GlucOS: Security, correctness, and simplicity for automated insulin delivery

Hari Venugopalan
University of California, Davis
Davis, USA
hvenugopalan@ucdavis.edu

Shreyas Madhav Ambattur Vijayanand
University of California, Davis
Davis, USA
smvijay@ucdavis.edu

Caleb Stanford
University of California, Davis
Davis, USA
cdstanford@ucdavis.edu

Stephanie Crossen
University of California, Davis
Davis, USA
scrossen@ucdavis.edu

Samuel T. King
University of California, Davis
Davis, USA
kingst@ucdavis.edu

Abstract

We present GlucOS, a novel system for trustworthy automated insulin delivery. Fundamentally, this paper is about a system we designed, implemented, and deployed on real humans and the lessons learned from our experiences.

GlucOS introduces a novel architecture that allows users to personalize diabetes management using any predictive model (including ML) for insulin dosing while simultaneously protecting them against malicious models. We also introduce a novel holistic security mechanism that adapts to unprecedented changes to human physiology. We use formal methods to prove correctness of critical components and incorporate humans as part of our defensive strategy. Our evaluation includes both a real-world deployment with seven individuals and results from simulation to show that our techniques generalize. We highlight that our results are *not* from a lab study, with people using GlucOS to manage Type 1 Diabetes in their daily lives. Our results show that GlucOS maintains safety and improves glucose control even under attack conditions. This work demonstrates the potential for secure, personalized, automated healthcare systems. Our entire source code is available at this link.

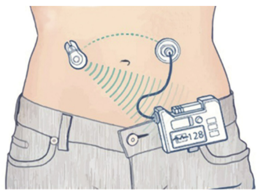
1 Introduction

Type 1 Diabetes (T1D) is a metabolic disorder where an individual's pancreas stops producing insulin. To compensate, they inject synthetic insulin. Mobile applications, called automated insulin delivery systems, use subcutaneous sensors to continuously monitor glucose concentrations (glucose is the body's primary energy source) and regulate glucose by automatically injecting insulin via an insulin pump (Figure 1). However, insulin is a dangerous hormone as too much insulin can kill people in a matter of hours [22] and too little insulin can kill people in a matter of days [13]. For automated insulin delivery systems, the key challenge is to maintain a balance: provide enough insulin to prevent dangerously high glucose levels while avoiding excessive insulin that could lead to life-threateningly low glucose levels. In this context, security means ensuring integrity for insulin dosing against malicious attacks or inadvertent errors.

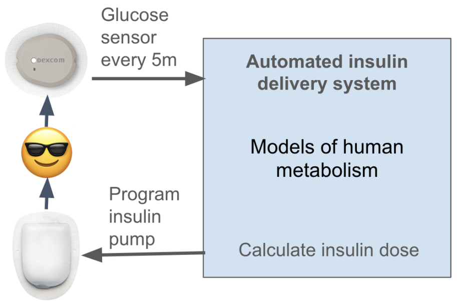
Several commercial [3, 5, 18] and open-source [47, 55] automated insulin delivery systems exist today, but none of them consider security as a primary design constraint. The complexity of these systems makes it difficult to reason about their correctness, which is crucial for security. For example, the OpenAPS [55] core function for calculating insulin doses consists of 1192 lines of Javascript code, 63 input and configuration parameters, and 90 branch statements. Furthermore, automated insulin delivery systems that use the OpenAPS algorithm embed it into native apps using a WebView for Javascript interpretation, increasing the overall complexity and attack surface. Vulnerabilities or attacks on such systems can be severe and fatal.

In this paper, we take on the challenge of building the first trustworthy automated insulin delivery system, called GlucOS. Our design includes: (1) an architecture where we apply separation principles for isolated and simple components, (2) a novel security mechanism and policy to enable secure insulin delivery, (3) a mechanism to account for unprecedented changes in human physiology, and (4) the application of formal methods to prove correct the implementation of our insulin delivery path, the most sensitive part of the system. Despite these efforts, we identify scenarios that require users to take action for security. Thus, we incorporate humans as part of our defensive strategy, providing them with agency and trust to take the required, corrective actions. In contrast to existing systems, GlucOS enables the use of any algorithm (including ML) to calculate insulin doses, while ensuring security by moderating insulin doses using 25 lines of formally verified Swift code.

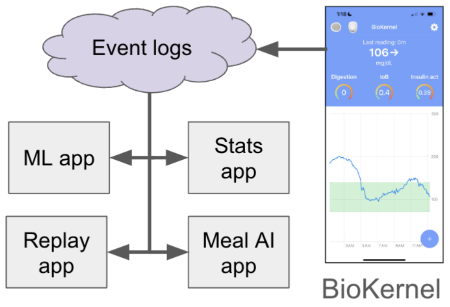
Our approach is distinguished by its holistic consideration of the entire problem of automated insulin delivery. We provide clear security boundaries, mechanisms and policies to mitigate attacks on vulnerable components, and formal methods to provide assurances that our implementation is correct. Unlike prior research that exclusively focuses on



(a) Devices for automated insulin delivery. This figure is from breakthrought1d.org.



(b) Closed-loop architecture.



(c) Overall GlucOS architecture.

Figure 1. Overview of automated insulin delivery devices, closed-loop architecture, and GlucOS.

simulation for evaluation [75, 76], we also evaluate our approach with an end-to-end system used by individuals to manage diabetes with all the uncertainties of the real world.

Given their life-threatening stakes, automated insulin delivery systems require special consideration beyond the traditional purview of the systems and security communities. Research on operating systems or web browsers can list scores of CVEs from current systems and show how their system improves security by eliminating their impact. Our position is that we need to make sure that automated insulin delivery systems are never assigned CVEs, because vulnerabilities can kill people. Thus, we focus on building a new system, using security-first principles, to avoid security vulnerabilities and mitigate their damage from the start.

Our main contribution lies in a new end-to-end system that we built and the lessons learned from our experience designing, implementing, and deploying GlucOS to humans to manage their T1D. Our novel contributions include:

- Security mechanisms and policies that protect individuals from malicious dosing algorithms, as well as drastic changes in human physiology.
- A case study describing our experiences from a real-world deployment with seven people, and results from simulation to show that our techniques generalize.
- A human-centric design that considers the user as an integral part of the system, influencing both the architecture and including them in our defensive strategy.

We give top priority to the health, safety, and ethics of the humans using GlucOS to manage their T1D. Section 10.1 summarizes our ethical considerations.

There are 8.4 million people living with T1D worldwide [33], underscoring the need for providing trustworthy systems for automated insulin delivery. Our cross-disciplinary work, carried out with rigorous medical and ethical oversight, is a first step in addressing this need.

2 Overview

Doctors and healthcare professional recognize Type-1 Diabetes as a predominantly self-managed condition and support the adoption of automated insulin delivery systems for management [23, 40, 51].

This paper describes our design for GlucOS, a system for trustworthy automated insulin delivery; we have two primary goals. First, we want the software to be simple and correct. Second, we want to identify the most vulnerable parts of the system and design a system to prevent attacks on these components or withstand successful attacks. This section describes our overall architecture for GlucOS that strives to achieve these goals, and our threat model.

2.1 Locking down the insulin delivery path

Our system architecture focuses on providing security mechanisms and policies on the insulin delivery path. The insulin delivery path in automated insulin delivery systems includes models for calculating the amount of insulin still active in the individual (insulin can take up to six hours for complete absorption), closed-loop algorithms for calculating and delivering insulin doses that could use deep neural networks to predict future metabolic states. Together, these components are both the most complex, and the most sensitive from a security perspective because they are responsible for insulin delivery.

2.2 Architecture

From a system design perspective, we strive to keep our implementation simple and enable our use of formal methods to prove correct the most critical parts. For our software system architecture, we use separation principles from the OS and microkernel areas [8, 36] applied to the application layer to push complexity away from the most critical parts of the system. This architecture is similar to secure web browsers [34, 59, 65, 70], which also apply OS separation principles at the application layer. Figure 1c shows how we decompose our automated insulin delivery system, where we strive to

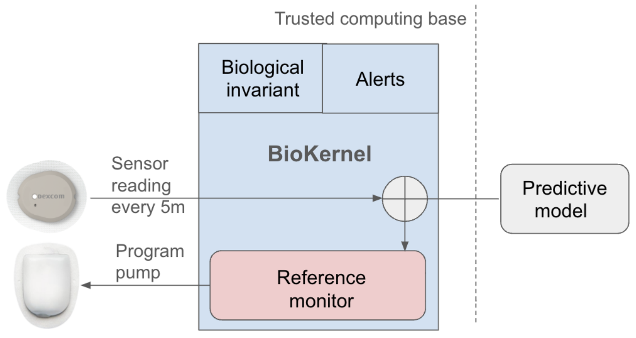


Figure 2. Overall architecture for the BioKernel.

keep our trusted computing base simple. GlucOS includes a BioKernel that forms our core trusted computing base. The BioKernel logs events that other components can ingest to learn the state of the system. These other components include apps for training ML models, visualizing statistics, an AI agent for suggesting dosing during meals, and an app for replaying closed-loop algorithm execution. Together, these components make up the overall GlucOS system.

Figure 2 shows our architecture for the BioKernel. The BioKernel is the component that interacts with the continuous glucose monitor (CGM) and insulin pump hardware, executes the closed-loop dosing algorithm, runs security and safety checks, and produces event logs that other components use to learn the state of the system.

The BioKernel handles the typical operations of automated insulin delivery systems. It runs the closed-loop algorithm every five minutes as the new CGM readings come in. Once the BioKernel receives new CGM readings, it collects the current metabolic state of the individual and sends it to a predictive model. The predictive model calculates the amount of insulin to inject based on its assessment of the individual's current metabolic state and predictions of future metabolic states. The BioKernel then issues commands to the pump to deliver the calculated amount of insulin. A pump driver converts these commands into low-level I/O to program the pump and adjust insulin delivery.

To withstand attacks, the BioKernel serves as a *reference monitor* [10] to enforce security policies on the predictive model's pump commands. This reference monitor architecture ensures that we can withstand attacks from malicious predictive models without having to know the internals of how they operate. Our reference monitor enforces algorithmic security, where we bound the amount of insulin injscted by the model to stay within dynamic safety bounds computed by a simple *reactive safe model*.

Our use of formal methods focuses on the software in the insulin delivery path. We use Hoare logic to prove key functions correct, define system states and transitions to handle runtime verification check failures safely, and we introduce the notion of a novel *biological invariant*. Our biological invariant is an end-to-end check of our theoretical calculations of how the individual's metabolism should behave compared to what we observe in practice. Violations of this invariant suggest the need to temporarily disable automated insulin delivery in response to an unprecedented change to human physiology.

2.3 Threat model

Automated insulin delivery systems inject synthetic insulin to regulate glucose concentration in the body. In the long run, high glucose or hyperglycemia leads to dangerous health outcomes such as kidney failure, heart disease, amputation etc. The most serious and immediate health risk for people who use synthetic insulin is low glucose, or hypoglycemia. Hypoglycemia results from an overdose of insulin. Once an automated insulin delivery system has delivered insulin, there is no way to remove it from the body. If untreated, severe hypoglycemia can lead to loss of consciousness, seizure, or death in a matter of hours. Thus, our primary security objective is to reduce or eliminate hypoglycemia.

Since personalization is important for T1D self-management [43], our design philosophy centers around humans. We provide flexibility for people to use any model for insulin dosing that suits their preferences. Our goal is to make sure that they stay safe even when they pick vulnerable or malicious models. Attackers may target automated insulin delivery systems to directly harm the human using the system.

For safety, we focus on protecting users from incorrect insulin doses that emerge from both, poorly chosen insulin dosing models, as well as inadvertent mistakes from benign models. Malicious or vulnerable models, if unchecked, can be devastating because they directly affect insulin delivery and can lead to life-threatening hypoglycemia.

We build GlucOS as a complete iOS app, and accordingly assume that the Swift type system, that forms the backbone of the iOS ecosystem, is secure. We only consider FDA-approved CGMs and pumps. With the FDA being stringent with security guidelines for medical devices [30], we assume that CGM and pump hardware are free of attacks. However, we assume that CGMs and pumps are prone to measurement errors within their standard error bounds [29, 64, 79].

We discuss malicious pump drivers and ways to overcome them in Appendix D.

3 Design principles

Four key principles guide our design of GlucOS:

- *Principle 1: Focus on simplicity for security mechanisms by using domain knowledge.* Rather than design a general security mechanism for a broad class of problems, we focus specifically on the highly-risky domain of automated insulin delivery. Using domain-specific insights, we provide simple security mechanisms that both humans and theorem proving software can easily reason through.

- *Principle 2: Design the system to empower humans to provide security.* Automated insulin delivery represents an extreme example of high-stakes healthcare. For people to be able to use a secure system to manage their health, they need to have agency over the security of their system. Rather than simply "alerting the human" if something unexpected happens, we empower the humans using GlucOS to be active participants in the security of their system.
- *Principle 3: Consider asymmetries in risk.* Insulin dosing possesses an inherent asymmetry where overdosing poses greater immediate danger than underdosing [13, 22]. With GlucOS, we optimize our mechanisms to be more aggressive when protecting individuals from insulin overdose.
- *Principle 4: Adaptability for security.* Although we carefully design our software to be correct, humans and glucose metabolism can change quickly. Any change that affects the correctness of models of glucose metabolism would compromise the security of the entire system. Thus, we design GlucOS to adapt to constantly changing human physiology to provide security both in our formulation of the problem and our use of formal methods.

Although we do strive to apply these principles to the entire design, there are times when there is tension between them. For example, our biological invariant (Section 5) helps with adaptability (Principle 4) by detecting when our physiological models are wrong and considers asymmetries in risk (Principle 3). But to ensure that the human remains empowered (Principle 2), we use necessarily complex logic to deal with cases when the biological invariant does *not* hold (violation of Principle 1). These types of tensions are fundamental to a real-world system that manages human health and underscores the importance of building real systems to gain experience to navigate these tradeoffs.

4 Algorithmic security: Withstanding malicious dosing algorithms

We have dual goals for our algorithmic security mechanism of being both defensive and enabling. Our security mechanism is defensive by protecting people from inappropriate insulin doses emerging from bugs, corner cases, or even attacks merging from complex predictive algorithms – be it deep neural networks or massive sets of hand-crafted rules. Our algorithmic security mechanism is enabling in allowing people to employ any predictive algorithm to personalize the system to suit their individual preferences.

To enable predictive algorithms, GlucOS's algorithmic security mechanism introduces a novel architecture that separates the use of predictive models for insulin dosing from accounting and safety logic to ensure that they are operating safely. This architecture provides flexibility in using any predictive model while enforcing bounds on insulin doses with a simple and formally verified model for safety.

4.1 Need for algorithmic security

Closed-loop automated insulin delivery systems have traditionally used reactive physiological models (such as PID controllers [42, 71]) to calculate insulin doses. These models essentially use standard physiological models to react to offset the amount of excess glucose in the body. This makes it easy to reason about the correctness of their insulin doses. However, because synthetic insulin is slow-acting and takes up to six hours to fully absorb, the reactive nature of these models limits their effectiveness in regulating elevated glucose concentrations. Thus, having the ability to inject insulin proactively by predicting glucose concentrations leads to more effective glucose regulation. But, predictive dosing opens up the individual to the risk of potentially dosing too much insulin, and current insulin delivery systems have been reluctant to adopt the most advanced forms of predictive models.

Researchers have shown that modern advances in ML, such as deep neural networks, work well in making predictions for automated insulin delivery [39, 67, 68, 74, 77, 78]. Despite their promise, none of the existing automated insulin delivery systems use deep neural networks. Instead, they use statistical methods, like linear regression or model predictive control [21, 48, 55, 58, 69]. The reason for using this more traditional form of ML is sound. It provides consistent, explainable results and has a physiological basis that people can use to vet its decisions. While advanced deep neural networks have more predictive power, current systems do not incorporate them due to their black-box nature and the potentially dire consequences of mispredictions. Even a well-tested ML model does not guarantee immunity to misprediction [25, 62], especially when predicting human metabolism that is influenced by numerous ever-changing factors such as food, exercise, stress, environment, allergies, puberty, temperature and so on [15]. An automated insulin delivery system's unchecked misprediction or "hallucination" [1] could have lethal consequences if it delivers an inappropriate insulin dose [13, 22].

Beyond the use of ML, the complexity of rule-based predictive models (such OpenAPS' thousands of lines of code [55]), make it difficult to reason about their correctness, even if they are generally effective at regulating glucose for diabetes management. Additionally, people requesting enhancements to such models [31, 32] based on their personal struggles further exacerbates model complexity.

Thus, avoiding predictive models for insulin delivery comes at the cost of effective diabetes management. On the other hand, incorporating them introduces safety concerns stemming from challenges in the explainability of complex code and deep neural networks.

4.2 Mechanism fundamentals

The key novel insight underpinning our security mechanism design is that all correct insulin dosing algorithms, regardless of whether they are predictive or reactive, will deliver the same amount of insulin over a sufficiently long period (such as a full digestion-absorption cycle). People inject insulin to facilitate glucose absorption after eating. Digestion converts food to glucose, and insulin facilitates the absorption of glucose by tissues (e.g., the brain) where the body can use it as an energy source. Since insulin dosing is dominated by food consumption, all correct algorithms should dose the same amount of insulin to cover glucose from digestion. But the timing of these insulin doses has a large impact on the individual's glucose regulation.

Our algorithmic security mechanism pairs two models. Our first model is a predictive model that anticipates future metabolic states and doses insulin proactively. Our second model is a reactive safe model that measures the current metabolic state and doses insulin based only on what it can measure currently. This pairing balances proactive insulin dosing for faster responses with a more conservative approach for safety. As per our insight, the reactive model would dose enough insulin eventually, so we use it as our baseline but provide predictive models with enough buffer to inject insulin earlier and safely.

Architecturally, in a novel approach for automated insulin delivery systems, the predictive model runs outside of our trusted computing base while the reactive safe model is part of the trusted computing base. This separation enables individuals to use whatever predictive models they want, while maintaining security. It also defends against malicious predictive models, as all insulin delivery decisions are ultimately vetted by the reactive safe model. This design also enables people to update their predictive models without violating core safety guarantees.

Conceptually, the predictive model is the primary model in our system and should control the pump most of the time. We use the reactive safe model for accounting so that we can track how far the predictive model deviates from a known-to-be-safe baseline. Then, we bound the size of this deviation. By combining these two models, we strive to get beneficial properties from both. Our approach combines the safety of a reactive model and the performance from a predictive model.

Our security mechanism draws inspiration from previous work on combining simple and complex models for safety [52, 61]. However, our approach extends these ideas by providing a novel security mechanism specifically designed for insulin delivery models. Our key contributions lie in our insight into the security properties of insulin delivery algorithms and

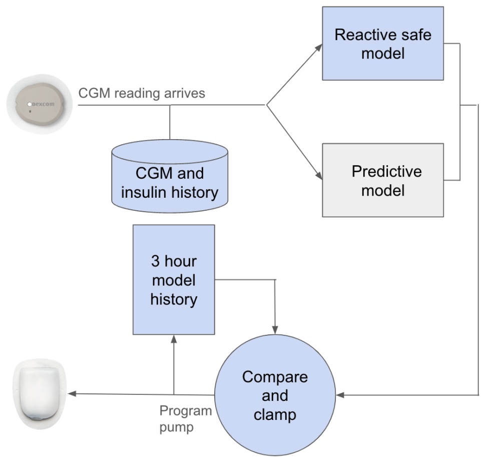


Figure 3. Algorithmic security mechanism for GlucOS.

our principles for how to apply our mechanism to automated insulin delivery systems (Section 4.3).

Figure 3 shows how our algorithmic security mechanism works. The whole process starts when a new CGM reading arrives. Once the BioKernel receives a new CGM reading, it sends the recent history of CGM readings and insulin doses to both the reactive safe model and the predictive model. These models independently produce pump commands (called a temporary basal command) that set the insulin delivery rate for the pump over the next 30 minutes. The BioKernel then collects both commands and looks back at the differences in doses between both models over the past three hours to calculate how much insulin the predictive model has dosed. If the predictive model is still within the safe insulin delivery bounds relative to the reactive safe model's doses, the BioKernel uses the predictive model's command to program the pump. If the predictive model has exhausted its limits, we clamp its calculated insulin to fit within the insulin bounds relative to the reactive safe model.

To set bounds, we base our calculations on human physiology. People produce a background level of glucose for energy. Our method for setting bounds allows the predictive model to use up to three hours worth of background insulin in advance. If the prediction is correct and glucose levels rise (e.g., from eating), the proactive insulin will manage it. If the prediction is wrong and no extra glucose appears, the body's ongoing glucose production will use the extra insulin over the next three hours.

4.3 Model principles

The mechanism described above can be implemented with various algorithms. Our contribution is *not* in the specific algorithms we use, but rather in the insight around insulin delivery being equal for all correct algorithms and how we use this insight for security. In this section, we outline the algorithms we use in our current implementation and define the principles for how to apply security to any predictive insulin dosing algorithms.

Algorithmically, we use a PID controller [27] from feedback control for our reactive safe model, and we use a deep neural network from the literature for meal predictions for our predictive ML model [53]. Although we use this model in our implementation, our system supports any predictive model. The design of the reactive safe model is crucial for algorithmic security, as it serves as the anchor for the security mechanism. The ideal reactive safe model should be safe, easy to understand, and have a formally verified implementation (Section 7). To ensure the safety of this model, we base its calculations solely on facts observed automatically from the CGM and the insulin pump. Given these high-quality sources of data, and a simple insulin absorption physiological model, we can calculate how much insulin an individual needs and how much previously injected insulin has not yet taken effect. From this calculation, we know how much insulin to inject or withhold.

These calculations are standard calculations that people who inject insulin manually use to determine dosing, making them easy for people who manage T1D or their medical care team to understand. The difference in our system is that we run these calculations automatically and every five minutes to adjust to the latest sensor readings. This basic formulation is standard and used in most automated insulin delivery systems.

Our evaluation shows that a benign predictive ML model outperforms the reactive safe model in terms of glucose regulation, both in simulation (Section 9) and for the individual who uses our system for their real-world insulin dosing (Section 10). Our evaluations in simulation also demonstrate the fatal consequences of using an incorrect or malicious ML for insulin dosing, as well as our mechanism's ability to withstand such models to protect individuals (Section 9).

5 Holistic security: Biological invariant

We recognize that human physiology is complex and it is difficult to ensure that the BioKernel's reactive safe model, which forms the basis of algorithmic security, can handle all changes to human physiology. Concretely, since the reactive safe model incorporates parameters that represent the user's physiological state, any divergence in their values could render the model ineffective in protecting the user. With GlucOS, we introduce a *biological invariant* to detect when such divergence could endanger the user, and shut off automatic insulin dosing to protect them.

From a high level, the biological invariant runs experiments every 5 minutes to empirically measure physiological parameters. These experiments compare the expected drop in glucose from prior insulin doses against the observed change in glucose sensor readings, thereby providing an end-to-end security mechanism. The biological invariant effectively serves as a "catch-all" off-switch for insulin delivery when the BioKernel's view of the user's physiology becomes inaccurate. Although our reactive safe model adapts to small changes in physiological parameters by design, the biological invariant detects large changes that operate outside of the bounds of our design.

Insulin sensitivity, which is the amount of glucose absorbed per unit of insulin, is a key physiological parameter that can change. Changes to insulin sensitivity happen from drastic changes to the user's glucose absorption, such as during exercise, when the body uses glucose without needing insulin [14], or if the pump insertion site hits a vein and delivers insulin directly into the blood stream [63]. In such cases, the user becomes more sensitive to insulin where the same amount of insulin will facilitate more absorption of glucose than expected, increasing the risk of insulin overdose. For example, if the user becomes twice as sensitive to insulin while jogging, the same insulin dose would decrease their glucose by twice the expected amount, pushing them towards dangerously low glucose (hypoglycemia).

To implement the biological invariant, we compare the actual change in the user's glucose, against the BioKernel's calculations of how much their glucose *should* have changed based on current insulin doses over a fixed time window. If the observed drop in glucose deviates beyond a clinically determined threshold of 30 mg/dl [38, 41] from the calculated drop in glucose within a window of 30 minutes, we consider the biological invariant to have been violated and shut off insulin delivery. Mathematically, we define the biological invariant as follows:

$$\Delta G^C - \Delta G^A \leq T \tag{1}$$

Here, ΔG^C denotes the BioKernel's calculated drop in glucose based on insulin doses over a 30 minute window, ΔG^A denotes the actual drop in the user's glucose over the same time window, and T is the threshold of 30 mg/dl. Crucially, we note that we cannot precisely calculate ΔG^A or ΔG^C since CGMs and pumps are prone to measurement errors. To account for these errors under standard use, we enforce error bounds on readings from CGMs [29, 64] and records from pumps [79] under standard conditions. Based on these bounds, we check the following condition on the BioKernel's current estimate of insulin sensitivity:

$$S \leq \frac{0.9 \cdot G^M_{t+\Delta t} - 1.1 \cdot G^M_t + T}{B \cdot \Delta t - 0.99 \cdot \Delta IOB^M + 0.01 \cdot \Sigma I_i^M} \tag{2}$$

Here, S is the BioKernel's estimate of the user's insulin sensitivity, G^M_t and $G^M_{t+\Delta t}$ are readings from the CGM, ΔIOB^M and ΣI_i^M are calculated from the insulin doses recorded by the pump, Δt denotes the 30 minute window, and B is the user's basal rate, a physiological parameter that often remains stable for days and is less prone to change than insulin sensitivity [46]. We refer to this condition as the *implementation*

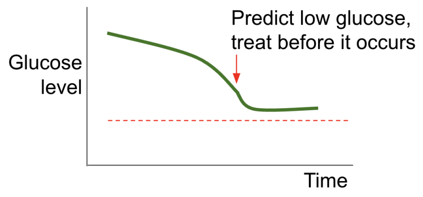


Figure 4. GlucOS predicts hypoglycemia and notifies the user before it happens so they can proactively prevent it.

invariant and use formal methods to prove that violations to this condition imply violations to the biological invariant (Section 7). Since all quantities in the implementational invariant are observable, we shut off insulin delivery when it is violated. We show how to adopt mathematical models from the literature [7, 69] to derive the implementational invariant in Appendix B.

We resume normal execution if the implementational invariant goes back to being satisfied within two hours, based on clinical guidance for exercise for people living with T1D [14, 37]. If it remains violated for more than two hours, we alert the user and transition to manual insulin delivery.

Appendix G discusses the less dangerous scenario when the actual change in glucose is higher than the calculated change.

6 Last line of defense: Humans

In this section, we describe our design and implementation for incorporating humans as a critical component of our multi-layered security system. While our algorithmic security mechanism and biological invariant defenses form the primary barriers against attacks, humans serve as the last line of defense, to take corrective action against hypoglycemia. We discuss our most interesting findings around user preferences and the architectural iterations to our design from real-world experiences.

6.1 Predictive alerts for humans

When an individual's glucose levels are not within the range of healthy values, they need to intervene to bring them back in range. Low glucose, or hypoglycemia, occurs from too much insulin, and there is no way to remove insulin that the automated insulin delivery system has already delivered. To recover from hypoglycemia, people need to eat sugar. Given that people with T1D are already active participants in managing diabetes, we focus on how we can use them to create a more comprehensive security system.

We use predictive yet deterministic mobile notifications to alert people about impending hypoglycemia before it happens (Figure 4). We focus on hypoglycemia since it presents the most immediate and dangerous risk. Alerts for hypoglycemia ae not new. All CGM software comes with non-predictive hypoglycemia alerts by default [2] and glucose prediction algorithms have been well covered in the literature [11, 68, 74]. Our contribution is using predictive alerts before someone reaches hypoglycemia as a security defense.

In a user study (Section 10.4), we reduced the amount of time six people spent in hypoglycemia as a result of using GlucOS, despite them already having CGM alerts set up prior to our study. We highlight that we did not conduct our user study in a controlled environment, but had participants use our app for predictive alerts as they managed their diabetes in the real world. We also highlight that The number of participants in our study is also roughly consistent with recent human factors in computing research [12, 19, 43, 73]. Our results show that predictive alerts can be effective as a security measure against hypoglycemia.

Using a simple and deterministic algorithm is important since it runs as a part of our trusted computing base. Algorithmically, we predict hypoglycemia using linear regression over the last 20 minutes of CGM readings and extrapolate forward 15 minutes. We trigger alerts if our prediction crossed the hypoglycemia threshold in the next 15 minutes.

The most interesting finding from our study was that personalization was important. We provided people with the ability to set the prediction threshold and specify how often they wanted the alert to repeat while they remained in hypoglycemia. Five of the six participants customized their alerts and the remaining participant noted that the default values we picked coincided with their alerting preferences. In our first implementation we did not provide the ability to customize their settings and we tried to make the usability vs security tradeoff algorithmically and automatically. But in the end, simplifying the system to enable the personalization of alerting policy was important for our participants.

Our finding has the potential to have broader implications for security in healthcare systems. Often, security systems favor uniformity and consistency [24, 28], but when managing people's health we found that allowing people to balance alert fatigue and security increased engagement and potentially increased protection, based on each individual's needs.

6.2 Architectural implications

One interesting design iteration we made was moving the alerting logic into the BioKernel itself. In our first design, consistent with our separation principles, the alerting logic resided in a separate app. This alerting app learned the state of the system using our cloud-based event logging system. However, one of our participants went on a canoeing trip and was disconnected from the internet, so they could not get alerts. Ideally iOS would provide inter-process communication mechanisms to pass data from the BioKernel directly to the alerting app, but iOS does not have the appropriate APIs for this style of communication. Thus, we moved the alerting logic into the BioKernel, violating our principle of separation, but enabling our users to get alerts even when they are disconnected from the internet.

Moving the alerting logic into the BioKernel did have security implications. It adds complexity to our trusted computing base, but we carefully manage this tradeoff. We minimize the added complexity by using traditional, well-understood algorithms for prediction rather than complex ML models and reuse existing functionality within the BioKernel when possible to keep the implementation simple. This approach allowed us to maintain a high level of confidence in the security and correctness of our alerting system while still providing the necessary functionality.

7 Formal verification

We formally verify the most critical components of GlucOS to prevent unexpected and undesired outcomes [66]. Specifically, we verify the implementation for: (1) delivering bolus insulin doses (a single dose delivered immediately), (2) delivering basal insulin doses (multiple doses delivered at a constant rate over a specified time period), (3) clamping insulin doses using the reactive safe model, and (4) enforcing the biological invariant. We consider these components to be the most critical since they implement our security mechanisms and directly control insulin delivery.

Formally verifying our GlucOS's implementation in Swift presented an interesting challenge since automatic translators do not exist between Swift and languages for verification, such as Dafny. While automatic translators exist between Dafny and other languages (such as C) we could not pick those languages over Swift for implementation. This is because Swify provides the most support to build iOS applications and our goal was to build a complete automated insulin delivery application for real-world use.

To overcome this limitation, we manually ported our Swift implementation to Dafny. While manual translation is prone to human error, we took important steps to reduce the likelihood of such errors. Concretely, we ported our Swift code line-by-line to Dafny, preserved variable and function names across implementations [20], and wrote unit tests for each function in both languages to ensure behavioral consistency.

We establish invariants among the aforementioned critical components along the insulin delivery path [66]. These invariants enforce critical constraints such as maintaining appropriate units insulin doses, disallowing immediately successive bolus doses, ensuring basal doses match doses supported by the pump etc. Collectively these invariants are important to ensure the expected safe operation of the BioKernel. We discover 9 bugs in our original implementation by verifying 31 invariants across 8 functions. Other than the biological invariant (Section 5), some of our other invariants enforced that there should be a gap of at least 4.2 minutes between bolus doses, the amount of insulin on board should also take into account the insulin from basal doses etc. We list all our invariants and bugs in Appendix F.

From our experience with formal verification, we report two key takeaways about our implementation for secure automated insulin delivery. First, we were able to prove the correctness of even those components that involve unobservable quantities (such as the user's true physiological state in the biological invariant) under standard operation. Second, while we expected most bugs to reside within the implementation of core algorithms, they were fully concentrated among peripheral components such as user settings.

7.1 Capturing true physiological state

Glucose readings from CGMs and insulin doses recorded from pumps do not reflect the user's true physiological state. This is because CGMs and pumps are subject to measurement errors, making precise glucose concentrations and insulin doses unavailable in Swift.

To model this discrepancy, we use ghost variables in Dafny [44] to represent the true glucose values and insulin doses. We then introduce *device invariants* that bound the maximum measurement error within standard use, i.e., 10% for CGMs [29, 64] and 1% for pumps [79]. We enforce these bounds as preconditions when verifying the implementation of all our core components. Most importantly, we proved the correctness of the implementational invariant (Equation (2) in Section 5) to verify the biological invariant (Equation (1) in Section5 as a post condition in Dafny).

7.2 Discovering bugs outside the core algorithms

7 out of the 9 bugs we discovered with formal verification were related to issuing insulin delivery commands with invalid parameters. With these bugs, users could specify negative amounts of insulin for basal and bolus doses, set 0-second duration for basal doses, allowing the time to stop a basal dose to be preced the start time of the dose etc. Other bugs involved insfficient error handling, for example when disallowing certain invalid bolus doses. We did not discover any bugs in GlucOS's core algorithms, such as those used by the reactive safe model. We resolved bugs by updating our Swift implementation to match the formally verified Dafny code.

8 Implementation

In our implementation, the BioKernel is an iOS app that consists of 5.1k lines of code in our trusted computing base. It communicates with a Dexcom G6 or G7 CGM and Omnipod DASH insulin pump via Bluetooth low energy, and runs the core closed-loop algorithm. For the BioKernel, we use the Swift programming language, the Actor abstraction for data-race-free concurrency, and JSON files stored on disk to capture persistent state. Although the BioKernel is a fully featured and stand alone automated insulin delivery system, GlucOS uses an event logging system (Figure 1c) that enables separate apps to extend functionality while remaining isolated from the BioKernel.

9 Evaluation

In this section, we evaluate GlucOS's algorithmic security mechanism in simulation. In the next section (Section 10), we evaluate all components of GlucOS in the real world. We do not evaluate the biological invariant in simulation since existing simulators do not support changes to human physiology.

9.1 How effective is GlucOS's algorithmic security mechanism?

We evaluate GlucOS's algorithmic security mechanism on 21 virtual humans from an FDA-approved simulator [7]. The mechanism should protect users against untrusted models that administer incorrect insulin doses (overdose or underdose), while minimally impeding algorithms that are efficient at managing T1D. While subtle attacks are also possible, we focus on extreme attacks in our evaluation since they have the most adverse consequences.

We first run simulations with a predictive ML model inspired from prior research [53], without incorporating our security mechanism to set a baseline of an effctive algorithm in managing T1D. Then, in the same simulation scenarios, we introduce an order of magnitude increase to the baseline algorithm's insulin doses to quantify the risks of untrusted algorithms. We then repeat these experiments with our security mechanism to evaluate its impact.

The American Diabetes Association (ADA) recommends that individuals with T1D spend at least 70% of their time in the range (TIR) of 70-180 mg/dl to remain healthy [6, 26, 54]. Glucose levels less than 70 mg/dl (hypoglycemia) put individuals at risk for acute complications like impaired consciousness, seizure, and death, while levels greater than 180 mg/dl (hyperglycemia) increase risks for long-term vascular damage. We use TIR as our metric in these evaluations.

9.1.1 Without algorithmic security mechanism. Predictive ML model baseline With the predictive ML model, we followed all ML best practices in carefully tuning hyperparameters and evaluating the model on simulation scenarios that were different from those used in training. Figure 5 shows the average proportion of time spent in the recommended healthy range (TIR) by virtual humans from different age groups when employing the predictive ML model. We see that individuals across all groups spend over 85% of their time in range compared to 77% when using only the reactive safe model. These results provide a baseline for a predictive algorithm that is effective in managing T1D.

Incorrect ML dosing In this section, we discuss an experiment with a malicious algorithm that intentionally doses too much insulin. Concretely, our malicious algorithm doses ten times the amount of insulin computed by the previously described predictive ML model. We ran this algorithm within the simulator on the same virtual humans on the same scenarios as the previous experiment.

Age group	Reactive safe model	TIR with predictive ML model	TIR with clamped ML model
Adults	88.25%	93.49%	92.88%
Adolescents	81.98%	85.23%	82.38%
Children	77.31%	86.6%	85.42%

Figure 5. Average proportion of time spent in range (TIR) across 21 virtual humans with a reactive safe model, a predictive ML model, and clamping the predictive ML with the reactive safe model for security.

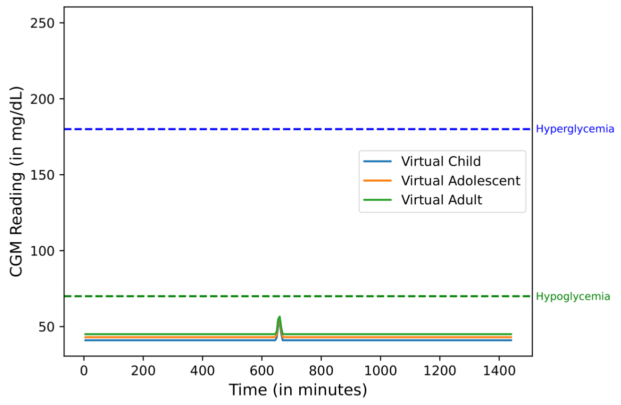


Figure 6. Variation in CGM readings for three virtual humans when running a predictive ML algorithm that intentionally doses ten times the amount of the required insulin. The CGM readings staying below the hypoglycemic bound would have likely been fatal to all three individuals.

Figure 6 shows the CGM readings on three randomly chosen virtual humans. The readings fully go below the hypogycemic bound, which would have killed them in the real world. Across all virtual humans, we see an average drop in TIR from 88.44% to 0% when dosing too much insulin. Correspondingly, we observe an average increase in the time spent in hypoglycemia from 0% to 100%. We report similar results with intentional underdosing in Appendix C. These results demonstrate the need for algorithmic security.

9.1.2 With algorithmic security mechanism. Impact on incorrect dosing. We repeat experiments from the previous section with the same simulation scenarios. We run the same malicous model to overdose insulin, but also run our reactive safe model (Section 4).

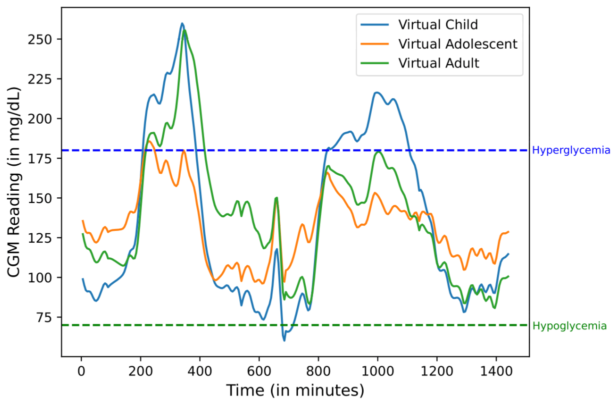


Figure 7. Variation in CGM readings for three virtual humans when clamping an algorithm that intentionally tries to dose ten times the amount of the required insulin. On average, CGM readings stay in range over 70% of the time.

Figure 7 show CGM readings on the same three virtual humans on the same scenarios as the experiment in the previous section. We see that all of them spend over 70% of their time in the healthy range, showing that GlucOS protects individuals from malicious overdosing. On average, across all virtual humans, we only see a drop in TIR from 88.44% with correct dosing to 78.22% when employing GlucOS while overdosing. We also report similar results on experiments with a malicious algorithm that underdoses insulin in Appendix C.

While the times spent in range are less than those of correct dosing, the impact of incorrect dosing is significantly dampened with individuals spending the recommended amount of time in the healthy range, despite active attack.

Impact on effective benign algorithms. We evaluate the impact of GlucOS's algorithmic security mechanism on benign algorithms that are effective in managing T1D such as the baseline predictive ML model. We repeat the experiment of running the predictive ML model on all virtual humans in the same scenarios with the same parameters for the clamps and reactive model used to evaluate incorrect dosing.

Comparing data in Figure 5 we see that there is only a slight drop in the average proportion of time spent in range by virtual humans across all age groups when employing GlucOS's algorithmic security mechanism. When employing the reactive safe model as is to calculate insulin doses, we report an average TIR of 82.48% which is lower than the average TIR of 86.84% observed when running the predictive ML model with GlucOS's security mechanism. This shows that GlucOS's algorithmic security mechanism does not impede effective T1D management.

10 Using GlucOS in the real world

In this section, we report on our experiences working with seven individuals using GlucOS to help manage their T1D. Six people are students who only used the predictive alerting feature of the BioKernel (Section 6) for two months, from June 2024 - August 2024. One individual, who we will call Bob, has been using the full GlucOS system since November 2023 for insulin delivery and predictive alerting. The number of participants in our study is roughly consistent with existing research in human factors in computing [12, 19, 43, 73]. More importantly, we report results on indiviuals using GlucOS to manage their diabetes in the real world.

10.1 Ethical considerations

In this section, we outline the steps we take to ensure that we uphold high ethical standards and ensure that the people using our system, are safe. Our user study for the six students was approved by our university's IRB. Bob's use GlucOS was determined by our university's IRB to be "self evaluation" and exempt from IRB.

One of the co-authors of this paper is a board-certified Endocrinologist, who specializes in T1D. They designed the safety protocol, which defines the specific criteria we use to stop the study, if needed. Appendix A includes a more detailed discussion of our ethics and safety protocols.

10.2 Is an ML-based closed-loop system safe and effective in practice?

Bob has been using GlucOS in a closed loop since November 23rd, 2023. From November 23 to January 28th, 2024 we used our reactive safe model to make control decisions. Starting on January 28th Bob started to use our predictive ML model for periods of time to get used to the new system, and then starting on January 31st, 2024 starting using our predictive ML model exclusively. We report on one week's worth of data from the predictive ML model starting on January 31st ending on February 6th, 2024.

Overall, Bob had the best results when using ML. His time spent in range increased from 96.3% to 97.2%, and notably he spent no time in hypoglycemia (below 70 mg/dl), going down from 0.67% when not using ML. More importantly, Bob's A1C, which captures his long-term glucose concentration, dropped to 5.8% as a result of using GlucOS. This result demonstrates a significant health outcome since it matches up to non-diabetics who have an A1C of 5.6% or lower.

Next, we measure how often the predictive ML model was responsible for programming the pump vs our reactive safe model. For the one week when Bob was using the predictive ML model, 33.0% of the time both the predictive ML model and reactive safe model issued the same commands, 39.9% of the time the BioKernel used the predictive ML model's commands, and 27.1% of the time the BioKernel used the reactive safe model's commands. The reactive safe model having to take over to issue commands is not surprising since ML models are not immune to mispredictions, especially when taking the variability of the real world into account. We believe that this distribution represents a suitable balance for providing the ML with enough flexibility to improve outcomes, while having tight enough bounds to limit potential damage from bad or malicious ML predictions.

As of July 2025, Bob has swapped the predictive ML model with a rule-based model similar to OpenAPS' model described in Section 1.

10.3 Do biological invariant violations trigger in practice?

To determine if biological invariant violations trigger in practice, we review 90 days of data for Bob, from May 29th, 2024 to August 27th, 2024. During this period of time, biological invariant violations occurred 1.6k times, or 9.0% of the time. Of these violations there were 49 sequences of 30 minutes or longer with the longest sequence being 65 minutes.

These results suggest that although natural variations in glucose metabolism do lead to biological invariant violations, none of them were long enough to trigger our most severe action of transitioning to manual insulin dosing after two hours. We manually reviewed one weeks worth of Bob's data and discovered that all five violations within this period were caused by exercise.

10.4 Are predictive alerts useful for security?

To measure predictive alerts as a security mechanism to avert hypoglycemia (Section 6), we recruited six students who live with T1D and had them use the GlucOS predictive alerts to complement their default CGM alerts. Our primary metric is time spent below range (i.e., below 70 mg/dl), where a decrease in the amount of time spent in hypoglycemia suggests that predictive alerts are effective to empower people to mitigate the most immediate and dangerous security risks.

For the two month study, our participants decreased their average time in hypoglycemia from 2.7% down to 1.6%, and all individuals experienced a decrease in the amount of time spent in hypoglycemia. For context, participants in a state-of-the-art automated insulin delivery clinical trial experienced a decrease of time in hypoglycemia of 0.9% [16], showing that our 1.1% decrease is substantial.

We also report positive feedback from all participants during monthly interviews. Other than customization, participants acknowledged the importance of being able to reason about the alerts once we explained that we used linear regression for prediction. They reported that this helped forge trust in our system which in turn reduced their cognitive load with diabetes. All participants have continued using our system even after the study's conclusion.

Overall, our results show how predictive alerts are useful for automated insulin delivery.

10.5 How does the BioKernel perform?

We evaluate two aspects of GlucOS's system performance in the real world. First, we measure how long our closed-loop algorithm takes to run and second, we count how often the closed-loop algorithm runs successfully end-to-end. We use Bob's iPhone 14 to collect all of the data we show and these numbers are averages across the entire eleven month period.

On average, the BioKernel spends 4.3s running the closed loop algorithm, where almost half of that time is dominated by CGM and pump communications. Of the components within the closed-loop algorithm, the ML predictions take 2.2s on average and the safety logic takes 19ms (0.019s). Since the system runs in the background, we believe that iOS runs the device with reduced computational capacity to save power. But overall, 4.3s of wall-clock time for every five minutes our closed-loop algorithm runs with only a tiny fraction of that coming from our safety logic represents practical system performance.

For successful closed-loop end-to-end execution, over the evaluation period the closed-loop algorithm runs successfully 96.5% of the time, with 2.1% of the runs failing due to pump communication, which is our largest class of failure.

11 Related work

The complexity of automated insulin delivery systems and the fact that they make real-time healthcare decisions, makes their security paramount. Current research on security for these systems focuses primarily on evaluating algorithms using simulations [75, 76]. Our experience shows that when faced with the practical challenges of real systems, algorithms designed in simulations often prove insufficient or inapplicable. These real-world complexities require a fundamentally different approach to designing security for automated insulin delivery systems. GlucOS moves beyond simulation and tackles the challenges of real-world systems.

Previous research has looked at the security of implanted medical devices in general [17, 35, 60], in addition to looking at insulin pumps in particular [45, 57], with more recent work looking at providing improved security [9, 50]. Also, recent work has looked at applying formal methods to insulin pumps for high assurance [56]. They focus on devices and their communication channels. With GlucOS, we assume that these devices are secure and focus on automated insulin delivery software.

12 Conclusion

The design of GlucOS, while grounded in established OS and security principles, incorporates several non-obvious elements that emerged from our deep engagement with the problem domain and real-world testing.

First, our approach to algorithmic security used classic security techniques, like reference monitors and the Simplex architecture. However, our novel insight around all algorithms dosing the same amount of insulin led to the application of these classic techniques. Plus, from our exploration, the most interesting part was *not* the use ML, but the principle of repurposing reactive models for security.

Second, the biological invariant emerged as a key check in our system, addressing a blind spot in typical simulations that ignore changing human dynamics. This forced us to develop novel formal methods to handle scenarios where our glucose metabolism models could be drastically wrong.

Third, our approach to human interaction in the system was nuanced. We found that simply alerting humans was not sufficient. For security alerts, personalization, customization, and simplicity was important.

These non-obvious design decisions underscore the complexity of creating a truly trustworthy automated insulin delivery system. They reflect the value of combining theoretical security principles with practical, real-world testing and domain-specific knowledge. This interdisciplinary research drew on deep expertise from operating systems, computer security, medicine, and ethics to create a practical system.

References

[1] Chatbots may ‘hallucinate’ more often than many realize. https://www.nytimes.com/2023/11/06/technology/chatbots-hallucination-rates.html.

[2] Dexcom g7 alarms and alerts. https://www.dexcom.com/faqs/how-do-i-customize-dexcom-g7-alert-settings.

[3] Insulet omnipod 5. https://www.omnipod.com/.

[4] Omnipod bluetooth pumpmanager for loop. https://github.com/LoopKit/OmniBLE/tree/dev.

[5] Tandem control iq. https://www.tandemdiabetes.com/products/automated-insulin-delivery/control-iq.

[6] Time-in-range and diabetes, 2022. https://www.endocrine.org/patient-engagement/endocrine-library/time-in-range-and-diabetes.

[7] simglucose, 2024. https://github.com/jxx123/simglucose.

[8] Mike Accetta, Robert Baron, William Bolosky, David Golub, Richard Rashid, Avadis Tevanian, and Michael Young. Mach: A new kernel foundation for unix development. 1986.

[9] Usman Ahmad, Hong Song, Awais Bilal, Shahzad Saleem, and Asad Ullah. Securing insulin pump system using deep learning and gesture recognition. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 1716–1719. IEEE, 2018.

[10] James P Anderson et al. Computer security technology planning study. Technical report, ESD-TR-73-51, 1972.

[11] Sunny Arora, Shailender Kumar, and Pardeep Kumar. Multivariate models of blood glucose prediction in type1 diabetes: A survey of the state-of-the-art. *Current Pharmaceutical Biotechnology*, 24(4):532–552, 2023.

[12] Clara-Maria Barth, Jürgen Bernard, and Elaine M. Huang. "it's like a glimpse into the future": Exploring the role of blood glucose prediction technologies for type 1 diabetes self-management. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery.

[13] Stephen R Benoit, Yan Zhang, Linda S Geiss, Edward W Gregg, and Ann Albright. Trends in diabetic ketoacidosis hospitalizations and in-hospital mortality—united states, 2000–2014. *Morbidity and Mortality Weekly Report*, 67(12):362, 2018.

[14] LB Borghouts and HA Keizer. Exercise and insulin sensitivity: a review. *International journal of sports medicine*, 21(01):1–12, 2000.

[15] Adam Brown. 42 factors that affect blood glucose?! a surprising update, 2022. https://diatribe.org/42-factors-affect-blood-glucose-surprising-update.

[16] Sue A Brown, Boris P Kovatchev, Dan Raghinaru, John W Lum, Bruce A Buckingham, Yogish C Kudva, Lori M Laffel, Carol J Levy, Jordan E Pinsker, R Paul Wadwa, et al. Six-month randomized, multicenter trial of closed-loop control in type 1 diabetes. *New England Journal of Medicine*, 381(18):1707–1717, 2019.

[17] Wayne Burleson, Shane S Clark, Benjamin Ransford, and Kevin Fu. Design challenges for secure implantable medical devices. In *Proceedings of the 49th annual design automation conference*, pages 12–17, 2012.

[18] Luz E. Castellanos, Courtney A. Balliro, Jordan S. Sherwood, Rabab Jafri, Mallory A. Hillard, Evelyn Greaux, Rajendranath Selagamsetty, Hui Zheng, Firas H. El-Khatib, Edward R. Damiano, and Steven J. Russell. Performance of the Insulin-Only iLet Bionic Pancreas and the Bihormonal iLet Using Dasiglucagon in Adults With Type 1 Diabetes in a Home-Use Setting. *Diabetes Care*, 44(6):e118–e120, 06 2021.

[19] Yoon Jeong Cha, Yasemin Gunal, Alice Wou, Joyce Lee, Mark W Newman, and Sun Young Park. Shared responsibility in collaborative tracking for children with type 1 diabetes and their parents. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery.

[20] Nathan Chong, Byron Cook, Konstantinos Kallas, Kareem Khazem, Felipe R Monteiro, Daniel Schwartz-Narbonne, Serdar Tasiran, Michael Tautschnig, and Mark R Tuttle. Code-level model checking in the software development workflow. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, pages 11–20, 2020.

[21] Claudio Cobelli, Eric Renard, and Boris Kovatchev. Artificial Pancreas: Past, Present, Future. *Diabetes*, 60(11):2672–2682, 10 2011.

[22] Philip E Cryer. Severe hypoglycemia predicts mortality in diabetes. *Diabetes care*, 35(9):1814–1816, 2012.

[23] Pooja M. Desai, Elliot G. Mitchell, Maria L. Hwang, Matthew E. Levine, David J. Albers, and Lena Mamykina. Personal health oracle: Explorations of personalized predictions in diabetes self-management. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery.

[24] Gurpreet Dhillon, Tiago Oliveira, Santa Susarapu, and Mario Caldeira. Deciding between information security and usability: Developing value based objectives. *Computers in Human Behavior*, 61:656–666, 2016.

[25] Elvis Dohmatob. Generalized no free lunch theorem for adversarial robustness, 2019.

[26] Nuha A. ElSayed, Grazia Aleppo, Vanita R. Aroda, Raveendhara R. Bannuru, Florence M. Brown, Dennis Bruemmer, Billy S. Collins, Marisa E. Hilliard, Diana Isaacs, Eric L. Johnson, Scott Kahan, Kamlesh Khunti, Jose Leon, Sarah K. Lyons, Mary Lou Perry, Priya Prahalad, Richard E. Pratley, Jane Jeffrie Seley, Robert C. Stanton, and on behalf of the American Diabetes Association Gabbay, Robert A. 6. glycemic targets: Standards of care in diabetes—2023. *Diabetes Care*, 46:S97–S110, 2022.

[27] Gene F Franklin, J David Powell, Abbas Emami-Naeini, and J David Powell. *Feedback control of dynamic systems*, volume 4. Prentice hall Upper Saddle River, 2002.

[28] Steven Furnell. The usability of security – revisited. *Computer Fraud and Security*, 2016(9):5–11, 2016.

[29] Satish K Garg, Mark Kipnes, Kristin Castorino, Timothy S Bailey, Halis Kaan Akturk, John B Welsh, Mark P Christiansen, Andrew K Balo, Sue A Brown, Jennifer L Reid, et al. Accuracy and safety of dexcom g7 continuous glucose monitoring in adults with diabetes. *Diabetes technology & therapeutics*, 24(6):373–380, 2022.

[30] Will Garvin, Buchanan Ingersoll Michael McLaughlin, and Rooney. Understanding the fda's new medical device cybersecurity guidelines. https://www.meddeviceonline.com/doc/understanding-the-fda-s-new-medical-device-cybersecurity-guidelines-0001, 2023.

[31] Trio GitHub. Hypo protect. https://github.com/nightscout/Trio/issues/449, 2025.

[32] Trio GitHub. Offer custom bg target when cob=0. https://github.com/nightscout/Trio/issues/631, 2025.

[33] Gabriel A Gregory, Thomas I G Robinson, Sarah E Linklater, Fei Wang, Stephen Colagiuri, Carine de Beaufort, Kim C Donaghue, Jessica L Harding, Pandora L Wander, Xinge Zhang, Xia Li, Suvi Karuranga, Hongzhi Chen, Hong Sun, Yuting Xie, Richard Oram, Dianna J Magliano, Zhiguang Zhou, Alicia J Jenkins, Ronald CW Ma, Dianna J Magliano, Jayanthi Maniam, Trevor J Orchard, Priyanka Rai, and Graham D Ogle. Global incidence, prevalence, and mortality of type 1 diabetes in 2021 with projection to 2040: a modelling study. *The Lancet Diabetes & Endocrinology*, 10(10):741–760, 2022.

[34] Chris Grier, Shuo Tang, and Samuel T King. Secure web browsing with the op web browser. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 402–416. IEEE, 2008.

[35] Daniel Halperin, Thomas S Heydt-Benjamin, Kevin Fu, Tadayoshi Kohno, and William H Maisel. Security and privacy for implantable medical devices. *IEEE pervasive computing*, 7(1):30–39, 2008.

[36] Hermann Härtig, Michael Hohmuth, Jochen Liedtke, Sebastian Schönberg, and Jean Wolter. The performance of μ-kernel-based systems. *ACM SIGOPS Operating Systems Review*, 31(5):66–77, 1997.

[37] UCLA Health. Diabetes exercise guidelines. https://www.uclahealth.org/medical-services/endocrinology/diabetes/type-1-diabetes/exercise-guidelines, 2025.

[38] Irl B Hirsch, Dana Armstrong, Richard M Bergenstal, Bruce Buckingham, Belinda P Childs, William L Clarke, Anne Peters, and Howard Wolpert. Clinical application of emerging sensor technologies in diabetes management: consensus guidelines for continuous glucose monitoring (cgm). *Diabetes technology & therapeutics*, 10(4):232–246, 2008.

[39] Peter G. Jacobs, Pau Herrero, Andrea Facchinetti, Josep Vehi, Boris Kovatchev, Marc D. Breton, Ali Cinar, Konstantina S. Nikita, Francis J. Doyle, Jorge Bondia, Tadej Battelino, Jessica R. Castle, Konstantia Zarkogianni, Rahul Narayan, and Clara Mosquera-Lopez. Artificial intelligence and machine learning for improving glycemic control in diabetes: Best practices, pitfalls, and opportunities. *IEEE Reviews in Biomedical Engineering*, 17:19–41, 2024.

[40] X.Y. Khor, J.M. Pappachan, and M.S. Jeeyavudeen. Individualized diabetes care: Lessons from the real-world experience. *World Journal of Clinical Cases*, 11(13):2890–2902, may 2023.

[41] Boris P Kovatchev, Linda A Gonder-Frederick, Daniel J Cox, and William L Clarke. Evaluating the accuracy of continuous glucose-monitoring sensors: continuous glucose–error grid analysis illustrated by therasense freestyle navigator data. *Diabetes Care*, 27(8):1922–1928, 2004.

[42] Taisa Kushner, David Bortz, David M Maahs, and Sriram Sankaranarayanan. A data-driven approach to artificial pancreas verification and synthesis. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 242–252. IEEE, 2018.

[43] Sabrina Lakhdhir, Chehak Nayar, Fraser Anderson, Helene Fournier, Liisa Holsti, Irina Kondratova, Charles Perin, and Sowmya Somanath. Glucomaker: Enabling collaborative customization of glucose monitors. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery.

[44] K Rustan M Leino. Dafny: An automatic program verifier for functional correctness. In *International conference on logic for programming artificial intelligence and reasoning*, pages 348–370. Springer, 2010.

[45] Chunxiao Li, Anand Raghunathan, and Niraj K Jha. Hijacking an insulin pump: Security attacks and defenses for a diabetes therapy system. In *2011 IEEE 13th international conference on e-health networking, applications and services*, pages 150–156. IEEE, 2011.

[46] A. M. Lindmeyer, J. J. Meier, and M. A. Nauck. Patients with type 1 diabetes treated with insulin pumps need widely heterogeneous basal rate profiles ranging from negligible to pronounced diurnal variability. *Journal of Diabetes Science and Technology*, 15(6):1262–1272, November 2021. Epub 2020 Aug 18.

[47] Loop. An automated insulin delivery app for ios, built on loopkit. https://github.com/LoopKit/Loop.

[48] Katrin Lunze, Tarunraj Singh, Marian Walter, Mathias D Brendel, and Steffen Leonhardt. Blood glucose control algorithms for type 1 diabetic patients: A methodological review. *Biomedical signal processing and control*, 8(2):107–119, 2013.

[49] Haohui Mai, Edgar Pek, Hui Xue, Samuel Talmadge King, and Parthasarathy Madhusudan. Verifying security invariants in expressos. *SIGPLAN Not.*, 48(4):293–304, mar 2013.

[50] Eduard Marin, Dave Singelée, Bohan Yang, Ingrid Verbauwhede, and Bart Preneel. On the feasibility of cryptography for a wireless insulin pump system. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, pages 113–120, 2016.

[51] L.K. Midyett. One size fits all versus individualized medicine in type 1 diabetes management. *Diabetes Technology & Therapeutics*, 25(S3):S42–S47, jun 2023.

[52] Sibin Mohan, Stanley Bak, Emiliano Betti, Heechul Yun, Lui Sha, and Marco Caccamo. S3a: Secure system simplex architecture for enhanced security and robustness of cyber-physical systems. In *Proceedings of the 2nd ACM international conference on High confidence networked systems*, pages 65–74, 2013.

[53] Clara Mosquera-Lopez, Leah M Wilson, Joseph El Youssef, Wade Hilts, Joseph Leitschuh, Deborah Branigan, Virginia Gabo, Jae H Eom, Jessica R Castle, and Peter G Jacobs. Enabling fully automated insulin delivery through meal detection and size estimation using artificial intelligence. *npj Digital Medicine*, 6(1):39, 2023.

[54] David M Nathan and DCCT/Edic Research Group. The diabetes control and complications trial/epidemiology of diabetes interventions and complications study at 30 years: overview. *Diabetes care*, 37(1):9–16, 2014.

[55] OpenAPS. The open artificial pancreas system project. https://github.com/openaps.

[56] Abhinandan Panda, Srinivas Pinisetty, and Partha Roop. A secure insulin infusion system using verification monitors. In *Proceedings of the 19th ACM-IEEE International Conference on Formal Methods and Models for System Design*, pages 56–65, 2021.

[57] Nathanael Paul, Tadayoshi Kohno, and David C Klonoff. A review of the security of insulin pump infusion systems. *Journal of diabetes science and technology*, 5(6):1557–1562, 2011.

[58] Griselda Quiroz. The evolution of control algorithms in artificial pancreas: A historical perspective. *Annual Reviews in Control*, 48:222–232, 2019.

[59] Charles Reis, Alexander Moshchuk, and Nasko Oskov. Site isolation: Process separation for web sites within the browser. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1661–1678, 2019.

[60] Michael Rushanan, Aviel D Rubin, Denis Foo Kune, and Colleen M Swanson. Sok: Security and privacy in implantable medical devices and body area networks. In *2014 IEEE symposium on security and privacy*, pages 524–539. IEEE, 2014.

[61] Lui Sha. Using simplicity to control complexity. *IEEE Software*, 18(04):20–28, 2001.

[62] Tom F. Sterkenburg and Peter D. Grünwald. The no-free-lunch theorems of supervised learning. *Synthese*, 199(3–4):9979–10015, June 2021.

[63] Leah V Steyn, Delaney Drew, Demetri Vlachos, Barry Huey, Katie Cocchi, Nicholas D Price, Robert Johnson, Charles W Putnam, and Klearchos K Papas. Accelerated absorption of regular insulin administered via a vascularizing permeable microchamber implanted subcutaneously in diabetic rattus norvegicus. *Plos one*, 18(6):e0278794,

2023.

[64] Tandem Diabetes Care. Accuracy of Dexcom G7. https://www.tandemdiabetes.com/support-center/cgm-sensors/dexcom-g7/article/accuracy-of-dexcom-g7, 2024.

[65] Shuo Tang, Haohui Mai, and Samuel T King. Trust and protection in the illinois browser operating system. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, 2010.

[66] Muffy H. Thomas. The story of the therac-25 in lotos. 1994.

[67] N. S. Tyler, C. M. Mosquera-Lopez, L. M. Wilson, and et al. An artificial intelligence decision support system for the management of type 1 diabetes. *Nat Metab*, 2:612–619, 2020.

[68] Josep Vehí, Iván Contreras, Silvia Oviedo, Lyvia Biagi, and Arthur Bertachi. Prediction and prevention of hypoglycaemic events in type-1 diabetic patients using machine learning. *Health Informatics Journal*, 26(1):703–718, 2020. PMID: 31195880.

[69] Roberto Visentin, Michele Schiavon, Rita Basu, Ananda Basu, Chiara Dalla Man, and Claudio Cobelli. Chapter 6 - physiological models for artificial pancreas development. In Ricardo S. Sánchez-Peña and Daniel R. Cherñavvsky, editors, *The Artificial Pancreas*, pages 123–152. Academic Press, 2019.

[70] Helen J Wang, Chris Grier, Alexander Moshchuk, Samuel T King, Piali Choudhury, and Herman Venter. The multi-principal os construction of the gazelle web browser. In *USENIX security symposium*, volume 28, 2009.

[71] Stuart A Weinzimer, Garry M Steil, Karena L Swan, Jim Dziura, Natalie Kurtz, and William V Tamborlane. Fully automated closed-loop insulin delivery versus semiautomated hybrid control in pediatric patients with type 1 diabetes using an artificial pancreas. *Diabetes care*, 31(5):934–939, 2008.

[72] Dan Williams, Patrick Reynolds, Kevin Walsh, Emin Gün Sirer, and Fred B Schneider. Device driver safety through a reference validation mechanism. In *OSDI*, volume 8, pages 241–254, 2008.

[73] Tian Xu, Emily Jost, Laurel H. Messer, Paul F. Cook, Gregory P Forlenza, Sriram Sankaranarayanan, Casey Fiesler, and Stephen Voida. "obviously, nothing's gonna happen in five minutes": How adolescents and young adults infrastructure resources to learn type 1 diabetes management. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery.

[74] Meng Zhang, Kevin B. Flores, and Hien T. Tran. Deep learning and regression approaches to forecasting blood glucose levels for type 1 diabetes. *Biomedical Signal Processing and Control*, 69:102923, 2021.

[75] Xugui Zhou, Bulbul Ahmed, James H Aylor, Philip Asare, and Homa Alemzadeh. Data-driven design of context-aware monitors for hazard prediction in artificial pancreas systems. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 484–496. IEEE, 2021.

[76] Xugui Zhou, Maxfield Kouzel, Chloe Smith, and Homa Alemzadeh. Knowsafe: Combined knowledge and data driven hazard mitigation in artificial pancreas systems. *arXiv preprint arXiv:2311.07460*, 2023.

[77] T. Zhu, C. Uduku, K. Li, and et al. Enhancing self-management in type 1 diabetes with wearables and deep learning. *npj Digital Medicine*, 5:78, 2022.

[78] Taiyu Zhu, Kezhi Li, Pau Herrero, and Pantelis Georgiou. Basal glucose control in type 1 diabetes using deep reinforcement learning: An in silico validation. *IEEE Journal of Biomedical and Health Informatics*, 25(4):1223–1232, 2021.

[79] Howard Zisser, Marc Breton, Eyal Dassau, et al. Novel methodology to determine the accuracy of the omnipod insulin pump: A key component of the artificial pancreas system. 5(6):1509–1518, 2011.

A Ethical and safety considerations

One individual, who we will call Bob, used GlucOS for 9 months. Six university students also used the predictive alerts from GlucOS for 2 months. We put in place a protocol where we would stop our study and ask people to stop using our system and revert back to their previous management regime. In our protocol, we have formal in-person meetings with Bob every week to check his data and make sure that he is not incurring risk as a result of using GlucOS, and weekly surveys for the six students, plus monthly meetings. We would have stopped our study if any of the following conditions occurred:

- *Excessive hypoglycemia.* We define excessive hypoglycemia as having spent 8% or more of time during a week with CGM readings below 70 mg/dl or more than 1% below 54 mg/dl. These amounts of time spent in hypoglycemia would have represented a significant increase in time spent in hypoglycemia for participants.
- *Insufficient CGM data.* If they had less than 70% of the time during a week with CGM readings, then the lack of CGM data would represent a decrease for the participants.
- *Insufficient closed-loop runs.* We designed GlucOS to run closed-loop algorithms every five minutes, or 288 times per day. If Bob had a day where our closed-loop algorithm ran successfully less than 200 times, it would represent a fundamental flaw in the system.
- *Serious complications due to diabetes.* Serious complications include any hospitalization for diabetes related issues, severe hypoglycemia where a participant was unable to recover from hypoglycemia themselves and had to get help from someone else, diabetic ketoacidosis, or seizure.
- *New medical diagnosis requiring significant attention.* If anyone had received a new medical diagnosis during the study, the study would have had the potential to be a distraction for participants.
- *New mental health diagnosis.* If anyone were struggling with mental health during the study, the study would have the potential to be a distraction.

In addition to our safety protocol, we followed best practices for ethical research to ensure that we conducted our study ethically.

- *IRB.* Our study went through our university's IRB process, where the study including six university students who used GlucOS's alerting software was approved, and our case study on Bob was determined to be "self evaluation" and exempt from IRB.
- *Informed consent.* All participants went through a written informed consent process.
- *Data management.* GlucOS does *not* store any personally identifiable information and uses access control

on our Google Cloud Platform infrastructure to ensure that only the authors of the paper can access this anonymized data.

- *Risk/benefit analysis.* All participants are already living with T1D and use CGMs and inject insulin, so they already take on the risks inherent in managing T1D. The benefits include improved control over their T1D. There are risks in using experimental software, but all participants continued to rely on their CGM alerts as a last line of defense.
- *Participant selection.* We selected participants who were university students living with T1D.
- *Adverse event reporting.* Had any participants experienced adverse events during the study, we would have reported it to our IRB for guidance and addressed it according to our safety protocol.
- *Communication.* All surveys were done through our university's medical school infrastructure for private communication with individuals.

Since Bob used the full GlucOS software, including for insulin delivery, we included several other safety mechanisms:

- Bob increased the frequency of his visits to his Endocrinologist while using the GlucOS software. He went from yearly appointments to quarterly appointments.
- During the study, twice Bob got lab work based on blood tests to provide an independent measure of his overall metabolic health. This lab work helped to ensure that he was in good health and that there were no hidden negative impacts from his use of GlucOS.

Overall, GlucOS had a large positive impact on the individuals who used it. Bob's lab-based A1C test (a blood test that measures average glucose levels over time), which includes his time using GlucOS, was 5.8%. That is nearly non-diabetic levels of control – healthy individuals will have an A1C of 5.6% or lower. *All* of the students decreased the amount of time spent in hypoglycemia. These results are profound for the seven people using GlucOS, and it had a positive impact on their health and quality of life.

Finally, Bob was a willing and enthusiastic user of the GlucOS system. To be clear, Bob is *not* a graduate student who was forced to use GlucOS so that his advisor could publish a paper. Rather, Bob struggled with the cognitive load of T1D management and wanted an automated insulin delivery system to offload part of the burden. However, he was unwilling to use any of the other automated insulin delivery systems out there due to concerns over their security. Bob found that GlucOS, with its focus on security, correctness, and simplicity, met his needs.

B Derving the implementational invariant

In this section, we show how we derive the implementational invariant from the biological invariant. We introduce the following equations to capture the error bounds on CGMs and pumps under standard conditions:

$$0.9 \cdot G_t^A \leq G_t^M \leq 1.1 \cdot G_t^A \tag{3}$$

$$0.99 \cdot I_t^A \leq I_t^M \leq 1.01 \cdot I_t^A \tag{4}$$

Here, I_t^M and G_t^M denote the measured insulin dose and glucose concentration respectively at time t, while I_t^A and G_t^A represent their true values.

We now recall the biological invariant (Equation (1) in Section 5):

$$\Delta G^C - \Delta G^A \leq T, \quad \text{or} \quad \Delta G^C - T \leq \Delta G^A \tag{5}$$

$\Delta G^A = G_{t+\Delta t}^A - G_t^A$ is the actual drop in glucose over the same period. We note that we do not know the value of ΔG^A since we cannot capture the user's true glucose concentration.

Here, ΔG^C is the calculated drop in the user's glucose over period Δt and T is the clinical threshold of 30 mg/dl.

We adapt mathematical models from prior literature [7, 69] to compute ΔG^C:

$$\Delta G^C = S \cdot (B \cdot \Delta t - \Delta IOB^A + \Sigma I_i^A) \tag{6}$$

where S is the BioKernel's estimate of the user's insulin sensitivity and B is the user's basal rate (a physiological parameter representing the amount of insulin required per hour to absorb the background glucose produced by the body). Basal rate does not change as frequently as the insulin sensitivity and often remains stable for days [46]. ΔIOB^A captures the change in the amount of insulin in the user's body over the period Δt and ΣI_i^A is the sum of insulin doses delivered over the same period. ΔIOB^A at time t is calculated as the dot product of insulin doses across the 6-hour window ending at t with coefficients from standard exponential curves. In other words, both ΔIOB^A and ΣI_i^A depend on insulin doses which cannot be captured precisely.

To account for these unknowns, we combine (3), (4), (5) and (6) to get:

$$S \leq \frac{0.9 \cdot G_{t+\Delta t}^M - 1.1 \cdot G_t^M + T}{B \cdot \Delta t - 0.99 \cdot \Delta IOB^M + 0.01 \cdot \Sigma {I_i}^M} \tag{7}$$

Here, ΔIOB^M and ΣI_i^M are the measured equivalents of ΔIOB^A and ΣI_i^M, recorded from the pump. We note that Equation (7) is the implementational invariant(Equation (2) in Section 5), thereby showing how we derive it from the biological invariant.

C Evaluating algorithmic security when intentionally underdosing insulin

We repeat the experiment described in Section 9.1, but intentionally dose one-tenth the required amount of insulin. This experiment represents an algorithm that intentionally underdoses insulin as opposed to overdosing insulin. Figure 8 shows that the same virtual humans described in Section

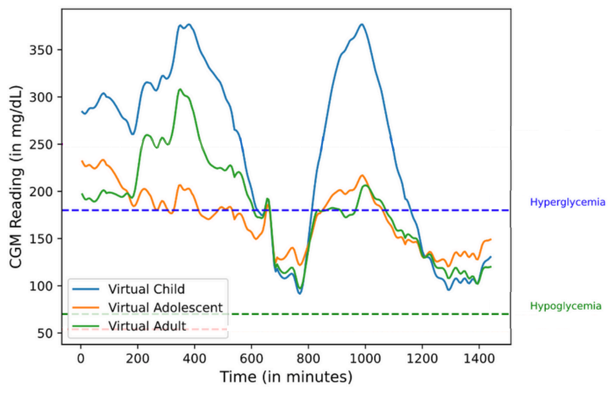


Figure 8. Variation in CGM readings for three virtual humans when running a predictive ML algorithm that intentionally doses one-tenth the amount of the required insulin. The CGM readings staying mostly above the hyperglycemic bound would have had drastic health consequences in all three individuals.

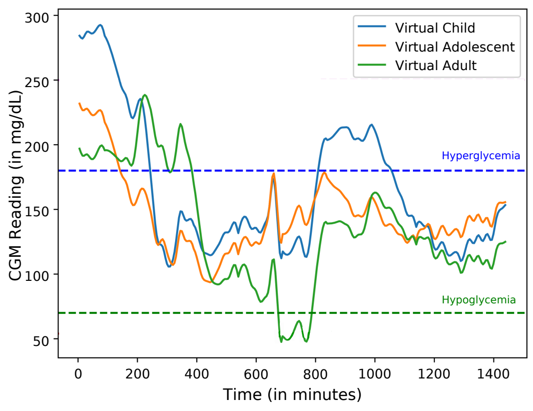


Figure 9. Variation in CGM readings for three virtual humans when clamping an algorithm that intentionally tries to dose one-tenth the amount of the required insulin. On average, CGM readings stay in range over 70% of the time.

9.1 spend most of their time above the hyperglycemic bound which would have led to severe health consequences in all three individuals.

Figure 9 shows the same virtual humans in the same simulation scenarios with the same malicious algorithm dosing one-tenth the required amount of insulin, but with GlucOS's algorithmic security mechanism enforced. We now see all virtual humans spending over 70% of their time in the recommended healthy range.

D Driver security: Withstanding malicious pump drivers

Pump drivers provide abstractions for insulin delivery applications to communicate with insulin pumps. Much like traditional device drivers, they have complex implementations since they help accomplish a wide range of tasks including pump setup/teardown, managing pump UX, handling insulin delivery/suspension, managing pump errors, accurately tracking insulin doses, and so on.

Given this complexity, there is a strong possibility for bugs or vulnerabilities to be present in existing pump drivers. These vulnerabilities can harm or even lead to the death of individuals since pump drivers control insulin pumps. For example, the user could be under risk of insulin overdose if the pump driver inadvertently or intentionally issues the same insulin dosing command twice.

Our goal is to protect users from inadvertent bugs in pump drivers as well as malicious pump drivers. We introduce a shim between the pump driver and the pump, which validates all communication between them. We design our mechanism to interpose on existing pump driver-pump communication instead of directly programming the pump. This designs ensures that our trusted computing base remains small and simple by not including complex pump driver implementations. This design also provides flexibility with altering or introducing new functionality to pump drivers.

Interestingly, to protect the individual, we had to overrule our principle of not programming the pump in cases pertaining to commands for canceling ongoing insulin delivery. We could not directly adopt the design of other device driver security mechanisms [49, 65, 72] for our driver security mechanism since these cases are specific to insulin pumps.

We focus on pump drivers because they are complex (14.5k LoC), multi-threaded, event-driven, and use two-way communication between the pump and the driver. In contrast, CGM drivers are simple (1.9k LoC) and only read sensor values from the device. In our future work, we will formally verify CGM drivers using the same techniques we use for the insulin delivery path (Section 7).

D.1 Driver security mechanism design

Figure 10 provides an overview of our driver security mechanism. The BioKernel registers high-level commands, like "deliver 2U of insulin" with our driver security mechanism, and sends them to the pump driver. The pump driver converts these commands into I/O messages to program the pump. The driver security mechanism then checks to make sure that these I/O messages are consistent with the previously registered high-level pump commands before forwarding them to the pump hardware. We only validate commands for insulin delivery since they pose the biggest threats to the user. On detecting added, modified or dropped commands from the pump driver, the driver security mechanism rejects

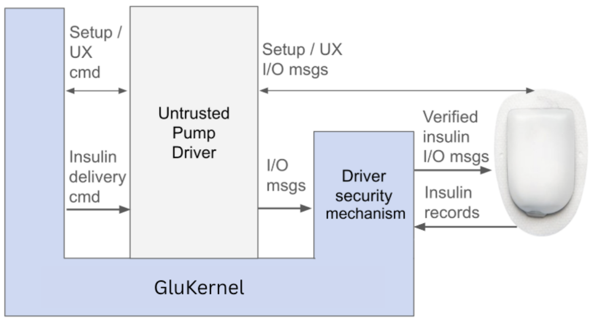


Figure 10. We interpose on communications between the pump driver and the pump to handle untrusted pump drivers.

Figure 11. We attached an Omnipod DASH insulin pump to a stuffed animal and modified the OmniB LE pump driver to intentionally add, drop and modify insulin commands. Our driver security mec hanism detected all manipulations.

further commands and transitions the system to manual insulin delivery where the user takes control. We list insulin message types pertaining to insulin delivery, pump setup and UX in Figure 12.

Dropped commands pertaining to canceling ongoing insulin delivery require separate consideration since excess insulin is dangerous and cannot be withdrawn once it has been delivered. Users will cancel insulin delivery if they accidentally issued an insulin delivery command or incorrectly set the insulin dosage. In cases where a malicious driver drops a cancel command, the driver security mechanism directly issues I/O messages to the pump to cancel insulin delivery before transitioning the pump to a manual mode. We make the tradeoff of increasing the size of our trusted computing base to handle these cases in the interest of protecting the user. Fortunately, there are only two commands to cancel insulin delivery in our current implementation, so the increase to our trusted computing base is small.

Message name	Message type	Message description
getStatus	Insulin delivery	Request insulin delivery status from the pump
statusResponse	Insulin delivery	Insulin dose delivery information from the pump
errorResponse	Insulin delivery	Insulin dose delivery error from the pump
setBasalSchedule	Insulin delivery	Set default rate of insulin delivery
setTempBasal	Insulin delivery	Set temporary rate of insulin delivery
bolus	Insulin delivery	Deliver instant insulin dose
cancelDelivery	Insulin delivery	Cancel insulin delivery
setupPod	Setup	Setup a new pump
assignAddress	Setup	Pair pump with phone
deactivatePod	Setup	Teardown pump
acknowledgeAlert	UX	Acknowledge alert from pump
configureAlerts	UX	Configure alerts from pump
beepConfig	UX	Configure pump beeps

Figure 12. Types of messages exchanged between a pump driver and pump pertaining to delivery, setup and UX.

D.2 Does GlucOS ensure pump operation while overcoming malicious pump drivers?

We evaluate the driver security mechanism in allowing communications from a benign pump driver to go through while blocking manipulated communications from a malicious pump driver.

Malicious pump drivers We modified the OmniBLE pump driver [4] to intentionally add, drop and modify insulin commands sent to an Omnipod DASH insulin pump [3] attached to a stuffed animal (Figure 11). We report that the driver security mechanism detected all manipulated commands, resulting in 0 false negatives.

Benign pump drivers For one week, we logged messages passed between the BioKernel, the unmodified OmniBLE pump driver and the pump on an individual running GlucOS (Section 10). We report 100% consistency between all commands showing that GlucOS's driver security mechanism does not incur any false positives.

These results show that GlucOS can ensure pump operation while withstanding attacks from malicious drivers.

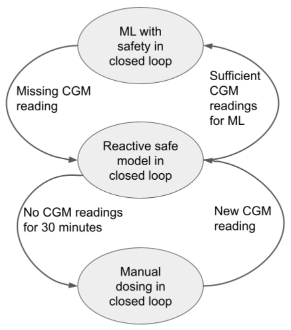

ML with safety in closed loop
Missing CGM reading
Sufficient CGM readings for ML
Reactive safe model in closed loop
No CGM readings for 30 minutes
New CGM reading
Manual dosing in closed loop

Figure 13. GlucOS's CGM state machine to handle missing readings from CGMs.

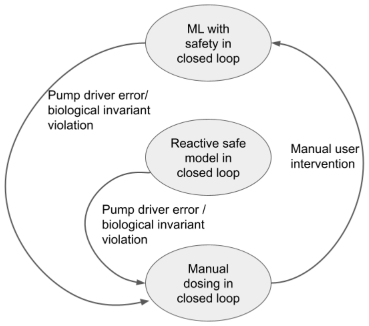

ML with safety in closed loop
Pump driver error/ biological invariant violation
Reactive safe model in closed loop
Manual user intervention
Pump driver error / biological invariant violation
Manual dosing in closed loop

Figure 14. GlucOS's state machine to transition users to the safest manual dosing state when losing trust in insulin delivery (i.e., with a buggy or malicious pump driver) or when the biological invariant is not satisfied.

E Runtime failure management

In this section, we describe GlucOS's *state machine* that combines its security components, i.e., algorithmic security (Section 4), driver security (Section D) and end-to-end security (Section 5) together to protect users in light of their corresponding failures.

GlucOS's state machine consists of three operating states: (1) ML with safety operating in closed loop, (2) reactive safe model operating in closed loop and (3) manual dosing in closed loop. ML provides the tightest level of control in managing glucose but poses the largest immediate threat from excess insulin since it preemptively injects insulin by anticipating rise in glucose. The reactive safe model provides slightly lesser control, but poses milder immediate threat since it only injects insulin to account for measured excess glucose. Manual dosing offers the least amount of control but poses no threat since it only injects the minimum insulin needed for sustenance. GlucOS predominantly operates in the state running ML (with safety) in closed loop.

We transition to the safest manual dosing state when we do not trust insulin delivery (i.e., with a buggy or malicious pump driver). We also transition to the safest manual operation when the biological invariant is not satisfied to ensure that we do not risk the possibility of removing too much glucose from our insulin doses. If we end up in the manual dosing state, we enforce that only manual user intervention can transition back to other states. We visualize these state transitions in Figure 14.

Inability to retrieve readings from a CGM also count as runtime failures. However, these failures are easier to handle.

Since ML needs a stream of CGM readings as input, we transition from ML to the reactive safe model if we don't receive an input from the CGM. As long as the number of missing CGM readings is low (say, within 30 minutes), we can interpolate to account for the missing readings to operate the reactive safe model. Once we have too many missing CGM readings that make us lose confidence in interpolation, we transition to manual dosing. From manual dosing, we transition back to the reactive safe model on receiving a CGM reading. Lastly, from the reactive safe model, we transition back to ML once we have enough CGM readings that are required by the ML model. Figure 13 summarizes these transitions.

F Formal verification: invariants and bugs

In this section, we list the invariants used to formally verify critical functions of GlucOS. Figure 15 lists the bugs we found and fixed during formal verification.

F.1 Invariants

The MicroBolusing function ensures that micro boluses are always administered at safe time intervals while making sure that they are actually needed (the glucose level of the person is in fact high).

- Ensure no micro bolus in the last 4.2 minutes.
- Ensure glucose is at least 20 mg/dl above the target glucose level.
- Ensure predicted glucose is greater than the current glucose level minus 2.
- Ensure micro bolus amount is within the allowed range.
- Ensure bolus amount is rounded correctly if *pumpManager* is available.

We assume that the BioKernel is the only software issuing bolus commands.

The GuardRails function is responsible for ensuring a safe basal rate value, mindful of the user's glucose levels. It enforces several invariants:

- Ensure the result of rounding *newBasalRateRaw* to the supported basal rate is within the allowable range.
- Ensure the new basal rate does not exceed the maximum basal rate in the settings.

Bug Type	Description
Division by Zero Error	Potential division by zero error when calculating insulin dosage if the user entered the same start and end time for a basal rate, resulting in a duration of zero.
Negative Value Handling	The insulin dosing calculation for microbolusing and basal rate erroneously allowed negative values.
Out-of-Range Value Handling	The microbolusing function failed to handle cases where one of its calculation components was out of the expected range or resulted in a negative value.
Violation of Intended Postcondition	The function responsible for setting the insulin delivery duration violated its intended postcondition by allowing the end date to be earlier than the start date.
Violation of Intended Postcondition	The function responsible for calculating the total insulin dose violated its intended postcondition by returning an incorrect upper bound value under certain conditions.
Violation of Intended Postcondition	The function responsible for updating the user's insulin dosage violated its intended postcondition by failing to validate the input parameters correctly.
Violation of Intended Postcondition	The function responsible for calculation of ‘unitsDelivered‘ needed additional postconditions accurately reflect the amount of insulin delivered over the time gap.
Lack of Error Handling	Additional error handling was added to insulin delivered over a time segment function to gracefully catch undesirable state.
Lack of Error Handling	Additional error handling was added to microbolusing function to gracefully catch undesirable states.

Figure 15. Report of bugs fixed by formally verifying GlucOS.

- Ensure the new basal rate is not negative.
- Ensure that if either the current glucose level or the predicted glucose level falls below or equals the shut-off glucose threshold, the new basal rate is set to 0.0.

The function operates under the assumption that all paths must go through the guard rails, with a specific evaluation comparing with the pump invariant. Additionally, it assumes the integrity of settings, trusting the underlying OS and file system and storage stack.

The insulinDeliveredForSegment function ensures that *intersectionStart* is not greater than *intersectionEnd* when computing the intersection between *self.startDate* and *self.endDate* with *startDate* and *endDate*.

- Verifies that *intersectionDuration* is non-negative and does not exceed *doseDuration*, ensuring it accurately reflects the overlapping time duration.
- Checks that the units of insulin (*units*) used for calculation are either *deliveredUnits* or *programmedUnits*, ensuring consistency in how the insulin amount is derived.
- Verifies that the computation *(units * intersectionDuration / doseDuration)* accurately represents the amount of insulin delivered during the intersection period.

The function assumes that the input parameters (*self.startDate*, *self.endDate*, *startDate*, *endDate*, *units*, and *doseDuration*) are valid and within the expected ranges. It is also assumed that the underlying data structures and calculations are accurate and consistent throughout the system.

The CreateBasalDose function requires a time gap greater than 1 second to proceed and return a valid *DoseEntry*.

- Ensures that the calculation of *basalRatePerSecond* from *basalRate* accurately represents the rate of insulin delivery per second.
- Ensures that the calculation of *unitsDelivered* accurately reflects the amount of insulin delivered over the time gap.
- The *DoseEntry* constructed should have consistent attributes and adhere to the specified type, units (*unitsPerHour*), and mutability (*false*).

The function assumes that the input parameters (*basalRate*, *startDate*, and *endDate*) are valid and within the expected ranges. It also assumes that the underlying time calculations and conversions between units are accurate and consistent.

The inferBasalDoses function ensures that *basalDoses* contains only *DoseEntry* instances where *type* is *.tempBasal*, *.resume*, or *.suspend*, sorted in ascending order by *startDate*.

- InsulinType should be determined correctly based on the last *.tempBasal* or *.bolus* type dose in *doses*, defaulting to *.humalog* if none are found.
- Each inferred basal dose added to *inferredBasalDoses* must be created using the *createBasalDose* function with valid parameters and added in the correct chronological order.
- If the last dose in *basalDoses* is not of type *.suspend*, a valid basal dose must be inferred from its *endDate* to *at*.

The function assumes that the input parameters (*doses*, *basalDoses*, *at*, and *pumpRecordsBasalProfileStartEvents*) are valid and within the expected ranges. It also assumes the correctness of the underlying data structures and the *createBasalDose* function.

The insulinOnBoard function ensures that *doses* contains unique and valid *DoseEntry* instances filtered up to *at*.

- The total *iob* should accurately represent the cumulative insulin on board from all valid dose entries in *doses*.
- *basalDoses* should contain inferred basal doses that accurately reflect insulin on board contributions when *pumpRecordsBasalProfileStartEvents* is false.
- The final return value of *insulinOnBoard* should be the sum of *iob* and *basalIob*, representing the total insulin on board at *at*.

The function assumes that the input parameters (*doses*, *at*, and *pumpRecordsBasalProfileStartEvents*) are valid and within the expected ranges. It also assumes the correctness of the underlying data structures and the *inferBasalDoses* function.

The ActiveBolus function ensures that data is read consistently from disk before proceeding with any operations.

- Verifies that *DeduplicatedDoses(events: eventLog, at: at)* correctly filters out duplicates and retains only relevant dose entries up to *at*.
- Ensures that doses.filter accurately selects bolus entries that were active at the specified *at*.
- Verifies that *doses.last* correctly returns the last active bolus entry, ensuring it is non-null if a bolus exists within the specified criteria.

The function assumes that the input parameters (*eventLog* and *at*) are valid and within the expected ranges. It also assumes the correctness of the underlying data structures and the *DeduplicatedDoses* function, as well as the consistency and reliability of disk read operations.

The TempBasal (Safety Clamps) function ensures that the events array includes only events from safetyStates that fall within the time range *[start - duration, at)*.

- Verifies that the historicalMlInsulin value represents the total units of insulin delivered by the machine learning system over the specified time horizon.
- Ensures that the machine learning and safety temp basal rates are correctly converted to units of insulin based on the given duration.
- Ensures that the upper and lower bounds for insulin units are correctly adjusted based on historicalMlInsulin.
- Ensures that the difference between machine learning and safety temp basal units is correctly clamped within the calculated bounds.
- The clamped delta units should be converted back to a temp basal rate and added to the safety temp basal rate.
- The returned *SafetyTempBasal* object should correctly encapsulate the adjusted temp basal rate and historical machine learning insulin.

The function assumes that the input parameters (*safetyStates*, *start*, *duration*, *at*, *mlTempBasal*, *safetyTempBasal*, and *lastHistoricalMlInsulin*) are valid and within the expected ranges. It also assumes the correctness of the underlying data structures and the conversion functions between units, as well as the proper configuration and appropriateness of the safety clamp parameters and thresholds for the user.

G Scenarios when the user's insulin sensitivity is lower than the BioKernel's estimate

We do not take any action when the actual change in the user's glucose is higher than the calculated change. Higher actual glucose does not necessarily reflect a shift in the user's physiological state (decreased insulin sensitivity) since it can also result from digestion when the user consumes food. Further, even if the user is less sensitive to insulin, their glucose absorption would be slower, which does not pose them with an immediate threat (hyperglycemia).