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

Hari Venugopalan
hvenugopalan@ucdavis.edu
UC Davis

Shreyas Madhav Ambattur Vijayanand
smvijay@ucdavis.edu
UC Davis

Caleb Stanford
cdstanford@ucdavis.edu
UC Davis

Stephanie Crossen
scrossen@ucdavis.edu
UC Davis

Samuel T. King
kingst@ucdavis.edu
UC Davis

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 combines algorithmic security, driver security, and end-to-end verification to protect against malicious ML models, vulnerable pump drivers, and drastic changes in 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. 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.

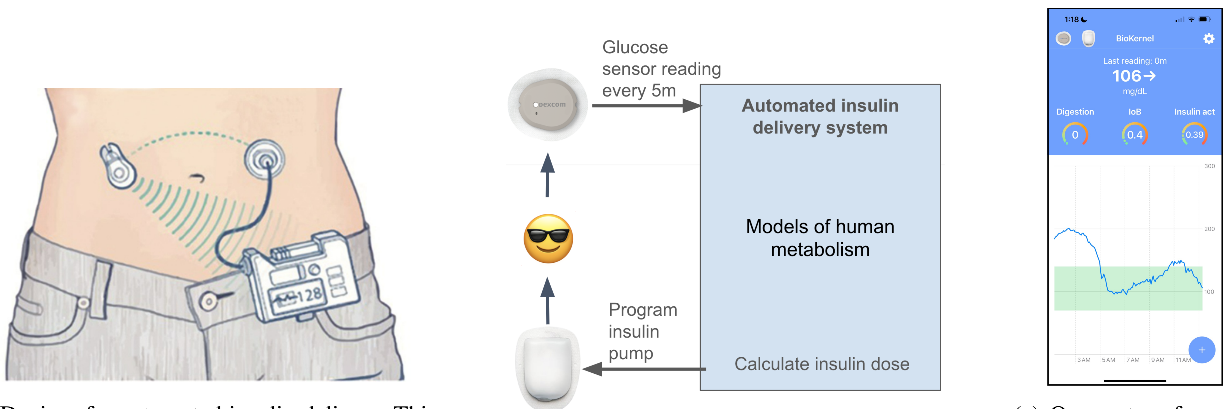
1 Introduction

Type 1 Diabetes (T1D) is a metabolic disorder where an individual's pancreas stops producing insulin. To compensate, they inject synthetic insulin. Computer systems, called automated insulin delivery systems, exist that use subcutaneous sensors to continuously monitor glucose concentrations (glucose is the body's primary energy source) and inject insulin automatically using an insulin pump device (Figure 1). However, insulin is a dangerous hormone where 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 inadvertant errors. Several commercial [3, 5, 17] and open-source [42, 54] automated insulin delivery systems exist today, but none of them consider security as a primary design constraint.

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 [71, 72]. We argue that this approach falls short of addressing the full spectrum of challenges inherent in building trustworthy systems that operate within real-world constraints. Real-world systems must cope with runtime platform limitations, changes in human physiology, and the necessity of human interaction with the system. Plus, algorithms and simulations model attacks abstractly through their glucose response, missing out on details like which components of the system are likely to be compromised or how the system architecture can prevent damage from these potentially vulnerable components. Our experience shows that when faced with these practical challenges, 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. To address these challenges, we need practical systems that move beyond simulation-based designs and tackle the challenges of real-world systems.

Current open-source automated insulin delivery systems are complex. For example, the OpenAPS [54] core function for calculating insulin doses consists of 1192 lines of Javascript code, 63 input and configuration parameters, and 90 branch statements. All of this logic influences the ultimate dosing decision, making reasoning about its correctness, both manually and automatically, challenging. Automated insulin delivery systems that use the OpenAPS algorithm embed it into native apps using a WebView for Javascript interpretation, adding to the overall complexity and increasing the attack surface. In contrast, our implementation of this same function is 25 lines of native and formally verified Swift code. None of the open-source automated insulin delivery systems use formal methods, can withstand attacks on components, or have mechanisms or policies for security.

In this paper, we take on the challenge of building the first trustworthy automated insulin delivery system, called Glu-



(a) Devices for automated insulin delivery. This figure is from breakthrough1d.org.

(b) Closed-loop architecture.

(c) Our system for automated insulin delivery.

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

cOS. 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 ML-driven insulin delivery, (3) the ability to withstand attacks on insulin pump driver software, and (4) the application of formal methods to prove correct our insulin delivery path, the most sensitive part of the system. Despite these efforts, we identify scenarios in automated insulin delivery 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. 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.

Automated insulin delivery systems are different from traditional applications that the security community focuses on, but they are important. Research on operating systems or web browsers can list scores of CVEs from current systems and show how their system improves security by eliminating the impact of some or all of these vulnerabilities by virtue of their design. Automated insulin delivery systems do not have a long list of CVEs that we can reference, although there are a few [46–49]. However, our position is that we need to make sure that these systems *never* get CVEs, because if they do, security vulnerabilities can kill people. Thus, we focus on building a new system, using security first principles and formal methods, 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 ML, vulnerable pump drivers, and drastic changes in the human's glycemic response.
- The application of formal methods to prove correct the most critical parts of GlucOS.
- A biological invariant that combines end-to-end physiological security checks with formal methods.
- A case study describing our experiences from a real-world deployment with seven individuals, 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 overall 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 9.1 summarizes our safety protocol and Appendix A provides a detailed description.

There are 8.4 million people living with T1D worldwide [30], underscoring the need for providing trustworthy systems for automated insulin delivery. Our work, carried out with rigorous medical and ethical oversight, is a first step in addressing this need.

2 Overview

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 be able 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.

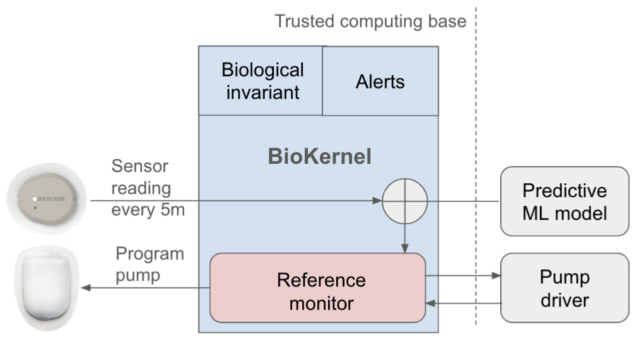


Figure 2: Overall architecture for GlucOS.

2.1 Locking down the insulin delivery path

We focus our system architecture on providing security mechanisms and policies on the insulin delivery path. The insulin delivery path includes models for calculating the amount of insulin still active in the individual (insulin can take up to six hours to fully absorb), closed-loop algorithms for calculating insulin dosing, deep neural network ML to predict future metabolic states, and drivers for programming the insulin pump. 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, 33] 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 [31, 59, 63, 67], which also apply OS separation principles at the application layer. By decomposing the overall automated insulin delivery system, we strive to keep our trusted computing base simple.

Figure 2 shows our overall system architecture. At the core of our overall architecture is the BioKernel. The BioKernel is the component that interacts with the 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 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 ML model. The predictive ML 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 ML model's pump commands and the pump driver's low-level I/O. This reference monitor architecture ensures that we can withstand attacks from malicious predictive ML as well as malicious pump drivers without having to know the internals of how they operate.

For predictive ML, our reference monitor enforces algorithmic security, where we bound the amount of insulin calculated by ML to stay within dynamic safety bounds computed by a simple *reactive safe model*. For the pump driver, we interpose on the I/O path to ensure that only the pump commands issued by the BioKernel are programmed into the pump. These checks constitute our *pump invariants*, which, when violated switch the pump to operate in a safe mode.

Our use of formal methods focuses on the software related to delivering insulin, which is the most risky operation in any automated insulin delivery system. 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, where violations of this invariant suggest a potentially catastrophic failure of the system.

2.3 Threat model

The most serious 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 that results from an attack on the system. Quantitatively, our goal is to maintain the American Diabetes Association's recommended time in hypoglycemia of less than 4% [12].

As our design philosophy centers around humans, providing flexibility for people to use whatever ML they want by loading personalized models or using any pump hardware they want is important. This flexibility helps guide our focus on providing novel security mechanisms and policies for these components. Our goal is to make sure that they stay safe even if the model they pick is malicious or the pump driver they choose has security vulnerabilities. Attackers may target automated insulin delivery systems because they can directly harm the human using the system.

We assume that the Swift type system is correct and that the underlying operating system, the CGM driver and hardware, and the pump hardware we use are free of attacks and their device communications maintain integrity. Thus, we do not consider attacks emerging from tampered CGM sensors or pump hardware in our threat model. These are important but complementary problems to our work, which focuses on the system architecture and algorithms for automated insulin delivery.

We consider attacks from malicious predictive ML algorithms. These attacks can come from trojan attack logic embedded directly in a model, or from an adversarial attack on predictive ML. For adversarial attacks, although we assume that the CGM and insulin pump remain attack free, an adversary can manipulate these entities by serving an individual who is using our system with calculated amounts of carbohydrates in their food (e.g., at a restaurant), thus influencing the resulting CGM readings and corresponding insulin commands. In our threat model, malicious predictive ML can issue arbitrary insulin doses.

We also consider malicious pump drivers. In our threat model, malicious pump drivers can issue arbitrary pump I/O operations to add, drop, or modify commands sent to the pump.

Successful attacks on predictive ML or pump drivers, if unchecked, can be devastating because they directly affect insulin delivery and can be used by an attacker to induce potentially life-threatening hypoglycemia.

3 Algorithmic security: Withstanding malicious ML

Currently, no automated insulin delivery systems use deep neural network ML. Some current systems use statistical methods, like linear regression or model predictive control [21, 43, 54, 58, 66]. 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. However, all of the recent advances in ML, like image classifiers and large language models, have come from deep neural networks, which researchers have shown also work well for predicting metabolic states for use in automated insulin delivery systems [35, 64, 65, 70, 73, 74]. Despite this promise, none of the current automated insulin delivery systems incorporate more advanced deep neural networks due to the black-box nature of deep neural networks and the potentially dire consequences of mispredictions – an automated insulin delivery system's unchecked "hallucination" [1] could have lethal consequences if it delivers an inappropriate insulin dose [13, 22].

In this section, we discuss the design and implementation of GlucOS's algorithmic security mechanisms to enable predictive models, including deep neural network ML, to run safely in automated insulin delivery systems.

3.1 Mechanism fundamentals

Our key novel insight underpinning our security mechanism design is that all correct insulin delivery algorithms will deliver the same amount of insulin over a sufficiently long period. People inject insulin to facilitate glucose absorption after eating. Digestion converts food to glucose, and insulin facilitates the absorption of glucose to tissue (e.g., the brain) where the body can use it as an energy source. Since insulin dosing is driven by food consumption, all correct algorithms should dose the same amount of insulin to cover glucose from digestion.

However, the timing of this insulin delivery matters for maintaining balanced glucose levels (Section 8). Insulin activation is a slow process, with injected insulin taking up to six hours to fully activate. To meet this challenge, 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 allows us to balance proactive insulin dosing for faster responses with a more conservative approach for safety.

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 guarentees.

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 safe model and the performance from a predictive model.

Our security mechanism draws inspiration from previous work on combining simple and complex models for safety [51, 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 our principles for how to apply our mechanism to automated insulin delivery systems (Section 3.2).

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 collects a recent history of CGM readings and insulin dosing

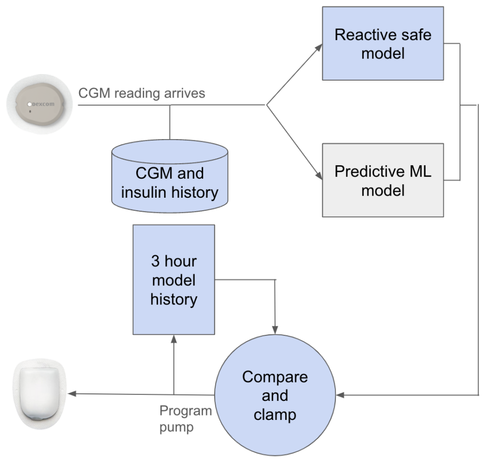


Figure 3: Algorithmic security mechanism for GlucOS.

and sends these as inputs to both the reactive safe model and the predictive ML model. These models independently produce pump commands, called a temporary basal command. The temporary basal command sets the insulin delivery rate for the insulin pump for a 30-minute duration. The BioKernel will collect both temporary basal commands and look back at the differences between the reactive safe model and predictive ML model for the past three hours to see how much insulin delivery it should attribute to the predictive ML model. If the predictive ML model is still within its safe insulin delivery limits, it uses the predictive ML model's temporary basal command to program the pump. If it has exhausted its limits, we clamp it to fit within the insulin bounds we set 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. People living with T1D need to inject insulin to facilitate the absorption of this background glucose.

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 help use the extra insulin over the next three hours.

In the worst case, an incorrect prediction might lead to low glucose. However, with our mechanism in place, we limit the impact by our three-hour bound and offset by the body's continuous glucose production.

3.2 Model principles

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 [25] 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 [52]. Although we use this model in our current implementation, our system supports any predictive model. PID controllers are used in automated insulin delivery systems [38, 68] and research predictive ML models for insulin delivery have been well covered in the literature [35, 64, 65, 70, 73, 74].

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 insulin they have injected but have not yet activated. 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 a standard formulation used in most automated insulin delivery systems.

In contrast, predictive models have no restrictions since their outputs are constrained by the reactive safe model and our security mechanism. The ideal predictive model anticipates future metabolic states and adjusts insulin proactively to avoid glycemic imbalance.

In our evaluation, our predictive model outperforms the reactive safe model both in simulation (Section 8) and for the individual that uses our system for their real-world insulin dosing (Section 9), showing why it is important for automated insulin delivery systems to have a practical way to adopt ML.

4 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. Different pump drivers use different implementations with varying amounts of customizations

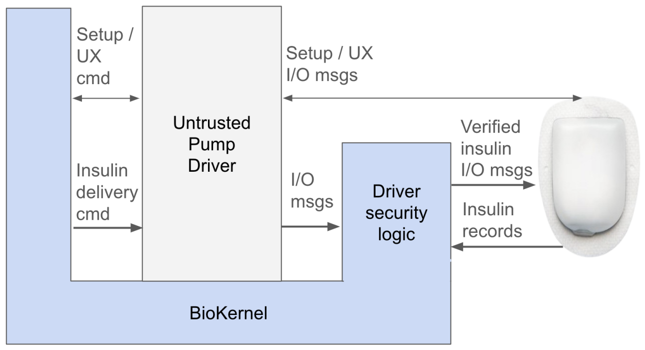


Figure 4: We interpose on communications between the pump driver and the pump to handle untrusted pump drivers to deliver insulin.

Message name	Message description
getStatus	Request insulin delivery status from the pump
statusResponse	Insulin dose delivery information from the pump
errorResponse	Insulin dose delivery error from the pump
setBasalSchedule	Set default rate of insulin delivery
setTempBasal	Set temporary rate of insulin delivery
bolus	Deliver instant insulin dose
cancelDelivery	Cancel insulin delivery

Figure 5: Types of insulin related messages exchanged between between a pump driver and an insulin pump.

and features to carry out these tasks.

While CVEs do not currently exist for pump drivers, there is strong possibility for bugs or vulnerabilities in their implementation given their complexity and the fact that they not been formally verified. Since drivers control insulin pumps, these vulnerabilities can harm people using them, even lead to the death. For example, consider a scenario where the user decides to inject 3 units (3U) of insulin using the BioKernel. The BioKernel would relay this as a command to the pump through abstractions provided by the pump driver. If the pump driver inadvertently or intentionally issues this command twice, 6U of insulin would be delivered to the user, which would put the user in danger by dosing too much insulin. We need a mechanism to verify that the pump driver is executing correctly.

Our security goals for GlucOS's driver security are to protect users from inadvertent bugs in pump drivers and safeguard against potentially malicious pump drivers. To achieve these goals, we (1) introduce driver security logic in the BioKernel that interposes on pump driver-pump communications, (2) validate all commands against known high-level pump commands, and (3) detect and handle anomalies by stopping any added/modified commands and proactively handling dropping critical commands.

This approach allows people to choose between various pump drivers while maintaining security, balancing user flexibility and safety with automated insulin delivery. It also keeps our trusted computing base small, simplifying formal verification.

4.1 Driver security logic design

Figure 4 provides an overview of our driver security logic. The BioKernel registers high-level commands, like "deliver 2U of insulin" with our driver security logic, and sends them to the pump driver. The pump driver converts these commands into I/O messages to program the pump. However, these I/O messages come back to the driver security logic, which checks to make sure that they are consistent with the previously registered high-level pump command, before forwarding them to the pump hardware.

We only validate commands for insulin delivery since they pose the biggest threats to the user. We do not validate communication pertaining to pump setup / UX since users can directly detect the respective manipulations. Figure 5 lists the different types of messages pertaining to insulin delivery exchanged between a pump driver and the OmniBLE insulin pump, which we use in our current implementation. We list other message types in Appendix B.

By interposing on the pump I/O message path, the driver security logic can ensure that the driver has not added, modified, or dropped any pump commands. Detecting added or modified commands is relatively straightforward. Our driver security logic in the BioKernel knows all high-level pump commands that it sends to the driver. Thus, we have a simple logic to confirm the corresponding I/O messages from the driver. The driver security logic stops any anomalous I/O messages before sending them to the pump.

Detecting dropped commands is more nuanced when the command that the driver drops is a command to cancel an ongoing insulin delivery. Users can cancel insulin delivery if they accidentally triggered an insulin delivery command or incorrectly set its parameters. Since excess insulin delivery is dangerous, the driver security logic automatically sends these cancel commands to the pump when it detects a dropped insulin cancellation command. There are two commands that deal with canceling insulin delivery: cancelDelivery and setting the suspend option on setTempBasal. The cancelDelivery command cancels a previous insulin delivery command (if the delivery is still ongoing) and the suspend option on the setTempBasal command suspends all insulin delivery until the user resumes delivery.

The most interesting lesson learned from our experience is in coping with dropped pump commands. In our original design, our goal was to ensure that our driver security logic only understood I/O commands and prevented any malicious I/O commands from going through. However, our driver security logic ended up needing to issue its own I/O commands to deal with dropped insulin delivery cancellation commands. This more proactive approach to driver security is different than previous approaches that also use specifications for runtime driver verification [44, 63, 69], which simply infer I/O semantics and do not issue their own I/O commands. In general, we avoid programming the pump directly from the driver security logic. But in this case, we decided to program the pump directly from the driver security logic for human safety.

When we detect a malicious driver, we transition the system into a safe manual mode where the individual needs to take over control of their own insulin delivery. They can still use the BioKernel to manage their insulin, but they would be unable to use the automated closed-loop algorithms. In our evaluation, our system had no false positives and detected all attacks (Section 8.2). See Appendix C for more details on how we maintain safety even when detecting an attack.

4.2 Driver security logic implementation

We focus our implementation on Loop's OmniBLE [4] pump driver. OmniBLE allows iOS applications to communicate with Omnipod pumps over Bluetooth. We include OmniBLE's cryptography and Bluetooth communication modules as part of the BioKernel and incorporate the reference monitor just before encrypting messages sent to the pump/ decrypting messages received from the pump.

OmniBLE does not incorporate automatic retry in its implementation. However, OmniBLE occasionally sends additional commands as part of its operation. For example, when sending a setTempBasal command, if OmniBLE's internal states indicate that delivery is still in progress as part of a previous setTempBasal command, it first issues a cancelDelivery command to cancel that delivery. To remove ambiguity in terms of whether the cancelDelivery command was issued as part of setTempBasal, we enforce OmniBLE to always issue the cancelDelivery command.

5 End-to-end security: Biological invariant

A key novelty in our overall approach is our introduction of an *end-to-end* check that ensures the BioKernel's glucose calculations against observations of the user's metabolic state. This raises an interesting challenge for formal verification as it requires modeling the true *biological state* of the user – a parameter that is both unknown at runtime and distinct from measured parameters which are part of the software implementation. We refer to the true safety invariant we enforce on the user's state as the *biological invariant*, and proving it is both a key part of our implementation and one of the key conclusions of our formal verification efforts.

If there is a large mismatch between our calculations and what we observe, then it means that our calculations are wrong. These deviations can happen from drastic changes in the user's glucose absorption, like 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 [62]. Regardless of the cause, the end-to-end check detects these conditions and prevents harm to the user. For the purposes of the following example, suppose the end-to-end check is that the measured glucose levels are always within 30mg/dl of the predicted level; we refer to this as the *implementation invariant* that is enforced in an end-to-end way across the system:

$$|g_{\mathsf{measured}} - g_{\mathsf{predicted}}| < 30\mathsf{mg/dl} \quad (1)$$

Naively, one might look at the $30\mathsf{mg/dl}$ on the right-hand side and think that the user's true glucose levels are within 30mg of the actual glucose levels. But this would be wrong! The problem is that the measured glucose level is subject to measurement error; the state-of-the-art Dexcom G7 CGM has a mean absolute relative difference of 8.2% from the true glucose level in the body [29]. For the sake of example, suppose that the measurement error on the CGM reading is around 5%; we refer to this as the CGM invariant:

$$|g_{\mathsf{measured}} - g_{\mathsf{actual}}| < 5\mathsf{mg/dl} \quad (2)$$

and from this we can derive the *true* conclusion on the user's glucose levels, which is what we refer to as the biological invariant. We stress that this invariant involves the parameter g_{actual} that is *not available in software*; it instead ensures a property (which we are able to formally prove) about the real level of glucose in the biological system. That statement we prove in the software is actually the following, *not* (1):

$$|g_{\mathsf{measured}} - g_{\mathsf{predicted}}| < 35\mathsf{mg/dl} \quad (3).$$

When we run BioKernel on a real human, we don't want to assume safety with respect to the measured glucose levels (invariant (1)); we actually want to ensure safety with respect to the real glucose levels in the human that is using the system (invariant (3)). Failing to distinguish these cases could be a severe software bug, and would jeopardize the end-to-end safety that we aim to provide by failing to account for CGM error. In the rest of the section, we describe the invariants that we enforce in the implementation (including invariant (1) as a simple case) and what additional invariants we prove in our formal verification development (including invariant (3)) below.

Implementation invariant. In the implementation, we check that changes in glucose levels that we observe from

CGM readings and the individual's insulin delivery history are consistent with our theoretical calculations of how much the person's glucose *should* change given these same inputs. If their glucose is dropping by more than our theoretical calculations show, then they are more sensitive to insulin then our system believes they should be. We set a threshold of 30 mg/dl per hour, for the maximum allowable divergence based on clinical evaluations showing detectability of significant glycemic deviations above this level [34, 37].

We take a three-tiered approach to dealing with deviations in our glucose change calculations. First, for small differences of less than 30 mg/dl, we accumulate errors and adjust the target glucose level to compensate, clamping the adjustment to 10 mg/dl. For example, if the individual's insulin sensitivity is elevated and their target glucose level is 90 mg/dl, GlucOS will update their target to 100 mg/dl dynamically to adjust. Second, when the differences are 30 mg/dl or greater, we shut off insulin delivery. Our rationale is that the individual is more sensitive to insulin than our models believe they should be, thus we want to avoid overdosing. If their difference goes back to within the expected range, normal execution resumes. Third, if they remain at 30 mg/dl or more for more than two hours, we alert the individual and transition the system into a manual mode that requires the user to dose insulin manually.

Verification. In the verification development (described in Section 7), we prove that the biological invariant holds, given error bounds set by the CGM manufacturer.

First, in the ported Dafny code, we locate the portions which enforce implementation invariants described above, including invariant (1). The code includes the two parameters g_{measured} and g_{actual}, as these are available to the software implementation. Second, we model actual glucose by putting error bounds on our CGM readings, in the CGM invariant (2). This CGM invariant is inserted as a precondition on all functions that operate on CGM values. Third, we track the individual's actual glucose level using a *ghost variable* in Dafny. Ghost variables are variables that are only available to the prover, these do not exist in the real software system. Fourth and lastly, we express as a postcondition the biological invariant (4), which asserts that the absolute difference between the theoretical model glucose and the ground-truth actual glucose cannot exceed a threshold of 35 mg/dl per hour. When the code is compiled, the Dafny verifier proves that the biological invariant holds on all inputs to the verified function.

Evaluation results. Our evaluation shows that the biological invariant is effective at keeping humans safe when they face drastic changes in insulin sensitivity (Section 8.3), and that our thresholds of 30 mg/dl per hour and two hours were effective for the human that used GlucOS to manage their T1D without any false positives (Section 9.3).

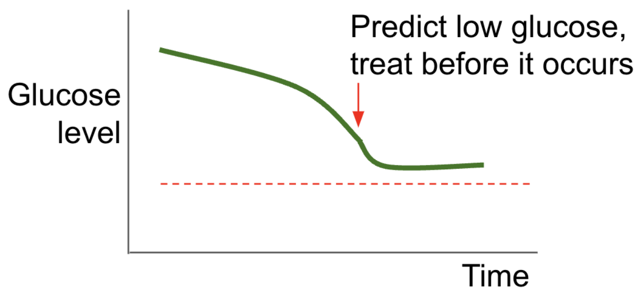


Figure 6: GlucOS predicts hypoglycemia and notifies the individual before it happens so they can proactively prevent it.

Novelty. We beleive that having a parameter real biological state of the system is an important modeling decision; and it diverges, for example, from previous work which only models the biological data in simulation [71, 72]. Traditional software verification techniques like model checking [36] and deductive verification [27] have been explored in the past for hybrid system verification [26] that interact with the human body, but they do not completely account for physiological variability between individuals or model inaccuracies during the insulin delivery process. Also, researchers have used real-world invariants for security in the context of autonomous vehicles [18], but these were based on quad-copter physics not human physiology and did not include formal verification of their implementation.

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, system-level, and biological invariant defenses form the primary barriers against attacks, humans serve as the last line of defense, capable of mitigating attacks that manifest as glycemic imbalance.

When an individual's glucose levels are out-of-range, 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 glycemic imbalance, we focus on how we can use this fact to our advantage to create a more comprehensive security system.

We use predictive yet deterministic mobile notifications to alert people when they are likely to experience hypoglycemia before they reach it (Figure 6), as hypoglycemia presents the most immediate health risk and could be an indicator of an attack. Alerts for hypoglycemia are 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 [11,65,70]. Our contribution is using predictive alerts before someone reaches hypoglycemia as a security defense.

In a user study (Section 9.4), 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. These results demonstrate a reduction in hypoglycemia for the non-attack case, but the most dangerous successful attacks will result in hypoglycemia. Thus, these results suggest that predictive alerting can be an effective security measure.

Using a simple and deterministic algorithm is important since it runs as a part of our trusted computing base. Algorithmically, we predict hypoglycemia by doing a least squares fit for the last 20 minutes of CGM readings and extrapolate forward 15 minutes. We trigger alerts if our prediction shows hypoglycemia is likely 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 out-of-range. Five of the six participants customized their alerts and the 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 security settings and we tried to make the usability vs security tradeoff algorithmically and automatically. But in the end, simplifying the system and allowing users to personalize their security 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 [23,28], 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.

7 Formal verification

To provide additional assurance that our implementaiton is correct, we port our Swift code to Dafny and use Hoare logic. As the system we are verifying is complex, our goal is to locate the pieces and target verification in places where formal methods can have the greatest impact; this is a common reality of applying formal methods at scale, see for example [19]. Since we are porting the Dafny code to Swift, to reduce the chance of human error, we use a line-by-line port as much as possible and add unit tests for each function to provide confidence that the two implementations are equivalent. We port updated code back to Swift manually because Dafny does not include code generation for Swift.

These efforts paid off: we found and fixed nine bugs in our original implementation (Appendix D); in addition to proving various invariants in the code for our insulin delivery path and security mechanisms. However, the most interesting part was where these bugs were: all of our core algorithms were correct, but we found bugs in seemingly less critical parts of the system, like user settings, which had subtle interactions with the core algorithms. For a full list of the invariants that we define and prove, see Appendix E.

8 Evaluation

8.1 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]. We use simulation since this evaluation involves injecting dangerous amounts of insulin.

To be effective, GlucOS's algorithmic security mechanism should protect users against untrusted ML algorithms that administer incorrect insulin doses (overdose or underdose), while minimally impeding algorithms that are efficient at managing T1D. Accordingly, we first run simulations with a predictive ML model inspired from prior research [52], without incorporating our security mechanism to set a baseline of an algorithm that is efficient at managing T1D. Then, in the same simulation scenarios, we introduce an order of magnitude difference (increase and decrease) to the insulin doses computed by the baseline algorithm, without incorporating our security mechanism to quantify the risks of untrusted algorithms. We then repeat both sets of simulations while incorporating our security mechanism to quantify the impact of the mechanism on correct and incorrect dosing.

The goal of T1D management is to maintain glucose levels that are in a safe, target range (70-180 mg/dl) for the majority of the time – an outcome that has been linked to positive long-term health outcomes [53]. Glucose levels 54-69 mg/dl (level 1 hypoglycemia) or less than 54 mg/dl (level 2 hypoglycemia) put individuals at risk for acute complications like impaired consciousness, seizure, and death, while levels 181-250 mg/dl (level 1 hyperglycemia) or greater than 250 mg/dl (level 2 hyperglycemia) increase risks for long-term vascular damage. The American Diabetes Association (ADA) recommends that individuals spend at least 70% of their time in range [6,24] in order to remain healthy.

8.1.1 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 7 shows the average proportion of time spent in range (TIR) by virtual humans from different age groups (adults, adolescents and children) when employing the predictive ML model. We see that individuals across all groups spend over 85% of their

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 7: 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.

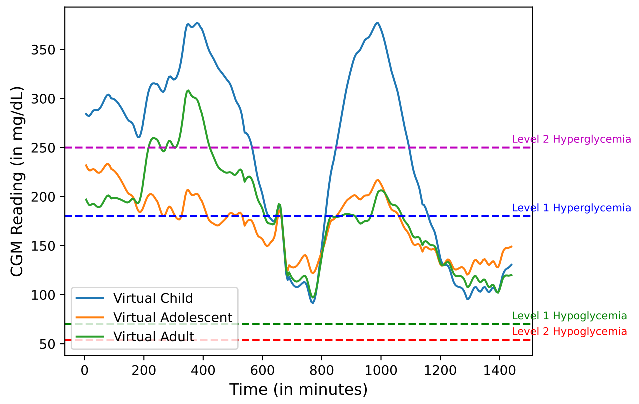


Figure 8: Variation in CGM readings for three virtual humans when running a predictive ML algorithm that intentionally doses one-tenth the amount of the required insulin. The CGM readings staying mostly above the Level 1 hyperglycemic range.

time in range compared to 77% when using only the reactive safe model. These results provide a baseline for an algorithm that is effective in managing T1D.

Incorrect ML dosing Since GlucOS is designed to be flexible with any predictive algorithm, it could be subject to attacks from malicious ML algorithms. To simulate attacks, we intentionally introduce an order of magnitude difference to the amount of insulin predicted by the baseline predictive ML model. In one experiment, we dose one-tenth of the amount of insulin and in another experiment, we dose ten times the amount of insulin computed by the predictive ML model. We run these malicious ML algorithms within the simulator on each virtual human on the same scenarios as the previous experiment.

Figure 8 and Figure 9 show the CGM readings on three randomly chosen virtual humans on a particular scenario for the experiments where we dose less and dose more respectively. When dosing less, we see that all of them spend most of their time above Level 1 hyperglycemia. When dosing more, all individuals fully go below Level 2 hypoglycemia, which would have killed them.

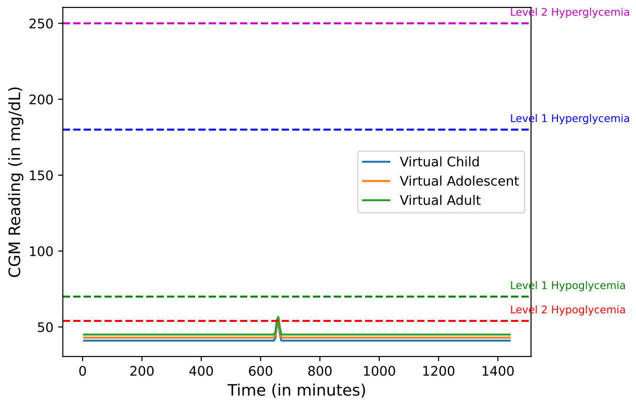


Figure 9: Variation in CGM readings for three virtual humans when running a predictive ML algorithm that intentionally doses ten times the amount of the required insulin. The CGM readings staying below the Level 2 hypoglycemic range would have likely been fatal to all three individuals.

Across all virtual humans, we see an average drop in TIR from 88.44% with correct dosing to 23.32% when dosing less and an average drop in TIR from 88.44% to 0% when dosing more. Correspondingly, when dosing less, we observe an average increase in the time in hyperglycemia from 3.77% with correct dosing to 65.05%. Similarly, when dosing more, we observe an average increase in the time in hypoglycemia from 0% to 100%.

These results further motivate the need for security mechanisms when providing individuals with the flexibility to use any algorithm to manage T1D.

8.1.2 With algorithmic security mechanism

Impact on incorrect dosing. We repeat the experiments from the previous section with the same scenarios. We run the same models for incorrect dosing, but also run our reactive safe model (Section 3) to set bounds on the amount of insulin the malicious algorithms can inject.

Figure 10 and Figure 11 show CGM readings on the same three virtual humans on the same scenarios from the experiment in the previous section. With both types of incorrect dosing, we see that all of them spend over 70% of their time in range. These results show that GlucOS protects individuals even in presence of malicious incorrect dosing. On average, across all virtual humans, we only see a drop in TIR from 88.44% with correct dosing to 73.89% when employing safety while dosing less and a drop in TIR from 88.44% to 78.22% when dosing more. While the times spent in range is less than that of correct dosing, the impact of incorrect dosing is significantly dampened with individuals spending the recommended amount of time in range, despite facing an active attack.

Impact on effective benign algorithms. We evaluate the

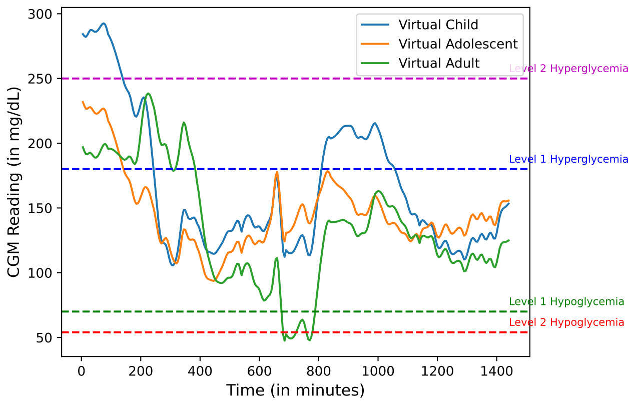


Figure 10: Variation in CGM readings for three virtual individuals when clamping an algorithm that intentionally tries to dose one-tenth the amount of the required insulin. On average, the CGM readings stay in range over 70% of the time.

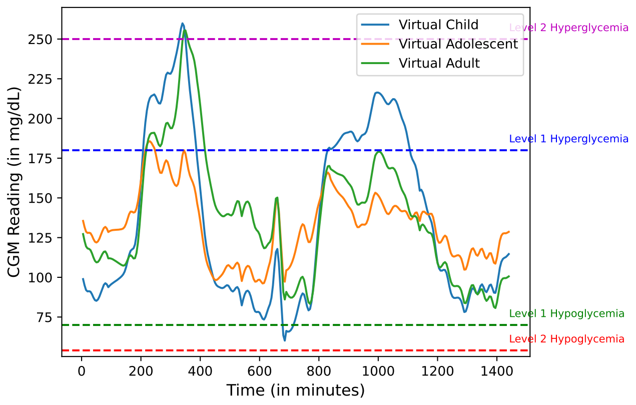


Figure 11: Variation in CGM readings for three virtual humans when clamping an algorithm that intentionally tries to dose ten times the amount of the required insulin. On average, CGM readings stay in range over 70% of the time.

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 random scenarios while maintaining the same parameters for the clamps and reactive model used to evaluate incorrect dosing.

Comparing data in Figure 7 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 guarantees do not come at the cost of effective T1D management from predictive ML.

8.2 Does GlucOS ensure pump operation while overcoming malicious pump drivers?

We first evaluate GlucOS's driver security mechanism in terms of its impact on benign pump drivers that accurately relay pump communications. We then evaluate the impact on detecting additional, modified, or dropped communication from a malicious pump driver. Our mechanism should ensure that all communications go through with a benign pump driver and should catch all manipulated communication with a malicious pump driver. We do not use the simulator for this evaluation since it does not incorporate a pump driver.

Impact on malicious pump drivers. As shown in Figure 12, we attached an Omnipod DASH insulin pump to a stuffed animal. We then modified the source code of the OmniBLE pump driver to intentionally add, drop and modify insulin commands. We detected all manipulated pump commands showing that our driver security mechanism has 0 false negatives in detecting manipulated commands.

Figure 12: We attached an Omnipod DASH insulin pump to a stuffed animal and modified the OmniBLE pump driver to intentionally add, drop and modify insulin commands. Our driver security mechanism detected all manipulations.

Impact on benign pump drivers. For one week, we logged messages from the BioKernel to unmodified OmniBLE as well as messages sent from OmniBLE to the pump on an individual running GlucOS (See Section 9). We report 100% consistency between the commands sent to OmniBLE and commands issued from OmniBLE showing that GlucOS's driver security mechanisms do not incur any false positives.

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

Age Group	**TIR without biological invariant**
Adults	49.93%
Adolescents	55.83%
Children	67.41%

Figure 13: Average proportion of time spent in range across 21 virtual humans when fluctuating their insulin sensitivity. The low values for TIR stress the importance of an end-to-end security mechanism, i.e., the biological invariant.

8.3 Is the biological invariant important?

We run experiments on the simulator to assess the risks of not considering our end-to-end security mechanism, i.e., the biological invariant. Figure 13 shows the average TIR across age groups. Across all virtual humans, we record an average TIR of 57.56%, even with our algorithmic security mechanism, when fluctuating insulin sensitivity. This corresponds to an average drop of 29.28% as a result of not considering the biological invariant. These results emphasize the importance of incorporating an end-to-end invariant and undermines results obtained from pure simulation. We present case-studies involving the biological invariant with a real user using GlucOS in Section 9.3.

9 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.

Our user study for the six students was approved by our university's IRB, and Bob's use GlucOS was determined by our university's IRB to be "self evaluation" and exempt from IRB.

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 that enables separate apps to implement complementary features while remaining isolated from the BioKernel.

Glucose range (mg/dl)	**Range label**	**Reactive 67 days**	**Reactive last wk.**	**ML 1 wk**
<54	L2 Hypo	0.08%	**0%**	**0%**
54-69	L1 Hypo	0.59%	0.48%	**0%**
70-180	In range	96.35%	91.90%	**97.22%**
181-250	L1 Hyper	2.79%	7.62%	**2.78%**
>250	L2 Hyper	0.18%	**0%**	**0%**

Figure 14: The percent of time spent in different glucose ranges for the reactive safe model and the predictive ML model. Bold values represent the best results for each range.

9.1 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.

One of the co-authors of this paper is a board-certified Endocrinologist, who specializes in T1D. They are both a clinician with an active practice seeing patients living with T1D and a researcher who focuses on research on T1D. They designed the safety protocol, which defines the testing we conduct to ensure that the individuals using our system are safe, and the specific criteria we use to stop the study, if needed. Appendix A includes a more detailed discussion of our ethical considerations and safety protocol.

9.2 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.

Figure 14 shows the amount of time Bob spent in different glucose level ranges over the duration of our study. Data from the reactive safe model covers 67 days, followed by one week of data from our predictive ML model. We also show data for the last week of using the reactive safe model to show trends over time. Overall, Bob had the best results when using ML.

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. 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.

9.3 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 mode after two hours.

We reviewed one weeks worth of violations with Bob manually to determine the cause. During the most recent week of data, Bob had five sequences of 30 minutes or longer and all were caused by exercise.

9.4 Are predictive alerts useful for security?

To measure predictive alerts as a security mechanism (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 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 security attacks.

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% [15], showing that our 1.1% decrease is substantial.

Overall, our results suggest that predictive alerts can be a useful last line of defense for trustworthy automated insulin delivery systems.

10 Related work

In addition to the work already discussed in this paper, there are several other related works.

Previous research has looked at the security of implanted medical devices in general [16, 32, 60], in addition to looking at insulin pumps in particular [39, 56], with more recent work looking at providing improved security [9, 45]. Also, recent work has looked at applying formal methods to insulin pumps for high assurance [55]. These works focus on devices and their communication channels. In contrast, with GlucOS we assume that these devices are correct and secure and focus our efforts on the software we use to run the automated insulin delivery system.

Researchers have started to look at combining untrustworthy ML with more traditional forms of automated decision making, like Markov Decision Processes, from a theoretical perspective [20, 40, 41, 50, 57]. These theoretical works focus on providing algorithms to prove desirable properties about a combined ML / Markov system, like robustness and stability. In contrast, we take advantage of domain knowledge and formulate a simple and practical system for one specific problem: automated insulin delivery.

11 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, it was our novel insight around all correct algorithms dosing the same amount of insulin that led us to these classic techniques. Plus, from our exploration, the most interesting part was *not* the ML, it was around the principles of reactive safe models and in particular our principle that these safety-critical algorithms should be reactive.

Second, our pump driver security went beyond traditional specification and checking. We needed to actively program the pump to handle dropped cancel commands, which was an unexpected but crucial insight.

Third, 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.

Fourth, our approach to human interaction in the system was nuanced. We found that simply alerting humans or suspending insulin delivery were not always appropriate. For security alerts, personalization and customization were important. Stopping insulin delivery could be dangerous if the user has high glucose and insufficient insulin on board, which required more context-aware responses.

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 computer security, medicine, and ethics to create a practical system.

Our work improves the security of automated insulin delivery systems and also provides a framework for thinking about security in other critical healthcare technologies. Our exploration is an extreme example of personalized, distributed, and automated healthcare using trustworthy computer systems.

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] American Diabetes Association. Pharmacologic approaches to glycemic treatment: Standards of medical care in diabetes. *Diabetes Care*, 44(Supplement 1):S111–S124, 2021.

[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] 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.

[16] 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.

[17] 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.

[18] Hongjun Choi, Wen-Chuan Lee, Yousra Aafer, Fan Fei, Zhan Tu, Xiangyu Zhang, Dongyan Xu, and Xinyan Deng. Detecting attacks against robotic vehicles: A control invariant approach. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 801–816, 2018.

[19] 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.

[20] Nicolas Christianson, Tinashe Handina, and Adam Wierman. Chasing convex bodies and functions with black-box advice. In *Conference on Learning Theory*, pages 867–908. PMLR, 2022.

[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] 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.

[24] 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.

[25] 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.

[26] Martin Fränzle and Christian Herde. Hysat: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30:179–198, 2007.

[27] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. Keymaera x: An axiomatic tactical theorem prover for hybrid systems. In *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*, pages 527–538. Springer, 2015.

[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] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] Sumit K Jha, Edmund M Clarke, Christopher J Langmead, Axel Legay, André Platzer, and Paolo Zuliani. A bayesian approach to model checking biological systems. In *Computational Methods in Systems Biology: 7th International Conference, CMSB 2009, Bologna, Italy, August 31-September 1, 2009. Proceedings 7*, pages 218–234. Springer, 2009.

[37] 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.

[38] 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.

[39] 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.

[40] Tongxin Li, Yiheng Lin, Shaolei Ren, and Adam Wierman. Beyond black-box advice: Learning-augmented

algorithms for mdps with q-value predictions. *Advances in Neural Information Processing Systems*, 36, 2024.

[41] Bev Littlewood and John Rushby. Reasoning about the reliability of diverse two-channel systems in which one channel is" possibly perfect". *IEEE Transactions on Software Engineering*, 38(5):1178–1194, 2011.

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

[43] 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.

[44] 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.

[45] 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.

[46] MITRE. CVE-2011-3386. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3386, 2011.

[47] MITRE. CVE-2019-10964. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-10964, 2019.

[48] MITRE. CVE-2020-10627. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-10627, 2020.

[49] MITRE. CVE-2020-27268. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-27268, 2020.

[50] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *Communications of the ACM*, 65(7):33–35, 2022.

[51] 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.

[52] 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.

[53] 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.

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

[55] 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.

[56] 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.

[57] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. *Advances in Neural Information Processing Systems*, 31, 2018.

[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] 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.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] 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.

[68] 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.

[69] 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.

[70] 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.

[71] 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.

[72] 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.

[73] 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.

[74] 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.

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.

Message name	**Message type**	**Message description**
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 15: Types of messages exchanged between a pump driver and pump pertaining to setup/UX.

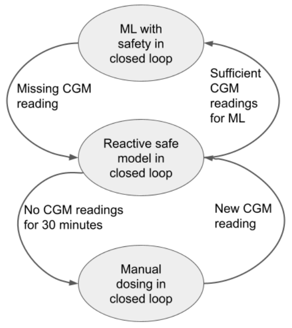

ML with safety in closed loop
Missing CGM reading
Sufficient CGM readings for ML
Reactive safe model in closed loop
No CGM readings for 30 minutes
New CGM reading
Manual dosing in closed loop

Figure 16: GlucOS's CGM state machine to handle missing readings from CGMs.

B List of pump setup and UX communication messages

We list insulin message types pertaining to pump setup and UX in Figure 15.

C Runtime failure management

In this section, we describe GlucOS's *state machine* that combines its security components, i.e., algorithmic security (Section 3), driver security (Section 4) 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

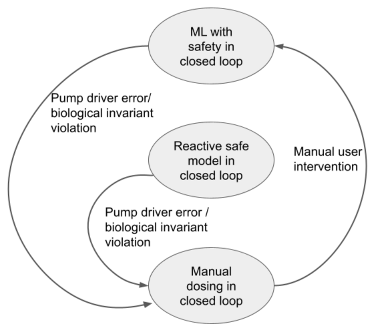

ML with safety in closed loop
Pump driver error/ biological invariant violation
Reactive safe model in closed loop
Manual user intervention
Pump driver error / biological invariant violation
Manual dosing in closed loop

Figure 17: 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.

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 17.

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 16 summarizes these transitions.

D Did formal verification help with correctness?

During the formal verification process of our secure insulin delivery system, we uncovered various bugs and inconsistencies that could have compromised the safety and reliability of the system in niche scenarios. We identified a total of 9 bugs across different categories. 3 of these bugs were related to invalid ranges for variables such as negative value returns, domain-specific out of range values, potential division by zero errors etc. in the insulin dosing calculations which were resolved by adding appropriate checks and error handling mechanisms to prevent such operations and ensure graceful handling of these cases. Additionally, we found 4 instances where the implemented functions violated their intended postconditions along the insulin delivery path, which were addressed by thoroughly reviewing and modifying the implementations to correctly satisfy the postconditions, preserving the intended behavior and meeting our claims for critical funcrions. Moreover, our analysis uncovered 2 situations where certain functions lacked proper error handling mechanisms, potentially leading to undefined behavior or system failures. By addressing these bugs and incorporating formal verification throughout the development lifecycle, we significantly enhanced the robustness, reliability, and safety of our secure insulin delivery system.

E Formal verification: invariants and bugs

In this section, we list the invariants used to formally verify critical functions of GlucOS. Figure 18 lists the bugs we found and fixed during formal verification.

E.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.

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 18: Report of bugs fixed by formally verifying GlucOS.

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.
- 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.