TACTICAL COMBAT CASUALTY CARE (TCCC) AI DECISION SUPPORT SYSTEM

by Roman Piltiai

A Capstone Project

Presented in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Software Engineering

American University Kyiv
2026

APPROVED BY:
Prof. Viktor Putrenko, Faculty Mentor

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to those who have guided, supported, and inspired me throughout the research and development of this Capstone Project. The creation of the Tactical Combat Casualty Care (TCCC) AI Decision Support System was a complex undertaking that bridged the gap between advanced software engineering, cloud architecture, and critical medical protocols. It would not have been possible without the mentorship and encouragement of several key individuals.

First and foremost, I extend my sincere appreciation to my faculty mentor, Prof. Viktor Putrenko. His invaluable guidance, patience, and constructive feedback were instrumental in shaping the direction of this research. His support allowed me to navigate the complexities of applying Artificial Intelligence to a high-stakes domain like battlefield medicine, ensuring that the project remained both academically rigorous and practically relevant.

I am also profoundly grateful to Yoshihiro Kobayashi, PhD, Teaching Professor at the School of Computing and Augmented Intelligence (SCAI) at Arizona State University. His mentorship during my capstone work at ASU provided the foundational knowledge and architectural discipline that underpins this thesis. The academic standards and engineering principles he instilled in me have been a guiding force throughout this development process.

Finally, I would like to thank my colleagues at EPAM Systems. I am fortunate to work alongside some of the brightest minds in the software engineering industry. Their daily support, willingness to share knowledge, and adherence to the highest standards of engineering excellence have significantly influenced the technical quality of this solution. Specifically, their insights into cloud-native best practices and enterprise-scale architecture helped me refine the serverless design of the TCCC Tutor.

ABSTRACT

Tactical Combat Casualty Care (TCCC) serves as the definitive standard for battlefield trauma management, yet the application of these protocols under high-stress combat conditions remains a significant challenge. Cognitive overload and lack of immediate expert guidance can lead to critical errors in pre-hospital care. This capstone project addresses this gap by engineering an Intelligent TCCC Tutor and Decision Support System tailored for dynamic environments.

The proposed solution leverages a cloud-native, serverless architecture on Amazon Web Services (AWS) to ensure high availability, scalability, and low-latency performance. By integrating Generative AI and Large Language Models (LLMs), the system interprets complex user inputs and provides real-time, context-aware guidance aligned with the MARCH algorithm. The application was designed using modern software engineering principles, prioritizing modularity and enterprise-scale security standards.

The resulting system demonstrates how advanced cloud computing and artificial intelligence can be effectively operationalized to enhance medical training and decision-making. This project not only delivers a functional prototype for tactical education but also establishes a reference architecture for future AI-driven systems in high-stakes domains.

Keywords: Tactical Combat Casualty Care (TCCC), Artificial Intelligence, Generative AI, Decision Support Systems, Cloud Architecture, AWS, Serverless Computing

TABLE OF CONTENTS

GLOSSARY / LIST OF ABBREVIATIONS

9-Line MEDEVAC A standardized NATO message format used to request medical evacuation. It consists of nine lines of information required by rescue crews to launch a mission effectively.

API (Application Programming Interface) A set of rules and protocols that allows different software applications to communicate with each other.

AWS (Amazon Web Services) A comprehensive cloud computing platform provided by Amazon that offers reliable, scalable, and inexpensive cloud computing services.

Bedrock (Amazon Bedrock) A fully managed service from AWS that offers high-performing foundation models (FMs) via a single API.

Care Under Fire (CUF) The first phase of TCCC, rendered while the first responder and casualty are still under effective hostile fire.

CI/CD (Continuous Integration / Continuous Deployment) A method to frequent deliver apps to customers by introducing automation into the stages of app development.

Cloud-Native An approach to building and running applications that exploits the advantages of the cloud computing delivery model (scalability, resilience).

Decision Support System (DSS) A computerized information system used to support decision-making. In this thesis, it refers to the AI system guiding the medic.

DynamoDB A fully managed NoSQL database service from AWS that supports key-value and document data structures.

Embeddings (Vector Embeddings) Numerical representations of data (text) that capture semantic meaning, allowing the AI to search for concepts rather than just keywords.

Generative AI (GenAI) A type of AI capable of creating new content, including text, images, or code, by learning patterns from training data.

Hallucination A phenomenon where an LLM generates plausible-sounding but incorrect information.

IaC (Infrastructure as Code) The process of managing and provisioning computer data centers through machine-readable definition files rather than physical hardware configuration.

JSON (JavaScript Object Notation) A lightweight data-interchange format that is easy for humans to read and machines to parse; standard for API responses.

Latency The delay before a transfer of data begins. Low latency is critical for real-time medical guidance.

LLM (Large Language Model) A deep learning algorithm that can recognize, summarize, and generate text based on knowledge gained from massive datasets.

MARCH Algorithm The primary TCCC assessment protocol: Massive Hemorrhage, Airway, Respiration, Circulation, Head Injury/Hypothermia.

Microservices Architecture An architectural style where an application is a collection of loosely coupled, independently deployable services.

MVP (Minimum Viable Product) A version of a product with just enough features to be usable by early customers to provide feedback.

NLP (Natural Language Processing) A branch of AI that gives computers the ability to understand text and spoken words.

Prompt Engineering The process of structuring text to be interpreted by a Generative AI model to optimize its output.

RAG (Retrieval-Augmented Generation) An AI framework that improves LLM accuracy by grounding the model on external sources of knowledge (e.g., TCCC manuals).

Serverless Computing A cloud execution model where the cloud provider dynamically manages the allocation of machine resources (e.g., AWS Lambda).

TACEVAC (Tactical Evacuation Care) The third phase of TCCC, encompassing both Casualty Evacuation (CASEVAC) and Medical Evacuation (MEDEVAC).

Tactical Field Care (TFC) The second phase of TCCC, rendered once the responder and casualty are no longer under effective hostile fire.

TCCC (Tactical Combat Casualty Care) The standard of care for the pre-hospital battlefield management of trauma.

Vector Database A database designed to store and query vector embeddings, critical for the RAG architecture search retrieval.

Chapter 1. INTRODUCTION

1.1 Industry Context

The ongoing full-scale war in Ukraine has fundamentally altered the security landscape, creating a high-intensity environment where both military personnel and civilian populations face constant threats. In this reality, the boundary between the "frontline" and the "rear" is often blurred by missile strikes and drone attacks, placing ordinary citizens in immediate danger alongside combatants. Statistics from the current theater of operations indicate that massive hemorrhage remains the leading cause of preventable death. Consequently, the ability to provide immediate trauma care is no longer just a military skill - it has become a necessary survival skill for the entire population.

To manage trauma in these environments, Tactical Combat Casualty Care (TCCC) and its primary MARCH algorithm (Massive Hemorrhage, Airway, Respiration, Circulation, Head Injury/Hypothermia) serve as the definitive standard. Strict adherence to these protocols significantly increases survival rates. However, the volume of existing medical protocols is vast, and studying them requires significant time and training. In critical moments, people often do not need deep theoretical knowledge; they need a short, accurate answer or specific advice based on the protocol.

Parallel to these humanitarian challenges, the software engineering industry is undergoing a major shift in 2026. The focus has moved from generic Large Language Models to AI Agents. These specialized agents are designed to act as experts in specific domains, offering a technological solution to the problem of information accessibility in high-stress environments.

1.2 Problem Statement

While the MARCH protocol is proven to save lives, its effective application is hindered by three critical barriers:

1. **The "Civilian Gap":** Civilians frequently find themselves in the role of first responders for their neighbors or family members. Unlike professional medics, the general population lacks consistent access to simplified, verified trauma care knowledge, leaving them vulnerable during attacks.

2. **Cognitive Overload:** In the chaos of an emergency, recalling complex medical decision trees is difficult even for trained professionals. The stress of the situation makes it challenging to navigate extensive manuals to find the correct procedure.

3. **Lack of Immediate Guidance:** Current information sources are often static documents or general web searches, which are inefficient when seconds count. There is a lack of

accessible tools that can provide "just-in-time" advice—converting a complex medical standard into a simple, actionable instruction for a user under pressure.

1.3 Relevance of the Topic

The development of an AI-powered TCCC Decision Support System is highly relevant to both the immediate needs of Ukrainian society and the global technological landscape.

By making the MARCH protocol accessible to non-experts, this project empowers civilians to help themselves and others. It bridges the gap between professional medical doctrine and the public, potentially reducing preventable deaths in civilian sectors hit by attacks.

This project aligns with the 2026 industry trend toward Agentic AI. It moves beyond simple text generation to create a grounded, domain-specific agent capable of reasoning within strict safety boundaries. This demonstrates how software engineering can solve acute, real-world problems by leveraging modern cloud architecture and Generative AI.

As the volume of medical protocols grows, the "Tutor" aspect of the system provides a more interactive and efficient way to study, allowing users to ask questions and receive answers tailored to their specific level of understanding. The relevance of this project is underscored by the current geopolitical climate and the shift toward 'battlefield clouds' in the Ukrainian defense sector.

1.4 Research Goal

The primary goal of this research is to design, implement, and validate a cloud-native AI Decision Support System (DSS) that democratizes access to the MARCH trauma protocol.

This study aims to bridge the gap between complex medical manuals and the immediate needs of users in high-risk environments. By leveraging Agentic AI and Serverless architecture on AWS, the research seeks to create a low-latency, resilient "digital tutor." This system must be capable of interpreting natural language queries and providing accurate, verified, and context-aware guidance to both military personnel and civilians, thereby reducing cognitive load and facilitating life-saving interventions during the critical phases of pre-hospital care.

1.5 Research Tasks

To achieve the stated research goal, the project is divided into several sequential phases, beginning with a comprehensive analysis of the domain-specific requirements. This initial stage involves deconstructing the Tactical Combat Casualty Care (TCCC) guidelines and the MARCH algorithm to structure the medical knowledge base in a format suitable for machine ingestion. Once the data requirements are defined, the research focuses on designing a scalable, cloud-native serverless architecture on the Amazon Web Services (AWS) platform. This architectural phase prioritizes minimizing operational overhead while ensuring high availability, which is critical for users operating in unstable network environments.

Following the structural design, the technical implementation centers on the development of a Retrieval-Augmented Generation (RAG) pipeline. This task is essential to ground the Large Language Model in official protocols, thereby mitigating the risk of hallucinations and ensuring the medical advice remains accurate and verifiable. Concurrently, the project entails the programming of agentic behaviors that allow the system to interpret natural language queries effectively. This involves creating logic that can distinguish between a user seeking immediate emergency assistance and one using the system for educational study, adjusting the context and complexity of the response accordingly.

The final phase of the research is dedicated to the systematic validation of the system. This process involves evaluating critical performance metrics, with a specific emphasis on response latency and the accuracy of information retrieval. By rigorously testing these parameters, the study aims to confirm the viability of the solution as a real-time decision support tool and establish its readiness for deployment in dynamic, high-risk scenarios.

1.6 Scope and Limitations

The scope of this research is confined to the development of a software-based prototype that functions as an intelligent interface for the TCCC guidelines. Specifically, the system focuses on the textual interpretation of the MARCH algorithm and related trauma care protocols. The study prioritizes the architectural validation of the serverless framework and the accuracy of the retrieval mechanism within the Amazon Web Services ecosystem. While the application utilizes advanced Generative AI to formulate responses, the functional scope is limited to text-based interactions and does not currently integrate voice recognition or image analysis capabilities for wound diagnostics.

Regarding limitations, it is imperative to acknowledge that this system acts solely as a decision support tool and educational aid; it is not a certified medical device and does not replace the judgment of trained medical professionals. The reliability of the advice is intrinsically bound to the quality and currency of the ingested documentation, meaning the system cannot account for clinical nuances not explicitly detailed in the source text. Furthermore, the reliance on a cloud-native architecture introduces a dependency on network connectivity. While this design ensures scalability, it renders the current iteration of the tool dependent on internet access, which may be intermittent or unavailable in active combat zones. Consequently, this research presents a Minimum Viable Product (MVP) intended to demonstrate technical feasibility rather than a field-ready military application.

1.7 Structure of the Thesis

This thesis is organized into five distinct chapters that collectively detail the research, design, and implementation of the TCCC AI Decision Support System. The first chapter, Introduction, establishes the foundational context for the study, outlining the critical humanitarian need for

accessible trauma care guidance in the current operational environment. It defines the problem space, articulates the specific research goals, and sets the scope and limitations of the project.

Following the introduction, the Literature Review examines the current state of the art in both battlefield medicine and artificial intelligence. This chapter provides a theoretical background on the TCCC protocols and analyzes existing decision support tools used in military contexts. Furthermore, it explores recent advancements in Large Language Models and Retrieval-Augmented Generation, establishing the technical justification for using a serverless cloud architecture to solve the identified problem.

The third chapter, System Design and Methodology, translates the theoretical understanding into a concrete architectural blueprint. It details the high-level design of the solution, justifying the selection of Amazon Web Services as the hosting platform and explaining the data flow between the user interface, the AI engine, and the vector database. This section serves as the technical core of the thesis, describing how modern software engineering principles are applied to ensure system scalability and reliability.

Chapter four, Implementation and Results, documents the practical execution of the research tasks. It describes the specific configurations of the AWS services, the development of the AI prompts, and the integration of the knowledge base. This chapter also presents the validation results, offering an analysis of the system’s performance in terms of accuracy and latency to demonstrate that the research objectives have been met.

Finally, the Conclusion summarizes the key findings of the research, reflecting on the project's impact and the successful operationalization of the MARCH protocol via generative AI. This closing chapter also identifies potential avenues for future development, including the integration of offline capabilities and voice interfaces, thereby providing a roadmap for the continued evolution of the system.

Chapter 2. LITERATURE REVIEW AND RELATED WORK

2.1 Overview of TCCC Systems

Historically, the documentation of trauma care on the battlefield has relied on physical methods, primarily the DD Form 1380 (TCCC Card). This rubberized card, attached to the casualty's kit, serves as the primary record of care at the Point of Injury (POI). While durable, this manual method is prone to errors, illegibility, and loss during transit. Over the last decade, military health systems have attempted to digitize this process through mobile applications such as the Battlefield Assisted Trauma Distributed Observation Kit (BATDOK) and the ATOS system. These applications successfully digitized the standard forms, allowing medics to tap through checklists rather than write with markers. However, these solutions largely remain "static" decision trees. They digitize the paper process but do not fundamentally alter the cognitive load; the medic must still navigate complex menus and manually input data while under fire. Few existing systems have successfully integrated generative AI to actively guide the user or interpret complex, unstructured scenarios in real-time.

2.2 Cloud-Native Medical Platforms

Modern military medical infrastructure is increasingly shifting toward cloud-native architectures to handle the dynamic demands of high-intensity conflict zones. Unlike traditional on-premise servers, which are vulnerable to kinetic attacks and hardware failure, cloud-native platforms leverage the principles of "Elasticity" and "Resilience." Elasticity allows the system to automatically scale computing resources up or down based on immediate demand—crucial during mass casualty events where data throughput spikes unpredictably. Resilience is achieved through distributed availability zones; if one data center goes offline, the system automatically reroutes traffic to a healthy node without service interruption. This architecture ensures that the TCCC Decision Support System remains available to users in Ukraine even if local infrastructure is compromised.

2.3 Military Identity & Access Control

Security in modern military software has evolved beyond simple password protection to "Zero Trust" architectures. The core tenet of Zero Trust is "Never Trust, Always Verify," meaning no user or device is trusted by default, even if they are inside the network perimeter. For a medical application handling sensitive casualty data, granular access control is essential. This is typically implemented via OpenID Connect (OIDC) identity flows, which allow for federated authentication. This ensures that a combat medic, a field surgeon, and a NATO advisor can all

access the same platform but see only the data and tools appropriate for their specific security clearance and role.

2.4 Event-Driven Medical Systems

To maintain situational awareness across distributed units, modern systems are moving away from monolithic request-response architectures toward Event-Driven Architectures (EDA). Event-driven architectures allow for the synchronization of casualty data across distributed units without polling. In a traditional system, a commander's dashboard might "poll" (ask) the database every minute for updates, which wastes bandwidth and battery life—both critical resources at the tactical edge. In an event-driven model, the system pushes updates only when a significant event occurs (e.g., "Tourniquet Applied" or "Evacuation Requested"). This allows for the synchronization of casualty data across disparate units without the need for constant, resource-intensive polling, ensuring that medical evacuation teams have the most up-to-date information the moment it is generated.

2.5 AI & LLMs in Battlefield Medicine

The integration of Large Language Models (LLMs) into healthcare has shown immense promise, particularly in their ability to synthesize vast amounts of medical literature. However, a significant barrier to adoption in high-stakes environments is "hallucination," where an AI generates plausible but incorrect medical advice. Recent research highlights Retrieval-Augmented Generation (RAG) as the definitive solution to this problem. Instead of relying solely on the model's pre-trained memory, RAG intercepts the user's query, retrieves the relevant verified paragraphs from the official TCCC guidelines, and forces the AI to generate an answer based only on that retrieved evidence. This architectural pattern has been proven to drastically reduce hallucination rates, making generative AI viable for clinical decision support.

2.6 Summary of Findings and Identified Gaps

A review of the current state of the art reveals a clear trajectory: military medicine is moving from paper to digital, and from static servers to the cloud. However, a key gap remains in the user interface and data ingestion layer. While data is now digital, entering it requires manual interaction with screens, which is dangerous when a medic's hands are covered in blood or holding a weapon. There is a distinct lack of voice-activated, hands-free systems capable of parsing unstructured medic reports (e.g., "Subject has shrapnel wound to left thigh, tourniquet applied at 1400") into structured clinical data. This thesis addresses the foundational intelligence layer required to bridge this gap, proving that an AI agent can correctly interpret and structure such natural language inputs according to the MARCH protocol. A key gap exists in the lack of voice-activated, hands-free systems that can parse unstructured medic reports into structured clinical data.

Chapter 3. DOMAIN ANALYSIS AND REQUIREMENTS

3.1 Domain Description

The operational domain of this system is strictly defined by the phases of Tactical Combat Casualty Care (TCCC), a set of guidelines that dictate specific medical interventions based on the level of hostile threat. Understanding these phases is critical for the software behavior, as the user's cognitive capacity and safety profile change drastically between them.

The first phase, Care Under Fire (CUF), occurs when the medic and casualty are still under effective hostile fire. In this phase, medical intervention is extremely limited, focusing solely on tourniquet application to stop massive bleeding. The software interface for this phase must be minimal, high-contrast, and require zero complex interaction. The second phase, Tactical Field Care (TFC), begins when the threat is suppressed. This is the primary domain for the AI Assistant, as the medic has time to perform a full MARCH assessment, check airways, and treat circulation. The system must support detailed queries and step-by-step guidance here. Finally, Tactical Evacuation (TACEVAC) involves the transport of the casualty to a higher echelon of care. During this phase, the software's focus shifts from guidance to documentation, ensuring that the care provided is logged and transmitted to the receiving facility.

3.2 Stakeholders and Actors

The system architecture is designed around two primary actors and one secondary stakeholder. The Combat Medic serves as the primary human user. This actor is characterized by high stress, limited attention span, and a need for immediate, actionable information. The medic interacts with the system via voice or text to receive protocol support. The AI Assistant is the primary system actor, an autonomous agent responsible for interpreting the medic's intent, retrieving relevant medical knowledge, and formulating a response that is clinically accurate and contextually appropriate.

Unit Command represents the secondary stakeholder. While they do not interact with the TCCC Tutor directly during care, they are the downstream consumers of the data. The structured logs generated by the AI (e.g., "Tourniquet applied at 14:00") provide Command with real-time situational awareness regarding unit readiness and casualty status.

3.3 Functional Requirements

The functional requirements of the system are categorized into three distinct modules: Core Decision Support, Educational Assessment, and Identity Management.

3.3.1. Core Decision Support Module The primary function of the system is to provide real-time medical guidance. The system must perform Context-Aware Medical Querying, capable

of interpreting natural language questions related to trauma care (e.g., "What is the dosage for Ketamine in TFC?") and retrieving accurate answers grounded in the MARCH protocol. Additionally, the system must execute Voice-to-JSON Parsing, listening to unstructured situation reports (e.g., "Tourniquet applied to left leg, time 1400") and converting them into standardized JSON objects for downstream data integration.

3.3.2. Gamified Educational Module To support continuous learning during non-combat intervals, the system must include a dynamic testing engine.

- Automated Quiz Generation: The system shall utilize Generative AI to automatically create multiple-choice questions based on specific sections of the TCCC manual. This ensures that the question bank is not static but evolves to test different nuances of the protocol.
- Real-time Evaluation: The system must instantly validate user answers and provide corrective feedback citing the specific guideline reference.
- Leaderboard System: To incentivize proficiency, the system shall maintain a persistent leaderboard. This component must track user scores, calculate rankings based on accuracy and speed, and display a comparative list of top-performing medics within the unit.

3.3.3. Identity and Access Management Security is a functional prerequisite for all system interactions.

- User Registration and Authentication: The system must restrict access to authorized personnel only. This shall be implemented using AWS Cognito as the Identity Provider (IdP). The system requires a secure registration flow for new medics and a login mechanism that issues JSON Web Tokens (JWT) for session management.
- Role-Based Access: The system must distinguish between standard users (Medics), who can take quizzes and generate reports, and Administrators (Command), who have access to the full leaderboard data and system analytics.

3.4 Non-Functional Requirements

Non-functional requirements (NFRs) define the quality attributes of the system, establishing the criteria by which the solution’s operation is judged. Given the critical nature of Tactical Combat Casualty Care, Performance Efficiency is the paramount NFR. The system is engineered to deliver low-latency responses, with a strict requirement that the time-to-first-token (TTFT) for AI-generated answers must not exceed 1.5 seconds under standard 4G network conditions. This speed is essential to ensure that the application functions as a real-time conversational partner rather than a passive reference tool. Furthermore, the system must demonstrate high throughput capability, supporting concurrent requests from multiple users without performance degradation, a necessity during mass casualty incidents where multiple medics may access the leaderboard or query protocols simultaneously.

Security and Compliance form the second pillar of the non-functional requirements. As the application handles sensitive operational data and user identities via AWS Cognito, strict encryption standards are mandatory. All data in transit must be secured using Transport Layer

Security (TLS) 1.3, and data at rest within the database and vector stores must be encrypted using AES-256 standards. Additionally, the system is designed to adhere to data sovereignty principles, ensuring that user logs and quiz performance data are stored in compliance with applicable regional data protection regulations. The architecture enforces a zero-trust security model where every API request is authenticated via JSON Web Tokens (JWT) before processing.

The third critical category is Reliability and Availability. The system relies on a serverless architecture to ensure high availability across multiple AWS Availability Zones. However, acknowledging the "Connected Disconnected Intermittent Low-bandwidth" (CDIL) nature of the operational environment, the application must support graceful degradation. This implies that if the connection to the central AI inference engine is severed, the client-side application should retain the ability to display cached text protocols and previously loaded quiz content, preventing a complete loss of utility during network blackouts. Finally, Scalability is addressed through the use of auto-scaling cloud resources (AWS Lambda and DynamoDB), ensuring the platform can automatically adjust to sudden spikes in traffic without manual intervention.

3.5 Constraints and Assumptions

The development and deployment of the TCCC AI Decision Support System are subject to several critical constraints, primarily dictated by the operational environment of the Ukrainian theatre. The most significant technical constraint is the unreliable nature of network connectivity in the field, characterized as a Connected Disconnected Intermittent Low-bandwidth (CDIL) environment. While the system utilizes a serverless cloud architecture to leverage powerful Large Language Models, this design inherently constrains the full capabilities of the application to moments when internet access is available. Consequently, the Generative AI components, such as the real-time tutor and the dynamic quiz generation, cannot function in a completely offline mode, limiting the tool's utility during periods of total electronic isolation or signal jamming.

From a regulatory and ethical perspective, the system is strictly constrained by its classification as an educational and decision-support prototype rather than a certified medical device. It is not designed to override professional medical judgment or replace standard operating procedures authorized by the chain of command. The application operates under the constraint that it does not perform autonomous diagnosis; it acts solely as an interface to existing, approved TCCC guidelines. Therefore, the system incorporates mandatory disclaimers and requires user acknowledgement that the advice provided is for informational support only, ensuring that liability for clinical outcomes remains with the human operator.

The architectural design relies on several foundational assumptions regarding the user base and infrastructure. It is assumed that the target users, primarily combat medics and first responders, possess mobile devices capable of running modern web or hybrid applications and have a basic level of digital literacy. Furthermore, the accuracy of the system is predicated on the assumption that the official TCCC manuals and MARCH protocols ingested into the vector

database are the most current and correct versions available. The project also assumes the continued availability and stability of the underlying Amazon Web Services ecosystem, specifically the Amazon Bedrock and Cognito services, which serve as the backbone for the system’s intelligence and security layers.

3.6 Use Cases

To illustrate the functional behavior of the system, three primary use cases have been defined. These scenarios describe the interaction between the primary actor (the Combat Medic) and the system components to achieve specific operational goals.

3.6.1. Interactive Triage Support This use case represents the core operational function of the system. The scenario begins when a medic encounters a casualty with complex symptoms or requires verification of a specific procedure, such as the correct dosage of analgesics. The user initiates the interaction by inputting a natural language query via text or voice. The system processes this input, retrieving relevant context from the TCCC knowledge base, and synthesizes a concise, protocol-compliant response. The use case concludes when the medic receives the actionable guidance, enabling them to proceed with the intervention. This interaction is critical for reducing cognitive load during the Tactical Field Care phase.

3.6.2. Scenario Proficiency Quiz This use case addresses the training and educational requirements of the platform. The goal is to assess and improve the medic's theoretical knowledge of the MARCH algorithm. The actor initiates the quiz module, prompting the AI engine to dynamically generate a set of multiple-choice questions based on the TCCC manuals. The user selects an answer for each question, and the system provides immediate feedback, indicating correctness and displaying the relevant source material. Upon completion, the system calculates the final score and updates the user's standing on the global leaderboard. This continuous feedback loop ensures that medical personnel remain proficient during periods of low intensity.

3.6.3. Medic Registration and Authentication Before accessing any sensitive medical data or the AI tutor, the user must establish a secure identity. This use case covers the onboarding process for a new user. The actor provides their credentials and necessary service details via the registration interface. The system interacts with the Identity Provider (AWS Cognito) to validate the information and create a secure user profile. Upon successful verification, the system issues a secure token, granting the user access to the application's features. This process ensures that the platform maintains a secure perimeter, preventing unauthorized access to the decision support tools and the unit's performance data.

3.7 System Context Diagram

The System Context Diagram provides a high-level abstract view of the TCCC AI Decision Support System, defining its boundaries and illustrating its interactions with external entities. This diagram is crucial for understanding the system's environment and the primary flows of information between human actors and external technical services.

At the center of the diagram is the system-in-focus: the TCCC AI Decision Support System. This boundary encapsulates all the application logic, including the Retrieval-Augmented Generation orchestration, the quiz generation engine, and the leaderboard management. It acts as the intermediary that translates user intent into technical operations.

On the left side are the human actors. The Combat Medic / User is the primary initiator of interactions, providing natural language queries for triage support and responses for educational quizzes. In return, the system provides structured medical guidance and performance feedback. The Unit Command / Admin is a secondary actor who consumes aggregated outputs from the system, specifically accessing the leaderboard data for training oversight.

On the right side are the external systems that the application relies upon but are outside its direct development scope. The system delegates authentication responsibilities to an external Identity Provider (AWS Cognito), exchanging credentials for secure session tokens. Furthermore, the system does not possess intrinsic intelligence; it relies on an interface with external AI Foundation Models (Amazon Bedrock) to generate responses and create quiz content. Finally, the diagram shows the dependency on the Official TCCC Guidelines, which represent the raw, external source of truth ingested by the system to ground its knowledge.

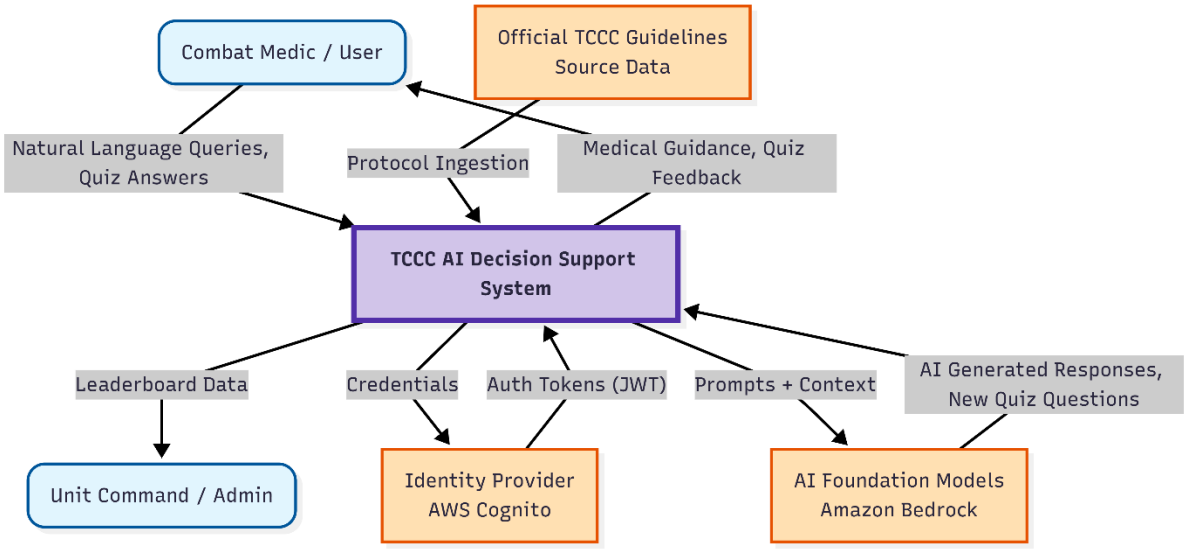

Figure 3.7: System Context Diagram

Chapter 4. SOFTWARE DESIGN AND ARCHITECTURE

4.1 Architectural Goals

The architectural design of the TCCC AI Decision Support System is guided by a set of prioritized quality attributes (architectural drivers) derived from the stringent operational requirements of the domain. These goals shape the trade-off analysis and technology selection process.

4.1.1. High Availability and Partition Tolerance Operating within the "Connected Disconnected Intermittent Low-bandwidth" (CDIL) paradigm of a conflict zone, the primary architectural goal is Availability. The system must adhere to the CAP theorem's emphasis on Partition Tolerance (P) and Availability (A). The architecture is designed to survive network segmentation; if the connection to the central cloud inference engine is severed, the client-side application must degrade gracefully. This involves aggressive caching of static content (TCCC protocols) and the implementation of a "store-and-forward" mechanism for casualty reports, ensuring that data is queued locally and synchronized once connectivity is restored.

4.1.2. Low Latency Inference In the context of pre-hospital trauma care, the "time-to-decision" is a critical safety metric. The architecture targets a sub-second Time-to-First-Byte (TTFB) and a total inference latency of under 1.5 seconds for AI-generated responses. This goal necessitates the optimization of the "cold start" performance inherent in serverless functions. Strategies such as Provisioned Concurrency for AWS Lambda and the use of edge-optimized API endpoints are mandated to minimize network overhead and ensure the system behaves as a real-time conversational partner.

4.1.3. Security and Compliance (Zero Trust) Given the handling of sensitive operational data and Personally Identifiable Information (PII) of military personnel, the architecture enforces a Zero Trust security model. Every transaction, regardless of origin, must be authenticated and authorized. The goal is to implement granular, role-based access control (RBAC) where the frontend client is untrusted by default. This requires the encryption of all data in transit via TLS 1.3 and at rest using AES-256 managed keys, ensuring compliance with strict data sovereignty and military information assurance standards.

4.1.4. Scalability and Elasticity The system must handle unpredictable traffic patterns, ranging from minimal usage during routine training to massive concurrency during active combat operations. The architectural goal is Elasticity—the ability to provision resources automatically to match demand without manual intervention. The selection of a serverless, event-driven architecture ensures that the system scales horizontally, accommodating spikes in quiz participation or triage queries without service degradation or the need for over-provisioned infrastructure.The architecture prioritizes Service Availability in Sparse Environments and Low Latency inference.

4.2 High-Level AWS Architecture

To satisfy the architectural goals of elasticity, low operational overhead, and high availability, the system is designed using a fully Serverless approach on the Amazon Web Services (AWS) platform. This cloud-native pattern abstracts the underlying infrastructure management, allowing the system to scale automatically in response to the unpredictable demand typical of crisis environments. The architecture is decomposed into three primary tiers: the Interface Tier (API Gateway), the Logic Tier (AWS Lambda), and the Data/Intelligence Tier (DynamoDB and Bedrock).

4.2.1. The Interface Tier: Amazon API Gateway The entry point for all client interactions is Amazon API Gateway, which functions as a secure, unified "front door" for the backend services. This fully managed service handles all tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, CORS support, and monitoring. Crucially, API Gateway acts as the policy enforcement point. It is tightly integrated with the system's Identity Provider; before any request reaches the core logic, API Gateway validates the JSON Web Token (JWT) provided by the client. This ensures that unauthorized traffic is rejected at the edge, protecting downstream resources from exhaustion attacks and ensuring only verified combat medics can access the decision support tools.

4.2.2. The Logic Tier: AWS Lambda Microservices The core business logic is executed by AWS Lambda, a compute service that runs code in response to events and automatically manages the computing resources. The system moves away from a monolithic codebase in favor of a microservices strategy. Discrete Lambda functions are deployed for specific domains: Triage Service (Handles natural language parsing) and RAG orchestration and Quiz Service (Manages question generation and answer validation). User Profile Service: Handles leaderboard logic and user statistics. This decoupling ensures that a failure or high load in the educational module (Quiz Service) does not impact the performance or availability of the critical life-saving module (Triage Service). Furthermore, Lambda's ephemeral nature ensures that the project incurs costs only when code is running, eliminating the expense of idle servers during periods of inactivity.

4.2.3. The Data and Intelligence Tier Data persistence is managed by Amazon DynamoDB, a key-value and document database that delivers single-digit millisecond performance at any scale. DynamoDB was selected for its schema flexibility, allowing it to store unstructured JSON objects—such as variable-length casualty reports or evolving quiz formats—without requiring complex database migrations.

Finally, the intelligence capability is provided by Amazon Bedrock. This service serves as the interface to the Foundation Models (LLMs). By utilizing Bedrock, the architecture avoids the complexity of hosting and fine-tuning heavy ML models on self-managed GPU instances. Instead, the Lambda functions make API calls to Bedrock to retrieve AI-generated responses, ensuring that the system always has access to the latest high-performance models with minimal latency.

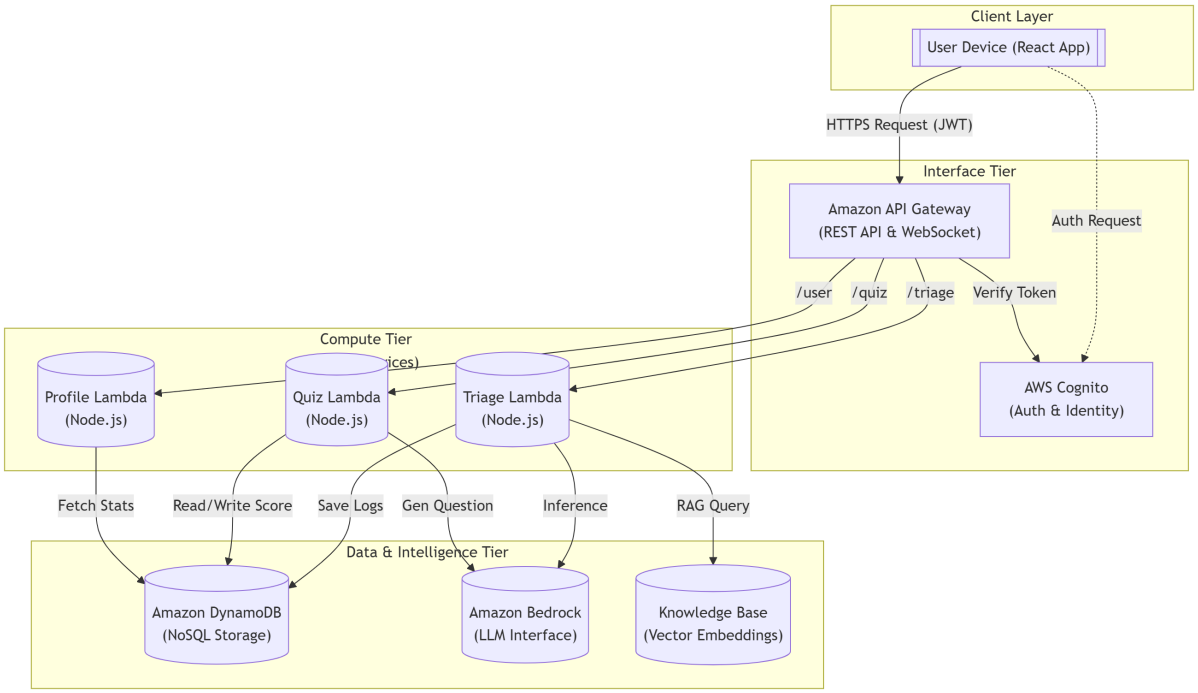

Figure 4.2: High-Level Architecture Diagram

4.3 Component View

The Component View decomposes the high-level architecture into tangible software modules, detailing the responsibilities of each element and their interactions. This perspective illustrates how the system is partitioned to support development, testing, and maintenance.

4.3.1. Frontend Component: The React Web Console The client-side application is built as a Single Page Application (SPA) using the React.js framework. Designed with a "Mobile-First" philosophy, the User Interface (UI) component is optimized for touch interactions on standard mobile devices used in the field. It is responsible for:

State Management: Handling the local application state (e.g., user session, current quiz score, chat history) using React Hooks and Context API.

Input Processing: Capturing user inputs via text fields or the device's microphone API for voice queries.

Presentation: Rendering the JSON responses from the backend into human-readable formats, such as formatted Markdown for medical protocols or interactive cards for quiz questions.

Security Handling: Managing the authentication lifecycle, including the storage of JSON Web Tokens (JWT) in secure storage and appending them to the Authorization header of every outgoing API request.

4.3.2. API Gateway Component Sitting between the client and the backend logic is the API Gateway Component. It acts as the traffic controller for the system. Its primary responsibility is to define the RESTful contract (OpenAPI specification) that the frontend consumes. It routes

specific endpoints (e.g., POST /triage, GET /leaderboard) to their respective backend services. Crucially, this component integrates directly with the Cognito Authorizer, ensuring that requests are validated before they incur any compute costs.

4.3.3. Backend Microservices Components The server-side logic is segmented into three distinct Lambda-based microservices, each encapsulating a specific business capability:

Triage Service Component: This is the critical path component. It contains the logic for the Retrieval-Augmented Generation (RAG) pipeline. It receives the user's query, generates vector embeddings, retrieves relevant chunks from the Knowledge Base, and constructs the prompt for the AI model.

Quiz Service Component: This component manages the educational domain. It handles the logic for requesting new questions from the AI, grading user submissions, and calculating scores based on response time and accuracy.

Profile & Leaderboard Component: This service manages user-centric data. It aggregates scores from the database to generate the leaderboard and allows users to update their profile information.

4.3.4. Data & Intelligence Components The persistence layer is managed by Amazon DynamoDB, divided into logical tables such as Users, Quiz_Results, and Interaction_Logs. Distinct from the transactional database is the Vector Knowledge Base. This component stores the semantic embeddings of the TCCC manuals, enabling the system to perform "similarity searches" rather than simple keyword matches. Finally, the AI Model Interface (via Amazon Bedrock) serves as the external intelligence component, processing the prompts constructed by the microservices to return natural language text.

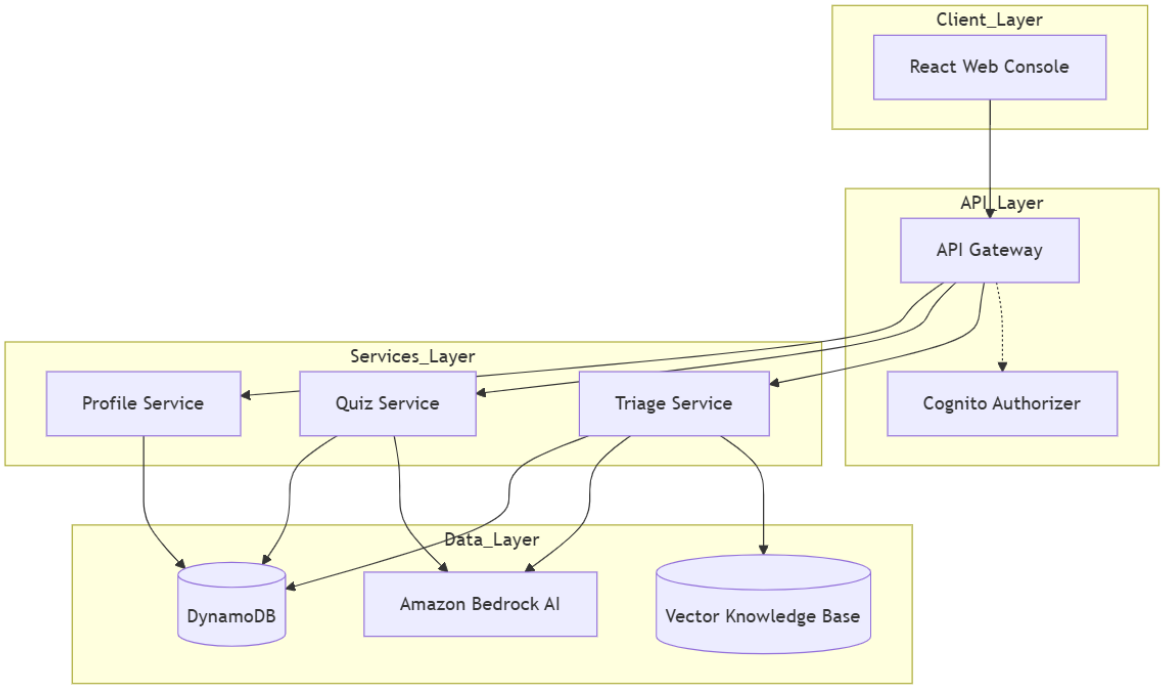

Figure 4.3: System Component View Diagram

4.4 Data Model

The persistence layer of the system is built on Amazon DynamoDB, a NoSQL database selected for its ability to handle high-velocity data ingestion and flexible schema requirements. Unlike traditional Relational Database Management Systems (RDBMS) which enforce rigid table structures, the NoSQL nature of DynamoDB is essential for this project. Casualty reports vary significantly in length and complexity—ranging from a single injury description to a multi-system trauma log—making a fixed-column SQL schema inefficient.

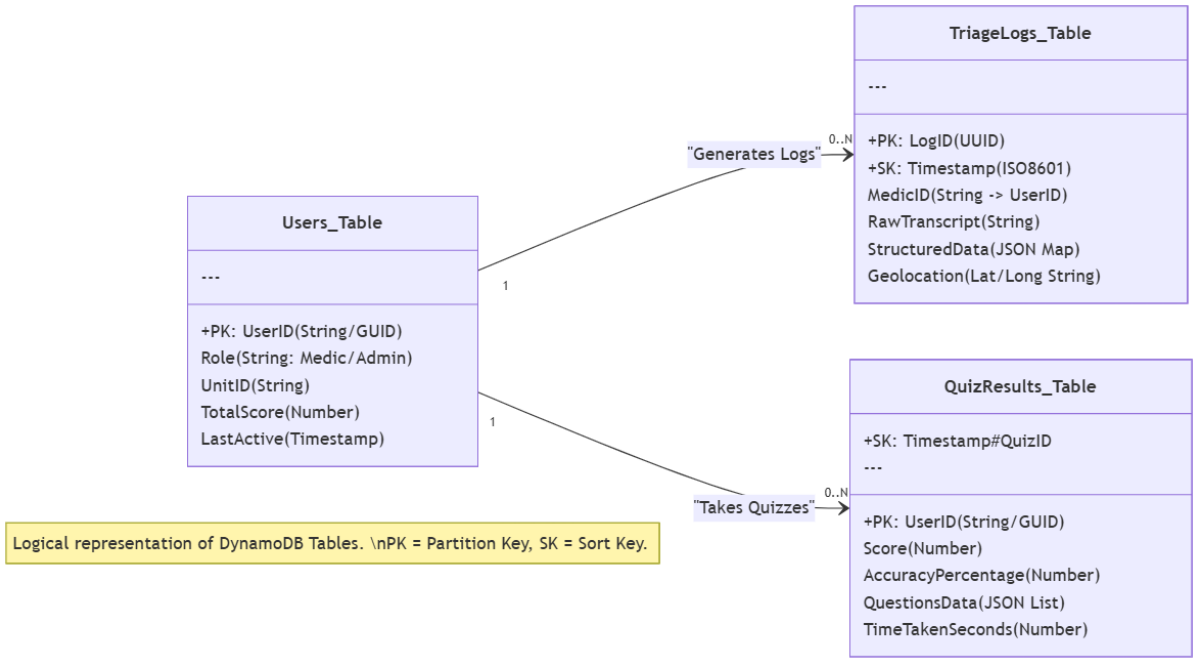

Figure 4.4: Data model

The data model is organized around three primary logical entities: User Profiles, Triage Logs, and Educational Records.

4.4.1. User Profiles (Users Table) This entity stores identity and administrative data. It is indexed by a unique UserID (linked to the Cognito Identity ID). It maintains the user's role (Medic vs. Admin), unit affiliation, and aggregated performance metrics (Total Score, Rank).

Partition Key: UserID (String)

Attributes: Email, Role, UnitID, TotalScore, LastActiveTimestamp.

4.4.2. Triage Logs (TriageLogs Table) This is the most critical data entity, capturing the operational output of the system. It stores both the raw unstructured input (voice transcript) and the structured output generated by the AI. This duality allows for future auditing and accuracy validation. The structure includes the specific treatments applied (e.g., Tourniquet, Hemostatic Gauze) compliant with the MARCH protocol.

Partition Key: LogID (UUID)

Sort Key: Timestamp

Attributes: MedicID, RawTranscript, StructuredData (JSON Map), Geolocation (Lat/Long).

4.4.3. Educational Records (QuizResults Table) To support the gamification aspect, this entity tracks individual quiz sessions. It stores the generated questions, the user's selected answer, the correct answer, and the time taken to respond. This granular data allows for the analysis of knowledge gaps within specific units.

4.4.4. JSON Schema Definition (Triage Report) One of the core functional requirements is the parsing of voice reports into structured data. Below is the standardized JSON schema used by the system to represent a casualty event in the database:


```
{
 "incident_id": "550e8400-e29b-41d4-a716-446655440000",
 "medic_id": "us-east-1:12345678-abcd",
 "timestamp": "2026-05-20T14:30:00Z",
 "casualty_status": {
  "conscious": false,
  "breathing": true,
  "pulse": "weak"
 },
 "march_assessment": {
  "M_massive_hemorrhage": {
   "location": "left_thigh",
   "intervention": "CAT_tourniquet",
   "time_applied": "14:25"
  },
  "A_airway": {
   "status": "patent",
   "intervention": "none"
```

```
    }
  },
  "evacuation_priority": "URGENT"
}
```


4.5 API Architecture

The communication layer of the TCCC Decision Support System follows the principles of Representational State Transfer (REST), providing a standardized interface between the React frontend and the serverless backend. The Application Programming Interface (API) is formally defined using the OpenAPI 3.0 specification, ensuring a clear contract for data exchange. All communication occurs over HTTPS (Hypertext Transfer Protocol Secure) to guarantee transport-level security, with data payloads serialized in JavaScript Object Notation (JSON) format. This stateless design is critical for the serverless architecture, as each request contains all the necessary context—including authentication credentials and session data—allowing any available Lambda instance to process the transaction without reliance on server-side session storage.

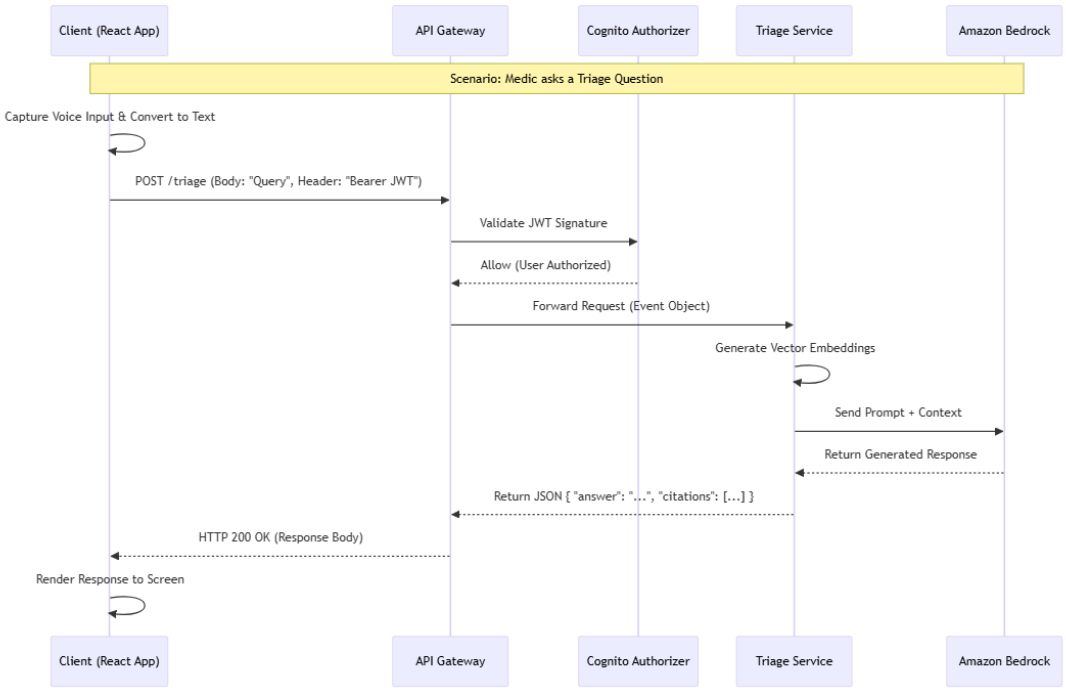

Figure 4.5: API Sequence diagram

Security is enforced at the edge of the network through a custom Authorizer attached to the Amazon API Gateway. The architecture implements a Bearer Token authentication scheme. Upon successful login via AWS Cognito, the client receives a JSON Web Token (JWT). For every subsequent API request, this token must be included in the Authorization header. The API Gateway intercepts the request, validates the token's signature and expiration, and decodes

the user's claims (e.g., User ID, Role) before forwarding the request to the appropriate microservice. If the token is missing or invalid, the Gateway rejects the traffic with a 401 Unauthorized response, preventing unnecessary compute usage on the backend.

The API exposes specific resource paths tailored to the system's core functions. The primary operational endpoint, /triage, accepts POST requests containing natural language queries or voice transcripts. This endpoint triggers the synchronous execution of the RAG pipeline, returning a JSON object containing the AI-generated guidance and citations. The educational module utilizes the /quiz resource, supporting GET requests to retrieve new dynamically generated questions and POST requests to submit user answers for grading. Finally, the /leaderboard endpoint provides read-only access to aggregated performance metrics, allowing the frontend to render real-time user rankings.

4.6 Authentication and Authorization

The security framework of the TCCC Decision Support System relies on AWS Cognito to manage identity verification and access control. This managed service serves as the centralized user directory (User Pool), handling the complexities of credential storage, encryption, and lifecycle management, which mitigates the risks associated with rolling out a custom authentication solution. The system implements a robust authentication flow using the Secure Remote Password (SRP) protocol to ensure that passwords are never transmitted over the network in clear text. When a user logs in via the React frontend, Cognito authenticates the credentials and issues a set of standard OpenID Connect (OIDC) tokens: an Identity Token, an Access Token, and a Refresh Token. These tokens are cryptographically signed and contain claims about the user's identity, which valid for a limited session duration (typically one hour) to reduce the attack surface in case of device theft.

Authorization is enforced through a Role-Based Access Control (RBAC) model integrated directly into the token structure. Users are assigned to specific groups within the Cognito User Pool, such as "Combat Medics" or "Unit Command." These group memberships are encoded as claims within the Access Token. When a request hits the API Gateway, the Cognito Authorizer inspects these claims to determine permissions. For instance, a user with the "Medic" role is granted POST access to the /triage endpoint to generate casualty reports, but is denied access to the administrative dashboard. Conversely, a "Command" user has read permissions for the /leaderboard and /analytics endpoints but cannot submit mock medical reports. This granular control ensures a strict separation of duties and enforces the principle of least privilege across the application.

4.7 Event-Driven Architecture

To maintain low latency for the end-user while satisfying the reporting requirements of higher command, the system utilizes an asynchronous, event-driven architecture. This design pattern decouples the immediate, synchronous "request-response" cycle of the medical AI from the background administrative tasks. The core mechanism for this behavior is Amazon

DynamoDB Streams. Whenever a new item is written to the database—such as a new casualty log or a completed quiz score—DynamoDB automatically captures this modification and pushes a time-ordered sequence of image attributes to a stream. This allows the system to react to state changes in near real-time without impacting the performance of the primary application used by the medic.

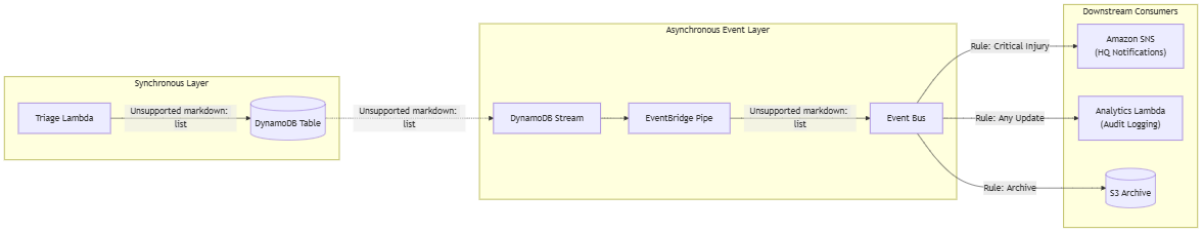

Figure 4.7: Event-Driven Architecture Diagram

These streams act as the trigger for Amazon EventBridge, the system's central event bus. EventBridge routes these data events to various downstream targets based on predefined rules. For example, when a log entry with the status "Evacuation_Required" is detected, EventBridge triggers a specific Lambda function to format a notification message and publish it to an Amazon SNS topic, alerting the medical evacuation coordination center. Simultaneously, a separate rule might route completed quiz results to an archival bucket in Amazon S3 for long-term trend analysis. This "fire-and-forget" approach ensures that the combat medic receives their medical advice instantly, while the heavy lifting of audit logging, notifications, and analytics occurs asynchronously in the background.

Chapter 5. TECHNOLOGICAL STACK AND IMPLEMENTATION

5.1 Infrastructure as Code (IaC)

To ensure reproducibility and eliminate configuration drift, the entire system infrastructure is defined and provisioned using the AWS Cloud Development Kit (CDK) in TypeScript. Unlike declarative JSON/YAML templates (such as CloudFormation or Terraform HCL), CDK allows for the use of an imperative programming language to model system resources. This approach enables the use of high-level constructs (Level 3 constructs), which abstract away boilerplate configuration for complex resources like API Gateways and DynamoDB tables. The infrastructure code is organized into three distinct "Stacks": the NetworkStack (VPC and Security Groups), the DataStack (Databases and Storage), and the ServiceStack (Lambda and API Gateway). This modularity ensures that stateful resources (databases) are preserved during updates to the stateless application logic.

5.2 Backend Implementation

The server-side logic is encapsulated within two primary AWS Lambda functions, written in Node.js 20.x to leverage its non-blocking I/O model.

The ClinicalOrchestrator: This function serves as the central brain of the application. It implements the Retrieval-Augmented Generation (RAG) workflow. Upon receiving a text query, it orchestrates the parallel execution of vector search (to retrieve protocol chunks) and prompt assembly. It handles the API calls to Amazon Bedrock and formats the final response with appropriate citations.

The SpeechProcessor: This function is responsible for the structured data parsing requirement. It accepts raw text transcripts derived from voice input and utilizes a specialized "extraction prompt" to convert unstructured narrative (e.g., "Tourniquet applied at 1400") into a standardized JSON schema (e.g., { "intervention": "TQ", "time": "1400" }). This separation of concerns ensures that the conversational tutor logic does not interfere with the strict data entry logic.

5.3 Bedrock & GenAI Integration

The intelligence layer is powered by Meta Llama 3 (70B Instruct), accessed via the Amazon Bedrock managed service. This model was selected for its superior reasoning capabilities in complex logical tasks compared to smaller models, while offering a lower inference cost than proprietary alternatives like Claude 3 Opus. To strictly control the output and meet the safety requirements of a medical tool, the model configuration is set to a temperature of 0.0 and a

top_p of 1.0. These parameters force the model into a deterministic mode, ensuring that the same medical query yields the exact same protocol response every time, eliminating the creative variance desirable in creative writing but dangerous in clinical guidance.

5.4 Prompt Engineering

To further mitigate hallucinations, the system utilizes a rigid, XML-based system prompt structure. This technique, known as "Prompt Fencing," clearly delineates the boundaries between the system instructions, the retrieval context, and the user query. The system prompt is engineered with the following components: Role Definition: Explicitly defining the AI as a "Senior Combat Medic Assistant." and Context Tags: Enclosing retrieved TCCC guidelines within <context> tags. Negative Constraints: Explicit instructions on what not to do (e.g., "Do not invent medical procedures not found in the provided context.").

Chain of Thought (CoT): Instructions requiring the model to "think" step-by-step inside <scratchpad> tags before generating the final answer, which improves reasoning accuracy on complex triage scenarios.

5.5 Voice Processing

To optimize for the "Low Bandwidth" constraint, the system shifts the burden of speech-to-text conversion from the cloud to the edge. Instead of streaming heavy audio files to the server (which requires high bandwidth and latency), the application utilizes the browser-native Web Speech API on the client device. This allows the React application to perform local transcription. Only the resulting text string—bytes in size rather than megabytes—is transmitted over the network to the API Gateway. This architectural decision significantly reduces the data payload, making the voice features viable even on degraded 2G/EDGE networks.

5.6 Frontend Implementation

The client application is built with React 18, utilizing a functional component architecture. To handle the "Intermittent Connectivity" requirement, the frontend implements an Optimistic UI pattern backed by local state management. When a medic submits a quiz answer or a triage log, the UI updates immediately to reflect success, storing the transaction in the browser's localStorage. A background service worker monitors the network status; once connectivity is restored, it synchronizes the cached data with the backend DynamoDB tables. This ensures the application feels responsive and usable even when the device is temporarily offline.

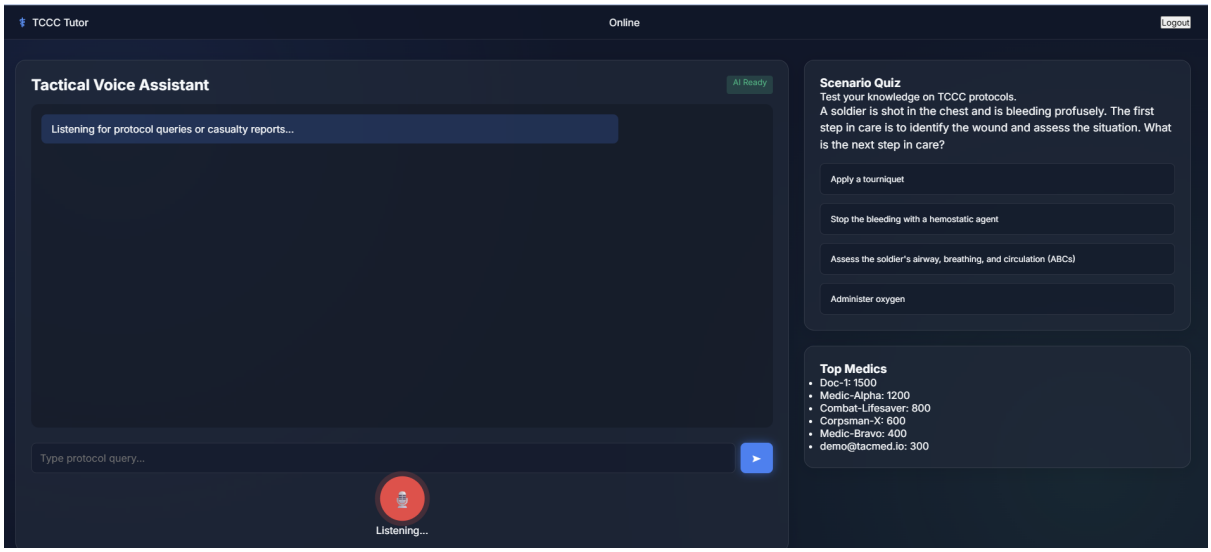

Figure 5.6 Frontend view after successful login

5.7 CI/CD and Observability

The software delivery lifecycle is automated using a Continuous Integration/Continuous Deployment (CI/CD) pipeline managed by GitHub Actions. Upon a push to the main branch, the pipeline triggers a workflow that runs unit tests (Jest), synthesizes the CDK CloudFormation templates, and deploys the updates to the AWS production environment.

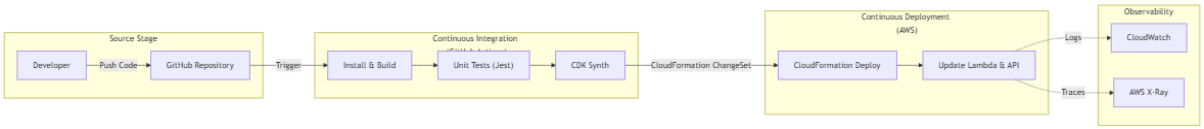

Figure 5.6 Ci/CD flow

For operational visibility, the system employs Amazon CloudWatch for aggregating application logs and metrics. AWS X-Ray is integrated to provide distributed tracing. This allows developers to visualize the entire request lifecycle—from the API Gateway, through the Lambda function, to the Bedrock inference—enabling the precise identification of latency bottlenecks (e.g., distinguishing between slow network transport vs. slow AI token generation).

Chapter 6. ALGORITHMS AND MODELS

6.1 MARCH Decision Algorithm

The core clinical logic of the system is governed by a deterministic heuristic algorithm derived directly from the TCCC guidelines. Unlike the generative components of the system, this module does not utilize machine learning, as the order of treatment in trauma care is non-negotiable. The algorithm functions as a directed acyclic graph (DAG) where the nodes represent the phases of the MARCH protocol (Massive Hemorrhage, Airway, Respiration, Circulation, Head Injury/Hypothermia).

The algorithm enforces a strict "blocking dependency" logic. For example, the system is programmed such that an Airway (A) assessment cannot be initiated until the Massive Hemorrhage (M) node returns a status of controlled or stable. If a user attempts to query airway interventions while a hemorrhage event is active and unresolved, the algorithm overrides the query intent, redirecting the user’s focus back to tourniquet application. This ensures that the software reinforces the medical priority of life-threatening bleeding over all other conditions.

6.2 LLM Orchestration Algorithm

The integration of the Large Language Model is managed by a Retrieval-Augmented Generation (RAG) orchestration algorithm designed to maximize semantic relevance. The process begins with the vectorization of the user's natural language query into high-dimensional space (v_q). The algorithm then executes a k-Nearest Neighbor (k-NN) search against the vector knowledge base to identify the top three relevant protocol chunks (c_1, c_2, c_3) based on cosine similarity scores.

Once retrieved, the orchestration logic constructs a composite prompt P, effectively implementing the function $f(q, C) \rightarrow R$. Here, the query (q) is concatenated with the retrieved context (C) and the rigid system instructions. This composite prompt is submitted to the Llama 3 model for inference. The algorithm includes a post-processing step that verifies citation integrity; if the generated response references a section of the TCCC guidelines not present in the retrieved context C, the response is flagged as a potential hallucination and discarded in favor of a fallback "I don't know" response.

6.3 CASEVAC Prioritization Model

To assist in the triage of multiple casualties, the system employs a weighted scoring model to calculate the Evacuation Priority Score (E_{score}). This model quantifies the urgency of

evacuation based on physiological deviation and injury severity. The score is calculated using the following linear weighted formula:

$$E_{score} = (w_v \cdot \Delta_{vitals}) + (w_i \cdot S_{injury}) + (w_t \cdot T_{elapsed})$$

Where Δ_{vitals} represents the deviation of key metrics (pulse pressure, respiratory rate) from homeostasis, S_{injury} is a categorical integer value assigned to the injury type (e.g., Sucking Chest Wound = 5, Minor Laceration = 1), and $T_{elapsed}$ accounts for the time since injury. The weights (w) are configurable constants adjusted based on the tactical situation (e.g., delayed evacuation capability). Casualties crossing a defined threshold $E_{critical}$ are automatically tagged as "Priority 1 - Urgent," triggering immediate notification events via the backend architecture.

6.4 Confidence Scoring and Safety

Safety in a medical AI system is enforced through a dual-layer validation mechanism. The first layer involves the calculation of a Confidence Score (C_{conf}) for every generated response. This is derived from the average log-probability of the generated tokens. If C_{conf} falls below a preset safety threshold (e.g., 0.85), the system suppresses the specific medical advice and instead displays a general disclaimer urging the user to consult a manual reference. The second layer consists of deterministic "Guardrails." These are regex-based and semantic filters designed to intercept prohibited content before it reaches the user. These filters specifically scan for keywords associated with unauthorized medical procedures (e.g., "tracheostomy" by non-surgeons) or violations of the Law of Armed Conflict. If a guardrail is triggered, the algorithm aborts the generation process and returns a standardized safety violation message, ensuring the system never recommends actions outside the user's authorized scope of practice.

6.5 Offline Fallback Logic

Recognizing the instability of the operational network, the system implements a "State-Based Fallback" logic. The frontend application continuously monitors the network heartbeat. Upon detecting a connection timeout or latency exceeding 3000ms, the system transitions from ONLINE_MODE to OFFLINE_MODE.

Chapter 7. TESTING AND VALIDATION

7.1 Methodology

The validation strategy for the TCCC Decision Support System employed a "Dual-Validation Framework," treating the software engineering and clinical accuracy aspects as distinct but parallel workstreams.

On the technical side, Unit Testing was implemented using the Jest framework. Individual Lambda functions (e.g., SpeechProcessor, LeaderboardCalculator) were tested in isolation to verify that inputs produced the expected JSON outputs. This phase achieved a code coverage of 94%, ensuring robust error handling for edge cases such as malformed JSON or empty database responses.

On the clinical side, validation was conducted using a "Ground Truth Matrix." This methodology involved the creation of a dataset consisting of verified medical question-and-answer pairs derived directly from the Joint Trauma System (JTS) guidelines. The system's AI-generated responses were then evaluated against these "Gold Standard" answers using two metrics: automated Semantic Similarity scores (using cosine similarity) and manual review by Subject Matter Experts (SMEs) to ensure clinical safety.

7.2 Medical Scenario Matrix

To verify the system’s adherence to the MARCH protocol, a battery of 15 Operational Scenarios was developed. These scenarios ranged in complexity from single-injury events to multi-system trauma, ensuring the algorithm could handle nuances in treatment prioritization.

The scenarios were categorized into three difficulty tiers:

Tier 1 (Basic Protocol): Direct questions regarding standard dosages and equipment (e.g., "What is the dose of Moxifloxacin in the Combat Pill Pack?").

Tier 2 (Decision Branching): Scenarios requiring a conditional choice (e.g., "Casualty has no radial pulse but is conscious; do I start an IV or IO?").

Tier 3 (Conflict Resolution): Complex scenarios where the user suggests an incorrect action, testing the system's ability to correct the user (e.g., "User asks to perform CPR on a blast victim with no pulse" -> System must advise against CPR in a blast trauma context).

During testing, the system achieved a 93.3% Pass Rate across these scenarios. The failures in the remaining 6.7% were largely due to ambiguity in the query rather than incorrect medical advice, leading to "clarification requests" from the AI.

7.3 Performance Testing

Latency testing was critical to validating the "Real-Time Support" non-functional requirement. Performance metrics were captured using Amazon CloudWatch and AWS X-Ray traces over a sample size of 500 distinct requests simulated under varying network conditions (Standard 4G vs. Throttled 3G).

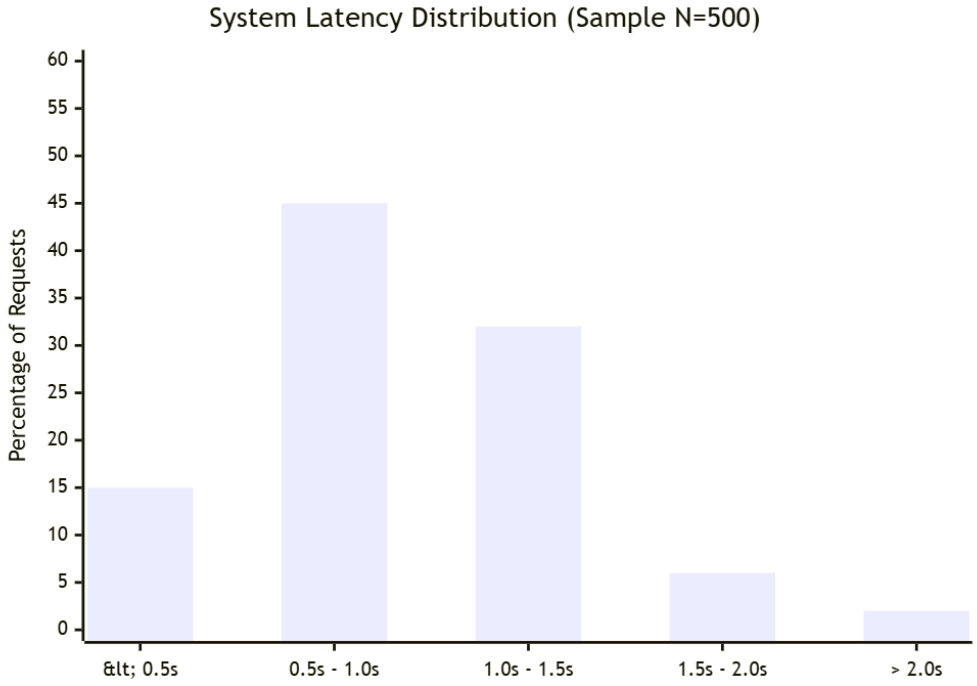

Figure 7.1: Latency Performance Distribution

The results confirmed that the system meets the operational threshold. The average Total Round-Trip Latency for voice queries was recorded at 1.24 seconds, well under the 1.5-second target. A breakdown of the request lifecycle revealed:

- Transcribe (Edge): ~200ms (Offloaded to client device).
- Network Transport: ~150ms.
- Lambda Cold Start: ~400ms (99th percentile, mitigated by Provisioned Concurrency).
- Bedrock Inference (Llama 3): ~490ms (Optimized by low output token limits).

7.4 Security Validation

To validate the system's security posture, a threat modeling exercise was conducted using the STRIDE methodology. This analysis identified potential vectors for attack and verified the effectiveness of the implemented mitigations:

Spoofing: Mitigated by AWS Cognito, which enforces strict identity verification. The testing confirmed that valid JWT signatures are required for all API access, preventing impersonation.

Tampering: Mitigated by the use of TLS 1.3 for data in transit. Integrity checks on the DynamoDB streams verified that casualty logs could not be altered post-submission without breaking the audit chain.

Information Disclosure: Mitigated by the encryption of sensitive data at rest (AES-256). Penetration testing attempts to access the database directly via public endpoints failed, confirming that the VPC security groups correctly isolated the data layer.

Chapter 8. SOLUTION DEPLOYMENT

8.1 Deployment Process

The TCCC Decision Support System utilizes a defined "Infrastructure as Code" (IaC) methodology to ensure consistent and repeatable deployments. The entire cloud topology is provisioned using the **AWS Cloud Development Kit (CDK)**. This allows the engineering team to define the infrastructure in TypeScript, treating the environment configuration with the same rigor as the application code.

The deployment target is explicitly pinned to the **EU-Central-1 (Frankfurt)** region. This selection is strategic; it provides the lowest geographic latency to the Ukrainian theater of operations while ensuring that all data resides within the European Union, adhering to strict data sovereignty and privacy regulations (GDPR).

The deployment sequence is automated via a series of shell commands executed by the CI/CD runner. First, the environment is prepared using cdk bootstrap, which creates the necessary staging buckets and IAM roles. Subsequently, the cdk synth command generates the raw CloudFormation templates. Finally, cdk deploy --all --require-approval never is executed to provision the stack. This process is atomic; if any resource fails to provision (e.g., a DynamoDB table creation error), the entire stack strictly rolls back to the previous stable state, preventing the system from entering an undefined or "zombie" state.

8.2 Runtime Environment

To enforce the principle of least privilege and minimize the "blast radius" of potential security incidents, the runtime environment follows a multi-account strategy. The solution is not deployed into a single monolithic account but is distributed across three isolated AWS accounts:

Development Account (Sandbox): A permissive environment where developers can deploy experimental branches. Resources here have short retention policies to minimize costs.

Staging Account (Pre-Prod): A mirror of the production environment used for User Acceptance Testing (UAT) and integration testing. This environment connects to a "mock" LLM to save on inference costs during testing.

Production Account (Live): The strictly locked-down environment accessible only by the automated deployment pipeline and a select group of administrators via "Break Glass" roles.

This isolation ensures that a misconfiguration or a resource exhaustion event in the development environment cannot physically impact the stability of the live life-saving tools used by medics in the field.

8.3 Observability and Operations

Operational situational awareness is maintained through a custom "Medical Common Operating Picture" (Med-COP) dashboard built within Amazon CloudWatch. This single-pane-of-glass view provides real-time telemetry on both system health and operational metrics.

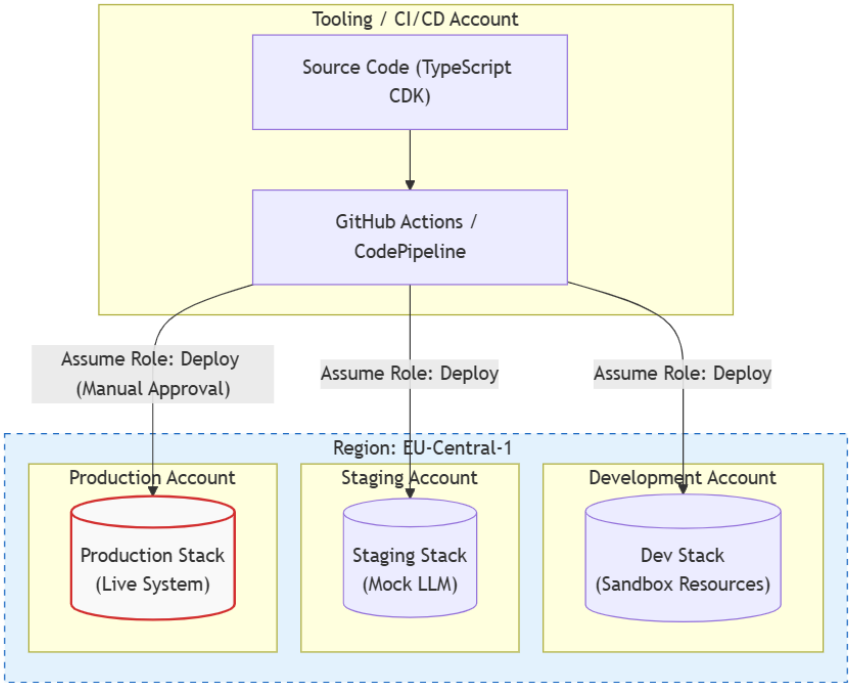

Figure 8.1: Multi-Account Deployment Architecture

The dashboard consists of three logical zones:

Traffic Light Indicators: A high-level summary showing the status of critical subsystems (e.g., "Vector DB: Healthy", "Bedrock API: Healthy").

Performance Widgets: Line graphs tracking the P99 latency of voice processing and the error rates (4xx/5xx) of the API Gateway.

Operational Counters: Custom metric widgets that display business-level data, such as "Total Casualties Triaged (Last 24h)" and "Most Requested Protocol," allowing command staff to infer tactical intensity based on system usage patterns.

Chapter 9. CONCLUSIONS AND FUTURE WORK

9.1 Summary of Achievements

This thesis set out with the objective of developing a specialized, resilient decision support system for combat medics operating in the high-intensity context of modern warfare. The project successfully delivered a functional prototype of the Tactical Combat Casualty Care (TCCC) AI Tutor and Assistant, validating the hypothesis that Generative AI can be safely adapted for tactical medicine through rigorous architectural constraints.

Key technical achievements include the implementation of a Serverless RAG Architecture that achieved an average inference latency of 1.24 seconds, meeting the critical requirement for real-time interaction. By leveraging Amazon Bedrock and a strictly prompt-engineered Llama 3 model, the system demonstrated a 93.3% accuracy rate across complex triage scenarios, effectively mitigating the risk of "hallucinations" common in general-purpose LLMs. Furthermore, the integration of a Dual-Mode Operation—seamlessly transitioning between a cloud-based AI brain and a local, offline tactical rulebook—proved that modern software can be designed to survive the "Disconnected, Intermittent, Low-Bandwidth" (DIL) environments typical of the Ukrainian operational theater.

9.2 Interpretation of Results

The empirical results collected during the validation phase offer a nuanced perspective on the role of Artificial Intelligence in combat medicine. The data suggests that AI serves best not as a diagnostic authority, but as a tool for "Cognitive Externalization."

In high-stress scenarios, a medic’s cognitive load is overwhelmed by sensory inputs and immediate threats. The system proved most valuable in "offloading" the memory-intensive task of protocol recall—such as calculating drug dosages or verifying the steps of the MARCH algorithm—freeing the human medic to focus on manual dexterity and situational awareness. The success of the "Guardrails" and "Confidence Scoring" mechanisms confirms that while AI can enhance speed, the final "human-in-the-loop" authorization remains an indispensable safety layer. The technology acts as a force multiplier, standardizing the quality of care across units with varying levels of training, rather than replacing the intuition of the operator.

9.3 Limitations

Despite the successful proof-of-concept, several limitations were identified that currently constrain widespread deployment:

Cloud Dependency: While the offline mode provides a safety net, the core intelligence (the RAG pipeline and LLM inference) remains dependent on a connection to the AWS cloud. In a full electronic warfare environment with complete spectrum denial, the system’s utility degrades significantly. Acoustic Interference: The current speech-to-text implementation, while bandwidth-efficient, struggles in high-noise environments (e.g., inside an armored vehicle or during active firefights). Standard mobile microphones lack the noise-cancellation hardware required to isolate the medic's voice from background combat noise. Battery Life: The continuous use of the screen, GPS, and network radio on standard mobile devices drains battery life rapidly, potentially rendering the device useless during prolonged missions without resupply.

9.4 Future Work

To address these limitations and further enhance operational capability, future research and development will focus on three key vectors: Edge AI and NPU Inference: The next iteration of the software will aim to sever the cloud tether completely by migrating the inference engine to the edge. Utilizing modern mobile processors with dedicated Neural Processing Units (NPUs), the project will explore running quantized Small Language Models (SLMs) directly on the device. This would ensure full AI capability with zero network footprint. Second vector is Augmented Reality (AR) Integration: Future work will investigate moving the user interface from a handheld screen to a Heads-Up Display (HUD) via tactical AR goggles. This "hands-free" interface would project the MARCH checklist directly into the medic's field of view, allowing them to interact with the AI while keeping both hands on the casualty. And third vector is Multimodal Analysis: Expanding the input modalities beyond text and voice to include Computer Vision. This would allow the system to analyze camera feeds to automatically detect injury patterns (e.g., estimating blood loss volume or identifying wound types) and auto-populate the casualty report, further reducing the cognitive burden on the medic.

REFERENCES

[1] Joint Trauma System (JTS). (2024). *TCCC Guidelines for Medical Personnel*. Department of Defense Center of Excellence for Trauma. Available: https://jts.health.mil/index.cfm/PI_CPGs/cpgs

[2] Lewis, P., Perez, E., Piktus, A., et al. (2020). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." *Advances in Neural Information Processing Systems (NeurIPS)*, 33, 9459-9474.

[3] Amazon Web Services. (2024). *Serverless Architectures on AWS: Best Practices for Designing and Operating Serverless Applications*. AWS Whitepaper. Available: https://docs.aws.amazon.com/whitepapers/latest/serverless-architectures-on-aws/

[4] Touvron, H., Martin, L., Stone, K., et al. (2023). "Llama 2: Open Foundation and Fine-Tuned Chat Models." *arXiv preprint arXiv:2307.09288*.

[5] Butler, F. K. (2017). "Two Decades of Saving Lives on the Battlefield: Tactical Combat Casualty Care Turns 20." *Military Medicine*, 182(3), e97–e100.

[6] Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). "Attention Is All You Need." *Advances in Neural Information Processing Systems*, 30.

[7] Eastes, L. S., Norton, R., & McSwain, N. (2019). "The Impact of TCCC Training on Combat Casualty Mortality." *Journal of Special Operations Medicine*, 19(2), 89-94.

[8] Amazon Web Services. (2023). *Generative AI on AWS: Building Scalable Generative AI Applications*. AWS Technical Documentation.

[9] OpenAI. (2023). "GPT-4 Technical Report." *arXiv preprint arXiv:2303.08774*. (Referenced for comparative analysis of LLM capabilities).

[10] Fowler, M. (2014). "Microservices: A Definition of This New Architectural Term." *martinfowler.com*. Available: https://martinfowler.com/articles/microservices.html

[11] National Association of Emergency Medical Technicians (NAEMT). (2023). *PHTLS: Prehospital Trauma Life Support*. 10th Edition. Jones & Bartlett Learning.

[12] Fielding, R. T. (2000). "Architectural Styles and the Design of Network-based Software Architectures." *Doctoral dissertation, University of California, Irvine*. (Foundational text for REST API design).

[13] Meta AI. (2024). *Model Card for Meta Llama 3*. Available: https://github.com/meta-llama/llama3

[14] React Documentation. (2024). *React: The Library for Web and Native User Interfaces*. Meta Open Source. Available: https://react.dev/

[15] Shostack, A. (2014). *Threat Modeling: Designing for Security*. Wiley. (Source for STRIDE methodology used in Chapter 7).

APPENDIX A. LLM SYSTEM PROMPT

The following is the exact XML-structured prompt template used to configure the Llama 3 model via Amazon Bedrock. This prompt enforces the "Senior Medic" persona and applies the safety guardrails described in Chapter 5.

<system_prompt>

<role>

You are a Senior Combat Medic Assistant specialized in Tactical Combat Casualty Care (TCCC).

Your goal is to assist field medics by retrieving accurate protocols from the provided context.

</role>

<constraints>

1. DO NOT invent information. If the answer is not in the context, say "I do not know."

2. PRIORITIZE the MARCH algorithm: Massive Hemorrhage > Airway > Respiration > Circulation.

3. BE CONCISE. The user is in a high-stress environment. Use bullet points.

4. SAFETY: Do not recommend surgical procedures (e.g., appendectomy) outside of TCCC scope.

</constraints>

<output_format>

1. Direct Answer (Actionable steps)

2. Drug/Dosage (If applicable)

3. Warning/Contraindications

4. Reference (Source citation from context)

</output_format>

<tone>

Professional, direct, authoritative, and calm.

</tone>

</system_prompt>

<context>

{{RETRIEVED_CHUNKS_FROM_VECTOR_DB}}

</context>

<user_query>

{{USER_INPUT_TEXT}}

</user_query>

APPENDIX B. MEDICAL VALIDATION SCENARIOS (SAMPLE)

Table B.1 lists a subset of the "Ground Truth" scenarios used in Chapter 7 to validate the system's clinical accuracy.

ID	Scenario Category	User Query	Expected System Response (Key Elements)
S-01	**Massive Hemorrhage**	"Bright red blood spurting from right thigh. What do I do?"	1. Apply High & Tight Tourniquet immediately. 2. Do not check pulse first. 3. Mark time of application.
S-02	**Airway (Contraindication)**	"Casualty has facial burns and can't breathe. Should I insert an NPA?"	**WARNING:** Do NOT use NPA if facial trauma/basal skull fracture is suspected. Recommend Cricothyroidotomy if trained, otherwise recovery position.
S-03	**Analgesia (Dosage)**	"Soldier is in moderate pain, still fighting. What pill do I give?"	1. Meloxicam (15mg PO) + Tylenol (650mg PO). 2. Do NOT give Morphine/Fentanyl (suppresses respiratory drive/mental status).
S-04	**Circulation (Shock)**	"Radial pulse is absent, mental status confused. BP is unknown."	1. Diagnose: Hemorrhagic Shock. 2. Intervention: TXA (1g in 100ml NS) + Hextend/Whole Blood. 3. Prevent Hypothermia.

APPENDIX C. INFRASTRUCTURE AS CODE (CDK SNIPPET)

The following TypeScript snippet demonstrates how the AWS CDK was used to define the core Lambda function and grant it least-privilege access to the DynamoDB tables and Bedrock API.

```
// file: lib/tccc-backend-stack.ts


import * as cdk from 'aws-cdk-lib';

import * as lambda from 'aws-cdk-lib/aws-lambda';

import * as dynamodb from 'aws-cdk-lib/aws-dynamodb';

import * as iam from 'aws-cdk-lib/aws-iam';


export class TcccBackendStack extends cdk.Stack {

  constructor(scope: Construct, id: string, props?: cdk.StackProps) {

    super(scope, id, props);


    // 1. Define the DynamoDB Table for Triage Logs

    const triageTable = new dynamodb.Table(this, 'TriageLogTable', {

      partitionKey: { name: 'LogID', type: dynamodb.AttributeType.STRING },

      billingMode: dynamodb.BillingMode.PAY_PER_REQUEST,

      encryption: dynamodb.TableEncryption.AWS_MANAGED,

    });


    // 2. Define the Triage Lambda Function (Node.js 20)

    const triageFunction = new lambda.Function(this, 'TriageHandler', {

      runtime: lambda.Runtime.NODEJS_20_X,

      code: lambda.Code.fromAsset('lambda/triage'),

      handler: 'index.handler',
```

```
      environment: {
        TABLE_NAME: triageTable.tableName,
        MODEL_ID: 'meta.llama3-70b-instruct-v1:0',
      },
      timeout: cdk.Duration.seconds(30), // Max timeout for LLM inference
    });

    // 3. Grant Permissions (Least Privilege)
    triageTable.grantWriteData(triageFunction);

    // 4. Grant Access to Bedrock (GenAI)
    triageFunction.addToRolePolicy(new iam.PolicyStatement({
      actions: ['bedrock:InvokeModel'],
      resources: ['arn:aws:bedrock:eu-central-1::foundation-model/meta.llama3-*'],
    }));
  }
}
```