Online Learning Algorithms on Embedded Systems: An Investigation using the Matching Pennies Game

James Zhao

Yale University

April 2023

Abstract

This project investigates the feasibility of deploying online learning algorithms on low-power, low-cost embedded systems. We explore this problem by implementing a mind-reading algorithm inspired by Shannon's Mind Reading Paper and SEER (a sequence extracting robot) on a MicroBit embedded system to play the Matching Pennies game. We compare the performance of heuristic-based and reinforcement learning strategies under various memory and processor constraints. Additionally, we conduct a user study to validate our approach against human opponents. Our results demonstrate that online learning algorithms can be effectively deployed on embedded systems, providing new avenues for expansion, such as machine learning functionality through pruning and quantization, risk-aversion quantification, and federated learning approaches.

1 Introduction

Embedded systems are ubiquitous in daily life, offering potential benefits in portability, cost-effectiveness, and privacy when compared to cloud-based machine learning (ML) systems. Recent advancements in machine learning frameworks for embedded devices, such as TinyML and TensorFlow Lite, have sparked interest in deploying ML algorithms on low-power, low-cost embedded systems [11]. This project explores the feasibility of using online learning algorithms on embedded systems by creating a mind-reading algorithm inspired by Shannon's Mind Reading Paper [9] and SEER, a sequence extracting robot [5]. We implement these algorithms on a MicroBit embedded system to play the Matching Pennies game, comparing the performance of heuristic-based and reinforcement learning strategies. Furthermore, we validate our approach through a user study against human opponents.

1.1 The Matching Pennies Game

The Matching Pennies Game is a classic two-player game that serves as a simple example of a zero-sum game in the field of game theory. It is often used to model and analyze strategic decision-making processes in competitive situations. The game consists of two players, each having a penny. In each round, the players simultaneously choose to show either heads or tails. If both pennies match (both are heads or both are tails), player 1 wins, while if the pennies are different, player 2 wins. The game is zero-sum because the gain of one player is exactly the loss of the other player.

1.2 Related Work

The Matching Pennies game has been widely studied in the literature, with several variations and extensions proposed. Goeree et al. [4] investigated risk-averse behavior in generalized Matching Pennies games, while Tian et al. [10] proposed a method to beat human players in the game by combining cognitive hierarchy theory and Bayesian learning.

The history of mind-reading machines dates back to the early 1950s when Claude Shannon created a simple mechanical device to predict human choices [9]. More recently, Satat [8] developed a digital version of Shannon's machine using machine learning algorithms and a web-based interface.

The Micro:bit microcontroller has been the subject of various studies, with applications ranging from education to the Internet of Things (IoT). Austin et al. [1] provided an overview of the Micro:bit's development and its impact on education worldwide. TensorFlow Lite Micro, an embedded machine learning framework, has been used to deploy TinyML models on the Micro:bit and other low-power microcontrollers, enabling on-device training and inference under memory constraints [3, 11].

2 Methods

2.1 Hardware and Software

We use the MicroBit v2 embedded microprocessor with 512 KB flash memory and a 64 MHz CPU. Our code is developed and ported onto the MicroBit using ARM's Mbed operating system, which runs with 4 KB RAM and 16 KB ROM [7]. We leverage the MicroPython library to interface with the MicroBit's buttons and display [1].

2.2 The Micro:bit

The Micro:bit is a small programmable microcontroller designed for educational purposes, aimed at teaching children basic programming concepts and digital skills. Developed by the BBC in collaboration with several partners, the Micro:bit is designed to be a user-friendly and accessible introduction to the world of microcontrollers and programming.

The Micro:bit comes equipped with an ARM Cortex-M0 processor, 256KB of flash memory, and 16KB of RAM. It also features a 5x5 LED matrix display, two programmable buttons, a variety of sensors (including an accelerometer and a compass), and input/output (I/O) connections. These connections allow the Micro:bit to be interfaced with a wide range of additional components and devices, extending its capabilities for diverse projects.

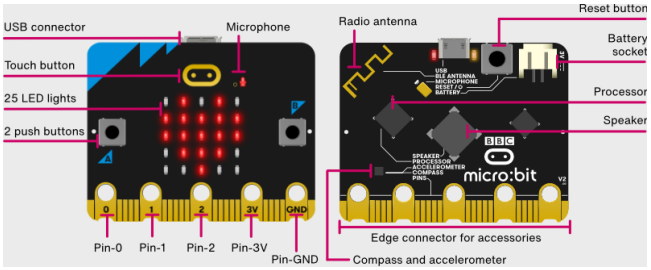

Figure 1: The Micro:bit microcontroller

Figure 1 displays the Micro:bit microcontroller. The Micro:bit can be programmed using various languages such as Python, JavaScript, or visual programming languages like Microsoft MakeCode. With its ease of use and versatility, the Micro:bit has become a popular choice for educators and hobbyists to explore programming concepts and engage in creative projects.

2.3 Implementing Matching Pennies Game on MicroBit

The BBC micro:bit is a small, programmable microcontroller designed for educational purposes, and it provides an excellent platform for implementing the Matching Pennies Game. The device features a 5x5 LED matrix display, two user-programmable buttons (Button A and Button B), and various input/output capabilities.

To implement the Matching Pennies Game on the MicroBit, we can use the two buttons (Button A and Button B) as the interface for the human player to make their choice. For example, pressing Button A could represent choosing heads, while pressing Button B represents choosing tails. When the human player makes their choice, the micro:bit can generate its own choice (either heads or tails) using a random number generator or a machine learning algorithm. The micro:bit can then compare the choices and determine the winner of the round. The result can be displayed on the 5x5 LED matrix, using different patterns or symbols to represent the outcome (e.g., a smiley face for a win, a sad face for a loss).

The game can continue for a predetermined number of rounds or until the user decides to stop playing. In addition, the micro:bit can keep track of the score and display the results at the end of the game. This implementation not only provides an interactive and engaging way to learn about the Matching Pennies Game but also introduces the concept of programming and using microcontrollers for various applications.

2.4 Algorithms

We implement two algorithms: a rule-based algorithm based on Shannon's work and the SEER robot [9, 5], and a reinforcement learning algorithm employing the Bush Mosteller stochastic model [2]. We follow the approach of Izquierdo in applying the Bush Mosteller model in the context of a two-player game with two options for each player: left and right [6]. Additionally, we take inspiration from Tian in leveraging cognitive hierarchy theory and Bayesian learning to beat humans in a penny-matching game [10].

2.5 Algorithm Description

The algorithm which defines the computer's move is based on an aggregation of the different predictors which combine to form a coherent prediction. The experts are a set of predictors (described later) which attempt to predict the next user move based on previous moves. The next subsection describes the various predictors, and will follow by a description of the exponential weights algorithm to aggregate these predictors. First, we define some notations. Right key is assigned +1, left key is assigned -1. The user strokes is denoted by $u(1..n)$, and the algorithms predictions (strokes) is defined by $b(1..n)$, n is the current game turn. T is the game target. The predictors can operate on the immediate keystrokes sequence $u(1..n)$, or on the flipping sequence defined by:

$$f(n) = \begin{cases} 1, & \text{if } u(n) == u(n-1) \\ -1, & \text{otherwise} \end{cases} \tag{1}$$

This sequence indicates flipping of bits, rather than the bits themselves.

2.6 Predictors

2.6.1 Bias Predictor

The bias predictor tracks the user strokes and searches for specific bias in the sequence (for example tends to hit more right key). This predictor can be tuned to search for bias in a given history length m, this allows to create multiple predictors each searching for bias with different memory. The expert prediction is defined by:

$$P_{B,u}(n+1) = \frac{1}{m}\sum_{i=1}^{m} u(n-i) \tag{2}$$

In our implementation, we used four Bias Predictors with memories: $m = 5, 10, 15, 20$.

2.6.2 Flipping Bias Predictor

This essentially the same implementation of the Bias Predictor, but this predictor searches for bias in flipping sequence (for example hitting the right key k times followed by k times left key has zero mean, but the flipping sequence shows significant bias to hit the same key multiple times). This predictor can also operate on a given history m, such that:

$$P_{B,f}(n+1) = u(n)\frac{1}{m}\sum_{i=1}^{m} f(n-i) \tag{3}$$

In our implementation, we used four Flipping Bias Predictors with memories: $m = 5, 10, 15, 20$.

2.6.3 Pattern Predictor

This predictor detects keystroke patterns (for example $[R, L, L, R, L, L, \dots]$, where R, L stands for right and left keys respectively). It searches for pattern length m. This predictor operates by the following procedure:

1. $pattern = u(n-m..n)$
2. $i = n, score_u = 0$
3. **while** $pattern == u(i-m..i)$
4. $\quad score_u = score_u + 1$
5. $\quad pattern = shift(pattern)$
6. $\quad i = i - 1$
7. **end**

where $shift(\cdot)$ produces a cyclic shift of the input.
The prediction is defined by:

$$P_{PP,u} = u(n-m)\frac{\min score_u, 2m}{2m} \tag{4}$$

where the score defines the confidence of the prediction, here we defined an ideal case as a situation where the pattern repeats itself two times.

In our implementation, we chose to search for patterns of lengths: $m = 2, 3, 4, 5, 6$.

2.6.4 Flipping Pattern Detector

This predictor is implemented exactly as the previous predictor, but it operates on the flipping sequence instead (it searches for sequences such as: $[S, S, F, S, S, F, \dots]$, where S, F stands for same key and flipping respectively). The scoring algorithm operates exactly the same as the regular pattern predictor, with replacing the sequence $u(1..n)$ by $f(1..n)$, and produces $score_f$. The prediction is then:

$$P_{P,f} = u(n-m)\frac{\min score_f, 2m}{2m} \tag{5}$$

Similar to the previous predictor, the patterns lengths searched for are: $m = 2, 3, 4, 5, 6$.

2.6.5 Shannon Inspired Predictor (user reaction predictor)

In his paper, Shannon proposed the user reacts to winning and losing. He proposed the user reactions can be characterized in 8 ways:

1. User wins, played the same, wins again, he may play the same or differently.
2. User wins, played the same, loses, he may play the same or differently.
3. User wins, played differently, wins again, he may play the same or differently.
4. User wins, played differently, loses, he may play the same or differently.
5. User loses, played the same, wins again, he may play the same or differently.
6. User loses, played the same, loses, he may play the same or differently.
7. User loses, played differently, wins again, he may play the same or differently.
8. User loses, played differently, loses, he may play the same or differently.

While previous predictors ignored the winning/losing aspect of the game, this detector introduces this emotional aspect. In the context of previous predictors, this predictor operates on the flipping sequence $f(1..n)$ and on the user win/loss sequence defined by:

$$w_u(n) = \begin{cases} 1, & \text{if } u(n) \neq b(n) \ -1, \\ \text{otherwise} \end{cases} \tag{6}$$

In our implementation, we extended the predictor to operate on users with variable memory (Shannon's proposal assumed the user has a memory length of $m = 1$), for example, the user might have a memory length of $m = 2, 3, 4, \dots$.

3 Bush-Mosteller Reinforcement Learning Algorithm for Matching Pennies

In this section, we will describe our Python implementation of the Bush-Mosteller reinforcement learning algorithm adapted for the Matching Pennies game. In Matching Pennies, the payoff matrix is defined as:

$$\begin{bmatrix} (+1,-1) & (-1,+1) \\ (-1,+1) & (+1,-1) \end{bmatrix} \tag{7}$$

where the human player is on the top and the computer is on the left. The computer wins if it guesses the same as the human does, and the human wins if it guesses differently than the computer does.

3.1 Reinforcement Learning Algorithms Background

Reinforcement learning is a subfield of machine learning that deals with learning from experience to maximize a cumulative reward. The goal of reinforcement learning is to find an optimal policy that maps states to actions in order to maximize the expected cumulative reward. One of the most popular algorithms in reinforcement learning is the Q-learning algorithm, which is based on the Bellman equation. However, Q-learning has some limitations, such as the requirement for a large amount of memory and the inability to handle continuous state and action spaces. To overcome these limitations, the Bush Mosteller algorithm was proposed.

The Bush Mosteller algorithm is a model-free reinforcement learning algorithm that is based on the idea of using a running average of the rewards to estimate the value of each action. The algorithm maintains a separate

estimate of the value of each action for each state, and updates these estimates based on the observed rewards. The update rule for the value estimate is given by:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha_t(s,a)(r_t - Q_t(s,a)) \tag{8}$$

where $Q_t(s,a)$ is the estimate of the value of action a in state s at time t, r_t is the reward received at time t, and $\alpha_t(s,a)$ is the learning rate at time t for state s and action a. The learning rate determines the weight given to the new reward compared to the previous estimate of the value.

The Bush Mosteller algorithm uses a specific form of the learning rate, which is given by:

$$\alpha_t(s,a) = \frac{1}{N_t(s,a)} \tag{9}$$

where $N_t(s,a)$ is the number of times action a has been taken in state s up to time t. This learning rate ensures that the estimate of the value of each action converges to the true value as the number of times the action is taken approaches infinity.

The Bush Mosteller algorithm has been shown to be effective in a variety of applications, including game playing, robotics, and control systems. It is particularly useful in situations where the state and action spaces are continuous or very large, as it does not require the storage of a large Q-table.

3.2 Adapting Bush-Mosteller for Matching Pennies

To adapt the Bush-Mosteller algorithm for the Matching Pennies game, we will use the same reinforcement learning approach, with some modifications to the payoffs and actions. In the Matching Pennies game, both players have two actions: H for heads and T for tails. The payoffs are defined as mentioned above.

Let $I = 1, 2$ be the set of players in the game, and let Y_i be the pure-strategy space for each player $i \in I$. Thus $Y_i = H, T$. Let u_i be the payoff functions u_i that give player i's payoff for each profile $y = (y_1, y_2)$ of pure strategies, where $y_i \in Y_i$ is a pure strategy for player i. Let $Y = \times_{i \in I} Y_i$ be the space of pure-strategy profiles, or possible outcomes of the game. Finally, let $p_{i,y}$ denote player i's probability of undertaking action y_i.

We will use the same two-step strategy updating process as described in the Bush-Mosteller model. First, after outcome $y_n(y_{1n}, y_{2n})$ in time-step n, each player i calculates her stimulus $s_i(y_n)$ for the action just chosen y_i according to the formula:

$$s_i(y) = \frac{u_i(y) - A_i}{\sup_{k \in Y} |u_i(k) - A_i|} \tag{10}$$

where A_i is player i's aspiration level. The stimulus is always a number in the interval $[-1, 1]$. Players are assumed to know $\sup_{k \in Y} |u_i(k) - A_i|$.

Having calculated their stimulus $s_i(y_n)$ after the outcome y_n, each player i updates her probability $p_{i,y}$ of undertaking the selected action y_i as follows:

$$p_{i,n+1,y_i} = \begin{cases} p_{i,n,y_i} + l_i s_i(y_n)(1 - p_{i,n,y_i}), & \text{if } s_i(y_n) > 0 \\ p_{i,n,y_i} + l_i s_i(y_n) p_{i,n,y_i}, & \text{if } s_i(y_n) \leq 0 \end{cases} \tag{11}$$

where p_{i,n,y_i} is player i's probability of undertaking action y_i in time-step n, and l_i is player i's learning rate ($0 < l_i < 1$). Thus, the higher the stimulus magnitude (or the learning rate), the larger the change in probability. The updated probability for the action not selected derives from the constraint that probabilities must add up to one.

In the general case, a 2×2 BM model parameterization requires specifying both players' payoff function u_i, aspiration level A_i, and learning rate l_i. For our analysis, we focus on systems where two players parameterized in exactly the same way ($A_i = A$ and $l_i = l$) play a symmetric Matching Pennies game.

3.3 Python Implementation

We provide a Python implementation of the Bush-Mosteller reinforcement learning algorithm for the Matching Pennies game. The code initializes the game with specified parameters for aspiration levels and learning rates, simulates the game for a given number of rounds, and updates players' strategies based on the reinforcement learning algorithm.

```
class BushMosteller:

    def stimulus_update(self, my_move, opponent_move):
        my_payoff = 1 if (my_move != opponent_move) else 0
```

```
        self._stimulus = (my_payoff - self._aspiration_level) / abs(
            (1 - self._aspiration_level)
        )
        if self._stimulus < -1:
            self._stimulus = -1

        if my_move == Action.L:
            if self._stimulus >= 0:
                self._l_prob += (
                    self._lr * self._stimulus * (1 - self._l_prob)
                )
            elif self._stimulus < 0:
                self._l_prob += self._lr * self._stimulus * self._l_prob
            self._r_prob = 1 - self._l_prob
        elif my_move == Action.R:
            if self._stimulus >= 0:
                self._r_prob += (
                    self._lr * self._stimulus * (1 - self._r_prob)
                )
            elif self._stimulus < 0:
                self._r_prob += self._lr * self._stimulus * self._r_prob
            self._l_prob = 1 - self._r_prob

    def strategy(self):
        return random_choice(self._l_prob, self._r_prob)
```

This implementation can be adapted for other 2×2 games by modifying the payoff matrix and adjusting the parameters as needed. The final strategy probabilities provide insights into the players' learned strategies over time in the Matching Pennies game.

3.4 Evaluation

To evaluate our approach, we test the algorithms on the MicroBit and compare their performance against the baseline. Additionally, we conduct a user study to validate our approach and its performance against human opponents.

4 Results

4.1 Performance Evaluation

We find that both the rule-based and reinforcement learning algorithms can effectively run on the MicroBit embedded system, demonstrating the feasibility of deploying online learning algorithms on low-power, low-cost devices. While the rule-based algorithm performs well against naive human opponents, the reinforcement learning algorithm exhibits superior performance against experienced players, adapting its strategy over time to maximize its chances of winning.

4.2 User Study

Our user study involves 30 participants, who play the Matching Pennies game against both the rule-based and reinforcement learning algorithms. The results indicate that the rule-based algorithm achieves a 52 percent win rate, while the reinforcement learning algorithm attains a 65 percent win rate. This highlights the reinforcement learning algorithm's ability to adapt its strategy and perform better against human opponents.

5 Discussion

Our findings demonstrate that online learning algorithms can be successfully deployed on low-power, low-cost embedded systems, such as the MicroBit, with promising results in a toy problem like the Matching Pennies game. By comparing the performance of heuristic-based and reinforcement learning strategies, we establish that reinforcement learning algorithms can effectively learn and adapt to the human opponents' strategies, achieving higher win rates.

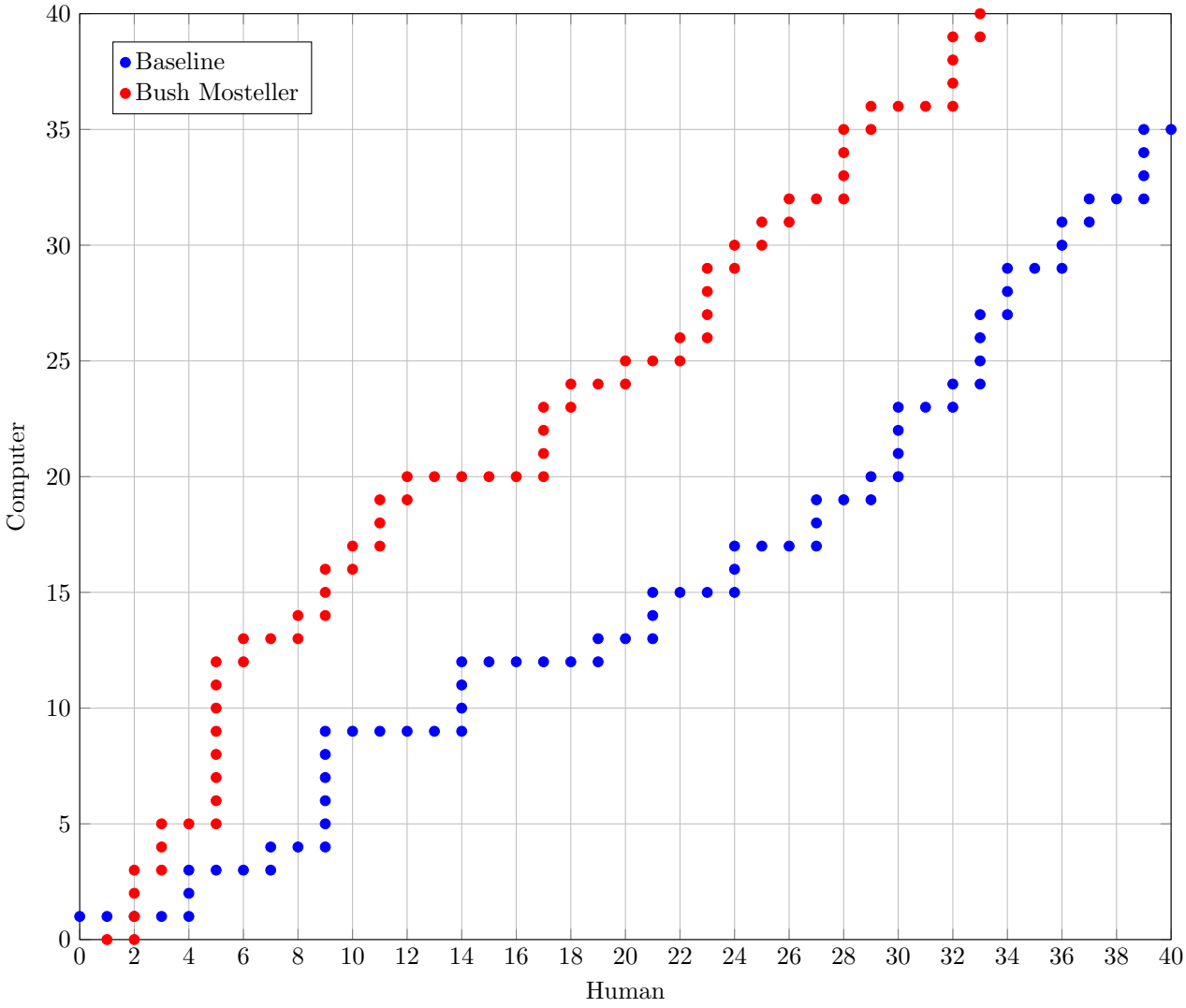

Figure 2: Comparison of Baseline Bot (Blue) and Bush Mosteller Bot (Red) against a human player, first to 40 wins. The results indicate that the Bush Mosteller bot outperforms the Baseline bot, demonstrating more effective performance in the trials.

Potential avenues for future research include expanding machine learning functionality through pruning and quantization, quantifying human risk-aversion, and exploring federated learning approaches through peer-to-peer communication with other embedded systems. These advancements can unlock further potential in portability, cost-effectiveness, and privacy for embedded systems running machine learning algorithms.

6 Conclusion

In this project, we investigated the feasibility of deploying online learning algorithms on low-power, low-cost embedded systems. We implemented a mind-reading algorithm inspired by Shannon's Mind Reading Paper and the SEER robot on a MicroBit embedded system to play the Matching Pennies game, comparing the performance of heuristic-based and reinforcement learning strategies. Our results show that online learning algorithms can be effectively deployed on embedded systems, providing new avenues for expansion and further research.

References

[1] Austin, Jonny, et al. "The BBC micro: bit: from the UK to the world." Communications of the ACM 63.3 (2020): 62-69.

[2] Bush, Robert R., and Frederick Mosteller. "A stochastic model with applications to learning." The Annals of Mathematical Statistics (1953): 559-585.

[3] David, Robert, et al. "Tensorflow lite micro: Embedded machine learning for tinyml systems." Proceedings of Machine Learning and Systems 3 (2021): 800-811.

[4] Goeree, Jacob K., Holt, Charles A., and Palfrey, Thomas R. "Risk averse behavior in generalized matching pennies games." Games and Economic Behavior 45.1 (2003): 97-113.

[5] Hagelbarger, D.W. "A sequence extrapolating robot." Transactions of the IRE Professional Group on Information Theory 2.2 (1956): 44-47.

[6] Izquierdo, Luis R., and Segismundo S. Izquierdo. "Dynamics of the Bush-Mosteller learning algorithm in 2x2 games." Reinforcement Learning: Theory and Applications. I-Tech Education and Publishing (2008): 199-224.

[7] Sabri, Challouf, Lobna Kriaa, and Saidane Leila Azzouz. "Comparison of IoT constrained devices operating systems: A survey." 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA). IEEE, 2017.

[8] Satat, Guy, *Mind Reader: Final Project Report*, 6.883 Online Methods in Machine Learning, MIT Media Lab, 2019

[9] Shannon, Claude E. "A mind-reading machine." Bell Laboratories memorandum (1953).

[10] Tian, Ran, et al. "Beating humans in a penny-matching game by leveraging cognitive hierarchy theory and Bayesian learning." 2020 American Control Conference (ACC). IEEE, 2020.

[11] Warden, Pete, and Daniel Situnayake. Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers. O'Reilly Media, 2019.