Improving App Launch Time with Deep Learning

Alan Zhang **Yikai Mao** **Mandy Lee**
UC San Diego, USA
{xuz017, yim003, mal001}@ucsd.edu

Abstract

Application launch time is a crucial element of the user experience. Long wait times can cause frustration and prompt users to upgrade to more powerful machines, resulting in increased electronic waste (e-waste) at landfills. Improving app launch time is vital in reducing e-waste by extending the average lifespan of computers. While software has become more resource efficient, it is still challenging to prevent large programs from being bloated and slow to run. In this paper, we propose utilizing neural networks to analyze system usage reports and pre-launch applications in the background before the user needs them. This approach can be universally applied to all computers, making it more economically viable than asking all developers to optimize their applications. We developed our data collection software to minimize resource usage and to identify applications that the system can pre-launch using Hidden Markov Model (HMM) and Long Short-Term Memory (LSTM) models.

1 Introduction

The Windows operating system is built to be compatible with a diverse range of hardware specifications. A study conducted by Intel revealed that many machines, including brand new laptops performed unexpectedly poor on application launch times. For instance, a brand new computer takes an average of 11 seconds to launch Google Chrome. Our speculation is that the study was conducted globally and many new laptops are still being shipped with slower hardware. In addition, developers have no incentive to improve performance as most companies strive to move fast and break things than to release the perfect application. This leads to longer app launch times on older or slower computers, which harms the overall user experience and hampers productivity. Our objective is to make devices feel more responsive through a data-driven solution.

Long app launch time could be attributed to a variety of factors such as the performance of the architecture the application is built on, the processing speed of the hardware, and poor coding practices that results in unoptimized code. Our solution to this issue is to pre-emptively launch applications before the user needs them. To achieve this goal, we created a software that collects system usage data. Two deep learning models are used to analyze user habits and trends, the Hidden Markov Model (HMM) and Long Short-Term Memory (LSTM) model. Hidden Markov Model is used to predict the sequence of applications to be used by the user and Long Short-Term Memory model will be used to predict application usage. The application with the highest predicted usage and frequency will be marked as candidates suitable for pre-launching. Our implementation and results are detailed below.

2 Methods

2.1 Data Collection

To ensure that other people can replicate the methods discussed in our method, we created our own software to collect the system usage data. This way, we can be as granular as we want in the collection process to capture more data or be more coarse to reduce file size. We build this data collector with the help of Intel's X Library Software Development Kit (XLSDK). The XLSDK package the data generated from our code and stores it in a database. To predict the sequences of applications used on a given day and their respective timeframe, we wrote code to capture the foreground window. The collection process is activated on two conditions.

The first condition is triggered when a user clicks on the screen. We accomplish this by creating a pure-event driven thread that monitors for mouse clicks. When we detect a mouse click, the software locks the mouse for a tenth of a millisecond by acquiring the keys to the critical section. It then captures the x and y coordinate of the mouse click and retrieves the handle to the application positioned at that coordinate through the use of Win32API. Sometimes, the foreground window changes without a mouse click such as alt-tabbing between windows. To address this concern, we have a second thread that is triggered at a set interval (i.e. every 10 seconds) to check if the current foreground window is the same. If the current window has a different process ID than the previous window, then we log the application into our database.

	MEASUREMENT_TIME	ID_INPUT	VALUE
1	2022-12-02 16:29:03.779	2	foreground_window-000010.db
2	2022-12-02 16:29:03.779	3	DB Browser for SQLite.exe
3	2022-12-02 16:29:03.779	4	Qt5QWindowIcon
4	2022-12-02 16:29:03.779	5	\DISPLAY1
5	2022-12-02 16:29:07.185	2	foreground_window-000012.db
6	2022-12-02 16:29:07.185	3	DB Browser for SQLite.exe
7	2022-12-02 16:29:07.185	4	Qt5QWindowIcon
8	2022-12-02 16:29:07.185	5	\DISPLAY1
9	2022-12-02 16:29:13.483	2	(58) Tesla Semi Delivery Event - YouTube — Mozilla Firefox
10	2022-12-02 16:29:13.483	3	firefox.exe
11	2022-12-02 16:29:13.483	4	MozillaWindowClass
12	2022-12-02 16:29:13.483	5	\DISPLAY1
13	2022-12-02 16:29:14.089	2	Friends - Discord
14	2022-12-02 16:29:14.089	3	Discord.exe
15	2022-12-02 16:29:14.089	4	Chrome_WidgetWin_1

Fig 1. Data Result Sample

After logging the essential information regarding the new foreground window, we send a signal to the desktop mapper input library to capture a snapshot of all the visible windows. This can help us remember the location of each window and launch the application in the right position and size. To see an example of the raw data we have collected, see **Fig 1**.

2.1.1 Data Collection Principles

Once we have developed the minimal viable product to collect the data, we made modifications to the program so that it can be deployed on a wide range of devices. We accomplish this goal by adhering to the following principles.

1. **Robustness and Resilience**
   To ensure that our program continues to run without requiring human intervention in the event of errors during deployment, we have implemented defensive coding practices. We have verified the data type and range of variables before feeding them into functions. If an error occurs, we log the error type, the file that generated the error, the line number within the file, and the timestamp. This enables us to identify faulty

code while reviewing error logs. Through this approach and rigorous testing, we were able to keep our collector running without errors for eight weeks.

2. **Privacy Compliance**
   To obtain the name of the foreground window application, we must locate the application's file path, which may contain Personal Identifiable Information (PII) such as a person's full legal name. To prevent the collection of PII, we have implemented a process to remove any PII from the file path before storing the collected information. For example, if a user has named their system after their legal name, we exclude the full file path to prevent the collection of PII.
3. **Efficiency**
   We have optimized the code to reduce the impact on system resources, such as CPU and memory usage, by minimizing unnecessary processing and data transfer. This is crucial as we want the application to run continuously in the background while the computer is on. To achieve this, we only allocate the minimum necessary memory to arrays and expand them as needed, ensuring efficient use of system resources.
4. **Compatibility**
   We have designed the data collector to be compatible with a wide range of languages by changing our collected strings from ANSI to UNICODE to capture foreign characters. This enables us to accurately capture and store data in different languages, ensuring compatibility with various software applications and operating systems.

2.1.2 Automation

To ensure the quality of our training dataset, we aim to collect continuous data whenever possible. To achieve this, we have created a task in the Windows 10 task scheduler to automatically run our data collection software whenever a user logs into their computer. The software is designed to run in the background to minimize any potential disruptions to the user's workflow. Additionally, this approach prevents accidental closure of the program, which would lead to incomplete data. As a result of our diligent approach over the first ten weeks of this study, we have successfully collected eight weeks of usage data.

2.2 Exploratory Data Analysis

We have collected over 8 weeks of data and **Fig 2** EDA result shows the top 10 processes and their corresponding total duration the user used.

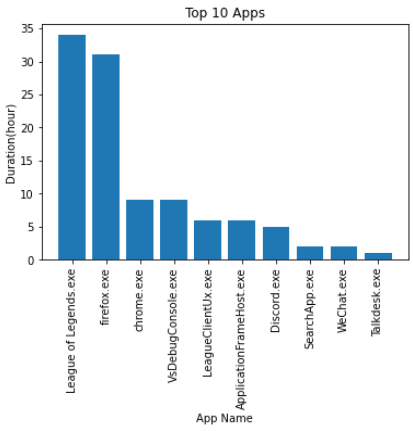

Fig 2. Top 10 Apps by Usage (measured in hours)

To create an effective Long Short-Term Memory (LSTM) model, we limited our case study to

analyzing the usage of the most frequently used app. League of Legends and FireFox were considered as the top contenders. However, we selected FireFox as our subject of study as it demonstrates more consistency in usage over time compared to League of Legends (**Fig 3** & **Fig 4**), which experiences a spike in usage during winter break but a sharp decline after the academic quarter begins. The time series chart of FireFox usage shows stable amplitude, which makes it an ideal candidate for the LSTM model. Our objective is for the model to understand these usage patterns and make accurate predictions, so we decided to focus on FireFox for our second task.

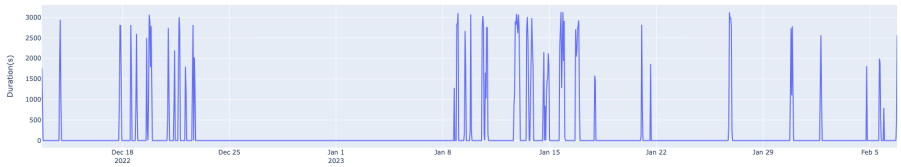

Fig 3. Usage over time for League of Legend

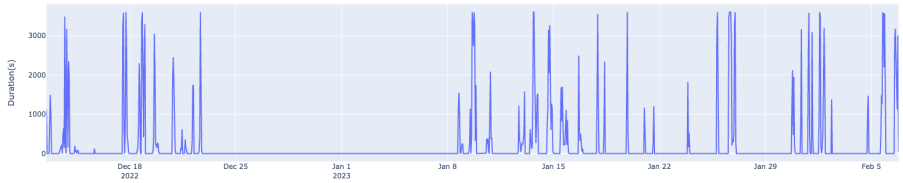

Fig 4. Usage over time for FireFox

2.3 Tasks, Models, and Evaluation Metrics

We tackled two forecasting objectives using the data we gathered over several weeks. Our first objective is to predict the subsequent application an individual will use, based on their past usage patterns. Then, our second task, which is also our primary objective, is to forecast the amount of time (in seconds) an individual will spend on a specific application within a specific hour.

2.3.1 Task 1: Next-App Prediction with Hidden Markov Model

In task one, our goal is to predict the next application the user will use based on the previous usage data. Hidden Markov Models (HMMs) are a type of machine learning predictive model used for modeling sequential data. In an HMM, the underlying system is assumed to be a Markov process, which means that the probability of the system being in a certain state at time t depends only on the state of the system at time t-1. In our scenario, we just need to tally up the number of times an application (e.g., Zoom) is used after Google Chrome has been used. This can help us generate a list of applications that are commonly used following the use of a specific application. Therefore, we can create a transition matrix, which is the probability of moving from one state to another for every application in the dataset. They specify the probability that the underlying system will transition from one state to another in the next time step. Just like in **Fig 5**, the number between the apps is the probability of one app to another and these are calculated with previous activity.

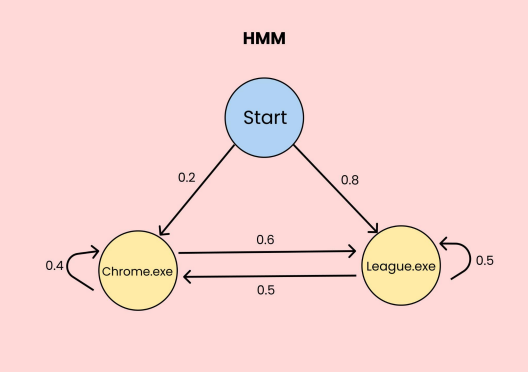

Fig 5. Simple Illustration of HMM

To assess the performance of our HMM Model, we used 20% of our dataset as a testing dataset. We evaluated our model's accuracy by checking if the ground truth falls within the top N predictions generated by the model. In other words, if the correct answer is among the top n guesses, the model gets credit for predicting correctly.

2.3.2 Task 2: App Duration Prediction with Long Short-Term Memory

We focused on using FireFox as the primary application for predicting the duration of usage during a specific hour. LSTM (Long Short-Term Memory) is a type of Recurrent Neural Network (RNN) architecture that is commonly used in machine learning for processing sequential data, such as speech, text, and time-series data. We have decided to use Keras, a high-level neural network API for Python, to implement LSTMs as a layer in a neural network. In an LSTM network, the hidden state is updated at each time step by combining the values of the input, forget, and output gates, and the memory cells are updated accordingly. This process allows the LSTM to selectively store or forget information over a long period of time, making it well-suited for tasks such as speech recognition, natural language processing, and time-series prediction. Just like in **Fig 6**, features such as process names and dates are imported as inputs and multiple hidden layers are updated to output the next name/duration of the processes.

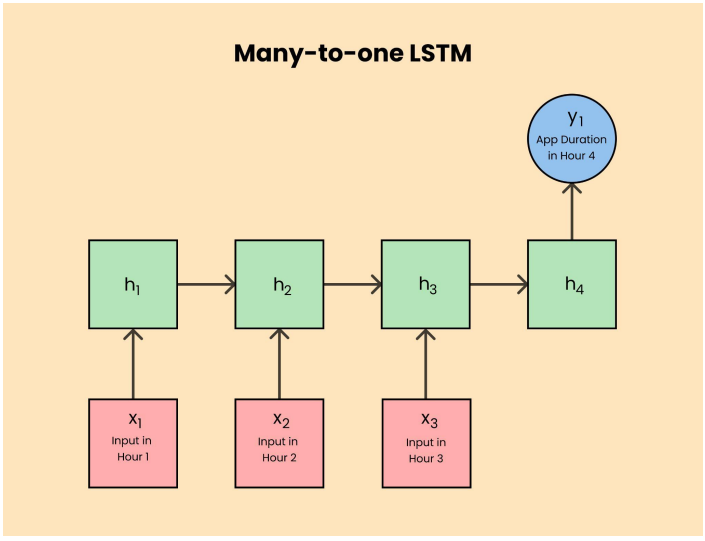

Fig 6. Simple Illustration of LSTM

We train LSTM with Mean Squared Error as the loss function. For a regression problem, MSE

allows the model to update its parameters through back-propagation due to its differentiability. In addition, we included another human-interpretable measurement, accuracy, to better understand our model's performance. This accuracy is based on whether the prediction and the target are within an arbitrary margin of error (we chose three thresholds, 5, 10, and 60 seconds). This is to give the model a bit of leeway for when it predicts the amplitude correctly but is off by a few seconds to a minute.

Furthermore, we removed periods from Dec. 23rd, 2022 to Jan. 7th, 2023 since the user was away from his PC for holidays, which could potentially bias our model's performance. The FireFox usage plot after the removal of aforementioned periods is shown in **Fig 7**.

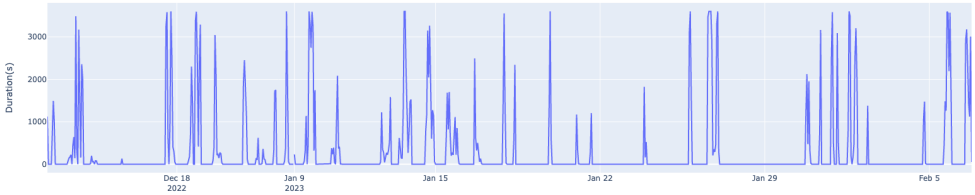

Fig 7. Usage over time for FireFox (with periods removed)

Then, we have divided logs that have a duration exceeding one hour into separate logs with durations corresponding to individual hours. This is to align with the specific requirements of our tasks, which ask for the duration within a specified hour **(Fig 8)**.

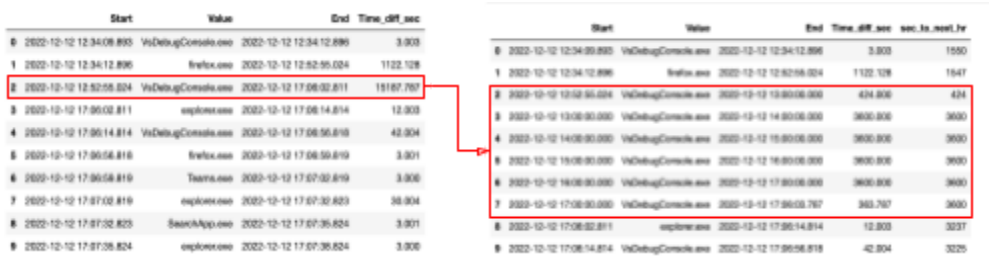

Fig 8. Split Up Logs Based on Duration

3 Results

3.1 Task 1 Results and Observations

In our analysis of the Hidden Markov Model, we evaluated the probability of applications being opened based on the current application being used. Our findings showed a strong correlation between the use of FireFox and League of Legends, Chrome, Discord, and Chrome and Talkdesk (**Fig 9**). This behavior aligns with my typical usage patterns where I use FireFox and League of Legends for leisure activities and Chrome and various productivity apps for work. This demonstrates that the Hidden Markov Model is able to effectively capture my usage habits and make accurate predictions.

Fig 9. Probability of the next application for the 10 most used apps

Based on the results shown in **Fig. 10**, we found that accuracy improved as N increased. This is because higher values of N allow for more tolerance for errors, resulting in increased accuracy. Eventually, we were able to achieve 75% accuracy with N=3 in predicting the next application by using HMM.

N	Accuracy
1	48%
2	65%
3	75%

Fig 10. HMM Top-N Accuracy Result

3.2 Task 2 Experiments, Results and Observations

For this task, we have experimented with the same model structure, but with several different modifications including feature engineering and activation function.

3.2.1 Experiment 1

In our initial experiment, we input numerical data such as day of the week, day of the month, hour, minute, date, and month, as well as binary data indicating whether it was a weekend or a winter holiday. We used an activation function that combined an LSTM with a Dense (linear) function. However, the results were unsatisfactory on both the training and testing sets, as we were only able to capture a few sporadic spikes (**Fig 11**).

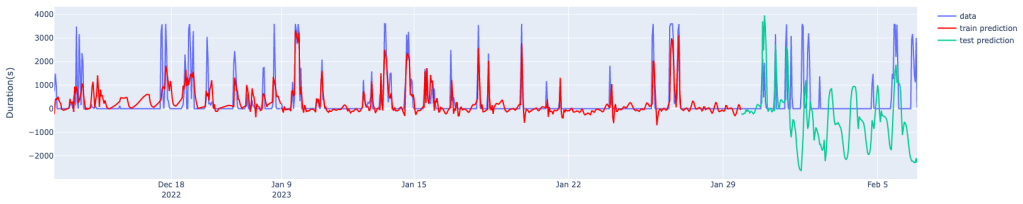

Fig 11. Experiment 1 Plot

3.2.2 Experiment 2

In our second experiment, we applied one-hot encoding to features that are better represented as categorical data, with the expectation that the model can update weights on one-hot-encoded features accordingly. This resulted in better performance compared to the previous experiment **(Fig. 12)**.

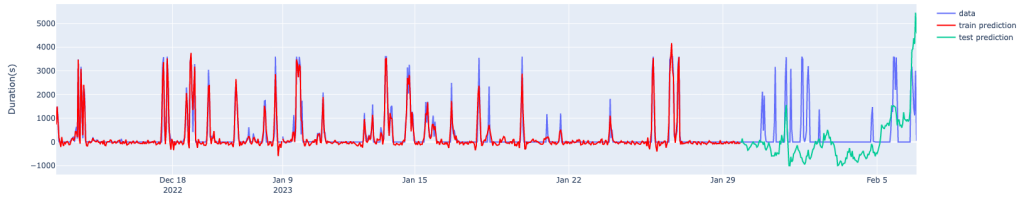

Fig 12. Experiment 2 Plot

3.2.3 Experiment 3

In our third experiment, we also noticed that these time-related data have cyclic nature, meaning that there are some relations between time that are close together. For example, events at 4 PM could be more related to events at 5 PM than events at midnight. Therefore, we applied sine and cosine transformations to the numerical input. While this approach did lead to a reduction in model size, the performance was slightly worse compared to the previous experiment (**Fig 13**).

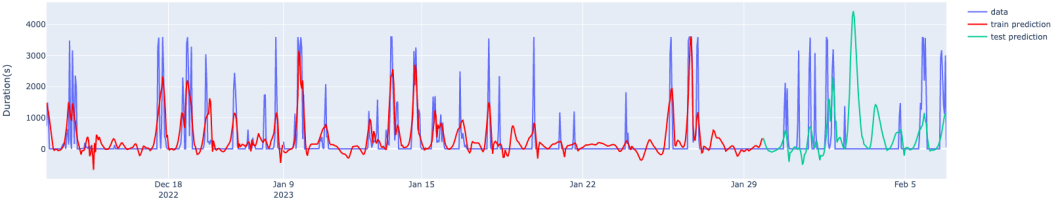

Fig 13. Experiment 3 Plot

3.2.4 Experiment 4

Lastly, we observed that the Dense (linear) function in our previous experiments produced some impossible predictions (values less than 0 and greater than 3600), so we switched to a Dense (sigmoid) function to address this issue. Additionally, based on our previous experiment, we reverted back to using one-hot encoding as it produced better predictions compared to sine and cosine transformations. As a result of these changes, we achieved good accuracy in both the train and test sets (**Fig14**).

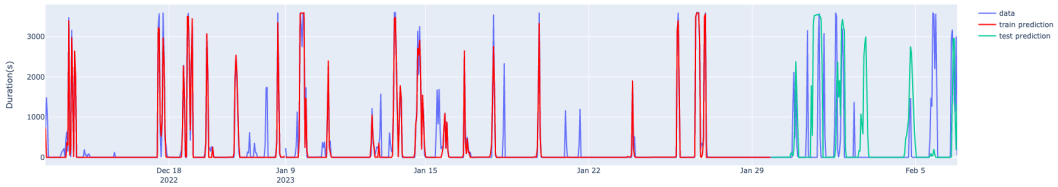

Fig 14. Experiment 4 Plot

3.2.5 Best Experiment and Observations

Fig 15 shows a summary of the experiments and experiment 4 gave us the best prediction with a mean square error loss of 0.138 and an accuracy of 82% (**Fig16**). Importantly, we noticed a tradeoff between accuracy and input size. Particularly, one-hot encoding increases the model size by almost three times larger than models in other experiments due to the large input required. Large input size will lead to the curse of dimensionality and the larger model size will cause higher time and memory complexity which would get worse as the dataset gets larger.

Experiment	Features	Activation Function	Observation
1	• **Numerical**: Day of Week, Day of Month, Day of Year, Hour, Minute, Date, Month • **Binary**: is Weekend, is Winter Holiday	• LSTM (tanh) • Dense (linear)	• Poor results on both train and test set • Only few spikes are caught
2	• **One-hot-encoded:** Day of Week, Day of Month, Day of Year, Hour, Minute, Date, Month • **Binary:** is weekend, is winter holiday	• LSTM (tanh) • Dense (linear)	• Better performance • BUT model size is larger due to input size (**30,145** params)
3	• **sin and cos transformed:** Day of Week, Day of Month, Day of Year, Hour, Minute, Date, Month • **Binary:** is weekend, is winter holiday	• LSTM (tanh) • Dense (linear)	• Performance is slightly worse but model is small (**12,481** params) • "Impossible" outputs (< 0 or > 3600)
4	• **One-hot-encoded:** Day of Week, Day of Month, Day of Year, Hour, Minute, Date, Month • **Binary:** is weekend, is winter holiday	• LSTM(tanh) • **Dense (Sigmoid)**	• Good accuracy in train and test set • High accuracy in the test set is not reflected in the plot (why?)

Fig 15. Experiment Trials and Results

Metric	Train Result	Test Result
MSE Loss	0.0038	0.1384
Accuracy within 5s	84%	79%
Accuracy within 10s	85%	80%
Accuracy within 60s	86%	82%

Fig 16. Experiment 4 Accuracy

In addition, when we compare the graph (**Fig 14**) of the model's performance to the table of results (**Fig 16**), we can see that the model does not perform well on the testing set. The graph shows that the model fails to capture most of the spikes, which is in contrast to the high test accuracy reported in the table. This discrepancy may be due to the fact that the model is correctly predicting many 0s, which suggests that the accuracy metric may not be a good evaluation tool.

To address this issue, we could adjust our evaluation metric to penalize incorrect amplitude predictions more severely while reducing rewards for correctly predicting 0s. This approach aims to address the class imbalance between active and inactive usage and encourage the model to focus on learning the amplitude of usage time more accurately.

Moreover, the graph suggests that the model is able to effectively learn from the training set, but struggles to forecast future patterns. We hypothesize that the poor performance on unseen data is due to the insufficient dataset, as there are features in the testing set, such as day of the year and month, which are not present in the training set and thus make the model difficult to generalize.

4 Conclusion and Discussion

We have developed software and models that serve as fundamental building blocks for We have developed software and models that serve as fundamental building blocks for constructing more complex and accurate models to identify suitable applications for pre-launch. By developing our own collector, we have gained insights into responsible data collection and good practices for acquiring and storing data. Our collector is memory-efficient and can run 24/7 without human intervention. If you wish to collect your own data and run it against our model, you can clone our GitHub repository[1] and add the script to your Task Scheduler. Detailed instructions are available in the repository.

Although our HMM model does not achieve the highest accuracy, it indicates that the model is generalizing well rather than simply memorizing and overfitting to the training data. Our LSTM model requires further work, and we suggest exploring modifications such as changing the metrics to one more suitable for this regression task, as the current accuracy measurement can be misleading. Moreover, the high accuracy of the model mostly results from correctly predicting zeros rather than timing and amplitude values. We can alter our test set to include only amplitudes or rebalance the amplitude and zero points to more effectively evaluate the model on an unseen dataset. Last but not least, we hope to collect additional data for the dataset to capture more consistent usage patterns.

We encourage those interested in the project to build upon what we have developed by following our GitHub repository. All instructions are available in the README.md file.

5 Acknowledgments

Jamel Tayeb and Bijan Arbab, Intel; Intel DCA Team

[1] Our github repository:
https://github.com/MikeM820/foreground_window_forcast.git