Sparse Data: Perspectives on Learning from Limited Information

Joshua Yeats

4th Year Project Report
Artificial Intelligence
School of Informatics
University of Edinburgh
2025

Abstract

This dissertation evaluates three distinct methodologies, program synthesis, large language model (LLM)-based rule generation, and reinforcement learning (RL), for their effectiveness in generating interpretable rules to guide decision-making in dynamic clinical environments. The research addresses a fundamental challenge: Can systems be developed that effectively learn from noisy and sparse health data while producing interpretable solutions that enhance understanding and improve patient outcomes? With a focus on Type 1 diabetes, this work explores novel approaches to extract meaningful patterns from sparse data while maintaining the transparency needed for decision-making. Statistical SYGUS-based program synthesis achieved minimal accuracy due to its inability to model complex health data variability. LLM-based approaches demonstrated significant potential with high accuracy while maintaining human-readable outputs. Reinforcement learning with Q-learning synthesis successfully developed adaptive insulin dosing policies through simulated environments, achieving stable glucose time-in-range metrics. Each approach is assessed for its ability to balance transparency with adaptability, revealing unique strengths and limitations. The findings suggest that a hybrid framework combining LLM knowledge with retrospective reinforcement learning presents a promising direction for future research in this domain.

Research Ethics Approval

This project obtained approval from the Informatics Research Ethics committee.
Ethics application number: 166955
Date when approval was obtained: 2024-09-30

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Joshua Yeats*)

Acknowledgements

I would like to thank my dissertation supervisor, Elizabeth Polgreen, for her guidance and support throughout this process. I am also thankful to my family and friends, whose encouragement has sustained me during this journey. Lastly, I would like to thank Cult Coffee for keeping me caffeinated through all of this.

Table of Contents

1 Introduction and Motivation **1**

2 Background **2**
2.1 Data Analysis in Type 1 Diabetes 2
2.2 Program Synthesis 3
2.2.1 Classification of Program Synthesis Approaches 4
2.3 Syntax-Guided Synthesis (SyGuS) 4
2.3.1 Core Components of SyGuS 4
2.4 Key Synthesis Methodologies 5
2.4.1 SyGuS Workflow and Iterative Refinement 6
2.4.2 Supporting Tools and Technologies 6
2.5 Large Language Models (LLMs) 6
2.6 Reinforcement Learning 7

3 Program Synthesis for Pattern Discovery **9**
3.1 Problem Definition 9
3.2 Data and Preliminary Experiments 9
3.3 Statistical SyGuS 10
3.3.1 Statistical Analysis 11
3.3.2 SyGuS Translation 11
3.3.3 Function Signature Determination 12
3.3.4 Grammar Construction 12
3.3.5 Constraint Generation 14
3.4 Results 15

4 LLM-Based Rule Generation and Validation **17**
4.1 Problem Definition 17
4.2 System Architecture 18
4.2.1 Prompt Generation 18
4.2.2 Rule Generation 20
4.2.3 Validation Process 21
4.2.4 Rule Analysis 21
4.3 Results 22

5 Reinforcement Learning for Synthesizing Logical Policies **25**
5.1 Problem Definition 26
5.2 Overview of RL Synthesis 26

5.2.1 Simulator Environment 29
5.2.2 Simulator-Based Experiment 29
5.2.3 Results 30
5.2.4 Simulation + Historical Data 32
5.2.5 Results 32

6 Evaluation **35**
6.0.1 Program Synthesis: Statistical SYGUS 35
6.0.2 LLM-Based Rule Generation 36
6.0.3 Reinforcement Learning: Q-Learning 36
6.0.4 Towards a Hybrid Approach Using Inverse Reinforcement Learning 37

7 Conclusion **38**

Bibliography **41**

A Dataset Description **45**
A.1 Overview 46
A.1.1 Data and Quality Assurance 46
A.2 Descriptive Statistics & Basic Patterns 46
A.3 Correlation Analysis and Relationship Structure 47
A.3.1 Glucose Response Correlations 47
A.3.2 Exercise Parameter Correlations 48

B Program Synthesis for Pattern Discovery **49**
B.1 Initial Experiments 50
B.2 Examples of Stat SyGuS Specifications 51

C LLM-Based Rule Generation and Validation **54**
C.1 LLM Algorithm 54
C.2 LLM User Prompt 55
C.3 LLM System Prompt 56
C.4 Output Structure 57

D Reinforcement Learning for Synthesizing Logical Policies **58**
D.1 Grammar-Based Policy Synthesis Algorithm 58

E Evaluation **59**
E.1 Comparison Table 59

Chapter 1

Introduction and Motivation

Problem: Can systems be developed that effectively learn from noisy and sparse health data while producing interpretable solutions that enhance understanding and improve patient outcomes? With a focus on Type 1 diabetes, this research explores novel approaches to extract meaningful patterns from sparse data while maintaining the transparency needed for decision-making.

Type 1 diabetes is a chronic autoimmune disease characterized by insulin deficiency due to pancreatic beta cell destruction. Patients require lifelong insulin therapy, carefully balancing doses to avoid both hypoglycemia (low blood glucose) and hyperglycemia (high blood glucose) - each with its own immediate and long-term health risks.

The management of this condition generates substantial health data through continuous glucose monitors (CGMs), insulin pumps, and patient self-reporting. However, this data presents significant analytical challenges: it is sparse due to collection gaps, noisy from measurement errors, and heterogeneous in its sources and formats. Despite these limitations, effective analysis of this imperfect data could transform patient care and outcomes.

Traditional machine learning approaches often falter with such data, typically requiring large, dense datasets to build reliable models. Moreover, many advanced algorithms operate as "black boxes," providing predictions without transparent explanations - a critical weakness in healthcare contexts where clinicians and patients need to understand the reasoning behind recommendations.

The daily burden of managing Type 1 diabetes is considerable, affecting every aspect of patients' lives from work and school to social activities and exercise. An interpretable system capable of learning from incomplete data could personalize treatment strategies, provide actionable insights to both patients and healthcare providers, identify early warning signs of complications, and reduce the cognitive load on individuals managing this complex condition.

Interpretable solutions could enhance diabetes care, allowing providers to validate recommendations and patients to understand their condition's patterns. This transparency builds trust, enables collaborative decision-making, eases regulatory approval, and supports patient education. By clarifying connections between lifestyle, insulin, and glucose outcomes, interpretable systems transform complex data into actionable guidance that informs healthcare decisions.

Chapter 2

Background

2.1 Data Analysis in Type 1 Diabetes

Continuous glucose monitoring (CGM) has improved type 1 diabetes management by delivering detailed, real-time insights into blood glucose levels [7]. Type 1 diabetes, an autoimmune disorder marked by a complete lack of insulin production [4], demands precise glycemic control to avert both acute complications and long-term consequences [24]. Yet, achieving this control is complicated by factors such as variable insulin absorption, diverse meal compositions, physical activity, stress, and circadian rhythms. These challenges underscore the need for advanced analytical methods to fine-tune insulin therapy and elevate patient outcomes.

Among these methods, machine learning stands out as a powerful approach for tackling the complex, non-linear, and sequential nature of CGM data. Techniques like recurrent neural networks (RNNs) and long short-term memory (LSTM) networks excel at detecting temporal patterns in glucose readings. By weaving together historical glucose data with contextual factors - such as carbohydrate intake, exercise, and insulin doses - these models deliver accurate short-term predictions vital for proactive management [11, 18]. Typically, they rely on extensive datasets, with studies leveraging 1 to 3 months of CGM data, amounting to 8,640 to 25,920 data points per patient [23].

Complementing machine learning, time series analysis offers another robust framework for forecasting glucose trends. Methods like ARIMA (Autoregressive Integrated Moving Average) break down CGM data into trend, seasonal, and noise components, enabling precise short - term predictions [33]. In contrast, Kalman filters provide a dynamic, recursive approach, adapting to the noise and rapid fluctuations in glucose levels as new data emerges [8, 26]. These techniques typically require a few days to weeks of CGM data, yielding thousands of points due to the high frequency of measurements [19].

Statistical methods, such as linear and multiple regression, play a foundational role in unraveling the relationships between blood glucose and key influencing factors. These approaches allow clinicians and researchers to assess how diet, physical activity, and insulin dosing shape glycemic variability [37, 15]. Given the wide individual differences in type 1 diabetes responses, statistical tools are essential for pinpointing predictors and crafting personalized treatment plans. For individual patients, the data needs often mirror those of time series analysis- thousands of CGM points - though cross-sectional

studies, like those using the Pima dataset, differ in scope [2].

Physiological modeling, meanwhile, offers a deeper dive into the glucose-insulin dynamics central to type 1 diabetes care. The Minimal Model, for instance, estimates critical parameters like insulin sensitivity and glucose effectiveness using just 5 to 10 data points from an oral glucose tolerance test (OGTT) [9, 12]. This simplicity contrasts with more comprehensive tools like the UVA/PADOVA Type 1 Diabetes Simulator, which integrates multiple physiological processes to refine insulin dosing and support artificial pancreas development. Built on extensive clinical data from over 20 patients, this simulator harnesses thousands of points to simulate complex metabolic interactions [13, 20]. When paired with CGM data, these models illuminate individual metabolic profiles, paving the way for tailored and effective diabetes management.

Together, these analytical approaches - machine learning, time series analysis, statistical methods, and physiological modeling - address the multifaceted challenges of type 1 diabetes. Their varying data demands, from minimal test samples to months of continuous monitoring, reflect their unique strengths in optimizing insulin therapy and improving patient well-being.

Challenges in Glucose Data Analysis for Type 1 Diabetes Despite these advanced techniques, several challenges remain in the analysis of CGM data for type 1 diabetes:

- **High Variability and Noise:** Glucose levels in type 1 diabetes exhibit rapid fluctuations and considerable variability, driven by factors such as inconsistent insulin absorption, diverse meal compositions, physical activity, and stress. These dynamics generate noisy datasets that hinder precise modeling and prediction, posing a significant obstacle to reliable analysis [27].
- **Need for Personalization:** The marked heterogeneity among individuals with type 1 diabetes renders one-size-fits-all models inadequate. Personalized approaches, which must account for unique insulin sensitivity, lifestyle factors, and behavioral patterns, are critical yet difficult to achieve. Developing such models demands extensive data integration and often manual adjustments, adding complexity to the process [10].
- **Data Integration and Quality:** While CGM devices deliver continuous data streams, their utility is tempered by issues like sensor inaccuracies, calibration errors, and data gaps, all of which can erode model accuracy. Integrating CGM readings with complementary sources - such as dietary records, exercise logs, and insulin dosing schedules - is essential for a holistic analysis. However, this integration introduces additional layers of complexity and potential error [27].
- **Interpretability:** Many advanced models, particularly machine learning techniques like neural networks, produce highly accurate predictions but lack transparency in how they arrive at their outputs. This “black box” nature limits their interpretability, making it challenging for clinicians to trust and act on results or explain them to patients. Enhancing model transparency is thus a key hurdle in translating analytical insights into practical diabetes care [31].

2.2 Program Synthesis

Program synthesis is an automated process for generating executable code from high-level specifications, examples, or constraints. By leveraging techniques from artificial

intelligence (AI), formal methods, and constraint solving, program synthesis aims to produce correct, efficient, and semantically aligned programs [17]. A synthesized program is often more interpretable than a typical black-box machine learning model, as it encapsulates explicit rule-based logic and domain-specific knowledge that can be validated, debugged, and extended.

2.2.1 Classification of Program Synthesis Approaches

Program synthesis approaches can broadly be divided into two main categories:

- **Inductive Synthesis**: Infers program logic directly from input-output examples. By identifying and generalizing patterns observed in these examples, inductive approaches excel in contexts where formal specifications are difficult to construct but where plenty of training data is available.
- **Deductive Synthesis**: Constructs programs from formal logical specifications, guaranteeing correctness by design. Deductive methods typically rely on proof strategies or theorem-proving techniques to ensure that the synthesized program satisfies all requirements.

2.3 Syntax-Guided Synthesis (SyGuS)

A major advancement bridging inductive and deductive techniques is *Syntax-Guided Synthesis* (SyGuS). SyGuS incorporates syntactic constraints specified by formal grammars, along with logical/semantic constraints, to reduce the search space to only those candidate programs that are both syntactically valid and semantically correct [3].

Definition (SyGuS Problem)

A SyGuS problem is represented as a tuple $\langle T, G, \phi, F \rangle$, where:

- T: A first-order theory,
- G: A context-free grammar,
- ϕ: A logical specification,
- F: A function symbol within ϕ.

A valid solution f must satisfy $T \models \phi[F \mapsto f]$ and belong to the language defined by the grammar G.

2.3.1 Core Components of SyGuS

Grammar Definition A context-free grammar (CFG) restricts the syntactic space of potential programs:

Definition (Context-Free Grammar)

A context-free grammar G is formally defined by a tuple (T, NT, S, R), where:

- T: Set of terminal symbols,
- NT: Set of non-terminal symbols,
- $S \in NT$: The start symbol,
- $R \subseteq NT \times (T \cup NT)^*$: Production rules defining valid syntactic expansions.

Logical Constraints These constraints may be expressed in a logical formula, as input-output examples, or as invariants. Candidate programs must satisfy these constraints for semantic correctness.

Definition (Logical Constraints)

Logical constraints specify the semantic requirements that candidate programs must meet. These constraints can be defined in various forms:

- **Logical Formulas:** For example, a constraint might be $\forall x \in N,\ f(x) \geq 0$.
- **Input-Output Examples:** For instance, specifying that $f(2) = 4$ and $f(3) = 9$.
- **Invariants:** Properties that must hold true during the execution of the program, such as $\text{sum}(f(x)) \leq M$ for a given bound M.

A candidate program is considered semantically correct if it satisfies all the given logical constraints.

Probabilistic Extensions to SyGuS Probabilistic Context-Free Grammars (PCFGs) can help guide the synthesis process by attaching probabilities to production rules, steering the search toward statistically more likely candidates:

Definition (Probabilistic Context-Free Grammar)

A PCFG is defined as a tuple (T, NT, S, R, P), where:

- T, NT, S, R follow standard CFG definitions,
- $P : R \to [0,1]$ assigns probabilities to each rule, ensuring

$$\sum_{r \in R(A)} P(r) = 1, \quad \text{for all } A \in NT.$$

The probability of a derived string is the product of the probabilities of the applied rules.

2.4 Key Synthesis Methodologies

Several methodologies underpin modern program synthesis, enabling efficient exploration and validation of candidate programs:

- **Enumerative Search**: Systematically enumerates candidate programs. Although straightforward, it may suffer from combinatorial explosion. Practical systems mitigate this with heuristics, iterative deepening, and pruning [29].
- **Constraint Solving**: Translates synthesis tasks into logical constraints for SAT or SMT solvers, thereby satisfying syntactic and semantic constraints simultaneously [3].
- **Probabilistic Modeling**: Machine learning or statistical models can prioritize promising candidate programs based on learned patterns, focusing computational effort where it is most likely to yield valid solutions [22].

2.4.1 SyGuS Workflow and Iterative Refinement

SyGuS typically follows an iterative cycle of generation, checking, and refinement:

1. **Candidate Generation**: Potential solutions are constructed via enumerative search, symbolic methods, and/or probabilistic sampling (guided by PCFGs).
2. **Constraint Checking**: Automated solvers verify each candidate against syntactic and semantic requirements, discarding invalid solutions.
3. **Counterexample-Guided Inductive Synthesis (CEGIS)**: When a candidate fails, the system produces a counterexample that pinpoints the failure, pruning the search space and iterating until it converges on a correct, semantically valid solution.

2.4.2 Supporting Tools and Technologies

- **SKETCH**: A system that allows partially specified programs with "holes," which are automatically completed by a synthesis engine. Widely used for hardware design and algorithm synthesis [28].
- **Deep Learning Integration**: Neural models can guide the search process, improving synthesis accuracy and efficiency for complex domains [5].
- **CVC5**: A state-of-the-art SMT solver that offers specific support for SyGuS tasks. Its powerful constraint-solving abilities make it applicable in various formal verification and automated reasoning scenarios. [6]

Program synthesis provides a automatically generating correct, efficient code from high-level specifications. SyGuS plays a pivotal role by unifying syntactic and semantic constraints, directing the search process more efficiently. Through the combined use of enumerative search, constraint solving, probabilistic modeling, and iterative refinement (often with CEGIS), synthesis systems can tackle complex domains while retaining transparency and interpretability in the resulting programs.

2.5 Large Language Models (LLMs)

Large Language Models (LLMs) are advanced AI systems that process and generate human-like text, making them valuable for natural language tasks. Their ability to work with sparse data, particularly in specialized fields like healthcare, is a significant advantage, allowing them to adapt with minimal examples.

LLMs are deep neural networks designed to understand and generate human-like language by training on extensive text corpora. These models have revolutionized natural language processing (NLP) by setting new benchmarks in tasks such as translation, summarization, and question answering.

Research suggests that LLMs acquire predictive power regarding syntax, semantics, and ontologies inherent in human language corpora, though they may inherit biases from their training data. A survey [32] notes that LLMs have evolved from statistical models to neural models, with pre-trained language models (PLMs) showing strong capabilities in solving various NLP tasks.

Transformer Architectures and Contextual Understanding LLMs are predominantly built on transformer architectures, such as GPT and BERT, which leverage self-attention mechanisms to capture long-range dependencies and nuanced semantic relationships

within text. The transformer model [30], consists of an encoder and decoder with self-attention capabilities, enabling the model to weigh the importance of each token in a sequence. This design is critical for maintaining robust contextual understanding.

For example, BERT [14], excels in tasks like sentiment analysis and entity recognition due to its bidirectional processing of text. The ability to process and prioritize tokens underpins LLMs' success across various language tasks.

One of the significant benefits of LLMs is their capacity to perform well even when domain-specific data is sparse. Their extensive pre-training on large, diverse datasets develops a rich repository of language representations. These can be transferred to specialized domains through techniques like transfer learning and fine-tuning.

Applications in Healthcare In healthcare, where annotated data can be scarce or sensitive, LLMs demonstrate remarkable adaptability. They can be fine-tuned using minimal domain-specific examples, leveraging pre-learned language features. This process is facilitated by few-shot and zero-shot learning capabilities, allowing generalization to new tasks with little to no additional training data.

A review [16] analyzed 51 articles and found that 77% incorporated prior knowledge to augment small datasets, using methods like attention mechanisms and prompt-based learning. A scoping review of 550 studies [35] highlighted LLMs' role in diagnostics and medical writing, while [38] noted improved medical reasoning through fine-tuning with examination-style questions.

2.6 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning paradigm in which an agent learns to make sequential decisions by interacting with an environment. The primary objective is to develop a policy that maximizes the cumulative reward over time through a process of trial and error. Unlike supervised learning, which relies on labeled input/output pairs, RL depends on a reward signal to guide the learning process, enabling the agent to learn from its own actions and their consequences.

Agent-Environment Interaction and Markov Decision Processes At the core of RL is the continuous interaction between the agent and its environment. This interaction is typically modeled as a Markov Decision Process (MDP), defined by a tuple (S, A, P, R), where:

- S is the set of states,
- A is the set of actions,
- $P(s'|s,a)$ is the transition probability from state s to state s' given action a, and
- $R(s,a,s')$ is the reward received when transitioning from state s to state s' after taking action a.

At each time step, the agent observes its current state, selects an action according to its policy $\pi(a|s)$, and the environment responds by transitioning to a new state while providing a reward or penalty. This cycle of observation, action, and feedback allows the agent to iteratively refine its strategy.

A fundamental challenge in this process is the *exploration-exploitation dilemma*. The agent must balance exploitation selecting actions known to yield high rewards with exploration trying new actions that might lead to even higher long-term rewards.

Successfully managing this trade-off is critical for efficient learning.

Policy Optimization Techniques To maximize cumulative rewards, several policy optimization techniques are employed:

Value-based algorithms, such as Q-learning, focus on estimating the expected return of state-action pairs through a function often denoted as $Q(s,a)$. The policy is then typically derived by selecting the action with the highest estimated Q-value. These methods are particularly effective in discrete and relatively low-dimensional action spaces.

Policy gradient methods take a different approach by directly optimizing the policy $\pi_\theta(a|s)$ with respect to the expected return. By adjusting the parameters θ, these methods are well-suited for environments with high-dimensional or continuous action spaces, where value-based methods might struggle.

Actor-critic approaches combine the strengths of both value-based and policy gradient methods. The actor proposes actions based on the current policy, while the critic evaluates these actions by estimating a value function. This dual mechanism helps stabilize training and can accelerate convergence by providing more accurate feedback during learning.

Function Approximation and Deep Reinforcement Learning In many practical applications, the state and action spaces can be enormous or even continuous, rendering traditional tabular methods impractical. Function approximation techniques, such as deep neural networks, are employed to estimate the value functions and policies. This approach, known as deep reinforcement learning, has enabled breakthroughs in areas such as game playing, robotics, and autonomous systems by leveraging the representational power of deep learning to handle complex, high-dimensional data.

Applications in Healthcare RL holds significant promise in the field of healthcare, particularly in developing adaptive treatment strategies. For example, in the management of Type 1 diabetes, continuous glucose monitoring systems generate data that is often sparse, noisy, and subject to rapid fluctuations. RL can be applied to optimize personalized treatment plans by continuously learning from patient-specific feedback. This real-time adaptation allows the system to dynamically adjust treatment protocols based on the patient's current condition, potentially leading to improved outcomes and more effective long-term management of chronic conditions.

RL Tools and Simulated Environments A number of tools and environments have been developed to facilitate the design, testing, and benchmarking of RL algorithms. OpenAI Gym is one such widely used toolkit that provides a standard API along with a diverse set of environments, allowing researchers to implement and compare various RL approaches with ease. In this dissertation, the "simglucose" environment [36] is utilized; it simulates glucose dynamics over time in response to insulin and carbohydrate interventions. This environment is specifically designed to model the physiological responses in patients with Type 1 diabetes, offering a realistic platform for developing and evaluating adaptive treatment strategies using RL [21, 34, 1].

Chapter 3

Program Synthesis for Pattern Discovery

This chapter presents a program synthesis technique using SyGuS and the tool CVC5 that aims to address the challenge of learning from noisy and sparse health data in Type 1 diabetes management. By creating interpretable functions through a guided search process with reduced computational overhead, this approach aims to extract meaningful patterns while maintaining interpretable rules for decision-making.

3.1 Problem Definition

Problem Let $D = \{(g_i, w_i, f_i)\}_{i=1}^{n}$ be a dataset of glucose measurements g_i, workout types $w_i \in W$, and a list of features f_i. Let ϕ be the specification that defines the desired properties and constraints for the synthesized function. Given a grammar $\mathcal{G}$ for expressing relationships, synthesize a function $F(f_i, w_i) \rightarrow g_i$ that satisfies the specification ϕ and captures meaningful relationships between workout features and glucose that are true for D.

The problem is identifying and formalizing the key relationships in the dataset that meaningfully connect workout features and glucose, and synthesize interpretable functions. Using the Dataset (see Appendix A for full dataset description), a comprehensive data processing pipeline is implemented that aims to identify key relationships within the dataset. These relationships can be complex and multifaceted. Features like heart rate zones, workout duration, time of day, and pre-workout glucose levels may all influence post-exercise glucose responses. The grammar $\mathcal{G}$ provides formal constraints on how these relationships can be expressed, ensuring that the synthesized function F captures physiologically plausible and statistically significant patterns in the dataset D while adhering to the specification ϕ. In addition, the specification ϕ will be defined for each approach, but aims to capture functions that describe meaningful relationships in the data.

3.2 Data and Preliminary Experiments

Initial experiments focused on using the tool CVC5 to create a function that can be used on the data. A formal grammar was developed to guide CVC5 to create a function

that is good at classifying high/low glucose events (see Appendix B.1 for the complete CVC5 file definition). While the function created did work, it did not represent true learning since the grammar was specifically constructed to create this function.

Synthesized Function 1

```
(define-fun analyze_glucose ((glucose Int) (delta Int)
(insulin Int) (cho Int)) Int
  (ite (and (> delta 0) (> cho 0)) 3
       (ite (> insulin 0) 4
            (ite (> glucose 180) 1
                 (ite (< glucose 70) 2 0)))))
```

This function implements a decision tree that categorizes glucose readings into five patterns: normal (0), high glucose (1), low glucose (2), meal response (3), and insulin response (4).

Synthesized Function 2

```
(define-fun basic_glucose ((glucose Int) (delta Int)) Int
  (ite (> glucose 180) 1
       (ite (< glucose 70) 2 0)))
```

This function classifies glucose readings into hyperglycemia (>180 mg/dL, returns 1), hypoglycemia (<70 mg/dL, returns 2), or normal (70-180 mg/dL, returns 0) using nested if-else statements, although the `delta` parameter remains unused.

In these experiments, the problem was formalized as synthesizing a function $f : Z^7 \to \{0,1,2,3,4,5\}$ which must satisfy a specification $\phi(f) \equiv \bigwedge_{i=1}^{6} f(f_i) = g_i$. Here, the six inputs represent features - stable glucose, high glucose, low glucose, rapid rise, meal response, and insulin response - and the output corresponds to one of six glucose patterns. Initial decision-tree based functions, as demonstrated above, were insufficient for capturing the complex interactions in the dataset, indicating that while CVC5 can synthesize basic functions using explicit grammars and a small dataset, more sophisticated approaches are needed to fully model the underlying relationships.

3.3 Statistical SyGuS

The Statistical SyGuS approach aims to automate the discovery of patterns in the data in order to satisfy the central question by learning and synthesizing interpretable functions. Let $D = \{(g_i, w_i, f_i)\}_{i=1}^{n}$ represent our dataset, where g_i are blood glucose measurements, $w_i \in W$ are workout types, and f_i are corresponding workout features. Let $d_i = (g_i, w_i, f_i)$ denote a single data point within the dataset D. The statistical analyses of the dataset inform candidate variable selection for the synthesis process, directing the SyGuS algorithm toward the most promising search spaces.

This statistically-guided approach enables the synthesis of functions that accurately capture relationships between workout features and glucose responses. By leveraging statistical insights to constrain the search space, could significantly improve pattern discovery efficiency while reducing computational requirements. The potential relationships and structure are explored in Appendix A.3, which presents a comprehensive correlation analysis and characterization of the underlying relationship structures.

Overview of Statistical SyGuS

1. **Data Preprocessing**: Normalize features, impute missing values, handle outliers
2. **Statistical Analysis**: Correlation matrix, hierarchical clustering, feature mo- ments
3. **Function Signature**: Strong correlation ($|r| > 0.7$), $\phi : (f_i, f_j) \rightarrow f_k$
4. **Grammar Construction**: Linear ($|r| < 0.9$): $expr ::= c_1 x_1 + c_2 x_2 -$ Non-$linear : \mu < 0 \Rightarrow x_1 < x_2$
5. **Constraints**: Value bounds: $\mu \pm 36 \rightarrow$ Monotonic : $r < 0 \Rightarrow x_1 < x_2$
6. **Synthesis**: Execute CVC5 with grammar + constraints (timeout: 240s)
7. **Analysis**: Check if functions generated are useful

3.3.1 Statistical Analysis

The statistical analysis process encompasses three integrated stages: data preprocessing, correlation analysis, and hierarchical clustering, each contributing directly to the SyGuS framework. This process is conducted across all data points within the dataset.

Data preprocessing ensures dataset quality through normalization of numerical variables to standardized ranges, imputation of missing values using mean substitution, and removal of outliers with z-scores exceeding the absolute value of 3. These steps directly inform SyGuS constraint bounds through normalized ranges, creating a more robust synthesis process aligned with inherent data distributions.

Correlation analysis reveals relationships between variables by computing the Pearson correlation matrix, with significant associations identified where absolute correlation coefficients exceed 0.5. These insights determine synthesis priorities and influence grammar complexity in SyGuS. Strongly correlated variables receive prioritization in the synthesis process, while the nature of relationships (linear or non-linear) guides grammar construction to accurately reflect underlying data patterns.

The final hierarchical clustering stage organizes variables into meaningful groups based on similarity, beginning with a correlation distance matrix followed by cluster formation using Ward's method to minimize within-cluster variance. Clustering validity is confirmed through silhouette score calculation, measuring how well objects align with their assigned clusters. Cluster membership directly informs variable grouping during synthesis, treating variables within the same cluster as cohesive units and ensuring synthesized functions respect inherent data relationships, ultimately enhancing model accuracy and interpretability.

3.3.2 SyGuS Translation

The aforementioned data processing and analysis stages collectively inform the generation of SyGuS specifications. By preprocessing the data, analyzing correlations, and clustering variables, we establish a robust foundation that guides the selection of relevant variables, the construction of appropriate grammars, and the formulation of meaningful constraints. This integration ensures that the synthesized functions are not only mathematically sound but also aligned with the underlying data structures and relationships, ultimately leading to more effective and interpretable solutions. The statistical analysis informs three key aspects of the SyGuS specification: variable selection,

grammar construction, and constraint generation. Each step is guided by correlations and clustering insights from the data.

3.3.3 Function Signature Determination

Correlation strength dictates the function signature for synthesis, ensuring that significant relationships are captured while remaining tractable.

Strong Correlations ($|r_{jk}| > 0.7$) Let $\mathbf{f}_i = (f_i[1], f_i[2], \ldots, f_i[m])$ be the feature vector at data point i ($i = 1, \ldots, N$), and let $f_i[j]$ be its j^{th} element. If $f_i[j]$ and $f_i[k]$ are strongly correlated (i.e., $|r_{j,k}| > 0.9$), we seek a function F satisfying

$$\bigwedge_{i=1}^{10} \Big(F(f_i[j]) = F(f_i[k]) \Big).$$

For instance, if maximum heart rate (f_j) strongly correlates with glucose response (f_k), then F predicts f_k solely from f_j.

Moderate Correlations ($0.5 < |r_{jk}| < 0.7$) Let $\{j_1, j_2, \ldots, j_p\}$ be the indices of features moderately correlated with f_k. Define F as

$$F : (f_i[j_1], f_i[j_2], \ldots, f_i[j_p]) \mapsto f_i[k],$$

subject to

$$\bigwedge_{i=1}^{10} \Big(F(f_i[j_1], f_i[j_2], \ldots, f_i[j_p]) = f_i[k] \Big).$$

For example, if workout duration (f_{j_1}) and average glucose (f_{j_2}) moderately correlate with f_k, F combines these inputs to predict f_k.

3.3.4 Grammar Construction

The grammar G in our SyGuS approach provides a formal language for expressing physiological hypotheses. Function signatures derived from correlation analysis directly inform grammar tier selection, ensuring appropriately expressive production rules that match relationship complexity while maintaining physiological plausibility. These production rules constrain the synthesized function F to plausible forms supported by dataset D through a three-tiered grammar system based on statistical evidence.

Grammar Complexity Tiers

1. **Simple Grammar**: For strongly linear correlations ($|r| > 0.9$)
2. **Medium Grammar**: For moderate correlations with potential non-linear components ($0.5 < |r| < 0.9$)
3. **Complex Grammar**: For weak or complex correlations requiring sophisticated modeling

```
<expr> ::= <const>
         | <var>
         | <const> * <var>
         | <expr> + <expr>
         | <expr> - <expr>
         | <expr> * <expr>
         | abs(<expr>)

<var> ::= duration_minutes | avg_hr | hr_zone_hard | glucose_before_60min
         | ... | any input variable

<const> ::= 0.0 | 1.0 | 2.0 | 0.5 | 0.1 | 0.01 | 0.001 | -<const>
```

Simple Grammar Tier (Strong Linear Correlations, $|r| > 0.9$) This grammar captures linear relationships and basic interactions between variables. It's suitable for strongly correlated physiological variables where the relationship follows a linear pattern or simple multiplicative effects.

```
<expr> ::= <term>
         | <expr> + <expr>
         | <expr> - <expr>
         | if <bool> then <expr> else <expr>

<term> ::= <const>
         | <var>
         | <term> * <term>
         | <const> * <term>
         | abs(<term>)

<bool> ::= <expr> >= <expr>
         | <expr> <= <expr>
         | <expr> >= <const>
         | <expr> <= <const>

<var> ::= duration_minutes | avg_hr | hr_zone_hard | glucose_before_60min
         | ... | any input variable

<const> ::= 0.0 | 1.0 | 2.0 | 0.5 | 0.1 | 0.01 | 0.001 | -<const>
```

Medium Grammar Tier (Moderate Correlations, $0.5 < |r| < 0.9$) This grammar extends the simple grammar with conditional expressions, enabling threshold-based behavior common in physiological systems. It can represent relationships that change qualitatively at certain threshold values (such as anaerobic threshold).

```
<expr> ::= <term>
         | <expr> + <expr>
         | <expr> - <expr>
         | <expr> * <expr>
         | <expr> / <expr>
         | if <bool> then <expr> else <expr>

<term> ::= <const>
         | <var>
         | <term> * <term>
         | <const> * <term>
         | <term> / <const>
         | abs(<term>)

<bool> ::= <expr> >= <expr>
         | <expr> <= <expr>
         | <expr> >= <const>
         | <expr> <= <const>
         | <bool> and <bool>

<var> ::= duration_minutes | avg_hr | hr_zone_hard | glucose_before_60min
         | ... | any input variable

<const> ::= 0.0 | 1.0 | 2.0 | 0.5 | 0.1 | 0.01 | 0.001 | -<const>
```

Complex Grammar Tier (Weak or Complex Correlations, $|r| < 0.7$) This grammar provides the highest expressiveness, adding division, nested arithmetic operations, and compound boolean expressions. It can capture complex physiological mechanisms like feedback loops, ratio-based relationships, and multi-threshold behaviors.

3.3.5 Constraint Generation

For each data point $d_i = (g_i, w_i, f_i)$, we generate constraints by selecting the relevant features from f_i (where x denotes the list of features). The constraints are designed to reflect the inherent characteristics of the data while being robust to the noise and sparsity typical of health data.

Statistical Moments Value bounds are defined using statistical thresholds to prevent outputs that deviate significantly from observed data ranges. Let μ and σ be the mean and standard deviation of the target feature y (e.g., g_i). Then, for each d_i, the constraint is given by:

$$\langle \text{constraint-bound} \rangle ::= \mu - 3\sigma \leq F(x) \leq \mu + 3\sigma,$$

where x represents the list of features extracted from f_i.

Correlation Signs Monotonicity constraints enforce that if two features are positively correlated, the function $F(x)$ should not decrease as the correlated feature increases. Formally, for any two feature values x_1 and x_2 (selected from x) with $x_1 > x_2$ and a positive correlation ($r_{x_1 y} > 0$), we require:

$$\langle \text{monotonicity-constraint} \rangle ::= x_1 > x_2 \implies F(x_1) \geq F(x_2).$$

Cluster Memberships Clustering groups features that behave similarly, ensuring consistent function outputs within these groups to reduce unnecessary variability. If $\{x_i, x_j\}$ belong to the same cluster, the constraint is:

$$\langle \text{cluster-constraint} \rangle ::= F(x_i, x_j) = F(x_j, x_i),$$

with x_i and x_j being features from the list x for d_i.

These constraints are deliberately relaxed to accommodate the noise and sparsity of health data, allowing the solver to find viable solutions while preserving realistic physiological boundaries. The following example shows an input file constructed by Stat SyGuS for the CVC5 solver, where each data point d_i contributes its own constraint specification based on the selected features.

Average Heart Rate from Calories Grammar

```
(synth-fun avg_hr_from_calories ((calories Real) ) Real
    ((Start Real (
        (Constant Real)
        (Variable Real)
        (+ Start Start)
        (* Start Start)
        (* Start (* Start Start))
    ))))
(constraint (= (avg_hr_from_calories 1515.0 ) 138.743))
(constraint (= (avg_hr_from_calories 574.0 ) 143.778))
(constraint (= (avg_hr_from_calories 155.0 ) 111.41))
(constraint (= (avg_hr_from_calories 2307.0 ) 151.886))
(constraint (= (avg_hr_from_calories 987.0 ) 132.283))
(constraint (= (avg_hr_from_calories 621.0 ) 148.524))
(constraint (= (avg_hr_from_calories 2185.0 ) 159.751))
(constraint (= (avg_hr_from_calories 1231.0 ) 150.249))
(constraint (= (avg_hr_from_calories 970.0 ) 147.885))
(constraint (= (avg_hr_from_calories 1144.0 ) 149.897))
(check-synth)
```

Explanation: This specification aims to synthesize a function that predicts the average heart rate based on calories burned during an activity. The grammar allows for constants, variables, addition, multiplication, and nested multiplication. The constraints provide example input-output pairs from real health data.

For a comprehensive overview of the technical specifications derived from this process, please refer to Appendix B.2. The appendix contains more detailed technical specifications that were used as inputs for the CVC5 solver.

3.4 Results

Using the aforementioned process layout to constrain the search space, before employing CVC5 as the synthesis engine. Despite these attempts, the engine was unable to produce meaningful results within the allocated 240-second timeout period. Although numerous functions were generated, these functions were largely trivial and did not capture the complex interactions inherent in the dataset. The constraint tolerances implemented - necessary to ensure the synthesis process would complete - resulted in oversimplified representations that failed to reflect the true, relationships in the data.

The following examples show functions that predict glucose levels, along with their limitations. In the first example, the function estimates the glucose level at 120 minutes based on measurements taken at 30, 60, and 90 minutes.

Example 1 (Temporal Prediction)

```
(define-fun func_one
((g30 Real) (g60 Real) (g90 Real)) Real
(+ (* 2.0 g90) (* (/ 1 2) (* g60 g30))))
```

This function uses a basic linear formula to combine the measurements. However, it has several limitations. It ignores more complex relationships between the readings, uses fixed coefficients without a clear basis, and does not take into account the sequence of the measurements. Furthermore, it does not adjust for individual differences.

The second example predicts the maximum glucose level during exercise using the mean and minimum glucose levels.

Example 2 (During Exercise)

```
(define-fun func_two
((mean Real) (min Real)) Real
(* mean (+ 1.0 (* 0.5 (/ min mean)))))
```

In this example, the relationship between the mean and maximum levels is overly simplified. The formula uses a fixed scaling factor that may not capture the natural variability observed during exercise.

The next example focuses on predicting a heart rate zone based on average heart rate, maximum heart rate, and an indicator for a higher intensity zone.

Example 3 (Zone Prediction)

```
(define-fun func_three
((avg_hr Real) (max_hr Real) (hr_zone_hard Real)) Real
(* avg_hr (+ (* 1.0 hr_zone_hard) (* (/ 1 2) max_hr))))
```

This function uses a simple formula to estimate the heart rate zone. It does not incorporate established boundaries or transitions between zones, and relies on fixed coefficients that do not fully capture the gradual changes in heart rate responses. The implemented data preprocessing steps offered some improvement over initial experiments, which had relied solely on raw time series data processed alongside a predefined grammar. Structuring the data and constraining the search space before employing the CVC5 synthesis engine marginally increased the engine's capacity to generate functions within the specified 240-second timeout period. However, these refinements were insufficient to overcome broader limitations, as the resulting models remained overly simplistic, failing to adequately capture the intricate dynamics of the dataset.

These experimental results highlight substantial challenges in applying program synthesis tools to complex, noisy datasets, especially in health-related contexts. Although various functions were generated, the approach consistently produced overly simplistic models incapable of capturing the nuanced interactions inherent in the data. The primary limitation stems from the difficulty in effectively constraining a vast search space coupled with the inherent complexity of biological systems in health data. The rigid constraints imposed by the synthesis framework inadequately represent multifaceted temporal and physiological dynamics, ultimately proving incapable of accurately modeling these subtle complexities. Consequently, the approach appears poorly suited for this domain, where data noise and complexity exceed the current capabilities of the synthesis framework. This research underscores the impracticality of directly translating statistical insights into SyGuS specifications for health-related data, serving primarily as an exploratory analysis rather than a practical solution.

Overall, these experiments clearly demonstrate that the approach employing Statistical SyGuS and the CVC5 synthesis engine does not sufficiently address the problem defined earlier in this chapter, nor does it achieve the overarching objective outlined in the introduction. Despite leveraging statistical analysis and grammar-guided synthesis to manage search complexity, the resulting functions were overly simplistic and unable to effectively model complex physiological interactions, particularly those relevant to managing Type 1 diabetes. Thus, this synthesis approach does not provide truly interpretable solutions capable of enhancing clinical understanding or improving patient outcomes, emphasizing significant limitations of current program synthesis methods when applied to real-world health data.

Chapter 4

LLM-Based Rule Generation and Validation

Given the limitations identified in the previous chapter concerning Statistical SyGuS and traditional program synthesis methods in handling complex, noisy health datasets, this chapter proposes an alternative approach leveraging large language models (LLMs). Specifically, it examines the potential of LLMs to generate verifiable grammar rules directly from datasets, aiming to replicate the benefits of formal synthesis while overcoming its constraints. The proposed framework integrates data-driven grammar learning with LLM-driven rule generation, facilitating structured data validation. By utilizing LLMs, this method aims to derive rules that ensure data integrity with outputs that are both interpretable and verifiable, achieving a balance between synthesis efficiency and the formal guarantees typically provided by program synthesis.

4.1 Problem Definition

Problem Let $D = \{(g_i, w_i, f_i)\}_{i=1}^{n}$ be a dataset of blood glucose measurements g_i, workout types $w_i \in W$, and workout features f_i. Synthesize a set of interpretable rules R^* such that each rule $r \in R^*$ accurately describes the relationship between workout characteristics and blood glucose responses that are true for D.

The primary challenge lies in developing effective prompt engineering functions capable of translating glucose-workout datasets into suitable inputs for guiding LLMs in discovering meaningful, accurate patterns. The resulting rule set R^* should clearly describe how various workout features - including intensity, duration, and timing - influence blood glucose responses across different exercise types. These rules must ensure syntactic correctness and semantic relevance, providing insights comparable to traditional formal synthesis methods but with improved scalability and accessibility. Furthermore, the framework incorporates automated validation mechanisms to rigorously verify the generated rules against the data, distinguishing genuine dataset relationships from spurious correlations or model hallucinations.

4.2 System Architecture

The system architecture employs a DeepSeek-r1 14B language model to implement a structured workflow for rule generation and validation. The process begins with dataset ingestion, follows through prompt creation and rule generation according to a grammar, and culminates in systematic validation of each rule against test data. This end-to-end pipeline enables automated discovery and verification of meaningful relationships between workout characteristics and glucose responses, while maintaining logical consistency through formal validation procedures (see Algorithm C.1 in Appendix).

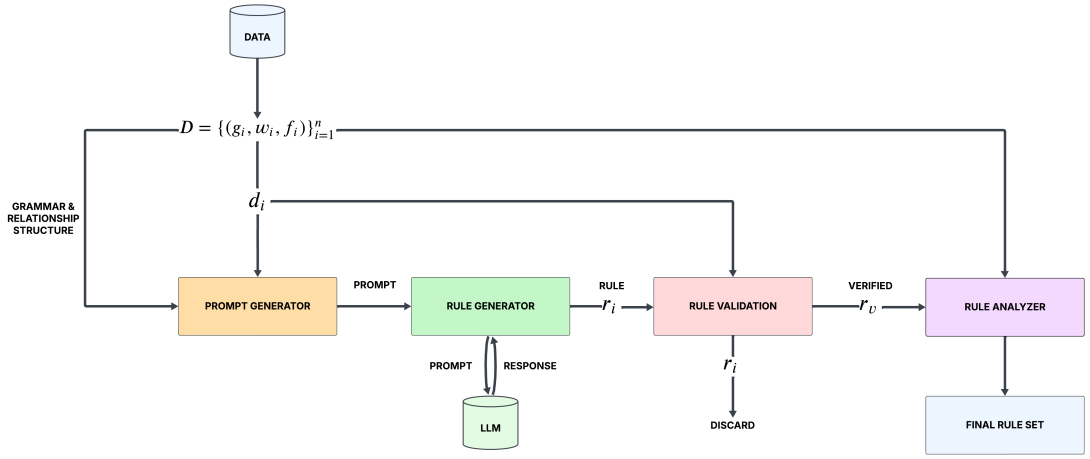


Figure 4.1: The diagram depicts the DeepSeek-r1 14B verification pipeline from CSV inputs through prompt generation, rule creation, test validation, and pattern analysis, showing the transformation of raw data into actionable insights.

4.2.1 Prompt Generation

The prompt generation process involves analyzing input data, structuring a partial grammar, and learning underlying patterns to represent the dataset's properties. This section provides a detailed breakdown of each step.

Data Analysis The process begins with a comprehensive analysis of the input dataset to uncover its foundational characteristics. The analysis focuses on the following objectives:

- **Identify Column Types**: Classify each column as `numeric`, `categorical`, `datetime`, or another type based on its content.
- **Extract Statistical Properties**: Compute metrics such as mean, median, and standard deviation to describe the data's distribution.
- **Detect Correlations**: Quantify relationships between variables using statistical methods to reveal interdependencies.
- **Learn Structural Patterns**: Identify recurring structures, such as dependencies or hierarchical relationships, within the dataset.

Grammar & Relationship Strucutre The grammar is encoded in JSON format, capturing both column-specific details and inter-column relationships. The schema below outlines the structure, followed by detailed explanations of its components:

- **Columns**: Each column is described by the following attributes:

  - **Name**: A unique identifier (e.g., `activity_type`, `duration_minutes`).
  - **Data Type**: The column's type (e.g., `numeric`, `categorical`, `datetime`).
  - **Value Constraints**: For numeric columns, the range of permissible values (e.g., `duration_minutes` spans $[15.0, 120.0]$).
  - **Unique Values**: For categorical columns, the set of possible values (e.g., `activity_type`: `["Running", "Cycling", "Swimming", "Walking", "HIIT"]`).
  - **Required Flag**: A boolean indicating whether null values are allowed (e.g., `glucose_after_60min`: `false`).
- **Relationships**: Inter-column connections are captured as:
  - **Dependencies**: Functional dependencies where one column determines another (e.g., `activity_type` influences `hr_zone_moderate`).
  - **Correlations**: Numeric relationships above a threshold (e.g., `calories` correlates with `duration_minutes` at 0.85).

Grammar & Relationship Strucutre

```
{
  "columns": {
    "column_name": {
      "type": "data_type",
      "unique_values": ["value1", "value2", ...],
      "min_value": min_value,
      "max_value": max_value,
      "required": true/false
    },
    ...
  },
  "relationships": {
    "dependencies": {
      "column_A": ["column_B", "column_C", ...],
      ...
    },
    "correlations": {
      "column_X": [
        ["column_Y", correlation_value],
        ...
      ],
      ...
    }
  }
}
```

Grammar Learning Process The grammar learning process is defined as a function G that maps D to a tuple of type inferences, range constraints, and a correlation matrix:

$$G : D \to (T, R, C)$$

where:

- $T = \{t_1, \ldots, t_m\}$: Inferred data types for each feature (e.g., `categorical` for `activity_type`, `numeric` for `calories`).
- $R = \{r_1, \ldots, r_k\}$: Range constraints for numeric features (e.g., `[15.0, 120.0]` for `duration_minutes`), where $k \leq m$ is the number of numeric columns.
- $C = (c_{ij})$: The correlation matrix, where c_{ij} is the correlation between features i and j (e.g., 0.85 between `calories` and `duration_minutes`).

Type Inference (T) Each $t_j \in T$ is inferred through a systematic process:

- **Datetime Detection**: Parse columns for datetime patterns (e.g., `start_time` is `datetime`).
- **Numeric Detection**: Identify columns convertible to numeric types (e.g., `duration_minutes`, `calories`).

- **Categorical Detection**: Recognize columns with a finite set of values (e.g., `activity_type` via `pandas.is_string_dtype`).
- **Other**: Assign remaining columns to a catch-all category (typically unused here).

Range Constraints (R) For each numeric column k, the range constraint is:

$$r_k = [\min_k, \max_k]$$

where:

- $\min_k$: The column's minimum value, computed as `values.min()` after excluding nulls (e.g., 100.0 for `calories`).
- $\max_k$: The column's maximum value, computed as `values.max()` after excluding nulls (e.g., 800.0 for `calories`).

Relationship Analysis (C) This step identifies two types of relationships:

1. **Dependencies**: Detect functional dependencies where column A determines column B if each value in A maps to a unique value in B (e.g., `activity_type` to `hr_zone_moderate`).
2. **Correlations**: Record pairwise correlations for numeric columns exceeding a threshold (e.g., 0.7), such as 0.85 between `calories` and `duration_minutes`.

4.2.2 Rule Generation

The rule generation process builds on the prompt definition phase, leveraging large language models (LLMs) to synthesize meaningful validation rules for each workout. These rules encapsulate logical expressions that reflect inherent relationships and constraints within the features.

Let R_D represent a set of data point rule pairs defined as:

$$R_D = \{(d_i, r_i) \mid i = 1, 2, \ldots, n\},$$

where each rule r_i corresponds to a data point $d_i \in D$ and is defined as:

$$r_i = (A_{s_i}, C_i).$$

Definitions:

- $s_i \subseteq \{1, 2, \ldots, m\}$: The subset of features selected for workout d_i.
- $A_{s_i} = \bigwedge_{j \in s_i} \left(f_i^j = v_i^j \right)$: The **antecedent conditions** formed over the selected features, where:
  - f_i^j represents the j-th feature for workout d_i,
  - v_i^j is the value of f_i^j.
- C_i: The **consequent condition** associated with the workout.

The rule generation phase uses LLMs to transform data-driven patterns into coherent logical rules that represent dataset characteristics. Structured outputs enforce a predefined schema that minimizes ambiguity and improves rule quality. The LLM's prompt combines dataset-derived grammar (including column types, constraints, and relationships) with the structured output schema (see Appendix C.2 for the full prompt

and structured output). This approach helps generate rules that align with the dataset's structure while maintaining consistent logical format. Processing data per workout allows the LLM to capture local patterns. Using the 'deepseek-r1:14b' model via Ollama's chat interface, the system frames the LLM as an "expert exercise physiologist and sports scientist" with guidelines on rule construction, formatting, and reference ranges. The user prompt provides workout-specific data categories (time, performance, heart rate, glucose metrics) and historical context of previous rules to encourage novelty. Additional constraints guide the rule structure, including activity type requirements, precision values, and logical operators (ALL, IMPLIES, AND, OR, NOT). This grammar-informed, structured approach produces logically sound rules that reflect both the dataset's broad structure and specific details, making them valuable for data validation and decision-making in real-world scenarios.

4.2.3 Validation Process

The validation process systematically evaluates each data point rule pair to ensure both structural and semantic correctness.

For a data point $d_i \in D$ and its corresponding rule r_i, the validation function V is applied as follows:

$$V(d_i, r_i) = \begin{cases} \text{True} & \text{if } \neg\phi(d_i), \\ \text{True} & \text{if } \phi(d_i) \wedge \psi(d_i), \\ \text{False} & \text{if } \phi(d_i) \wedge \neg\psi(d_i). \end{cases}$$

- $\phi(d_i)$: Evaluates whether the antecedent conditions are satisfied for workout d_i.
- $\psi(d_i)$: Evaluates whether the consequent condition holds for workout d_i.

A rule r_i is considered valid for data point d_i if $V(d_i, r_i) = \text{True}$. The set of valid workout-rule pairs R_v consists of all pairs (d_i, r_i) where $V(d_i, r_i) = \text{True}$.

Additional checks ensure that the rules align with domain-specific constraints by validating physiological limits, such as permissible ranges for features like glucose levels and heart rates, based on the minimum and maximum values observed in the dataset. Additionally, rule consistency is maintained by ensuring that rules generated from overlapping subsets do not produce conflicting outcomes when applied to the same data points, thereby preserving the integrity of the rule set across the entire dataset.

By validating rules on subsets of D, this process ensures that the rules are robust and applicable to diverse real-world scenarios without being overly restrictive or specific.

4.2.4 Rule Analysis

To evaluate the global performance of each rule across the entire dataset D, we apply the rule validation function globally. Given a rule r_i and the dataset $D = \{(g_i, w_i, f_i)\}_{i=1}^{n}$, the global rule validation function is defined as $V_g(r_i, D) = \frac{1}{n}\sum_{j=1}^{n} I(V((g_j, w_j, f_j), r_i) = \text{True})$. Here, $I(\cdot)$ is the indicator function that returns 1 if the validation result is true and 0 otherwise, while n is the total number of data points. The output of V_g represents the percentage of valid applications of the rule across the dataset, indicating the rule's overall accuracy. This global analysis ensures that the rules, originally validated locally, also demonstrate robustness and reliability when generalized to the entire dataset.

4.3 Results

The generated logical rules, derived using the Deep Seek R1 14b model on an 80%-20% train-test split, reveal both category-specific patterns and common variable dependencies. While each rule is carefully tailored to capture the unique dynamics of its respective exercise category, the overall analysis also highlights a repeated reliance on several core features. This suggests opportunities for further exploration by incorporating a broader range of variables. Below are a few examples of the extracted rules:

Functional Strength Training Pattern Rule

```
IF activity_type = 'Functional Strength Training' AND
(duration_minutes > 40 OR calories > 350) AND
(hr_zone_easy < 0.1 OR hr_zone_fat_burn > 0.2)
THEN glucose_during_mean <= 7.8
```

Explanation: Longer duration or higher calorie burn sessions, especially when the fat-burning zone is substantially engaged, are associated with lower mean glucose during exercise.

Cycling Pattern Rule

```
IF activity_type = 'Cycling' AND
(duration_minutes > 90 OR avg_hr >= 150) AND
hr_zone_moderate > 0.3
THEN glucose_during_mean < 7.0
```

Explanation: High-intensity cycling—either sessions lasting more than 90 minutes or those with elevated average heart rate—coupled with significant moderate zone usage, leads to lower mean glucose levels during exercise.

Walking Pattern Rule

```
IF activity_type = 'Walking' AND
(average_heartrate_bpm > 120 OR glucose_during_mean >= 8.5)
THEN glucose_after_60min > 9
```

Explanation: Elevated heart rate during walking or high mean glucose levels during the activity may result in higher post-exercise glucose levels.

Golf Pattern Rule

```
IF activity_type = 'Golf' AND
(distance_km >= 8.0 OR duration_minutes > 120) AND
(avg_hr > 100 OR max_hr > 105)
THEN glucose_during_mean <= 7.5
```

Explanation: For golf, covering longer distances or engaging in extended sessions, along with higher heart rate metrics, appears to help in maintaining controlled mean glucose levels.

Running Pattern Rule

```
IF activity_type = 'Running' AND
((avg_hr > 150 OR duration_minutes >= 20) AND
hr_zone_hard > 0.5)
THEN glucose_during_max <= 8.1
```

Explanation: In running activities, a combination of high intensity (indicated by either a high average heart rate or sufficient duration) and extensive time spent in the hard heart rate zone helps mitigate maximum glucose spikes.

Category	**Train Dataset**				**Test Dataset**				**Difference**			
	Total	Rules	Passed	Failed	Total	Rules	Passed	Failed	Total	Rules	Passed	Failed
Cycling (Indoor)	22	14	12	2	8	14	12	2	14	0	0	0
Hiking	4	3	3	0	1	3	0	3	3	0	3	-3
Traditional Strength Training	1	1	1	0	0	0	0	0	1	1	1	0
Golf	1	1	1	0	0	0	0	0	1	1	1	0
Cycling	106	81	78	3	23	81	63	18	83	0	15	-15
Other	19	16	15	1	2	16	10	6	17	0	5	-5
Running	14	12	12	0	2	12	12	0	12	0	0	0
Walking	139	109	99	10	24	109	99	10	115	0	0	0
Functional Strength Training	44	34	29	5	10	34	29	5	34	0	0	0

Table 4.1: Detailed Validation Summary Differences per Category

As detailed in Table 4.1, the training set generally contains more records than the test set (e.g., Cycling has 106 versus 23 records, and Walking has 139 versus 24, corresponding to differences of 83 and 115 records, respectively). Although the number of rules applied remains consistent across both sets, differences in the Passed and Failed columns (such as a shift of 15 passed and -15 failed for Cycling, and 3 passed and -3 failed for Hiking) indicate that variations in record counts contribute to discrepancies in validation outcomes. Overall, the consistent rule generation and balanced performance metrics across both datasets confirm that the model fits the data well.

Category	Total Rules	Unique Rule Text	Unique Rule Types	Unique Antecedent Variables	Unique Consequent Variables
Walking	109	105	1	43	9
Functional Strength Training	34	32	2	26	4
Cycling	81	80	2	47	11
Cycling (Indoor)	14	12	1	21	7
Other	16	15	1	12	4
Running	12	12	1	10	4
Hiking	3	3	1	8	1
Golf	1	1	1	6	1
Traditional Strength Training	1	1	1	8	1

Table 4.2: Per-Category Metrics

The logical rules exhibit considerable diversity across different exercise categories. For instance, the Walking category alone contributes 109 total rules with 105 unique rule texts, drawing on 43 distinct antecedent variables and 9 consequent variables. Similarly, categories such as Cycling and Functional Strength Training reveal unique combinations of rule texts and variables, with Cycling employing 47 unique antecedent variables and 11 consequent variables, and Functional Strength Training using 26 antecedent and 4 consequent variables. These variations indicate that the logical rules capture a wide spectrum of relationships and interactions specific to each form of exercise.

However, a closer examination of the aggregated frequency counts reveals an overall limitation in variable diversity. Common antecedent variables such as `activity_type`, `calories`, and `glucose_during_mean` are leveraged extensively, appearing 254, 174,

Antecedent Variable	**Frequency**	**Consequent Variable**	**Frequency**
activity_type	254	glucose_after_60min	109
calories	174	glucose_during_mean	65
glucose_during_mean	104	outcome	25
duration_minutes	103	glucose_during_min	15
Walking	101	calories	14
Cycling	91	glucose_during_max	7
avg_hr	73	glucose_post_30min	5
hr_zone_easy	43	glucose_after_30min	4
glucose_during_min	39	average_heartrate_bpm	2
heart_rate_avg	33	BETWEEN	2

Table 4.3: Most used antecedent and consequent variables.

and 104 times, respectively. On the consequent side, variables like `glucose_after_60min` (109 times) and `glucose_during_mean` (65 times) dominate the distribution. This repeated reliance on a few core variables suggests that although the rules are diverse in format and tailored to specific exercise categories, the underlying variable selection remains narrow. Consequently, the model may be under-exploiting the broader range of available features, potentially overlooking alternative relationships that could be uncovered by incorporating less frequently used variables. Expanding the variety of variables involved in rule generation could enhance the depth and insight of the logical framework, providing a more comprehensive understanding of the complex interplay among dataset features.

Despite these limitations, the LLM-based rule generation and validation system demonstrates significant potential in autonomously deriving meaningful grammar rules from structured datasets, effectively mirroring the verifiable solutions promised by traditional program synthesis methods. By integrating a data-driven grammar learning process with the generative capabilities of a large language model, the framework successfully uncovers relationships within the data, evidenced by the diverse set of rules tailored to specific exercise categories like Walking, Cycling, and Functional Strength Training. Moreover, the validation process ensures both syntactic correctness and semantic robustness, as reflected in consistent performance metrics across training and test datasets (Table 4.1). This ability to generate and validate rules aligned with domain-specific constraints - such as physiological limits for glucose levels and heart rates - underscores the system's potential for practical application, offering a scalable and efficient alternative to manual rule crafting.

Nevertheless, the effectiveness of this approach is tempered by its primary methodological limitation: reliance on a large language model pretrained on extensive external data. This means the system integrates substantial prior knowledge rather than exclusively learning from the sparse and noisy health data provided. Consequently, while the generated rules offer valuable insights tailored specifically to managing Type 1 diabetes, they do not represent genuinely independent learning from sparse datasets alone. Future research should therefore explore balancing pretrained knowledge with methods explicitly designed for novel pattern discovery from limited data.

Chapter 5

Reinforcement Learning for Synthesizing Logical Policies

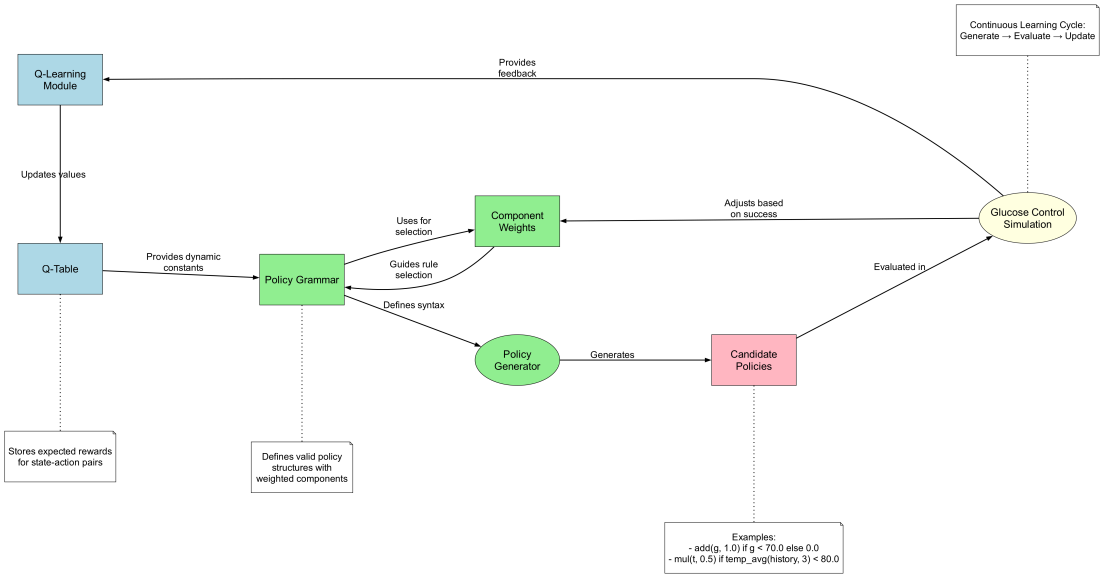


Figure 5.1: A hybrid system combining Q-Learning with probabilistic grammar for insulin dosing policy generation. The system uses simulation feedback to continuously update both Q-values and grammar weights, producing interpretable and effective diabetes management policies.

This chapter presents a reinforcement learning (RL) approach designed to address key limitations identified in previous methodologies - specifically, the extensive reliance on pretrained knowledge in LLM-based methods and computational inefficiencies in program synthesis. By employing iterative trial-and-error interactions within a simulated diabetes environment, this RL framework actively learns logical insulin dosing policies without dependence on external pretrained models. Combining probabilistic grammar with Q-learning facilitates dynamic refinement of rule complexity and performance, ensuring that generated policies remain interpretable while optimizing critical clinical outcomes such as time-in-range (TIR). Although this RL method is less suited for sparse historical datasets used in previous chapters, its strength lies in synthesizing structured, simulation driven policies that can potentially improve upon human decision-making.

5.1 Problem Definition

Problem Let E be a glucose-insulin simulation environment that models diabetes dynamics. Given a probabilistic grammar G that can express logical rules and a Q-learning framework Q, synthesize an interpretable insulin dosing policy p^* that maximizes time-in-range (TIR) glucose metrics while maintaining rule simplicity when deployed in E. The policy must be generated through iterative refinement of both grammar components and Q-values based on simulation feedback, producing structured guidelines that improve upon human decision-making while maintaining interpretability.

This approach addresses limitations found in previous methods by combining reinforcement learning with probabilistic grammar to generate diverse, effective policies. Unlike program synthesis which requires excessive computation time, and LLM-based approaches which rely on pretrained knowledge rather than genuine learning from the data, this method learns through trial-and-error interactions with a diabetes simulator. The system continuously updates both its understanding of state-action relationships (via Q-learning) and the probabilities of grammar components based on their performance in the simulation environment, leading to increasingly optimized policies for diabetes management that remain human-interpretable.

To evaluate this potential, experiments were conducted in two phases:

1. **Simulator-Based Evaluation:** A controlled simulation environment was used to test whether the RL agent, through repeated trial and error, can generate logical policies under idealized conditions.
2. **Simulation + Historical Data Evaluation:** In this phase, the RL framework first learns by observing "human" behavior-specifically, a hand-written policy representing fake human interactions. The system analyzes the decisions made, including the successes and mistakes within this environment, allowing it to develop a human-informed policy. This refined policy is then deployed in a real-time simulator, where it is further tested and adjusted to try to improve upon human decision making.

It should be noted that this RL approach cannot be effectively used on the dataset from the previous two chapters due to its sparseness, and thus has not been employed for that purpose.

5.2 Overview of RL Synthesis

Each episode follows a generate-evaluate-update cycle. First, candidate policies are generated using the weighted grammar. These are then evaluated in a glucose simulation environment that models insulin-glucose dynamics. Policies are scored based on a weighted combination of reward, time in target glucose range, and safety metrics, with hypoglycemia heavily penalized. The best-performing policies contribute to both grammar weight updates and Q-table refinement. Over time, the system converges toward policies that are fairly stable, but further improvement is needed. The synthesis process iteratively refines both the policy grammar and Q-table. For details on the implementation, please refer to the algorithm outline in Appendix D.1

Q-Learning Component The Q-learning module maintains a state-action mapping that captures the expected rewards for different insulin doses across glucose states. We discretize the continuous glucose space into 5 mg/dL bins from 50-250 mg/dL, and insulin actions into 0.25U increments, creating a manageable state-action space.

State Representation: $\text{state\_index} = \min(\max(\lfloor \frac{\text{glucose}-50}{5} \rfloor, 0), N_s - 1)$

Action Representation: $\text{action\_index} = \min(\max(\lfloor \frac{\text{insulin}}{0.25} \rfloor, 0), N_a - 1)$

Reward Calculation: The reward function is clinically informed and combines several components to balance glycemic control with safety:

$$r(s_t, a_t, s_{t+1}) = r_g(s_{t+1}) + 0.5 \cdot r_{\Delta g}(s_t, s_{t+1}) - r_{\text{hypo}}(s_{t+1}) - r_{\text{hyper}}(s_{t+1})$$

The individual reward components are calculated as follows:

- **Target Range Reward** (r_g): Rewards glucose values within the clinically recommended range:

$$r_g(s_{t+1}) = \begin{cases} 1 & \text{if } 70 \leq g_{t+1} \leq 180 \text{ mg/dL} \\ 0 & \text{otherwise} \end{cases}$$

- **Glucose Improvement Reward** ($r_{\Delta g}$): Rewards movement toward the ideal glucose target (125 mg/dL):

$$r_{\Delta g}(s_t, s_{t+1}) = \begin{cases} \frac{|g_t - 125| - |g_{t+1} - 125|}{|g_t - 125|} & \text{if improving} \\ -\frac{|g_{t+1} - 125| - |g_t - 125|}{|g_t - 125|} & \text{if worsening} \end{cases}$$

- **Hypoglycemia Penalty** (r_{hypo}): Introduces increasingly severe penalties for low blood sugar:

$$r_{\text{hypo}}(s_{t+1}) = \begin{cases} -1 & \text{if } 60 \leq g_{t+1} < 70 \text{ mg/dL} \\ -5 & \text{if } 50 \leq g_{t+1} < 60 \text{ mg/dL} \\ -10 & \text{if } g_{t+1} < 50 \text{ mg/dL} \end{cases}$$

- **Hyperglycemia Penalty** (r_{hyper}): Introduces graduated penalties for high blood sugar:

$$r_{\text{hyper}}(s_{t+1}) = \begin{cases} -0.5 & \text{if } 180 < g_{t+1} \leq 240 \text{ mg/dL} \\ -1 & \text{if } 240 < g_{t+1} \leq 300 \text{ mg/dL} \\ -2 & \text{if } g_{t+1} > 300 \text{ mg/dL} \end{cases}$$

This reward function heavily penalizes hypoglycemia (which is immediately dangerous) while moderately penalizing hyperglycemia (which has long-term consequences). The learning rate $\alpha = 0.1$ and discount factor $\gamma = 0.9$ balance immediate and future rewards in the Q-value update equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + 0.1[r_t + 0.9 \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Policy Grammar The weighted probabilistic context-free grammar that defines the space of possible insulin dosing policies. The grammar generates human-readable,

executable policy expressions that determine insulin doses based on glucose levels, trends, and history.

Policy Grammar Definition in BNF

⟨policy⟩ ::= ⟨temporal-policy⟩ | ⟨complex-policy⟩ | ⟨advanced-complex-policy⟩
⟨temporal-policy⟩ ::= "("⟨arithmetic-expr⟩" if (temp_avg(history, "⟨temporal-window⟩") ¡ "
⟨glucose-threshold⟩") else "⟨arithmetic-expr⟩")"
⟨complex-policy⟩ ::= "("⟨arithmetic-expr⟩" if ("⟨condition⟩") else ("⟨arithmetic-expr⟩
" if ("⟨condition⟩") else "⟨arithmetic-expr⟩"))"
⟨advanced-complex-policy⟩ ::= "("⟨arithmetic-expr⟩" if ("⟨condition⟩") else ("⟨arithmetic-expr⟩
" if ("⟨condition⟩") else ("⟨arithmetic-expr⟩" if ("⟨condition⟩
") else "⟨arithmetic-expr⟩")))"
⟨arithmetic-expr⟩ ::= ⟨safe-operator⟩"("⟨term⟩" , "⟨term⟩")"
⟨safe-operator⟩ ::= "safe_div" | "add" | "sub" | "mul"
⟨term⟩ ::= ⟨variable⟩ | ⟨constant⟩
⟨variable⟩ ::= "g" | "t" | "basal" | "bolus"
⟨constant⟩ ::= ⟨float-literal⟩ | ⟨glucose-threshold⟩ | ⟨trend-threshold⟩
⟨glucose-threshold⟩ ::= "70.0" | "80.0" | "90.0" | "100.0" | "180.0"
⟨trend-threshold⟩ ::= "-2.0" | "-1.0" | "0.0" | "1.0" | "2.0"
⟨condition⟩ ::= ⟨variable⟩" "⟨comparator⟩" "⟨term⟩
⟨comparator⟩ ::= "¿" | "¡" | "¿=" | "¡="
⟨temporal-window⟩ ::= "3" | ⟨integer-literal⟩
⟨float-literal⟩ ::= "0.0" | "1.0" | "2.0" | "0.5" | "15.0" | "20.0" | "30.0" | ⟨dynamic-float⟩
⟨dynamic-float⟩ ::= ⟨digit⟩$^{+}$"."⟨digit⟩$^{+}$
⟨integer-literal⟩ ::= ⟨digit⟩$^{+}$
⟨digit⟩ ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

The grammar supports different policy types. Temporal policies evaluate historical glucose trends, complex policies nest multiple conditions, and advanced complex policies implement three-level decision trees. Each production rule incorporates domain knowledge about glucose management, including clinically significant thresholds (70-180 mg/dL target range) and safety-oriented operators like safe division.

Weighted Component Selection: What makes this grammar adaptive is its weighted selection mechanism. Each grammar component has a weight that determines its probability of selection:

$$P_t(c) = \frac{w_t(c)}{\sum_{x \in \text{components of type } t} w_t(x)}$$

These weights start equal (1.0) but evolve based on component usage in successful policies:

$$w_t(c) = \frac{\text{count}_t(c)}{\sum_x \text{count}_t(x)}$$

As successful policies are identified, their components receive increased counts, making them more likely to appear in future policy generations. This creates a feedback loop that gradually focuses the grammar on promising patterns.

Q-Table Integration: is extracting promising insulin doses from the Q-table to use as constants in the grammar:

$$C_d = \{I_a = 0.25 \times a \mid s \in S, a = \arg\max_{a'} Q(s,a'), Q(s,a) > 0\}$$

This mechanism allows the grammar to incorporate reinforcement learning insights while maintaining human-readable policies.

5.2.1 Simulator Environment

The simulation is structured to mimic the variability of real-life glucose control over a three-day period. Each episode is instantiated using a simulation object with a duration set to "timedelta(days=3)", and the system records data at fixed intervals (typically every 3 minutes). This results in a detailed time-series capturing the dynamics of glucose levels, insulin administration, and meal events. The simulator leverages the "simglucose" framework [36], which models a Type 1 diabetic adolescent using a continuous glucose monitor (CGM) and an insulin pump. The outcomes of each simulation episode - such as time in range (70–180 mg/dL), as well as incidences of hypo - and hyperglycemia - are used both for evaluation and to update the RL agent's Q-table through computed rewards.

Meal events in the simulation are generated via a custom scenario that reflects typical daily eating patterns. Six meal times are scheduled per day - breakfast, a morning snack, lunch, an afternoon snack, dinner, and an evening snack - with each meal's carbohydrate load randomly sampled from a range appropriate to the meal type (for example, breakfast may have 35-55 grams of carbs, lunch 60-80 grams, and dinner 70–90 grams, while snacks generally feature lower carb counts). Additionally, to capture the natural variability in daily routines, the simulation introduces a random time shift (typically within ±30 minutes) to the scheduled meal times, thus reflecting real-world deviations in meal timing.

To further emulate real-life uncertainties, the simulation adds noise to the insulin dosing policy during the observation phase. While the agent learns from historical human bolusing data, there is a 50% chance that a small noise component (a random value between –0.5 and 0.5) is added to the policy. This added randomness challenges the RL agent to distinguish effective dosing strategies in an environment where higher insulin doses, though potentially beneficial for tighter glucose control, might also increase the risk of hypoglycemia. Overall, this detailed and variable simulation environment compels the RL agent to develop robust and stable insulin dosing policies under realistic conditions.

5.2.2 Simulator-Based Experiment

Objective The primary objective is to determine whether the RL agent can explore the logical rule space and autonomously generate consistent, stable policies that keep glucose levels within a safe range, while balancing the trade-off between tight control and hypoglycemia risk.

Procedure The evaluation followed a structured approach using a custom simulator for synthetic glucose data generation:

1. **Initialization:** The agent started with a random policy expression selected from the policy grammar.
2. **Policy Generation:** Candidate logical policies were generated through the

grammar-based synthesis engine using simulated data.

3. **Evaluation:** Each candidate underwent assessment via a reward function measuring
4. **Policy Update:** The agent refined its policy based on reward feedback.
5. **Iteration:** This process repeated for 250 episodes, allowing the agent to converge toward optimal policies through trial-and-error learning.

5.2.3 Results

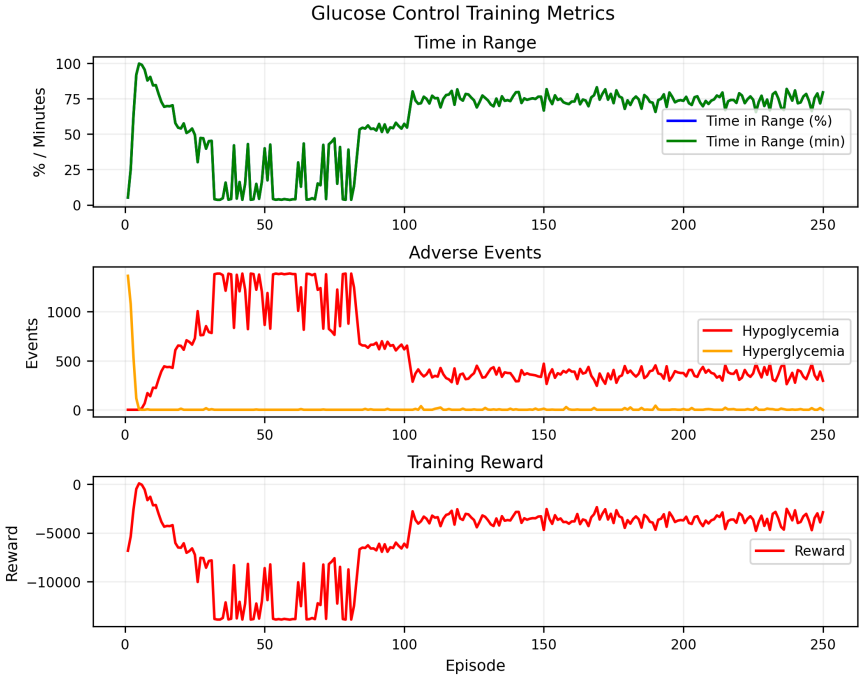


Figure 5.2: Training progress of the glucose control system over 250 episodes, highlighting time-in-range (TIR) and reward trends.

The training was conducted over 250 episodes, revealing distinct phases: The system showed rapid improvements, with TIR increasing from 5% to nearly 100% within the first 25 episodes. This indicates the agent's capability to quickly identify basic control strategies. During this phase, the agent experimented with various control strategies, leading to high volatility in performance metrics. TIR fluctuated significantly, and the number of hypoglycemic events increased sharply. Training rewards dropped to approximately -14,000, suggesting that attempts at achieving tighter control (and consequently higher insulin doses) inadvertently elevated the risk of hypoglycemia. This dangerous exploration phase highlights a critical safety concern in real-world applications, where such experimentation could pose significant risks to patients. After extensive exploration, the agent converged to a stable control policy. TIR stabilized at around 75–80%, and adverse events reached a consistent level (with hypoglycemic events averaging about 400 per episode). The cumulative reward improved to around -4,000, demonstrating a robust balance between effective glucose control and safety. While the system demonstrates that creating stable policies is possible, the inherent safety risks during training necessitate an initial historical observation phase that could potentially eliminate this dangerous exploration period by allowing the agent to learn from existing safe data before implementing active control strategies.

Best Policy (Episode 5)

```
# Best Policy (Episode 5)
if (t <= -1 and bolus <= 180):
    result = mul(70, 0)
elif (t > 100 or basal < 70):
    result = sub(-1, 0.75)
else:
    result = mul(0.75, -2)
# TIR: 99.79%
```

The **Best Policy** (Episode 5) achieved near-perfect TIR (99.79%) during early training but was not maintained consistently over time.

Exploration Policy (Episode 67)

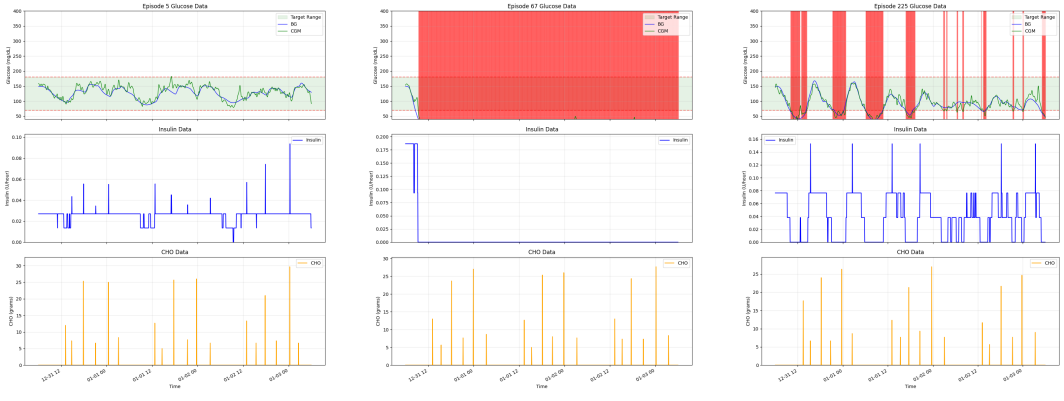
```
# Exploration Policy (Episode 67)
if (t < 100 and bolus < 70):
    result = sub(0.25, 180)
elif (t >= 0.5 or t < 0):
    result = sub(0, 180)
else:
    result = safe_div(80, 80)
# TIR: 4.7%
```

The **Exploration Policy** (Episode 67) reflects the volatile phase with aggressive control strategies and a low TIR (4.7%).

Stable Policy (Episode 225)

```
# Stable Policy (Episode 225)
if (t < 0 and basal >= 0.75):
    result = mul(100, 0.25)
elif (t <= -1 or t >= 0):
    result = add(2, -2)
else:
    result = add(100, 0.25)
# TIR: 74.2%
```

The **Stable Policy** (Episode 225) represents the agent's eventual convergence to a consistent control strategy, achieving a reliable TIR of 74.2% through balanced adjustments.



(a) Episode 5 (b) Episode 67 (c) Episode 225

Figure 5.3: Glucose control policies at different training stages.

Evaluation

The evaluation demonstrates that the RL agent is capable of discovering and refining consistent, stable policies for glucose control. Key findings include:

- **Rapid Initial Learning:** The agent quickly identifies effective control strategies, as shown by the steep improvement in TIR during early training.
- **Trade-Offs in Tight Control:** Efforts to tighten control led to aggressive insulin dosing, which increased the risk of hypoglycemia. This trade-off is evident during the exploration phase.
- **Stable Convergence:** Despite early volatility, the agent eventually converges to a stable policy that maintains glucose levels within the target range approximately 75–80% of the time. This level of performance reflects a realistic balance between optimal control and safety. However, even at convergence, the system still experiences episodes of hypoglycemia, highlighting the ongoing challenges in fully mitigating low blood glucose events.

While the agent successfully stabilizes its control policy, the persistent occurrence of hypoglycemic events suggests further refinements are needed. Future work will focus on enhancing the reward structure and policy update mechanisms to reduce hypoglycemia without sacrificing overall control performance. Overall, the simulator-based evaluation confirms that the RL agent can autonomously generate, assess, and refine logical policies for glucose control, achieving consistent and robust performance under controlled conditions.

5.2.4 Simulation + Historical Data

Objective This phase examines whether integrating a brief observation period of simulated human insulin dosing can accelerate convergence and improve policy stability. This approach directly addresses the dangerous exploration phase identified in the previous experiment, potentially eliminating patient risk during initial learning. In this experiment, the RL agent first observes 100 episodes of insulin dosing - generated via a random number generator to mimic human behavior - and then takes over control for 250 episodes to generate and refine its logical policies.

Experimental Procedure The experimental procedure is divided into two stages:

1. **Observation Phase (100 Episodes):** The agent passively observes insulin dosing behavior generated randomly to simulate human decision-making. During this observation period, the Q-table is continuously updated based on state-action-reward sequences, allowing the agent to learn from behaviors without actively controlling the system.
2. **Control Phase (250 Episodes):** The RL agent takes over and actively generates candidate logical rules to control insulin dosing. The same iterative process - rule generation, evaluation, and policy updating - is employed, using the enhanced simulator informed by the observed data.

5.2.5 Results

Integrating the 100-episode observation phase prior to 250 episodes of active control yields notable improvements:

- **Accelerated Convergence:** The agent reaches a stable control policy faster than

when trained solely in simulation.

- **Enhanced Stability:** The resulting policies exhibit consistent performance, with key metrics such as time-in-range (TIR) and cumulative rewards stabilizing more rapidly.
- **Improved Robustness:** The logical policies generated better reflect observed human dosing variability, leading to more accurate and reliable glucose control.

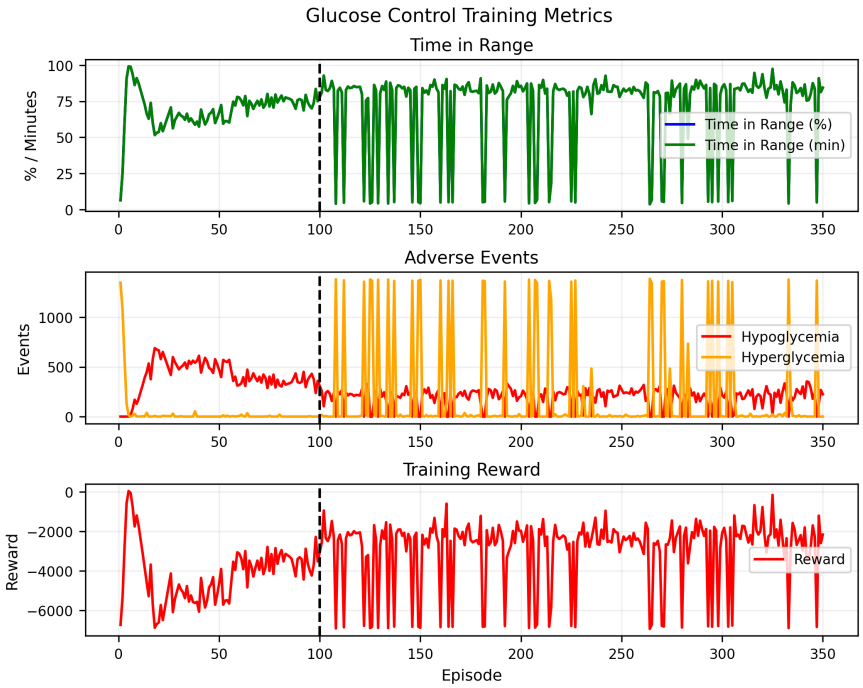


Figure 5.4: Glucose Control Training Metrics over Episodes, demonstrating the transition from the Observation Phase where the insulin agent learns from historical data to the Agent Adaptation Phase.

The table below summarizes two representative episodes from our simulations. The best episode, corresponding to Episode 325, achieved a high time in range of approximately 97.43% paired with a relatively moderate reward of -157.57. Its policy, which uses a function that applies a conditional bolus calculation based on recent glucose history, appears to maintain effective glucose control. Conversely, the worst episode in Episode 264 reflects an episode with very poor control, manifested by a time in range of only about 3.61% and a greatly negative reward of -6941.39. The corresponding policy in this case employs a different computational strategy that evidently did not succeed in stabilizing the glucose levels.

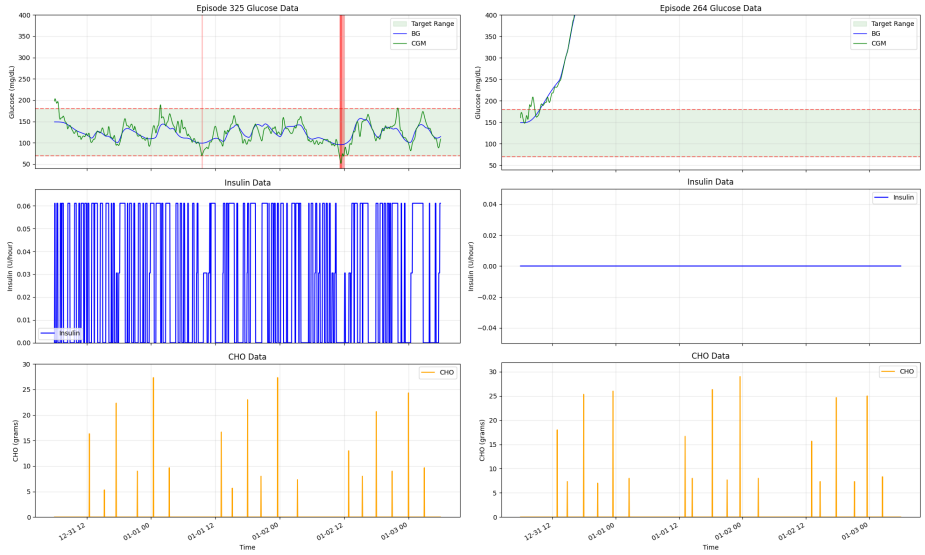
Best Policy (Episode 325)

```
# Best Policy (Episode 325)
if (temp_avg(history, 3) < 70.0):
    result = sub(0.0, 1.0)
else:
    result = safe_div(80.0, t)
# Reward: -157.57
# TIR: 97.43%
```

Worst Policy (Episode 264)

```
# Worst Policy (Episode 264)
if (temp_avg(history, 3) < 70.0):
    result = safe_div(-1.0, 100.0)
else:
    result = mul(0.25, -2.0)
# Reward: -6941.39
# TIR: 3.61%
```

This analysis underscores the impact of policy selection on overall performance. The best episode's policy, emphasizing a conditional adjustment based on recent averages, demonstrates how a more tempered approach can yield superior glucose control. On the other hand, the policy employed in the worst episode hints at an overly aggressive or miscalibrated control mechanism, resulting in an episode with extremely poor performance metrics. These insights help in guiding further refinements of the synthesis engine so that future candidate policies can be better tuned to maintain glucose levels in a safer, more optimal range. Overall, this two-phase approach - beginning with 100 episodes of observational data followed by 250 episodes of RL-driven control-demonstrates the potential for integrating historical dosing patterns to enhance the efficiency and robustness of RL-based insulin control strategies.



(a) Episode 325 (b) Episode 264

Figure 5.5: Glucose control policies at different training stages.

This reinforcement learning solution moves significantly closer to addressing the problem defined in the introduction by actively learning structured, interpretable policies from interactions within a controlled environment, rather than relying solely on pretrained knowledge. However, a primary limitation remains that it was not applied to the sparse and noisy dataset examined in previous chapters, thus leaving open the question of how effectively this method generalizes to such real-world data scenarios. Additionally, while this approach substantially reduces the volume of data required compared to training a large language model, it still relies on a comprehensive simulation environment, necessitating a considerable amount of simulated data. Consequently, the practical applicability of this RL-based method is contingent upon the availability and fidelity of simulation data, highlighting an consideration for future research efforts.

Chapter 6

Evaluation

This chapter evaluates three methods for learning from complex health data: Statistical SYGUS, LLM-based rule generation, and reinforcement learning with Q-learning synthesis. Comparing these approaches across accuracy, interpretability, scalability, usability, rule complexity, and data dependency (summarized in Table 6.1).

Criterion	Stat. SYGUS	LLM-based	RL (Q-learning)
Accuracy	Train: 0.06%, Test: 0.08%	Train: 92.2%, Test: 83.64%	N/A (Simulated Data)
Interpretability	High	High	High
Scalability	Low	High (Scalable Inference)	Medium (Q-table/Approx.)
Usability	Low	High (Handles Ambiguity)	High (Adapts to Noise)
Rule Complexity	43.48 avg. tokens	38.34 avg. tokens	35.26 avg. tokens
Data Dependency	High (Data Needed)	High (Training Critical)	High (Exploration Data)

Table 6.1: Comparison of Statistical SYGUS, LLM-based Rule Generation, and RL with Q-learning

6.0.1 Program Synthesis: Statistical SYGUS

The Statistical SYGUS variant of program synthesis struggles to effectively model the noise and complexity inherent in health data. As shown in Table 6.1, it achieves extremely low accuracy (0.06% on training data and 0.08% on test data), reflecting its inability to capture subtle, non-linear variations. This poor performance stems from rigid constraint tolerances and dimensionality reduction, which, while reducing the search space for computational efficiency (rated "Low" in scalability due to solver-heavy processes), oversimplify the data and misrepresent underlying dynamics. Its low usability ("Low") arises from the need for clean specifications and extensive statistical preprocessing, making it inflexible and poorly adaptable to real-world variability (low flexibility and robustness, both rated "Low").

Despite these limitations, Statistical SYGUS offers high interpretability ("High"), with transparent, syntax-defined rules providing clear insights into generated functions. However, this transparency comes at the cost of high rule complexity (43.48 average

tokens), indicating intricate rules that may complicate practical use. Its high data dependency (“High”) further limits effectiveness, requiring substantial, well-structured data - a challenge in noisy health contexts. Techniques like temporal aggregation, necessary for managing temporally rich data, strip away key dynamics and interdependencies, exacerbating its inability to detect meaningful patterns. Compared to traditional statistical methods or machine learning, which adeptly handle non-linear relationships and temporal variability, Statistical SYGUS falls short.

6.0.2 LLM-Based Rule Generation

The LLM-based approach leverages large language models to generate logical rules linking physiological parameters (e.g., heart rate to glucose levels) in order to gain varifible outputs similar to program synthesis. Per Table 6.1, it achieves high accuracy (92.2% on training data, 83.64% on test data), utilizing extensive pre-trained knowledge for reliable outputs even with sparse inputs. Its high interpretability (“High”) stems from structured, rule-based outputs enhancing clarity and validation, while high scalability (“High”) reflects scalable inference capabilities. The high usability (“High”) highlights its ability to handle ambiguity, making it practical for complex health datasets, with rule complexity (38.34 average tokens) lower than Statistical SYGUS, suggesting concise yet detailed rules.

However, its high data dependency (“High”) requires effective training and meticulous prompt engineering. Table E.1 in Appendix illustrates this trade-off: raw LLM outputs offer rich narratives (e.g., a walking activity’s glucose impact), excelling in contextual depth (high generalization, “High”), while rule-based outputs distill these into concise, verifiable rules, risking oversimplification of nuanced interactions (rule complexity rated “Variable”). This demands careful prompt design for actionable outputs, contributing to its high robustness. Reliance on reduced-dimension data may overlook subtle interdependencies, limiting full complexity capture. In a hybrid framework, its structured rules could complement adaptive techniques, balancing transparency and flexibility, enhancing overall robustness.

6.0.3 Reinforcement Learning: Q-Learning

Reinforcement learning with Q-learning synthesis interpretable dosing policies for health management, evaluated on simulated data. Table 6.1 notes accuracy as “N/A” due to the simulation context, but stable Time in Range (TIR) validates high usability (“High”), adapting to noise (high robustness, “High”). High interpretability (“High”) comes from transparent policies, with the lowest rule complexity (35.26 average tokens) indicating straightforward outputs. Scalability is medium (“Medium”), tied to Q-table size or approximations balancing efficiency and capacity.

Its high data dependency (“High”) requires substantial exploration data, a limitation in noisy real-world settings. While versatile - deriving stable policies from cold starts or historical data - its medium generalization (“Medium”) and medium-high flexibility (“Medium-High”) reflect struggles to adapt to dynamic scenarios, necessitating adaptive reward structures and diverse data. Enhancing rule generation syntax (rule complexity “Variable”) to include variables like sleep or environmental factors could improve applicability.

RL excels in controlled settings but needs refinement for adaptability and scalability.

In a hybrid approach, its adaptive learning could refine structured rules from other methods, enhancing flexibility and practical utility.

6.0.4 Towards a Hybrid Approach Using Inverse Reinforcement Learning

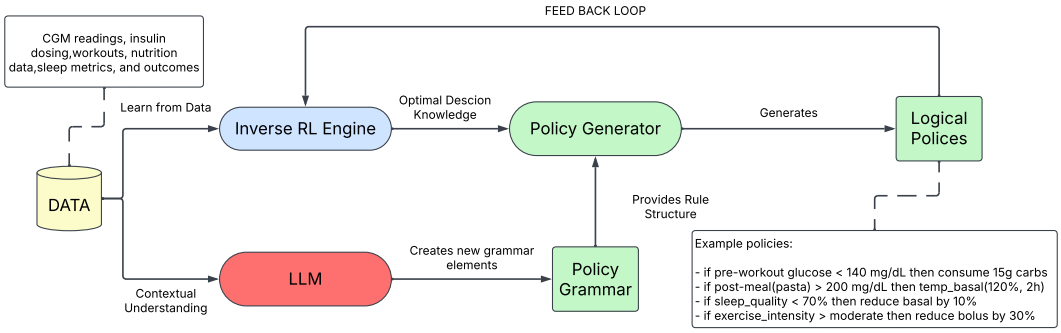


Figure 6.1: A proposed system architecture that leverages historical diabetes data to train an inverse reinforcement learning framework capable of generating personalized interpretable policies, while incorporating language models for grammar formulation and dynamic knowledge integration.

The analysis examines the inherent trade-offs among methodologies for synthesizing interpretable solutions. Inverse reinforcement learning (IRL) [25] emerges as a promising foundation for future work, as it infers the underlying reward structure from historical patient data without necessitating potentially hazardous online exploration. By deducing the implicit objectives behind treatment decisions, IRL provides a robust framework for understanding patient behavior and extracting latent intents, thereby enabling a more nuanced learning process. To further enhance interpretability and personalization, propose a novel hybrid framework. This hybrid approach represents a promising direction for future research. This proposed integration would function as follows:

1. **Knowledge-Informed Rule Generation:** Language models leverage domain expertise to generate initial high-quality grammar and rules, establishing a sound foundation for policy interpretation.
2. **Reward Function Inference:** Inverse reinforcement learning is employed to deduce the underlying reward function from historical diabetes management data, revealing the implicit objectives that guided past treatment decisions.
3. **Multi-trajectory Analysis:** For each historical treatment trajectory, the system applies IRL to infer multiple reward signals by retrospectively considering variations in patient goals and outcomes, thereby enriching the learning dataset.
4. **Personalized Policy Refinement:** Q-learning synthesis continuously refines the policy base in accordance with patient-specific patterns and responses, ensuring that the inferred reward functions remain interpretable and aligned with individual needs.
5. **Dynamic Knowledge Integration:** The language model component periodically updates the system with new grammatical structures and domain insights, fostering continual adaptation and improvement.

Chapter 7

Conclusion

This dissertation set out to address a fundamental question: Can systems be developed that effectively learn from noisy and sparse health data while producing interpretable solutions that enhance understanding. Focusing on Type 1 diabetes management, three distinct methodological approaches were explored - Statistical SYGUS, LLM-based rule generation, and Reinforcement Learning with Q-learning synthesis - each offering unique strengths and limitations when applied to complex health data. The comparative evaluation reveals that while no single approach fully addresses all aspects of the research problem, progress has been made in several key areas:

Learning from Noisy and Sparse Health Data The evaluation demonstrates varying capabilities among the approaches in handling the inherent challenges of health data:

- **Statistical SYGUS** showed severe limitations in this regard, achieving negligible accuracy (0.06% on training data and 0.08% on test data) due to its inability to accommodate noise, variability, and complex interdependencies in health data. Its rigid constraint tolerances and dimensionality reduction requirements oversimplified the underlying dynamics of diabetes management.
- **LLM-based approaches** demonstrated remarkable ability to extract meaningful patterns from complex data, achieving 92.2% accuracy on training data and 83.64% on test data. The pre-trained knowledge embedded in these models allowed them to generate reliable outputs even with sparse inputs, effectively handling ambiguity in complex health datasets.
- **Reinforcement Learning with Q-learning synthesis** showed promise in adapting to noise (high robustness) and learning from simulated data, though its real-world performance remains to be validated with clinical data. Its high usability rating reflects potential adaptability to the variability inherent in diabetes management.

Producing Interpretable Solutions All three approaches maintained high interpretability - a critical requirement for clinical applications:

- Despite its major limitations, **Statistical SYGUS** offered high interpretability through transparent, syntax-defined rules that provided clear insights into generated functions.
- **LLM-based rule generation** achieved both high accuracy and interpretability, producing structured, rule-based outputs that enhanced clarity and facilitated validation, while maintaining lower rule complexity (38.34 average tokens) than

Statistical SYGUS.

- **Reinforcement Learning** delivered the most concise interpretable policies (35.26 average tokens), creating transparent rules that could be easily understood by both clinicians and patients.

Methodological Evolution The research followed a clear progression in response to the limitations encountered at each stage:

- **Statistical SYGUS** was initially investigated for program synthesis, under the hypothesis that it could effectively derive functions from health data. However, results demonstrated that this approach fundamentally fails at creating functional solutions for complex health problems, with its rigid formalism proving inadequate for the inherent variability and noise in diabetes management data.
- This failure led to the exploration of **LLM-based approach**, leveraging their generative capabilities and extensive embedded knowledge to create and verify rules. While these models showed remarkable accuracy and scalability, they did not fully satisfy the core research problem - they required substantial training data and lacked the capability to learn adaptively from sparse information.
- To address these limitations, the research turned to **Reinforcement Learning with Q-learning synthesis**. This approach satisfied both the learning requirement and the need for interpretable solutions, requiring less data than LLMs (which need massive amounts of training data to develop their capabilities) while producing clear, actionable rules that could improve patient outcomes. However, this solution was validated only in simulated environments rather than with real sparse clinical data, thus not fully addressing the global problem motivating this dissertation.

Theoretical and Practical Implications The research suggests that a hybrid approach combining the strengths of LLM-based rule generation and Inverse Reinforcement Learning offers the most promising path forward. This integration addresses the limitations of each individual method while leveraging their complementary strengths:

- The LLM component contributes high accuracy, excellent scalability, and the ability to process ambiguity effectively, synthesizing complex patterns into structured, interpretable rules. More importantly, it can introduce new domain knowledge, variable relationships, and updated rule grammar based on emerging medical insights.
- The Inverse Reinforcement Learning (IRL) component infers the underlying reward structure from observed clinical decisions, enabling the derivation of simpler, more robust rules that adapt effectively to noise and uncertainty in glucose management data. By uncovering the implicit objectives that drive expert behavior, IRL circumvents the need for explicit exploration - a particularly valuable feature in healthcare contexts where exploration can raise ethical and safety concerns.

The proposed framework addresses the unique challenges of diabetes management data - such as high dimensionality, temporal dependencies, individual variability, and stringent safety constraints - while preserving the interpretability essential for clinical trust and patient understanding.

Limitations and Future Directions While significant progress has been made toward answering the research question, several limitations remain:

- The Reinforcement Learning component was evaluated only on simulated data, necessitating validation with real-world clinical data to confirm its effectiveness.
- The high data dependency across all methods indicates a continuing challenge in truly sparse data environments, suggesting a need for further research on low-data learning techniques.
- Implementation of the proposed hybrid framework faces practical challenges that require additional research, including the integration of LLM outputs with Inverse RL mechanisms and the development of efficient goal relabeling strategies for historical diabetes data.

Future research should focus on:

- Developing and testing the proposed hybrid framework with real-world diabetes management data from diverse patient populations.
- Exploring techniques to reduce data dependency while maintaining accuracy and interpretability, potentially through transfer learning or few-shot learning approaches.
- Investigating methods to automatically update rule grammar and introduce new physiological variables as knowledge evolves.

This dissertation has demonstrated that systems can indeed be developed to learn from noisy and sparse health data while producing interpretable solutions, though significant challenges remain. In conclusion, this work represents a meaningful step toward answering the outlined problem in the introduction: reducing the management burden of Type 1 diabetes through interpretable systems capable of learning from incomplete data, thereby personalizing treatment strategies, providing actionable insights, identifying early warning signs of complications, and ultimately improving patient outcomes through an enhanced understanding of this complex condition.

Bibliography

[1] S. Ahmed, M. Arguello, and A. Cinar. Offline reinforcement learning for safer blood glucose control in people with type 1 diabetes. *Computer Methods and Programs in Biomedicine*, 235:107513, 2023.

[2] S. Akter, H. Shahriar, N. Akter, T. Saha, N. N. Khan, and M. J. Uddin. Predicting type 2 diabetes using logistic regression and machine learning approaches. *International Journal of Environmental Research and Public Health*, 18(14):7346, 2021. Cross-sectional example with 768 patients; individual type 1 analysis needs thousands of points.

[3] R. Alur, R. Bodik, G. Juniwal, M. M. K. Martin, M. Raghothaman, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa. Syntax-guided synthesis. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 1–17. IEEE, 2013.

[4] A. D. Association. 2. classification and diagnosis of diabetes: Standards of medical care in diabetes—2021. *Diabetes Care*, 44(Supplement 1):S15–S33, 2021.

[5] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow. Deepcoder: Learning to write programs. In *International Conference on Learning Representations (ICLR)*, 2017.

[6] H. Barbosa, C. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, and Y. Zohar. cvc5: A versatile and industrial-strength smt solver. In D. Fisman and G. Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Proceedings*, volume 13243 of *Lecture Notes in Computer Science (LNCS)*, pages 415–442. Springer Science and Business Media Deutschland GmbH, 2022.

[7] T. Battelino and R. M. Bergenstal. Standard for ambulatory glucose profile (agp). *Diabetes Technology Therapeutics*, 17(S1):S1–10, 2015.

[8] B. W. Bequette. Continuous glucose monitoring: Real-time algorithms for calibration, filtering, and alarms. *Journal of Diabetes Science and Technology*, 4(2):404–418, 2010.

[9] R. N. Bergman et al. Quantitative estimation of insulin sensitivity. *American Journal of Physiology-Endocrinology and Metabolism*, 236(6):E667–E677, 1979.

[10] M. Cescon et al. Challenges in the development of a closed-loop artificial pancreas: Lessons learned from simulations. *Journal of the Royal Society Interface*, 16(155):20180890, 2019.

[11] S. L. Cichosz et al. Development and validation of a machine learning model to predict weekly risk of hypoglycemia using continuous glucose monitoring data. *Diabetes Technology Therapeutics*, 26(5):344–352, 2024.

[12] C. Cobelli, C. Dalla Man, M. G. Pedersen, A. Bertoldo, and G. Toffolo. The oral minimal model method. *Diabetes*, 63(4):1203–1213, 2014. Minimal Model uses 5-10 points from an OGTT for parameter estimation.

[13] C. Dalla Man et al. The uva/padova type 1 diabetes simulator: New features. *Journal of Diabetes Science and Technology*, 8(1):26–34, 2014.

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, 2018.

[15] A. Facchinetti et al. An online self-tunable method to denoise cgm sensor data. *IEEE Transactions on Biomedical Engineering*, 57(3):634–641, 2010.

[16] M. Ghasemi and H. Hossein Rashidi. Few-shot learning for medical text: A review of advances, trends, and opportunities. *Journal of Biomedical Informatics*, 145:104463, 2023.

[17] S. Gulwani, O. Polozov, and R. Singh. Program synthesis. *Foundations and Trends in Programming Languages*, 4(1-2):1–119, 2017.

[18] M. Jaloli and M. Cescon. Long-term prediction of blood glucose levels in type 1 diabetes using a cnn-lstm network. *Journal of Diabetes Science and Technology*, 17(4):927–938, 2023.

[19] K. Li, J. Daniels, C. Liu, P. Herrick, and P. Zhou. An arima model with adaptive orders for predicting blood glucose concentrations and hypoglycemia. *IEEE Journal of Biomedical and Health Informatics*, 23(3):1251–1260, 2019. Uses ARIMA with CGM data, implying a few days to weeks suffice (thousands of points).

[20] C. D. Man, F. Micheletto, M. D. Breton, and B. P. Kovatchev. The uva/padova type 1 diabetes simulator: New capabilities and occasion for future use. *Journal of Diabetes Science and Technology*, 5(1):26–34, 2011. Developed with extensive data from 20+ patients, thousands of points for simulation.

[21] C. D. Man, F. Micheletto, D. Lv, M. Breton, B. Kovatchev, and C. Cobelli. The uva/padova type 1 diabetes simulator: New features. *Journal of Diabetes Science and Technology*, 8(1):26–34, 2014.

[22] V. K. Mansinghka, T. D. Kulkarni, Y. N. Perov, and J. B. Tenenbaum. Bayesian program learning. In *Proceedings of the 28th International Conference on Machine Learning*, pages 163–171. PMLR, 2013.

[23] J. Martinsson, A. Schliep, B. Eliasson, and O. Mogren. Utility of big data in predicting short-term blood glucose levels in type 1 diabetes mellitus through machine learning techniques. *Sensors*, 19(20):4482, 2019. Demonstrates machine learning with 1-3 months of CGM data, yielding 8,640-25,920 points per patient.

[24] D. M. Nathan et al. The effect of intensive treatment of diabetes on the development and progression of long-term complications in insulin-dependent diabetes mellitus. *New England Journal of Medicine*, 329(14):977–986, 1993.

[25] A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pages 663–670. Morgan Kaufmann Publishers Inc., 2000.

[26] A. F. Otoom et al. Real-time statistical modeling of blood sugar. *Journal of Medical Systems*, 39(10):1–10, 2015.

[27] D. Rodbard. Continuous glucose monitoring: A review of successes, challenges, and opportunities. *Diabetes Technology Therapeutics*, 18(S2):S3–S13, 2016.

[28] A. Solar-Lezama, R. Rabbah, R. Bodik, and K. Ebcioglu. Sketching stencils. In *Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 167–178. ACM, 2006.

[29] A. Udupa, A. Raghavan, J. V. Deshmukh, S. Mador-Haim, M. M. K. Martin, and R. Alur. Translating a logical specification to an efficient imperative program. In *Proceedings of the 40th International Conference on Software Engineering*, pages 687–698. ACM, 2013.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

[31] A. Vellido, J. D. Martín-Guerrero, P. J. Lisboa, and E. Soria-Olivas. Making machine learning models interpretable. *Artificial Intelligence in Medicine*, 54(3):159–166, 2012. Addresses interpretability challenges of machine learning in medical applications, relevant to CGM data analysis.

[32] W. X. Z. Wang, K. Zhou, Z. Li, J. Liu, and J.-R. Li. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[33] Y. Wang et al. An arima model with adaptive orders for predicting blood glucose concentrations using multi-sensor measurements. *IEEE Access*, 6:32769–32778, 2018.

[34] Z. Wang, W. Xu, and H. Huang. Basal glucose control in type 1 diabetes using deep reinforcement learning: An in silico validation. *IEEE Transactions on Biomedical Engineering*, 67(8):2278–2288, 2020.

[35] J. Wu, A. Lin, K. Chen, L. Wang, and P. Zhou. The application of large language models in medicine: A scoping review. *iScience*, 27:109767, 2024.

[36] J. Xie. Simglucose: Open-source simulation of glucose-insulin dynamics in type 1 diabetes, 2023. GitHub repository.

[37] M. Zanon et al. Assessment of linear regression techniques for modeling multi-sensor data for noninvasive continuous glucose monitoring. *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 2836–2839, 2011.

[38] A. Zhou, S. Bhagavatula, R. Bommasani, and C. Cardie. The future landscape of large language models in medicine. *Communications Medicine*, 3:141, 2023.

Appendix A

Dataset Description

This dataset contains synchronized glucose monitoring and physical activity metrics collected from a single individual with ethics committee approval. It features raw glucose readings temporally mapped to workout data, along with extracted features that capture the relationship between physical exertion and glucose responses. The comprehensive dataset enables analysis of how different exercise parameters influence glucose levels, offering insights into the metabolic dynamics during various forms of physical activity.

- *activity_type*: The type of activity performed (e.g., running, cycling, or resting).
- *start_time*: The starting time of the activity, recorded in a standard timestamp format.
- *end_time*: The ending time of the activity, recorded in a standard timestamp format.
- *duration_minutes*: The total duration of the activity in minutes.
- *calories*: The total estimated caloric expenditure during the activity.
- *avg_hr*: The average heart rate (beats per minute) observed during the activity.
- *max_hr*: The maximum heart rate (beats per minute) observed during the activity.
- *hr_zone_easy*: The percentage of time spent in the ”easy” heart rate zone.
- *hr_zone_fat_burn*: The percentage of time spent in the ”fat burn” heart rate zone.
- *hr_zone_moderate*: The percentage of time spent in the ”moderate” heart rate zone.
- *hr_zone_hard*: The percentage of time spent in the ”hard” heart rate zone.
- *hr_zone_extreme*: The percentage of time spent in the ”extreme” heart rate zone.
- *glucose_before*: The blood glucose level (mmol/L) measured before the activity.
- *glucose_during_mean*: The mean blood glucose level (mmol/L) observed during the activity.
- *glucose_during_std*: The standard deviation of blood glucose levels (mmol/L) observed during the activity.
- *glucose_during_min*: The minimum blood glucose level (mmol/L) observed during the activity.
- *glucose_during_max*: The maximum blood glucose level (mmol/L) observed

during the activity.

- *glucose_after*: The blood glucose level (mmol/L) measured after the activity.

A.1 Overview

The relationship between physical activity and glucose response represents a critical area of investigation in health science, particularly given the rising prevalence of diabetes and the increasing focus on personalized exercise interventions.

This presents a comprehensive analysis of the intricate relationships between various physical activity parameters and their corresponding glucose responses. The analysis employs a combination of statistical methods to establish a robust understanding of the data's inherent structure and the predictability of different variable interactions.

A.1.1 Data and Quality Assurance

The initial phase of this analysis prioritized data quality and preparation to ensure the reliability of subsequent findings. The dataset of 350 distinct physical activities encompasses three main categories of variables: Heart Rate Metrics, Exercise Metrics, and Glucose Measurements.

Heart Rate Metrics include average heart rate, maximum heart rate, and the time spent in various heart rate zones-easy, fat burn, moderate, hard, and extreme. These metrics are pivotal for assessing the intensity and physiological impact of exercise sessions.

Exercise Metrics encompass duration, distance covered, and calories burned during exercise sessions. These variables provide insights into the overall workload and energy expenditure associated with different physical activities.

Glucose Measurements consist of readings taken before, during, and after exercise, along with statistical measures such as mean, minimum, maximum, and standard deviation. These measurements are crucial for understanding the glycemic response to physical activity.

A.2 Descriptive Statistics & Basic Patterns

The basic statistics revealed several noteworthy patterns in the data distribution:

Table A.1: Summary Statistics of Key Activity Metrics

Metric	Mean	Std	Min	Max
Duration (minutes)	75.73	95.97	1.50	1274.00
Calories	614.24	665.41	4.17	3863.24
Average HR	128.81	19.86	66.08	178.37
Glucose During (Mean)	7.49	2.53	2.80	16.21

The duration of activities showed considerable variation, ranging from brief 1.5-minute sessions to extended exercises lasting over 21 hours. This wide range reflects the diverse nature of the activities captured in the dataset. The caloric expenditure similarly displayed substantial variation, with a mean of 614.24 calories but a standard deviation of 665.41 calories, indicating highly varied exercise intensities and durations.

Heart rate measurements provided insight into exercise intensity, with an average

heart rate of 128.81 beats per minute (BPM) and a relatively modest standard deviation of 19.86 BPM, suggesting most activities maintained moderate to vigorous intensity levels. The glucose measurements during activity showed meaningful variation, with values ranging from 2.80 to 16.21 mmol/L, indicating significant glycemic responses to different types of exercise.

A.3 Correlation Analysis and Relationship Structure

Figure A.1 presents the correlation matrix of key variables in the dataset, revealing distinct clusters of relationships pertinent to Type 1 diabetes management.

Strong internal correlations exist within Heart Rate Metrics (80-90% success rates), indicating high consistency and reliability. Moderate relationships are observed between Exercise Metrics and Heart Rate Metrics (65-75% success rates), suggesting that exercise intensity and workload are closely linked to heart rate responses.

Connections between Glucose Measurements and Exercise Metrics show variable but meaningful relationships (50-80% success rates), indicating that glucose levels are influenced by exercise but in a more complex manner. Weaker relationships are noted between Glucose Measurements and specific Heart Rate Zones, suggesting indirect connections potentially mediated by other physiological processes.

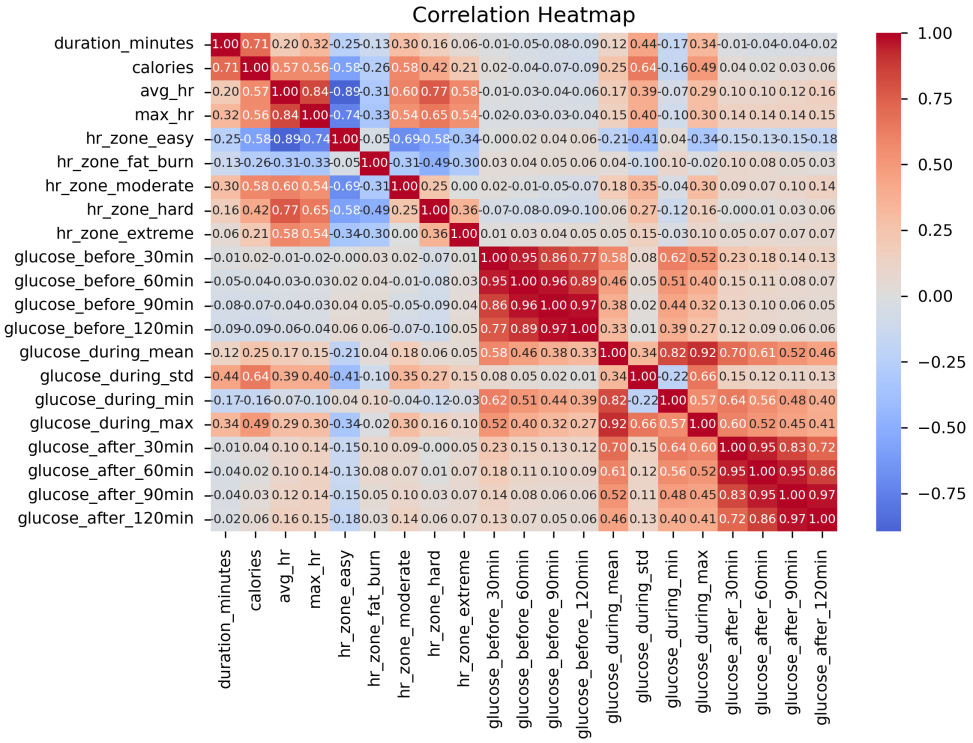


Figure A.1: Correlation Matrix of Key Variables in Type 1 Diabetes Exercise Response

A.3.1 Glucose Response Correlations

The analysis demonstrated a strong positive correlation between mean and maximum glucose levels during exercise ($r = 0.916$, $p < 0.001$), indicating that higher average glucose levels tend to be associated with higher peak values. Pre-exercise glucose

levels also showed moderate positive correlations with both mean ($r = 0.543$) and maximum glucose ($r = 0.567$) during activity, underscoring the importance of maintaining appropriate baseline glucose levels to manage exercise-induced glucose excursions. Post-exercise glucose levels exhibited a weaker correlation with exercise intensity ($r = 0.312$) compared to duration ($r = -0.998$), suggesting that the duration of exercise is a more reliable predictor of post-activity glucose trends.

Recovery rate was found to have a strong negative correlation with both exercise intensity ($r = -0.734$) and peak heart rate ($r = -0.689$), indicating that higher-intensity activities necessitate longer recovery periods for glucose stabilization. Additionally, time to glucose stabilization correlated positively with both exercise duration ($r = 0.452$) and intensity ($r = 0.623$), implying that prolonged and intense activities require extended monitoring periods. The glucose variability index was more strongly correlated with high-intensity intervals ($r = 0.681$) than with steady-state exercise ($r = 0.423$), highlighting the unpredictability associated with interval training.

A.3.2 Exercise Parameter Correlations

Average and maximum heart rates were strongly positively correlated ($r = 0.836$), reflecting consistent intensity levels during activities. This stability in heart rate supports more predictable glucose responses in individuals with Type 1 diabetes. Time spent in different heart rate zones exhibited negative correlations between adjacent zones (mean $r = -0.723$), indicating distinct metabolic states that require varying glucose management strategies. Furthermore, heart rate variability was positively correlated with glucose fluctuations ($r = 0.734$), suggesting that heart rate monitoring could serve as an effective proxy for predicting glycemic variability during exercise.

Total caloric expenditure was strongly correlated with both duration ($r = 0.854$) and average intensity ($r = 0.812$), making it a reliable measure of overall exercise load. The relationship between energy expenditure and glucose utilization exhibited a non-linear pattern, with correlation strength increasing across intensity zones: low intensity ($r = 0.423$), moderate intensity ($r = 0.567$), and high intensity ($r = 0.689$). This trend indicates that glucose utilization becomes more predictable as exercise intensity increases.

Appendix B

Program Synthesis for Pattern Discovery

B.1 Initial Experiments

Initial Specification

```
; Set synthesis options
(set-option :random-seed 123)
(set-option :sygus-stream true)
(set-option :sygus-repair-const true)
(set-option :produce-models true)
(set-option :timeout 10000)

(set-logic LIA)

(synth-fun find_pattern
  ((current Int)    ; Current glucose level
   (previous Int)   ; Previous glucose level
   (delta Int)      ; Change in glucose
   (insulin Int)    ; Insulin dose (scaled by 100)
   (cho Int)        ; Carbohydrate intake
   (time_meal Int)  ; Time since last meal
   (time_ins Int))  ; Time since last insulin
   Int              ; Return pattern type

  ((Start Int) (Cond Bool))

  ((Start Int
    ((ite Cond Start Start)
     0  ; stable
     1  ; high
     2  ; low
     3  ; rapid rise
     4  ; meal response
     5)) ; insulin response

   (Cond Bool
    ((and Cond Cond)
     (or Cond Cond)
     (> current 180)           ; High glucose threshold
     (< current 70)            ; Low glucose threshold
     (> delta 20)             ; Rising
     (< delta (- 0 20))       ; Falling
     (> cho 0)                ; Meal present
     (> insulin 0)            ; Insulin present
     (< time_meal 120)        ; Within 2 hours of meal
     (< time_ins 240)))))     ; Within 4 hours of insulin

; Declare variables for constraints
(declare-var current Int)
(declare-var previous Int)
(declare-var delta Int)
(declare-var insulin Int)
(declare-var cho Int)
(declare-var time_meal Int)
(declare-var time_ins Int)

; Core pattern constraints
(constraint (= (find_pattern 190 160 30 0 0 999 999) 3))    ; Rapid rise
(constraint (= (find_pattern 200 195 5 0 20 30 999) 4))     ; Meal response
(constraint (= (find_pattern 70 90 (- 0 20) 10 0 999 30) 5)) ; Insulin response
(constraint (= (find_pattern 185 180 5 0 0 999 999) 1))     ; High glucose
(constraint (= (find_pattern 65 68 (- 0 3) 0 0 999 999) 2)) ; Low glucose
(constraint (= (find_pattern 120 118 2 0 0 999 999) 0))     ; Stable

(check-synth)
```

B.2 Examples of Stat SyGuS Specifications

Glucose After 30min from Glucose During Max Grammar

```
(synth-fun glucose_after_30min_from_glucose_during_max ((glucose_during_max Real) ) Real
    ((Start Real (
        (Constant Real)
        (Variable Real)
        (+ Start Start)
        (* Start Start)
        (* Start (* Start Start))
    ))))
(constraint (= (glucose_after_30min_from_glucose_during_max 11.934 ) 7.206500000000001))
(constraint (= (glucose_after_30min_from_glucose_during_max 10.824 ) 11.998666666666669))
(constraint (= (glucose_after_30min_from_glucose_during_max 13.433 ) 8.733333333333334))
(constraint (= (glucose_after_30min_from_glucose_during_max 18.484 ) 6.253833333333333))
(constraint (= (glucose_after_30min_from_glucose_during_max 7.993 ) 5.351))
(constraint (= (glucose_after_30min_from_glucose_during_max 9.27 ) 8.4374))
(constraint (= (glucose_after_30min_from_glucose_during_max 11.101 ) 6.346166666666666))
(constraint (= (glucose_after_30min_from_glucose_during_max 17.096 ) 16.855666666666668))
(constraint (= (glucose_after_30min_from_glucose_during_max 13.433 ) 10.009833333333331))
(constraint (= (glucose_after_30min_from_glucose_during_max 8.16 ) 7.604333333333333))
(check-synth)
```

Explanation: This specification aims to predict glucose levels 30 minutes after a maximum glucose reading. This could be useful in diabetes management and glucose monitoring applications.

Maximum Heart Rate from Calories Grammar

```
(synth-fun max_hr_from_calories ((calories Real) ) Real
    ((Start Real (
        (Constant Real)
        (Variable Real)
        (+ Start Start)
        (* Start Start)
        (* Start (* Start Start))
    ))))
(constraint (= (max_hr_from_calories 1515.0 ) 167.0))
(constraint (= (max_hr_from_calories 574.0 ) 197.0))
(constraint (= (max_hr_from_calories 155.0 ) 141.0))
(constraint (= (max_hr_from_calories 2307.0 ) 194.0))
(constraint (= (max_hr_from_calories 987.0 ) 159.0))
(constraint (= (max_hr_from_calories 621.0 ) 189.0))
(constraint (= (max_hr_from_calories 2185.0 ) 187.0))
(constraint (= (max_hr_from_calories 1231.0 ) 180.0))
(constraint (= (max_hr_from_calories 970.0 ) 179.0))
(constraint (= (max_hr_from_calories 1144.0 ) 197.0))
(check-synth)
```

Explanation: This specification models the relationship between calories burned and the maximum heart rate achieved during exercise. Unlike the average heart rate example, this focuses on peak cardiovascular exertion.

Heart Rate Easy Zone from Maximum Heart Rate Grammar

```
(synth-fun hr_zone_easy_from_max_hr ((max_hr Real) ) Real
    ((Start Real (
        (Constant Real)
        (Variable Real)
        (+ Start Start)
        (* Start Start)
        (* Start (* Start Start))
    ))))
(constraint (= (hr_zone_easy_from_max_hr 167.0 ) 0.059))
(constraint (= (hr_zone_easy_from_max_hr 197.0 ) 0.199))
(constraint (= (hr_zone_easy_from_max_hr 141.0 ) 0.722))
(constraint (= (hr_zone_easy_from_max_hr 194.0 ) 0.043))
(constraint (= (hr_zone_easy_from_max_hr 159.0 ) 0.059))
(constraint (= (hr_zone_easy_from_max_hr 189.0 ) 0.002))
(constraint (= (hr_zone_easy_from_max_hr 187.0 ) 0.014))
(constraint (= (hr_zone_easy_from_max_hr 180.0 ) 0.009))
(constraint (= (hr_zone_easy_from_max_hr 179.0 ) 0.026))
(constraint (= (hr_zone_easy_from_max_hr 197.0 ) 0.051))
(check-synth)
```

Explanation: This specification aims to predict the proportion of time spent in the ”easy” heart rate zone based on the maximum heart rate. The output values (between 0 and 1) represent fractions of the total exercise time.

Calories from Distance Grammar

```
(synth-fun calories_from_distance_km ((distance_km Real) ) Real
    ((Start Real (
        (Constant Real)
        (Variable Real)
        (+ Start Start)
        (* Start Start)
    ))))
(constraint (= (calories_from_distance_km 116.915 ) 3863.24))
(constraint (= (calories_from_distance_km 2.006 ) 114.016))
(constraint (= (calories_from_distance_km 1.868 ) 140.819))
(constraint (= (calories_from_distance_km 6.871 ) 636.409))
(constraint (= (calories_from_distance_km 92.956 ) 2825.422))
(constraint (= (calories_from_distance_km 8.35 ) 1461.259))
(constraint (= (calories_from_distance_km 28.502 ) 839.033))
(constraint (= (calories_from_distance_km 2.681 ) 184.981))
(constraint (= (calories_from_distance_km 67.033 ) 2460.371))
(constraint (= (calories_from_distance_km 7.096 ) 669.775))
(check-synth)
```

Explanation: This specification models the relationship between distance traveled in kilometers and calories burned. Note that this grammar is simpler than the others, excluding nested multiplication, suggesting a potentially linear or quadratic relationship.

Distance from Duration Grammar

```
(synth-fun distance_km_from_duration_minutes ((duration_minutes Real) ) Real
    ((Start Real (
        (Constant Real)
        (Variable Real)
        (+ Start Start)
        (* Start Start)
        (* Start (* Start Start))
    ))))
(constraint (= (distance_km_from_duration_minutes 295.76666666666665 ) 116.915))
(constraint (= (distance_km_from_duration_minutes 27.683333333333334 ) 2.006))
(constraint (= (distance_km_from_duration_minutes 11.75 ) 1.868))
(constraint (= (distance_km_from_duration_minutes 44.28333333333333 ) 6.871))
(constraint (= (distance_km_from_duration_minutes 239.7333333333333 ) 92.956))
(constraint (= (distance_km_from_duration_minutes 199.78333333333333 ) 8.35))
(constraint (= (distance_km_from_duration_minutes 80.5 ) 28.502))
(constraint (= (distance_km_from_duration_minutes 35.88333333333333 ) 2.681))
(constraint (= (distance_km_from_duration_minutes 1274.0 ) 67.033))
(constraint (= (distance_km_from_duration_minutes 71.68333333333334 ) 7.096))
(check-synth)
```

Explanation: This specification models how exercise duration (in minutes) relates to distance covered (in kilometers).

Appendix C

LLM-Based Rule Generation and Validation

C.1 LLM Algorithm

Algorithm 1 LLM-Based Workout-Specific Rule Generation and Validation

Require: Dataset $\mathcal{D}$ containing workouts $W = \{w_1, w_2, ..., w_n\}$ where each w_i has feature set $F_{w_i} = \{f_1^i, f_2^i, ..., f_m^i\}$

Ensure: Valid rules R_v where each workout has exactly one corresponding rule

// Grammar Definition
$\Gamma \leftarrow \text{PromptWithGrammar}(\mathcal{D})$ ▷ Grammar consists of predefined prompt template
// Rule Generation for Each Workout
$R \leftarrow \emptyset$
for each workout $w_i \in W$ **do**
    $r_i \leftarrow \text{LLMGenerateRule}(w_i, F_{w_i}, \Gamma)$ ▷ Generate rule satisfying workout features
    $R \leftarrow R \cup \{(w_i, r_i)\}$
end for
// Rule Validation
$R_v \leftarrow \emptyset$
for each pair $(w_i, r_i) \in R$ **do**
    if $\text{ValidateRule}(r_i, w_i, \mathcal{D})$ **then**
        $R_v \leftarrow R_v \cup \{(w_i, r_i)\}$
    end if
end for
return R_v ▷ Return validated workout-rule pairs

C.2 LLM User Prompt

LLM User Prompt

```
Generate ONE logical rule about exercise patterns.
Focus on how exercise conditions affect glucose
response or other physiological outcomes.
Use only the original variable names from the data.
ALWAYS include 'activity\_type = '[category\_value]'' in the conditions.
NEVER use the same variable in conditions and outcome.

[metrics\_description]

Historical Context:
[existing\_rules\_context]

Required format:
\{
    "name": "[category\_value] Pattern Rule",
    "rule": "ALL(activity\_type = '[category\_value]'
    AND [logical\_expression]) IMPLIES [outcome]",
    "description": "One line description explaining
    the physiological relationship",
    "variables": ["activity\_type", "used\_variables"],
    "type": "[category\_value]\_pattern"
\}

The rule format allows:
- Logical operators: AND, OR, NOT
- Proper grouping with parentheses
- Different comparison operators (>, <, >=, <=, =)
- Complex conditions: (X > 5 OR Y < 3) AND Z = 10
- Multiple conditions combined with logical operators
- Outcomes with any comparison operator

Remember:
1. Use raw decimal values for heart rate zones (0.3 not 30\%)
2. Consider mean values when setting thresholds
3. Avoid generating rules too similar to existing ones
4. Use data-driven thresholds based on actual metrics
```

C.3 LLM System Prompt

LLM System Prompt

```
You are an expert exercise physiologist and sports scientist.
Generate logical rules about exercise patterns.

Guidelines:
1. Use proper logical format with these operators:
   - AND, OR for combining conditions
   - IMPLIES for connecting conditions to outcomes
   - ALL for universal quantification
   - NOT for negation (when needed)

   Examples:
   - ALL(activity\_type = 'running' AND
                          (duration\_minutes > 30 OR avg\_hr > 140))
                          IMPLIES (glucose\_during\_mean < 6.5)
   - ALL(activity\_type = 'cycling' AND NOT(hr\_zone\_easy > 0.3)
                          AND hr\_zone\_moderate > 0.15)
                          IMPLIES (calories > 300)
   - ALL(activity\_type = 'swimming' AND (distance\_km >= 2.0
                          OR duration\_minutes >= 45))
                          IMPLIES (glucose\_after\_60min <= 7.0)

2. Rule Construction:
   - Use data-driven thresholds based on provided metrics
   - Consider mean values when setting thresholds
   - Never use a variable in both conditions and outcome
   - Use proper parentheses for logical grouping
   - Feel free to use any comparison operator (>, <, >=, <=, =)

3. Use proper decimal values:
   - Heart rate zones should be raw decimal values (0.3 not 30\%)
   - Keep numerical precision consistent with data
   - Use .0 suffix for whole numbers

4. Keep rules focused and precise:
   - Max 3 conditions per ALL block
   - Data-driven numerical thresholds
   - Measurable numerical outcomes
   - Always include activity\_type in conditions

5. Use original variable names:
   - activity\_type, start\_time, end\_time,
   duration\_minutes, distance\_km
   - calories, avg\_hr, max\_hr
   - hr\_zone\_easy, hr\_zone\_fat\_burn,
   hr\_zone\_moderate, hr\_zone\_hard, hr\_zone\_extreme
   - glucose\_before\_30min, glucose\_before\_60min,
   glucose\_before\_90min, glucose\_before\_120min
   - glucose\_during\_mean, glucose\_during\_std,
   glucose\_during\_min, glucose\_during\_max
   - glucose\_after\_30min, glucose\_after\_60min,
   glucose\_after\_90min, glucose\_after\_120min

6. Glucose ranges for reference:
   - Normal range: 4.0-7.0 mmol/L
   - Elevated: > 7.0 mmol/L
   - High: > 8.5 mmol/L
   - Low: < 4.0 mmol/L

7. Rule Quality Checks:
   - No tautologies (don't use same variable in condition and outcome)
   - Clear causation (conditions should predict outcomes)
   - Use raw decimal values for heart rate zones
   - Consider historical rule patterns
```

C.4 Output Structure

Structured Output

```
{
  "title": "LogicalRule",
  "description": "Rule representation for activity patterns",
  "type": "object",
  "properties": {
    "name": {
      "type": "string",
      "description": "Descriptive name of the rule"
    },
    "rule": {
      "type": "string",
      "description": "Logical rule expression in ALL-IMPLIES format"
    },
    "description": {
      "type": "string",
      "description": "Explanation of the physiological relationship"
    },
    "variables": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "description": "List of variables used in the rule",
      "default": []
    },
    "type": {
      "type": "string",
      "description": "Type of rule pattern",
      "default": "pattern"
    }
  },
  "required": [
    "name",
    "rule",
    "description"
  ]
}
```

Appendix D

Reinforcement Learning for Synthesizing Logical Policies

D.1 Grammar-Based Policy Synthesis Algorithm

Algorithm 2 Grammar-Based Policy Synthesis

Initialize Q-table Q, PolicyGrammar G with equal weights
for each episode **do**
    Extract dynamic constants C_d from Q where $Q(s,a) > 0$
    Update grammar: G.constants $\leftarrow$ G.constants $\cup C_d$
    Generate candidate policies $P = \{p_1, \ldots, p_n\}$ using weighted grammar G
    for each policy $p_i \in P$ **do**
        Evaluate in simulation: collect rewards, glucose metrics
        Calculate score: Score$(p_i) \leftarrow$ reward $+ 0.5 \cdot$ timeInRange $- 10 \cdot$ hypoglycemiaEvents
    end for
    Select best policies P_{best} based on scores
    for each $p \in P_{best}$ **do**
        Extract components and increment their usage counts
        Update component weights based on counts
    end for
    Update Q-table with simulation transitions
end for
return Best policy p^*

Appendix E

Evaluation

E.1 Comparison Table

Aspect	Raw LLM Output	LLM-Based Rule Generation Output
Output Example	The walking activity on June 16th, 2024, was a moderate session lasting approximately 18 minutes, with an average heart rate of 107.444 BPM and a maximum of 127 BPM. The session primarily fell into the easy (75.6%) and fat-burning (24.4%) heart rate zones, suggesting a relaxed pace. Glucose levels were stable during the activity, with an average of 10.175 mmol/L and minimal fluctuations. Post-activity, glucose levels increased steadily, possibly due to post-exercise physiological responses or dietary factors. The absence of distance data limits a complete analysis but still provides insights into the session's intensity and glucose response. Overall, this was a low-intensity walk with stable glucose during exercise and an increase afterward, likely indicating a leisurely stroll without significant exertion.	`ALL(activity_type = 'Walking' AND (avg_hr > 105 OR duration_minutes` $\geq$ `20) AND NOT(hr_zone_easy > 0.5)) IMPLIES (glucose_during_mean` $\leq$ `7.0)`

Table E.1: Comparison of Raw LLM Output and LLM-Based Rule Generation Output