Adaptability, Extensibility and Simplicity of the MetabolicOS

Hari Venugopalan[*]
University of California Davis
hvenugopalan@ucdavis.edu

Shreyas Madhav Ambattur Vijayanand[*]
University of California Davis
smvijay@ucdavis.edu

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

Abstract

Biohacker /'bīō,haker/ *Noun*

1. A person who manipulates their metabolic state using sensors, injected hormones, nutrients, physical activity, computer systems, and artificial intelligence.

2. An enthusiastic and curious person who learns about their own biology and metabolism through experimentation on them self.

3. A person who uses computers to gain access to someone's metabolic state.

There are 8.4 million people living with Type 1 Diabetes (T1D) worldwide [7] and they are *all* hard-core biohackers. They inject a dangerous hormone, insulin, that could kill them in a matter of hours multiple times a day, they measure their metabolism using sensors, and they constantly run experiments to better understand how their bodies will respond to life – food, exercise, stress, sex, alcohol, drugs and so on. Based on what they learn from their experimentation, they strive to master their metabolism using synthetic insulin. Our hypothesis is that with the right software support for biohacking, we can turn a T1D diagnosis from a death sentence into an indicator of longevity, where people living with T1D will be expected to live longer than their otherwise healthy peers.

This paper we present MetabolicOS, a software system that we design and implement for biohacking and to manage T1D. This paper is about our real-world experiences building and running MetabolicOS.

1 Introduction

Our focus in this paper is on T1D. T1D is a metabolic disorder where an individual's pancreas stops producing insulin. To compensate for the lack of insulin production, they inject synthetic insulin. The dynamics of insulin and the impact of various intrinsic and extraneous factors on glucose complicate the management of T1D [29].

One common way to manage T1D is through automated insulin delivery systems. Automated insulin delivery systems read sensor values every five minutes. Using these sensor values, they calculate insulin needs and adjust an insulin pump automatically. Several commercial [1, 2, 6] and open source [13, 17] automated insulin delivery systems exist today.

However, current automated insulin delivery systems have two major shortcomings. First, they are complex. From the open source world, we look at the OpenAPS automated insulin delivery systems. OpenAPS' core dosing logic consists of 1192 lines of Javascript code, has 63 input and configuration parameters, and 90 branch statements – all of which have subtle interactions and can directly affect dosing decisions. Although it is effective at managing T1D, the complexity of its software makes it risky and difficult to update its implementation. Second, they are inflexible. Closed-loop systems provide no forms of extensibility – no apps to plugin, no ability to try new algorithms, no ability to personalize models – you simply get whatever they choose to provide to you. Open source automated insulin delivery systems can be updated, but their complexity makes it difficult to update the parts that matter: the dosing and prediction logic.

This paper presents MetabolicOS, a new system that we design and implement from the ground up specifically for adaptability, extensibility, and simplicity in automated insulin delivery systems. For adaptability, MetabolicOS introduces a novel, *added glucose* abstraction that uses the amount of glucose added to the body in a given time interval to calculate the amount of insulin to dose. The abstraction helps MetabolicOS adapt to all environmental factors since all of them result in a change in the amount of glucose in the body. For example, food intake increases added glucose until it is digested and aerobic exercise decreases added glucose as the muscles consume glucose for energy.

MetabolicOS supports diverse applications that help manage T1D. This provides extensibility to users to pick applications that are best suited to manage their T1D. Some users may prefer to use automated insulin delivery while others may prefer alerts to take manual injections. MetabolicOS

is extensible to support all such applications. MetabolicOS also allows applications to use any algorithm to calculate insulin doses. MetabolicOS restricts these algorithms to remain completely functional at confines them to only issue pump commands. On top of extensibility, this also security by ensuring that arbitrary algorithms cannot modify state and introduce undesired side-effects. MetabolicOS is also simple in that the core security and safety mechanisms that account for mispredictions are based on simple physiological models that only take up 4 lines of code.

We show results on one individual running different algorithms to manage their T1D using MetabolicOS for 3.5 months. Having a real human use MetabolicOS to manage their insulin needs meant that MetabolicOS had to take into account the challenges posed by the practicalities of daily life in the real world. Overall, the individual ran 4 different algorithms at different points in time, before settling on an ML algorithm based on the added glucose abstraction that proved to be most effective for them. To show that the added glucose abstraction generalizes to other physiological factors, we ran experiments with 30 virtual humans in a simulator. ML-based added glucose prediction proved to be more effective than physiological algorithms for managing T1D on 24/30 individuals (80%). This shows that the abstraction is general but is not a silver bullet for managing T1D, thereby reinforcing the need for MetabolicOS to safely and securely support algorithms from other abstractions to support all possible individuals.

Our contributions include:

- The design and implementation of MetabolicOS that provides adaptability, extensibility and simplicity in calculating insulin doses.
- A novel *added glucose* abstraction for insulin delivery that is compatible with all environmental factors influencing the amount of glucose in the human body.
- Account of our experience of running MetabolicOS with a real human which reveals the need to address practical challenges for successful adoption of algorithms for automated insulin delivery.

2 Background on T1D

Type 1 diabetes is a metabolic condition in which the body's immune system mistakenly attacks and destroys the insulin-producing cells in the pancreas. Insulin is a hormone crucial for regulating blood sugar levels by facilitating the uptake of glucose from the bloodstream into cells for energy production. In T1D, the absence or insufficient production of insulin results in uncontrolled high blood sugar levels (hyperglycemia).

People with Type 1 diabetes must actively manage their glucose levels by injecting synthetic insulin. They may choose to do so through calculated amounts of manual insulin injections or or use subcutaneous insulin delivery pump systems for automatic calculation which may be coupled with a suspension mechanism to deliver basal or bolus insulin. Basal insulin refers to the background insulin needed to maintain blood sugar levels between meals and overnight, typically administered as long-acting or intermediate-acting insulin. Bolus insulin, on the other hand, is taken with meals to manage the rise in blood sugar levels after eating. Too much insulin may lead to hypoglycemia and too little insulin will lead to hyperglycemia. Hypoglycemia occurs when blood sugar levels drop below a certain threshold, leading to symptoms such as shakiness, confusion, and sweating, and can be treated with fast-acting carbohydrates like glucose tablets or juice. Hyperglycemia, conversely, occurs when blood sugar levels are too high, causing symptoms like increased thirst, frequent urination, and fatigue. The extreme cases, classified as level 2, of both conditions can have immediate and long term detrimental effects on health that can potentially lead to death. Knowing how sensitive or resistant a person is to a fixed amount of insulin is important to plan timely delivery and treatment. Insulin sensitivity factor (ISF) represents how much one unit of insulin lowers blood sugar levels, helping to calculate the appropriate insulin dose needed to correct high blood sugar. Insulin active refers to the duration during which insulin remains effective in lowering blood sugar levels. Insulin on board (IOB) measures the amount of insulin still working in the body from previous doses, influencing subsequent insulin dosing decisions.

Glucose level	Importance
55 mg/dl	Below 55 mg/dl is considered severe low glucose
70 mg/dl	**Below 70 mg/dl is considered mild low glucose**
82 mg/dl	The average glucose for healthy people pre meal
90 mg/dl	**Our pre-meal glucose target for people living with T1D**
137 mg/dl	The average peak glucose for healthy people post meal
140 mg/dl	**Our goal for peak glucose for people living with T1D**
180 mg/dl	Above 180 mg/dl is considered high glucose
250 mg/dl	Above 250 mg/dl is considered severely high glucose

Figure 1: Importance of different blood glucose values

Time in range (TIR) refers to the percentage of time spent within the target blood sugar range, typically between 70-180 mg/dL (3.9-10 mmol/L). Recommended values for TIR aim for at least 70-80% of time spent in range, indicating effective blood sugar management and reducing the risk of diabetes-related complications.A general outlook on interpreting blood glucose values is provided in Figure 1. Understanding these concepts is crucial for individuals with T1D and their healthcare providers to optimize insulin therapy and maintain blood sugar levels within the target range for better overall health and quality of life.

3 Motivating Example

Let us consider the example of Bob, who lives with T1D. In his initial year grappling with the condition, he manually administered insulin using a syringe when required. However, he soon realized the unpredictability of his body's response to various factors like food, exercise, stress, and illness. Maintaining balanced glucose levels became a daunting task and this made Bob switch to a state of the art commercial insulin delivery system. While things did get better than manually calculating and injecting insulin, he soon found they couldn't offer the precise control he needed. They also remained closed systems, unable to accommodate Bob's unique physiological needs.

Taking matters into his own hands, Bob decided to explore open source insulin delivery systems made public by the T1D biohacking community, shared by people who have the same condition and have managed it successfully for several years. He adopted and tried a few of their most widely used insulin delivery systems but that was not the end of it. Every time

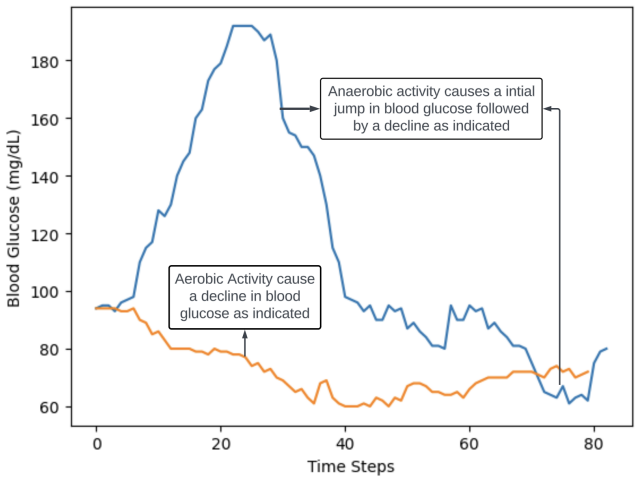

Figure 2: Anaerobic and Aerobic activity effects on the blood glucose of a person with Type-1 Diabetes. The blue line shows the blood glucose curve for anaerobic activity while the orange line shows the curve for aerobic activity.

Bob eats a meal, he has to announce his meal to his insulin delivery system and input the exact amount of carbohydrates he has eaten for accurate insulin dosing. If Bob overestimates the amount of carbohydrates in his meal, his blood glucose can drop below the recommended range and underestimation can lead to high blood glucose. Other factors like meal fat composition may also influence how and when the glucose absorption occurs from the digestion process. Expecting Bob to accurately predict his food composition to help administer insulin can be quite precarious. Bob also faces challenges when he hits the gym(anaerobic) and cycles to work(aerobic). During anaerobic exercise, such as weightlifting, Bob experiences an initial rise in blood glucose followed by a drop, while aerobic exercise, like cycling, causes a decrease in blood glucose levels initially, followed by a later increase as shown in Figure 2. Bob's insulin delivery system struggles to adapt to these fluctuations so Bob switches off his system, manually injects or inputs a temporary basal rate. These challenges prevent Bob from using his insulin delivery system in a complete 'closed-loop'.

Bob wanted these challenges to be averted. He was also open and eager to run ML on himself for insulin delivery, as he soon came to know of their predictive power through literature. Most of the previous open source systems based their decisions on complex physiological calculations coupled with complex software which made the system hard to interpret and understand. However he was unsure if a general one fit for all ML would be able to model his own physiology well enough to achieve tight control and was a bit hesitant in trusting these models considering mispredictions could have serious repercussions.

With MetabolicOS, Bob can run any ML model on himself in a trusted and safe manner. The model is personalized to Bob's unique physiology. It uses a different abstraction rather than carbohydrate count, enabling him to not worry about calculating his meal compositions manually and being able to establish control for any glucose addition event other than meals.

Bob is a real human. He has been running the MetabolicOS, which administers his insulin, on himself for the past 3.5 months.

4 Overview

In this paper, we present MetabolicOS, an adaptable, extensible and simple system that supports apps to manage T1D. MetabolicOS aims to provide extensibility to support different types of applications that help manage T1D.

From an algorithmic standpoint, adaptability provides the ability to navigate through differing scenarios introduced by dynamically changing factors such as food intake, exercise, and stress levels. However, implementing separate mechanisms to address each scenario poses challenges in terms of correctness and maintainability. Therefore, MetabolicOS advocates for the use of generalized abstractions in algorithm design. These abstractions allow for coverage of different scenarios while ensuring simplicity in implementation.

Viewed from a systems perspective, extensibility empowers applications to offer rich functionality to users, while simplicity serves to minimize potential attack vectors. Extensibility helps cater to a wide range of users with varied needs based on their physiologies and preferences. However, unrestricted code execution in pursuit of extensibility introduces vulnerabilities that malicious actors can exploit, posing grave risks, particularly in the context of T1D, where attacks can have life-threatening consequences. Thus, we introduce security

mechanisms that restrict applications from directly interacting with the CGM and insulin pump. These security mechanisms have to be simple since that would give us the ability to reason about their correctness and reduce the likelihood of vulnerabilities, thereby reducing the attack surface.

In this section, we provide a high-level overview of MetabolicOS's architecture. We provide a detailed account of MetabolicOS's adaptability, extensibility and simplicity aspects in the next 3 sections.

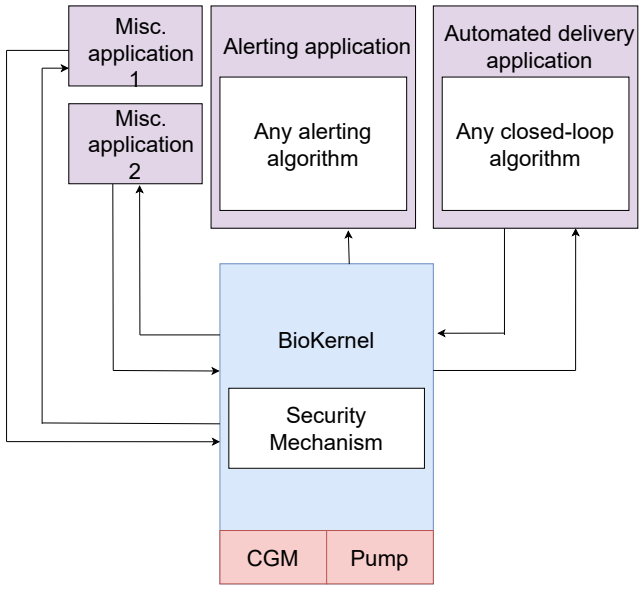

Figure 3: Overall architecture for MetabolicOS.

4.1 System architecture

Figure 3 shows our overall system architecture. At the core of our overall architecture is an iOS app, called the BioKernel. The BioKernel is the component that interacts with the CGM and insulin pump hardware, runs safety mechanisms, and produces event logs that other components use to learn the state of the system, which we will look into detail when we discuss the simplicity of MetabolicOS.

To support rich functionality, the BioKernel stores event logs which is then consumed by other applications. These could include a watchdog for monitoring an individual's metabolic state and triggering alerts in case of glycemic imbalance, an automated insulin delivery application running in closed-loop, an application to share metabolic updates with a health care professional etc.

5 Adaptability

Adaptability of a system is the ability of the system to accommodate and operate under changed circumstances. The MetabolicOS insulin delivery system has the ability to adapt to 1) different daily events 2) different humans and 3) different decision algorithms.

Motivated by the exaample of Bob, we build the adaptability of the MetabolicOS on three core principles (a) Move past the carbohydrate abstraction to model any glucose addition event (b) Personalizing the model to each individual for precise delivery (c) Run any machine learning model with any abstraction on the MetabolicOS.

Most current insulin delivery systems that run on real humans are powered by reactive control algorithms and base their decisions primarily on carbohydrate input and meal announcements from the user [20]. Some recent approaches have explored automated meal detection [23, 25]and carbohydrate estimation [14, 27] through predictive algorithms [11,28]. However, these systems require some level of manual intervention, carbohydrate count entry and meal announcements from the user, which takes away the system from being truly 'automated'. Another major challenge faced by carbohydrate based decision making in modeling glycemic control for events other than meals. These include controlling glucose levels during and after exercise, variable meal sizes/timings, unusual snacking, sleep disruption and irregular lifestyle activities that a user may indulge in.

To move past these challenges, we feel that different abstractions, other than carbohydrate count, need to be explored as the primary factor in determining how much insulin needs to be delivered. In this paper, we introduce the added glucose abstraction as the main motivator for the MetabolicOS on how much insulin needs to be delivered. Added glucose refers to the amount of glucose that is added to the body over a particular time frame. This can be calculated directly using just the CGM and insulin readings from the user.

5.1 Machine Learning with Added Glucose

By combining the predictive power of machine learning with added glucose, our goal is to establish tighter blood glucose control by predicting future metabolic states of people living with T1D. Our training data is derived from the historical CGM readings and insulin readings of an T1D individual. We have 30 simulated individuals provided by FDA-approved UV/PADOVA Type 1 diabetes simulator [15]- 10 adults, 10 adolescents and 10 children- all with different physiological and biological parameters. Each individual's historical data acts as its own dataset as we train personalized models. We calculate the amount of glucose added to the blood over a time period to give a value of added glucose at future time steps mathematically by deriving insulin on board, active insulin from the readings and using insulin sensitivity factor of the particular individual, providing us with the target labels. We look ahead one hour and predict added glucose in our implementations. Our approach eliminates the need for manual labeling or annotations from users, as we automate label

generation based on direct calculations from the individual's readings. This efficiency streamlines the training process for machine learning models. Consequently, we obtain a dataset for each individual, comprising of added glucose values at different time steps over a 30-day period, which serves as our target variable for prediction.

The goal of the training process is to build models to predict the added glucose of an individual one hour ahead of time in order to provide timely delivery of insulin, which is modeled as a regressive task. Different machine learning models were trained and evaluated on each of the individual's historical data and run on the MetabolicOS, to support the validity of our added glucose abstraction while ensuring extensibility and adaptability of our system. Each model type is trained on every individual's data separately and evaluated on that particular individual. Since we have 30 individuals and 4 types of models, a total of 120 models have been trained and tested.

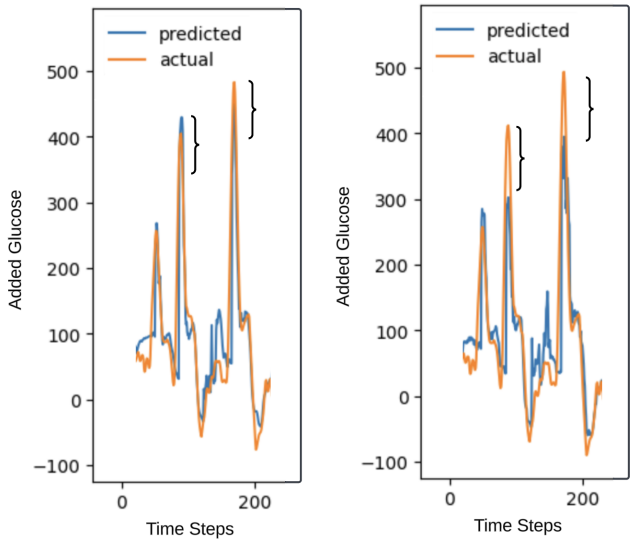

Figure 4: Depiction of added glucose vs time for one of the adult T1D individuals. The right side showcases neural network predictions trained with mean squared error loss while left side depicts our custom loss neural network model

---

Our first implementation is a traditional Support Vector Regressor(SVR). In ensemble methods, Random Forest Regressor and XG Boost Regressors were evaluated on the MetabolicOS. Our XG Boost regressor creates 100 decision trees sequentially, with each tree trying to correct the errors made by the previous ones. We use a learning rate of 0.1, controlling the contribution of each tree in the ensemble. Our Random Forest Regressor is created as an ensemble of 100 decision trees. Our final model used is also currently running on our real human is a neural network that was trained using a custom loss function. Our neural architecture is a three layer network with ReLU activations throughout with a single output neuron for regression. We have created a custom loss function termed 'peak loss' which places emphasis on the peaks of added glucose through weighted penalization. We aim to model the peaks as accurately as possible shown in Figure 4 as high amounts of added glucose cause large spikes in blood glucose values if insulin is not administered in a timely manner.

Our testing is done by simulating each individual for 6 continuous days. This is done with 5 random seeds, leading to a total of 30 days of testing period for each individual. The insulin delivery decisions are based on the added glucose predictions. While the direct output of our models is the added glucose one hour ahead, we place more emphasis on how well the complete system performs in maximizing blood glucose time in the range of each individual. We record the time in range, time in hyperglycemia, time in hypoglycemia and establish a comparative analysis between these models.

We also use a reactive model as a point of comparison. This is a simple insulin absorption physiological model that directly calculates the amount of insulin that needs to be delivered or withheld CGM and insulin readings. 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 our evaluation, even the most traditional model crushes the reactive delivery model on the simulated individuals and on a real human, showing why it is important for automated insulin delivery systems to have a practical way to adopt ML.

Alternatively, we could have designed a more sophisticated physiological predictive model, but this type of model has two shortcomings. First, more sophisticated models require a more complex implementation and deeper reasoning to understand the intuition behind how they work. In MetabolicOS, we value simplicity as a core first principle. Second, the underlying physiological models are approximations – humans are not second-order differential equations.

The reactive model suffers primarily from being unable to model postprandial hyperglycemia and alternate glucose addition events, a drawback which we initially mentioned. We choose to use ML to predict future metabolic state and our predictive ML model detects the entire digestion curve accurately. The added glucose abstraction hence does not require carbohydrate entry of meal announcements from the user. Since we model our prediction and mitigation on any form of glucose addition to the body, the MetabolicOS is not restricted to meals and can manage insulin delivery for all lifestyle events like exercise, sleep etc, providing a more generalized approach. The major concern of most systems using predictive models hindering widespread adoption is the safety of the user when mispredictions occur. The underlying security logic[cite biokernel] of MetabolicOS which is beyond

the scope of this paper allows the predictive ML model to borrow insulin from the future to inject insulin proactively. In general, it uses the predictive ML commands as long as they stay within a fixed bound of what the reactive safe model would have sent.

5.2 Model Personalization

Our system adapts its decisions to each individual as well rather than just the different events in life. Each person has a different metabolism and a unique physiology. Our system supports insulin delivery to humans with diverse physiology and lifestyles, eschewing a one-size-fits-all approach in favor of precision medicine through personalized models for each individual. Building upon our aforementioned benefit on automatic label generation with the added glucose abstraction,we use historical glucose and insulin data from each individual to generate added glucose labels and train a personalized model. This helps us understand each individual's own lived experiences and the uniqueness in their body's reaction to different events of glucose addition. This also helps us account for trends caused by differences in genetic, environmental, behavioral and lifestyle factors rather than a one decision for all approach.

Different users place importance on different aspects of a machine learning model that they want to control life or death decisions. While some users are fine with a black box model that provides them with the best TIR, some may prefer models that they can understand,interpret and trust enough to run on their own bodies. Different ML models may also work better for certain humans based upon the different underlying trends on how their bodies react to glucose, insulin and their unique lifestyle events. Users should be able to swap out, to upgrade or revert back and try models in a safe and trusted manner. MetabolicOS provides a safe[cite biokernel] and adaptable platform for users to try out and experiment different ML models on themselves. We have evaluated different models starting from conventional moving all the way to deeper networks, on the system . Our real human Bob started using a physiological model on the MetabolicOS, swapped to a simple ML model and now runs a custom loss neural network all over the period of just 4 months. While we firmly believe machine learning is the way forward and aim to provide MetabolicOS for users to run any ML model they want, we also strongly support an individual's biological free will to stick to a reactive algorithm. Hence the MetabolicOS system can adapt to and support any predictive(both machine learning and physiological) and reactive algorithms to help users safely biohack their bodies.

5.3 Other Abstractions

Added glucose is a strong step forward in the direction of exploring different abstractions that power the insulin delivery process and move past manual carbohydrate count. While added glucose exhibits higher TIR for most individuals than the reactive model, for specific simulated individuals, the reactive model was recorded to show better TIR in specific scenarios which motivates the exploration of other abstractions to move past both added glucose and carbohydrate counting for certain physiologies. Different people may need to use different abstractions to model their physiology better just how they may choose to incorporate different ML models into their systems. Some possible abstractions that can be explored further include modeling predictions and decision making based upon Insulin Sensitivity Factor(ISF) and basal rates. The insulin sensitivity of a person determines how much insulin they need to mitigate for a particular glucose event. A highly sensitive person requires far less insulin for a meal while a less sensitive(resistant) individual will require a far higher dose to bring them back into ideal blood glucose range. This ISF value changes throughout the day and modeling this factor directly could be a possible explanation to predict insulin deliveries. Predicting basal rates directly is also worth exploring in the future since this helps us tackle the insulin delivery problem directly at its core without other underlying calculations.

6 Extensibility

MetabolicOS provides two types of extensibility: algorithmic extensibility and systems extensibility. For any given application to manage T1D (such as automated insulin delivery), algorithmic extensibility enables the application to employ any algorithm that it sees fit to manage T1D. Systems extensibility enables any application to interface with a CGM and pump to help manage T1D (such as automated insulin delivery, collecting glycemic logs, alerts for manual injections etc). In this section, we first explain how MetabolicOS provides algorithmic extensibility and then explain how it provides systems extensibility.

6.1 Algorithmic extensibility

We take the example of automated insulin delivery in closed -loop to explain how MetabolicOS provides algorithmic extensibility. A proportional controller is a simple algorithm to calculate insulin doses. At any given instant, a proportional controller would keep track of the difference between the amount of glucose present in an individual's blood stream and their desired target for the amount of glucose. Using the individual's insulin sensitivity factor, the proportional controller injects insulin that would consume this difference. We visualize this algorithm in Figure 5. While this algorithm is safe in that it only injects insulin to account for excess glucose, it is reactive in nature which makes it difficult to maintain tight control. This is because synthetic insulin is slow and takes the order of hours to act. By the time the injected insulin acts

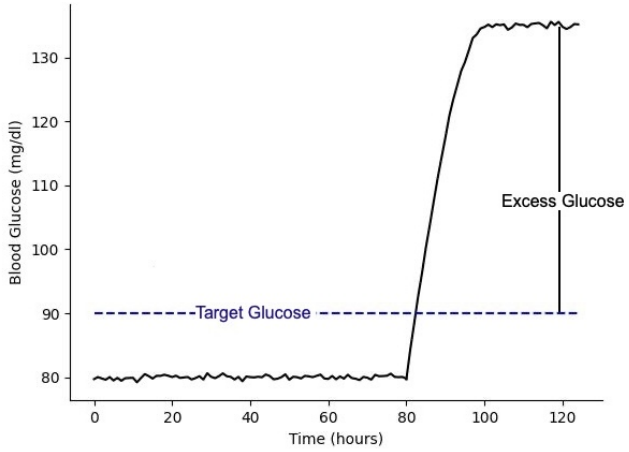

Figure 5: A proportional controller keeps track of the amount of excess glucose and injects insulin to consume the excess glucose. Such an algorithm can never put the user at risk, but only provides loose control over T1D.

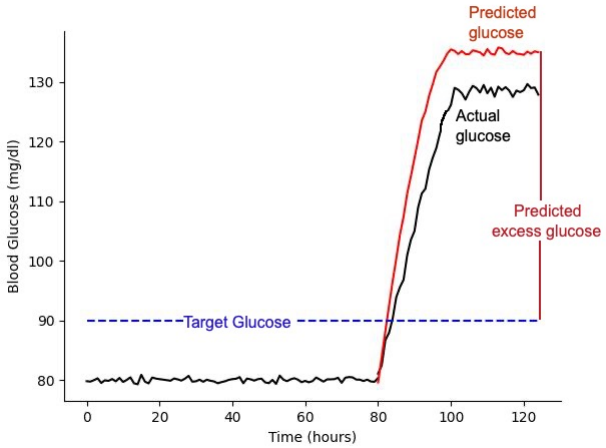

Figure 6: A predictive algorithm predicts the amount of glucose that will be in excess over a period of time and injects insulin to consume that glucose ahead of time. Such an algorithm can help provide tight control but can put individuals at risk when making mispredictions.

to account for the added glucose, the amount of glucose in the blood stream would have changed.

Users seeking tighter control could prefer an algorithm that predicts the amount of glucose that would be added in the next few hours and inject insulin to consume that glucose ahead of time. We visualize this algorithm in Figure 6. However, this algorithm would only effective when it is able to accurately predict changes to the amount of glucose. This algorithm would be dangerous for an individual whose glucose levels vary wildly thereby making it difficult to accurately predict changes to the amount of glucoses. For example, if the algorithm incorrectly predicts the amount of glucose to keep increasing (and thus, keeps injecting insulin), while the actual glucose values keep decreasing, this algorithm would push individuals to hypoglycemia.

Thus, different individuals may prefer to choose different algorithms for automated insulin delivery based on their physiology and desired level of control. A rigid system that incorporates a particular algorithm forces individuals to only use that algorithm even if it does not suit them. This system provides very high security since it only runs trusted code, but it offers limited flexibility and control to individuals. In contrast, a system that allows arbitrary code execution to support any algorithm is also dangerous since insulin delivery would inherit all vulnerabilities present in the arbitrary code which could be fatal to individuals. This system provides the most flexibility, but also provides the least security. With MetabolicOS, we take a middle ground where we allow arbitrary algorithms to administer insulin doses as long as the algorithms are purely functional beyond setting parameters to control the pump. This reduces the attack surface by eliminating side effects that can rise by executing arbitrary code.

At the same time, MetabolicOS enables flexibility to support diverse individuals by allowing them to use any algorithm to calculate insulin doses. We summarize this discussion in Figure 7.

To protect users from accidental or intentional algorithmic mispredictions, MetabolicOS regulates the amount of insulin administered by the algorithm using a safe, independent physiological model. From an architecture perspective, MetabolicOS isolates the physiological model within a component called the BioKernel which interacts directly with the pump.

6.2 Systems extensibility

MetabolicOS also offers extensibility to support different types of applications that help manage T1D. The BioKernel (described in the previous section) is at the heart of our design and enables extensibility. The BioKernel is the component that interacts with the CGM and insulin pump hardware, runs the safe physiological dosing algorithm and produces event logs that other applications consume for their operation.

The event logs captured by the BioKernel record events pertaining to the CGM and the pump. CGM logs only consist of a timestamp and a glucose reading, while pump logs include the commands issued to the pump along with their timestamp. Applications for automated insulin delivery periodically consume both CGM and pump logs to calculate the amount of insulin to dose based on CGM readings, past insulin doses and the amount of insulin retained by the body. Applications for alerting consume CGM readings to predict glucose levels going out of the glycemic range to send alerts

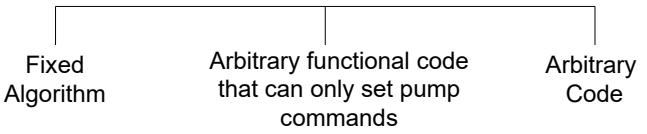

Figure 7: This scale shows the extremes to which algorithms can be provided with extensibility. The left extreme provides no extensibility by running a fixed algorithm for insulin delivery. However, it is the most secure. The right extreme allows for arbitrary code execution which makes it the most extensible to support different algorithms. However, allowing arbitary code execution makes it the least secure. MetabolicOS takes a middle ground of supporting arbitrary functional code execution that can only send commands to the pump. Allowing arbitrary functional code provides extensibility to algorithms while restricting it to only send commands to the pump provides security.

to users to take corrective actions to stay in range.

One implementation decision worth noting is storing our event logs in a cloud-based service. In our original design, everything ran on device for both improved privacy and availability. However, iOS (the platform where we implemented the BioKernel) is a general-purpose OS and its current abstractions mismatched what we needed in terms of background execution. However, this design choice provides us with an opportunity to independently verify the execution of critical components like the safe physiological dosing algorithm.

7 Simplicity

With simplicity it is easier to reason about correctness and we reduce the likelihood of bugs and vulnerabilities. In this section, we first explore the simplicity of two popular automated insulin delivery frameworks, OpenAPS and Loop. We then, present the simplicity in MetabolicOS's design.

The complexity behind OpenAPS' decision making stems from their multitude of parameters influencing basal rate calculations. The OpenAPS core function for calculating insulin doses consists of 1192 lines of Javascript code, 63 input and configuration parameters, and 90 branch statements that all influence the ultimate dosing decision. This logical complications and lack of abstraction makes updating this safety-critical code difficult and complicates the application of formal methods.This complexity poses challenges in modification, comprehension, and garnering trust in the system's accuracy.Loop, on the other hand, introduces the concept of Glucose Momentum, which lacks empirical basis.

In response to these challenges, we propose principles guiding the design of the BioKernel. The BioKernel, which resides in the center of our system, manages the CGM and insulin pump hardware, runs the safety algorithms, and saves event logs to the event log service to enable other apps to recreate its state and extend its functionality. Our primary aim is simplicity, achieved by minimizing the number of parameters required for basal rate calculations. Equally important is aligning these parameters with human physiology, ensuring they reflect inherent biological processes. Overall, the most important aspect of our system is keeping the BioKernel's implementation simple.

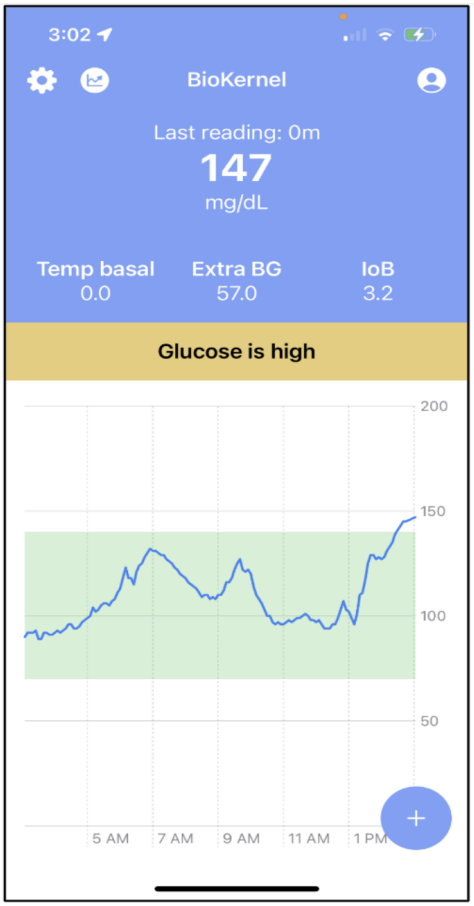

Figure 8: The main user interface for the BioKernel

7.1 The BioKernel

Although our design and implementation support extensibility, the BioKernel can serve as a stand-alone automated insulin delivery system. Using the main user interface for the BioKernel shown in Figure 8, the individual can configure their therapeutic settings, change their pump or CGM, view summary statistics of recent glucose readings, or deliver an insulin dose. First, the BioKernel uses LoopKit, an open-source library, for device drivers to interact with the underlying CGM and insulin pump hardware. Second, the BioKernel implements glucose, insulin, therapeutic settings, and alarm services that ingest

events from the LoopKit drivers and provide abstractions to the rest of the BioKernel. Third, the closed-loop run time queries the abstraction services and runs through the closed-loop safety calculation to update insulin delivery. Fourth, the BioKernel provides trusted UI components that other apps can invoke using Universal Links to update therapeutic settings or to dose insulin. Fifth, the BioKernel event logger sits in between the LoopKit drivers and BioKernel abstraction services to log these events.

7.2 Architecture Simplicity

Our innovation is in the architecture we use to implement T1D management features. We use separation principles from the OS and microkernel areas, applied to the application layer for strong isolation and simplicity of our software components, similar to secure web browsers. We decompose the system into isolated modules, and expose narrow and well-defined interfaces. These interfaces help provide the anchor for our security policies and form the foundation for our extensibility mechanisms. Our main approach for achieving simplicity is by logging events and offering interfaces for other applications to interact with the BioKernel. This extensibility allows us to incorporate the expected features of an automated insulin delivery system while maintaining the simplicity of the BioKernel intact. Another key approach to simplification we undertake is achieved by reimagining the core closed-loop algorithms, leveraging advancements in ultra-fast acting insulin. By utilizing these faster acting insulins, we can eliminate certain complexities such as core abstractions, predictions, and simulation functions typically present in automated insulin delivery systems. Additionally, since our software directly impacts a biological system (humans injecting insulin) operating on longer timescales, we can defer some verification tasks and correctness checks outside of the BioKernel, knowing we have hours until adverse effects occur.

8 Evaluation

In this section, we evaluate the adaptability, extensibility and simplicity of the MetabolicOS. The adaptability evaluation focuses on establishing the validity of using ML and the added glucose abstraction for the insulin delivery. We evaluate the algorithmic extensibility and dive deeper into the event logs and metabolic watchdog components of MetabolicOS to understand the systems extensibility. The final simplicity section focuses on evaluation of the complexities in the BioKernel.

Our evaluation data comprises 30 simulated patients from the UVA-PADOVA simulator and Bob, an actual human who uses MetabolicOS. The testing evaluation of each patient's model was done on a month's worth of unseen data for that patient. To run our real-life experiments, we use MetabolicOS running on Bob's iPhone 14. Bob uses a Dexcom G7 CGM and a combination of manual insulin injections using a syringe and Humalog insulin in addition to insulin injections from a Omnipod Dash insulin pump with Lyumjev insulin that he uses with MetabolicOS's closed-loop algorithms. Bob also wears an Apple Watch, which we use to deliver notifications from the Metabolic Watchdog.

8.1 Is added glucose a good abstraction for machine learning?

We aim to evaluate our added glucose abstraction by making it the target variable for ML predictions. We use this prediction to calculate the amount of insulin to be injected. We calculate each patient's Time in Range(TIR) as the determining factor of how good a model is rather directly considering model accuracy in added glucose predictions as we wish to evaluate both the model and the added glucose abstraction itself, in helping human stay in health blood glucose range. A total of two month's historical data(one month for training the model and one month for testing the model's performance) of CGM and insulin data collected throughout the day at different time steps was used as our starting point.

We can see that all ML models with added glucose outperform the reactive model significantly in helping adults, adolescents and children with T1D stay within the time in range shown in Figure 9 . On a comparative analysis we see that the best performance for adults and adolescents provided by our Neural Network(NN) architecture with custom loss function and best performance for children provided by random forest regressors. We attribute the better performance of random forest over the neural network to size and nature of data, high non linear fluctuations in glucose readings due to a child's metabolism and lower blood volume and its robustness toward noise/outliers.

8.2 Does ML decrease chances of extreme hypoglycemia?

While we have compared our models to the reactive physiological model in terms of TIR, we also emphasize the need for the system to not let the users hit extreme lower or higher values of blood glucose when they are not in range. Both hyperglycemia (high blood sugar) and hypoglycemia (low blood sugar) can be dangerous if not managed properly, especially if they reach level 2 severity. Extreme hyperglycemia can lead to long term health complications like ketoacidosis, cardiovascular conditions, nerve damage and other chronic conditions. Extreme hypoglycemia on the other hand is known its immediate adverse impact on a person's life in the form of seizures, loss of consciousness, cognitive impairment and potentially death and hence, it is much more imperative to make sure the patient spends negligible to no time in this particular level 2 range. In this section, we evaluate our MetabolicOS running the custom neural network against the reactive physiological

Patient Type	Reactive TIR %	Support Vector TIR %	Random Forest TIR %	XG Boost TIR %	NN TIR %
Adults	88.25%	91.16%	91.26%	91.15%	**91.49%**
Adolescents	81.52%	85.44%	85.76%	87.75%	**88.85%**
Children	76.24%	76.43%	**86.09%**	83.16%	84.16%

Figure 9: The average percentage of Time in Range (TIR) spent by each group of patients using different ML models with added glucose. We measure time spent in ranges using CGM data and the bolded values represent the best results for each range.

model on the time spent by the patients in level 2(extreme) hypoglycemia.

Metric	NN %	Reactive%
L2 avg	85	75
L2 max	90	80

Figure 10: Comparison of average and maximum time spent in level 2 hypoglycemia by patients using NN and Reactive Model

Considering the criticality of managing hypoglycemia over hyperglycemia, we explore the average amount of time spent by the 30 patients and the maximum amount of time spent by any patient in level 2 hypoglycemia as an evaluation how safe the models work even when they are out of the ideal blood glucose range. From the Figure 10, we can see that our neural network formed with our added glucose abstraction provide better outcomes for the patients than the reactive physiological model.

Overall, we can see that even when the patient is not in the recommended blood glucose range, ML protects them from spending a significant amount of time in the dangerous level 2 segments.

8.3 Is using a distributed event log effective for decomposing the system?

In this section, we evaluate our alerting app, called the Metabolic Watchdog, to measure how effective it was when running as a separate app as in Figure 11. Because in MetabolicOS we maintain simplicity through our event logging and distributed app architecture, evaluating how well this distributed system performs in practice is important.

The main idea behind the Metabolic Watchdog is that it is a separate app that reads CGM values and predicts hypoglycemia before it happens. When it predicts hypoglycemia, it alerts the user so that they can eat sugar to avoid low glucose from happening.

To evaluate the Metabolic Watchdog, we review four weeks of data from Bob, from February 22nd, 2024 through March 22nd, 2024. We measure the number of alerts that the Metabolic Watchdog sent Bob and compare against an idealized version of the same algorithm that would have run as each new CGM value arrives. This idealized algorithm shows what an alerting system built directly into the BioKernel would have done without managing delays and network connectivity issues from the distributed nature of MetabolicOS.

Over the period of time, the distributed Metabolic Watchdog sent Bob 68 alerts. In comparison, our idealized algorithm would have sent 70 alerts, so overall the Metabolic Watchdog sent alerts successfully 97% of the time. The two alerts that the Metabolic Watchdog missed where both due to network failures where the BioKernel was unable to sync its logs to the MetabolicOS server fast enough after reading in a CGM value that would have led to an alert. Upon reviewing the data, these missed alerts were both algorithmic false positives where the alert would have fired but Bob did *not* experience hypoglycemia.

We consider this success rate of 97% to be adequate for the Metabolic Watchdog. The reason we believe this is because our system has redundancy built in, where if Bob had experienced hypoglycemia his CGM app, which is separate from MetabolicOS, would have alerted him using non-networked and local data. Given this built in redundancy, and the simplicity gains we get from pushing non-critical functionality outside of the BioKernel, we believe that our architecture is appropriate.

An alternative implementation for the Metabolic Watchdog could be to use background execution and local IPC channels on the phone to run the alerting software. This alternative implementation would have been consistent with our architectural separation principles, but the reason we opted for our current implementation is that iOS does not have abstractions for running background tasks every five minutes unless these tasks connect to a Bluetooth device, like the BioKernel does.

8.4 Is the BioKernel complex?

To evaluate the complexity of our BioKernel, we count the lines of code in our implementation and compare against Loop, another open-source automated insulin delivery system. Using the "cloc" utility the BioKernel app has 4.6k lines of code compared to 39.8k lines of code in the Loop app, an order of magnitude reduction. We omit the lines of code coming from LoopKit and other drivers because these are shared between both projects. However, LoopKit and the drivers have a substantial amount of code, weighing in at 70.1k lines of

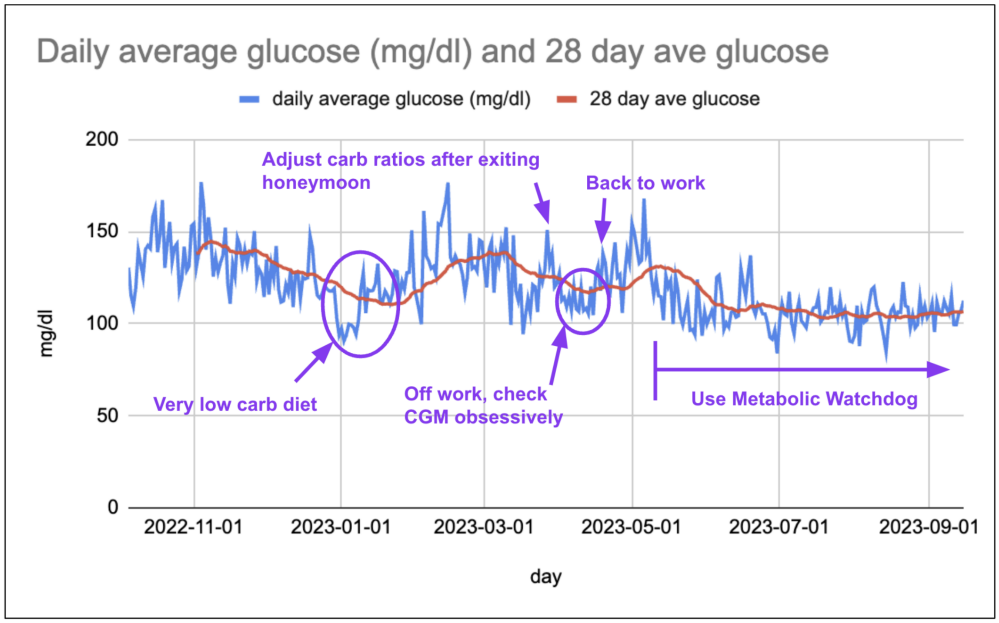

Figure 11: Depiction of added glucose vs time for one of the adult T1D individuals. The right side showcases neural network predictions trained with mean squared error loss while left side depicts our custom loss neural network model

code and is likely the next big opportunity for simplification. All those extra lines of code in Loop are useful, which is why we port Loop to run within MOS. Bob uses it for meal announcements and looking at their simulation and prediction results. However, for the core closed-loop algorithm we show how to keep it separated in an isolated protection domain while still providing the right interfaces to enable a fully featured automated insulin delivery system. To explain why we have such a large reduction in source code, we outline the differences between the two systems. First, Loop has several features that we move outside of the BioKernel. These features include a remote interface for insulin dosing (which we think it a bad idea in general), a Watch app, Siri command interfaces, third party libraries, tutorials, meal announcements, physiological simulations, and predictions of future metabolic states. Second, we simplify our implementation of features that are shared between the BioKernel and Loop.

9 Related work

Several automated insulin delivery systems exist today. With companies such as Tandem [2], Insulet [1], and Beta Bionics [6] all providing commercial closed-loop systems that connect CGMs to insulin pumps for automatic insulin delivery. From the open source world, OpenAPS [17] and Loop [13] also provide systems that people can use. To the best of our knowledge, none of these systems use ML. Our study builds on top of these works, where we use many of the safety principles from OpenAPS, the CGM and insulin pump drivers from Loop, and the push for user-facing simplicity from Beta Bionics. However, our focus is on how to provide security, safety, and extensibility for ML-based closed-loop systems while maintaining a small trusted computing base in our implementation.

For our software system architecture, we use separation principles from the OS and microkernel areas [3, 10] applied to the application layer for strong isolation and simplicity of our software components, similar to secure web browsers [8, 21, 24, 26]. Like work on secure web browsers, we apply OS principles at the application layer. In our implementation, we strive for simplicity for our trusted computing base first and foremost, while also facilitating extensibility. We want people to run whatever closed-loop algorithms they want to and to have a fully-featured automated insulin delivery system – we focus on providing mechanisms for rich functionality and flexibility securely and safely to this new application domain.

Previous research has looked at the security of implanted medical devices in general [5, 9, 22], in addition to looking at insulin pumps in particular [12, 19], with more recent work looking at providing improved security [4, 16]. Also, recent work has looked at applying formal methods to insulin pumps for high assurance [18]. These works focus on the device and their communication channel. In contrast, with MetabolicOS 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.

10 Conclusion

Individuals with T1D face significant challenges to keep up their metabolism to match that of their healthier counterparts. The complexities in T1D makes it difficult to devise a common, adaptable solution for all individuals. Computer systems have an opportunity to safely support customizable solutions for individuals to manage their T1D.

In this paper, we presented MetabolicOS, an adaptable, extensible and simple system to support diverse systems and algorithms to manage T1D. The extensibility and security mechanisms provided by MetabolicOS allow researchers and biohackers to safely experiment and build innovative systems to overcome T1D. Our long-term hope with this research is to turn a T1D diagnosis from a death sentence into an indicator of longevity, where people living with T1D will be expected to live longer then their healthy peers.

References

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

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

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] Muhammad Ibrahim, Aleix Beneyto, Ivan Contreras, and Josep Vehi. An ensemble machine learning approach for the detection of unannounced meals to enhance postprandial glucose control. *Computers in Biology and Medicine*, page 108154, 2024.

[12] 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.

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

[14] Zeinab Mahmoudi, Faye Cameron, Niels Kjølstad Poulsen, Henrik Madsen, B. Wayne Bequette, and John Bagterp Jørgensen. Sensor-based detection and estimation of meal carbohydrates for people with diabetes. *Biomedical Signal Processing and Control*, 48:12–25, 2019.

[15] Chiara Dalla Man, Federico Micheletto, Der Lv, Marc Breton, Boris Kovatchev, and Claudio Cobelli. The uva/padova type 1 diabetes simulator: New features. *Journal of diabetes science and technology*, 8(1):26–34, 2014.

[16] 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.

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

[18] 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.

[19] 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.

[20] Goran Petrovski, Judith Campbell, Maheen Pasha, Emma Day, Khalid Hussain, Amel Khalifa, and Tim van den Heuvel. Simplified meal announcement versus precise carbohydrate counting in adolescents with type 1 diabetes using the minimed 780g advanced hybrid closed loop system: a randomized controlled trial comparing glucose control. *Diabetes Care*, 46(3):544–550, 2023.

[21] 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.

[22] 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.

[23] Sediqeh Samadi, Kamuran Turksoy, Iman Hajizadeh, Jianyuan Feng, Mert Sevil, and Ali Cinar. Meal detection and carbohydrate estimation using continuous glucose sensor data. *IEEE Journal of Biomedical and Health Informatics*, 21(3):619–627, 2017.

[24] 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.

[25] Kamuran Turksoy, Sediqeh Samadi, Jianyuan Feng, Elizabeth Littlejohn, Laurie Quinn, and Ali Cinar. Meal detection in patients with type 1 diabetes: a new module for the multivariable adaptive artificial pancreas control system. *IEEE journal of biomedical and health informatics*, 20(1):47–54, 2015.

[26] 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.

[27] Ashenafi Zebene Woldaregay, Eirik Årsand, Ståle Walderhaug, David Albers, Lena Mamykina, Taxiarchis Botsis, and Gunnar Hartvigsen. Data-driven modeling and prediction of blood glucose dynamics: Machine learning applications in type 1 diabetes. *Artificial intelligence in medicine*, 98:109–134, 2019.

[28] Jinyu Xie and Qian Wang. Benchmarking machine learning algorithms on blood glucose prediction for type i diabetes in comparison with classical time-series models. *IEEE Transactions on Biomedical Engineering*, 67(11):3101–3124, 2020.

[29] 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.