolidates these achievements into a holistic account of the thesis's overall contributions. It integrates findings across the design, implementation, and evaluation phases to articulate the conceptual and technical value of the developed system, with explicit reference to the five objectives defined in Section 1.3.

#### Formalisation of Clinical Logic

The first contribution concerns the formalization of clinical logic using symbolic methods. Rather than translating the entire Bamberg SBT Safety Screen, the thesis focused on successfully encoding two representative questions from it (see Section 4.5) into HOL within Isabelle/HOL. Each question was modelled as a typed predicate that preserved its clinical semantics and supported formal verification. Although the remaining checklist items were not included in the prototype, the underlying approach is generalizable and supports dynamic creation of custom checklists by end users. This design ensures correctness-by-construction and eliminates interpretative ambiguity. Proof outcomes demonstrated semantic

fidelity: valid physiological states led to successful verifications, while guideline violations were correctly captured as failed proofs.

### Engineering of a Modular System

The second contribution addresses the engineering of a modular and verifiable system architecture. A prototype was implemented in Java, orchestrating HL7-compatible data ingestion, logic generation, batch-mode proof execution, and result rendering. The system integrates all layers of the decision-support pipeline, from raw patient data to formally verified readiness assessments. Profiling data confirmed that the architecture scales linearly with patient load and checklist complexity. The design enforces traceability and supports future extensibility, even though live HL7 streaming remains simulated in the current version.

### Empirical Evaluation and Clinical Relevance

The third area of contribution lies in the empirical evaluation of system viability and clinical relevance. Quantitative profiling across nine experimental configurations demonstrated predictable behaviour in CPU usage, memory allocation, and latency. Synthetic patient scenarios yielded clinically interpretable proof outcomes. Clinician-oriented features, such as explainable proof artifacts and tamper-evident logs, further supported transparency and auditability. The system's viability was thus confirmed both in technical terms and in alignment with clinical expectations.

### Ethical and Regulatory Alignment

A fourth contribution pertains to data protection and ethical infrastructure alignment. Security features such as pseudonymisation, AES-encrypted storage, and immutable logging were integrated into the prototype to support basic GDPR compliance. While comprehensive certification, such as a DPIA or B3S audit, remains out of scope, the architectural decisions reflect privacy-by-design principles and are consistent with infrastructural requirements defined under the KRITIS framework.

### Methodological Generalisability

The fifth and final contribution is methodological generalisability. Beyond the immediate case study, the thesis established a reusable method for formalising critical-care protocols through symbolic logic. The checklist-to-theory pipeline was designed to be template-driven and parameter-agnostic, supporting rapid adaptation to new clinical contexts. The resulting framework provides a scalable foundation for verifiable, explainable clinical decision support systems applicable to diverse domains.

These achievements not only validate the feasibility of formal, transparent AI in critical care but also establish a solid foundation for broader deployment, architectural refinement, and methodological expansion, as explored in the following section.

Building on these consolidated contributions, the following Section 7.2 outlines concrete opportunities to extend the system's capabilities, address remaining limitations, and broaden its applicability in clinical and research contexts.

## 7.2 Future Directions

The contributions of this thesis lay the groundwork for a range of future developments, both in terms of system capabilities and broader research trajectories. Building on the established prototype and methodological framework, several promising directions can be identified that address current limitations, extend the system's applicability, and explore new areas of integration within clinical environments.

#### Live Deployment and Real-World Validation

While the system currently operates on simulated patient data, a natural next step is the integration with live PDMS using HL7 v2 or FHIR interfaces. This would enable automated, real-time ingestion of clinical parameters and allow for prospective validation in hospital environments. Pilot studies or shadow trials in collaboration with clinical staff could further assess usability, diagnostic alignment, and workflow compatibility.

#### UI and Clinical Usability

To transition from a technical prototype to a clinically actionable tool, the development of a user-facing interface is essential. Such a graphical interface should support role-specific access control, interactive checklist visualization, and transparent feedback mechanisms, including visual representations of proof outcomes and explanatory traces. Emphasis should be placed on minimizing cognitive load while preserving the system's explainability.

#### Expansion to Additional Clinical Protocols

The formalization pipeline developed in this thesis is intentionally designed to be parameter-agnostic and template-driven. Future work could leverage this flexibility to encode additional ICU protocols, such as sedation scales, infection risk assessments, or hemodynamic stability checks. Each extension would not only broaden the system's clinical relevance but also serve to refine and stress-test the underlying logic generation mechanisms.

#### Integration of Symbolic and Subsymbolic AI

There is growing interest in combining the strengths of symbolic and subsymbolic AI to create systems that are both transparent and adaptable. While the current implementation is purely symbolic, future iterations could benefit from incorporating data-driven components capable of handling tasks such as trend prediction, parameter forecasting, or anomaly detection. These models would complement the symbolic layer by capturing empirical patterns and edge cases that are difficult to encode manually. Integrated as intelligent preprocessing stages or adaptive inputs, they could enhance the semantic depth and responsiveness of the system without compromising its formal rigour.

This direction builds on the thesis's broader vision of explainable yet adaptive AI, as outlined in Section 3.1. It also reflects the comparative insights of Section 5.4, where hybrid approaches were examined for their potential to

balance verifiability, clinical integration, and adaptability. By embedding data-driven mechanisms within a formally verifiable architecture, future versions of the system could retain correctness-by-construction while responding more flexibly to evolving clinical data and decision patterns. In safety-critical settings such as intensive care, this hybrid paradigm holds promise for delivering decision support that is not only intelligent, but also trustworthy and aligned with real-world complexity.

### Performance Improvements and Parallelization

Although the current system demonstrates acceptable performance under test conditions, further optimizations are necessary for larger deployments. Possible improvements include parallelization of proof generation, caching and reusing intermediate theory components, and asynchronous handling of patient data streams. These optimizations would reduce latency and increase throughput, making the system more robust for clinical-scale use.

### Regulatory Compliance and Certification Pathways

To enable deployment in regulated environments, future work should address the requirements for ethical and legal compliance in greater detail. This includes conducting formal DPIAs, documenting risk mitigation strategies, and preparing the system for external audits under KRITIS, ISO 13485, or MDR frameworks. Early engagement with regulatory bodies may streamline certification efforts and align system design with evolving governance standards.

### Transferability of the Formalization Method

Finally, the formalization methodology developed in this thesis may be transferable to other domains where verifiability, traceability, and interpretability are critical. Potential applications include oncology decision protocols, surgical checklists, or even non-clinical fields such as legal compliance automation. Each new domain would require domain-specific modeling, but the core translation principles could remain intact.

These directions outline a multidimensional trajectory for extending the impact of this research. Taken together, they represent a vision for the continued development of explainable, formally verified CDSSs that are both clinically meaningful and technically robust. As clinical environments increasingly demand transparency, safety, and accountability, the integration of symbolic reasoning into healthcare AI stands not only as a technical achievement, but as a foundational step toward trustworthy and future-ready decision support.

# Bibliography


Ajay, A., Bhatt, M., Soneja, M., Ray, A., Nischal, N., Ranjan, P., and Wig, N. (2022). Effect of implementation of protocol-based weaning practices in medical Intensive Care Unit of a high burden resource-limited setting.

Albahri, A., Duhaim, A. M., Fadhel, M. A., Alnoor, A., Baqer, N. S., Alzubaidi, L., Albahri, O., Alamoodi, A., Bai, J., Salhi, A., Santamaría, J., Ouyang, C., Gupta, A., Gu, Y., and Deveci, M. (2023). A systematic review of trustworthy and explainable artificial intelligence in healthcare: Assessment of quality, bias risk, and data fusion. 96:156–191.

Amann, J., Blasimme, A., Vayena, E., Frey, D., and Madai, V. I. (2020). Explainability for artificial intelligence in healthcare: A multidisciplinary perspective. 20(1):310.

Augusto, L. M. (2021). From Symbols to Knowledge Systems: A. Newell and H. A. Simon's Contribution to Symbolic AI. 2(1):29–62.

Balser, M., Coltell, O., van Croonenborg, J., Duelli, C., van Harmelen, F., Jovell, A., Lucas, P., Marcos, M., Miksch, S., Reif, W., Rosenbrand, K., Seyfang, A., and Teije, A. (2004). Protocure: Supporting the Development of Medical Protocols through Formal Methods. 101:103–107.

Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., and Herrera, F. (2020). Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. 58:82–115.

Belle, T., Jayaprakash, N., Jennings, J., Katie, G., Husnain, S., and Uduman, A. K. (2020). 1315: TIMELY EXTUBATION IN THE MEDICAL INTENSIVE CARE UNIT: A QUALITY IMPROVEMENT INITIATIVE. 48(1).

Benzmüller, C. and Andrews, P. B. (2024). Church's type theory. In Zalta, E. N. and Nodelman, U., editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2024 edition.

Bernardeschi, C., Domenici, A., and Masci, P. (2019). Logic-Based Formalization of System Requirements for Integrated Clinical Environments. In Liò, P. and Zuliani, P., editors, *Automated Reasoning for Systems Biology and Medicine*, volume 30, pages 215–242. Springer International Publishing.

Berner, E. S. and La Lande, T. J. (2007). Overview of Clinical Decision Support Systems. In Berner, E. S., editor, *Clinical Decision Support Systems: Theory and Practice*, pages 3–22. Springer.

Blanchette, J. C., Bulwahn, L., and Nipkow, T. (2011). Automatic Proof and Disproof in Isabelle/HOL. In Tinelli, C. and Sofronie-Stokkermans, V., editors, *Frontiers of Combining Systems*, pages 12–27. Springer.

Boldo, S., Lelay, C., and Melquiond, G. (2016). Formalization of real analysis: A survey of proof assistants and libraries. 26(7):1196–1233.

Chen, T., Xu, J., Ying, H., Chen, X., Feng, R., Fang, X., Gao, H., and Wu, J. (2019). Prediction of Extubation Failure for Intensive Care Unit Patients Using Light Gradient Boosting Machine. 7:150960–150968.

Cucchi, E. W., Burzynski, J., Marshall, N., and Greenberg, B. (2024). A dynamic customized electronic health record rule based clinical decision support tool for standardized adult intensive care metrics. 7(4):ooae143.

Czeschik, C. (2022). IT-Sicherheit: Gesetz nimmt auch kleine Kliniken in die Pflicht. Healthcare in Europe.

DeepMind (2022). AI solves IMO problems at a silver-medal level.

Dojat, M., Brochard, L., Lemaire, F., and Harf, A. (1992). A knowledge-based system for assisted ventilation of patients in intensive care units. 9(4):239–250.

Dougherty, R. (2025). Protect patient privacy: The definitive guide to GDPR compliance for healthcare companies. Kiteworks Blog.

Esmaeilzadeh, P. (2020). Use of AI-based tools for healthcare purposes: A survey study from consumers' perspectives. 20(1):170.

Fenske, O., Bader, S., and Kirste, T. (2024). Neuro-Symbolic Artificial Intelligence for Patient Monitoring.

Floridi, L. (2019). Establishing the rules for building trustworthy AI. 1(6):261–262.

ForeNova Technologies (2024). KRITIS requirements vs B3S standards for healthcare providers in germany. ForeNova blog.

Freitas, L., Scott, W. E., and Degenaar, P. (2020). Medicine-by-wire: Practical considerations on formal techniques for dependable medical systems. 200:102545.

Hommersom, A., Groot, P., Balser, M., and Lucas, P. (2008). Formal Methods for Verification of Clinical Practice Guidelines. 139:63–80.

Huang, P.-H., Chen, L.-Y., Chung, W.-C., Sheu, C.-C., Hsiao, T.-C., and Tsai, J.-R. (2022). Toward Evaluating Critical Factors of Extubation Outcome with XCSR-Generated Rules. 9(11):701.

Igarashi, Y., Ogawa, K., Nishimura, K., Osawa, S., Ohwada, H., and Yokobori, S. (2022). Machine learning for predicting successful extubation in patients receiving mechanical ventilation. 9:961252.

Ilkou, E. and Koutraki, M. (2020). Symbolic Vs Sub-symbolic AI Methods: Friends or Enemies? In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 3523–3524. ACM.

iNTERFACEWARE Inc. (2025). HL7 resource: Understanding HL7 standard versions. iNTERFACEWARE website.

Kaier, K., Heister, T., Motschall, E., Hehn, P., Bluhmki, T., and Wolkewitz, M. (2019). Impact of mechanical ventilation on the daily costs of ICU care: A systematic review and meta regression. 147:e314.

Kaier, K., Heister, T., Wolff, J., and Wolkewitz, M. (2020). Mechanical ventilation and the daily cost of ICU care. 20(1):267.

Kawamoto, K. and Fial, G. D. (2016). Clinical Decision Support Systems in Healthcare.

Kelly, C. J., Karthikesalingam, A., Suleyman, M., Corrado, G., and King, D. (2019). Key challenges for delivering clinical impact with artificial intelligence. 17(1):195.

Liu, C.-F., Hung, C.-M., Ko, S.-C., Cheng, K.-C., Chao, C.-M., Sung, M.-I., Hsing, S.-C., Wang, J.-J., Chen, C.-J., Lai, C.-C., Chen, C.-M., and Chiu, C.-C. (2022). An artificial intelligence system to predict the optimal timing for mechanical ventilation weaning for intensive care unit patients: A two-stage prediction approach. 9:935366.

Lloyd, J. (2017). *Encyclopedia of Machine Learning and Data Mining*. Springer US.

Markus, A. F., Kors, J. A., and Rijnbeek, P. R. (2021). The role of explainability in creating trustworthy artificial intelligence for health care: A comprehensive survey of the terminology, design choices, and evaluation strategies. 113:103655.

Marshall, J. C., Bosco, L., Adhikari, N. K., Connolly, B., Diaz, J. V., Dorman, T., Fowler, R. A., Meyfroidt, G., Nakagawa, S., Pelosi, P., Vincent, J.-L., Vollman, K., and Zimmerman, J. (2017). What is an intensive care unit? A report of the task force of the World Federation of Societies of Intensive and Critical Care Medicine. 37:270–276.

Montani, S. and Striani, M. (2019). Artificial Intelligence in Clinical Decision Support: A Focused Literature Survey. 28(01):120–127.

Murselović, T., Berić, S., and Makovšek, A. (2024). Pitfalls of difficult extubation in the ICU; when is the right time to extubate a patient? 20(2):22–26.

of Radiology, E. S. (2017). The new EU General Data Protection Regulation: What the radiologist should know. 8(3):295–299.

Oliveira, T., Neves, J., and Novais, P. (2013). Guideline Formalization and Knowledge Representation for Clinical Decision Support. 1(2):1–11.

Oluwalade, B. (2024). Brief history of HL7 interoperability standards. Oluwalade Blog.

Pérez, B. and Porres, I. (2010). Authoring and verification of clinical guidelines: A model driven approach. 43(4):520–536.

Quintard, H., l'Her, E., Pottecher, J., Adnet, F., Constantin, J.-M., Jong, A. D., Diemunsch, P., Fesseau, R., Freynet, A., Girault, C., Guitton, C., Hamonic, Y., Maury, E., Mekontso-Dessap, A., Michel, F., Nolent, P., Perbet, S., Prat, G., Roquilly, A., Tazarourte, K., Terzi, N., Thille, A., Alves, M., Gayat, E., and Donetti, L. (2017). Intubation and extubation of the ICU patient. 36(5):327–341.

Reinhardt, K. (2023). Trust and trustworthiness in AI ethics. 3(3):735–744.

Seidl, R. (2024). Correctness through formal methods. Richard Seidl Consulting Blog.

Shapiro, S. C. (2003). Artificial intelligence (AI). In *Encyclopedia of Computer Science*, pages 89–93. John Wiley and Sons Ltd.

Sozialstiftung Bamberg (2025). Sektion interdisziplinäre intensivmedizin – klinikum bamberg.

Stivi, T., Padawer, D., Dirini, N., Nachshon, A., Batzofin, B., and Ledot, S. (2024). Using Artificial Intelligence to Predict Mechanical Ventilation Weaning Success in Patients with Respiratory Failure, Including Those with Acute Respiratory Distress Syndrome. 13:1505.

Sunarti, S., Fadzlul Rahman, F., Naufal, M., Risky, M., Febriyanto, K., and Masnina, R. (2021). Artificial intelligence in healthcare: Opportunities and risk for future. 35:S67–S70.

Sutton, R. T., Pincock, D., Baumgart, D. C., Sadowski, D. C., Fedorak, R. N., and Kroeker, K. I. (2020). An overview of clinical decision support systems: Benefits, risks, and strategies for success. 3(1):17.

Teege, G. (2024). A Gentle Introduction to Isabelle and Isabelle/HOL.

University of Gothenburg (2024). AI model finds the cancer clues at lightning speed.

Vestrucci, A., Benzmüller, C., and Evangelatos, N. (2024). Symbolic Approach to Trustworthy AI: Exploration and Healthcare Case Study.

Vidal, M.-E., Chudasama, Y., Huang, H., Purohit, D., and Torrente, M. (2025). Integrating Knowledge Graphs with Symbolic AI: The Path to Interpretable Hybrid AI Systems in Medicine. 84:100856.

Vivek, N. and Martin Lloyd, S. P. (2021). Automation: An essential component of ethical AI?

Väänänen, J. (2024). Second-order and higher-order logic. In Zalta, E. N. and Nodelman, U., editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2024 edition.

Wenzel, M. (2024). The Isabelle System Manual.

Zein, H., Baratloo, A., Negida, A., and Safari, S. (2016). Ventilator Weaning and Spontaneous Breathing Trials; an Educational Review. 4(2):65–71.

Zwerwer, L. R., Kloka, J., Van Der Pol, S., Postma, M. J., Zacharowski, K., Van Asselt, A. D. I., and Friedrichson, B. (2024). Mechanical ventilation as a major driver of COVID-19 hospitalization costs: A costing study in a German setting. 14(1):4.