Discover User-App Interactions & Solutions to Reducing the Initial User-CPU Latency

Thy Nguyen Milon Chakkalakal Pranav Thaenraj

Advisors: Jamel Tayeb, Bijan Arbab, Sruti Sahani, Oumaima Makhlouk, Praveen Polasam, Chansik Im

Abstract

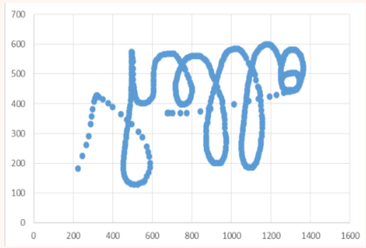
"Data loading" icons signal an unpleasant user-wait experience and can tear people away from using an app. This problem adversely affects the user experience, but limited research has been done, so we perform a study to collect user-app interaction data, analyze past behaviors to understand the user-wait events, and propose solutions to reduce such waiting times. Our study includes 2 phases: collecting data using Intel's XLSDK and analyzing data using EDA, HMM, and LSTM/RNN.

Methodology of Data Collection

We write the ***input libraries (ILs)*** using the software development kit ***XLSDK***, along with the ***Environment Server (ESRV)*** toolchain and the ***Intel® System Usage Report (SUR)*** framework to anonymously gather and analyze data usage from multiple devices. The ILs include:

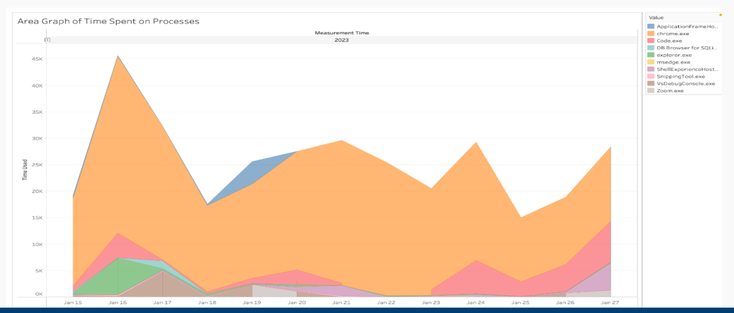
- **Mouse Input IL**: capture the (X, Y) positions of the mouse in pixels (+/- noise)
- **User Wait IL**: retrieve the cursor type (e.g. loading or working smoothly) and its timestamps
- **Mouse Hook IL**: track the UI objects clicked by the mouse
- **Foreground Window IL**: log the window's details detected by mouse clicks or time ticks
- **Desktop Mapper IL**: map all the open app windows in the z-order and store the relative position of an app on the screen as well as their individual size

We mainly focus on the data collected from the *Foreground Window IL* for further exploration.



Exploratory Data Analysis

Chrome is the top frequently used app of this user according to the time measurement in 01/2023



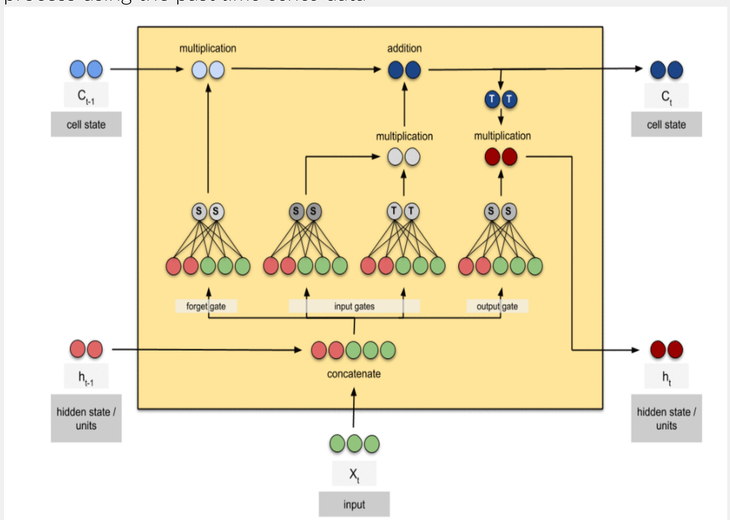
Methodology of Predictive Tasks

Hidden Markov Model (HMM)

- **Problem Statement**: Predict the likelihood of using an app *given* the former sequence of application usage
- **Basic Idea**: Utilize the conditional probability $P(A|B) = \frac{P(A \cap B)}{P(B)}$
- **HMM Assumptions**:
  1. **Markov Chain**: Only the *current* state plays the most crucial role in predicting the future in the sequence; other states before that will not influence the future states
  $$P(q_i = a|q_1 q_2 ... q_{i-1}) = P(q_i = a|q_{i-1})$$
  where q_k are the states for $k \in \{1, 2, ..., i\}$, e.g. a hidden state *chrome.exe*
  2. **Output Independence**: The probability of observing an event o_i only relies on the state q_i that *directly* produced o_i.
  $$P(o_i|q_1, ...q_i, ..., q_T, o_1, ..., o_i, ..., o_T) = P(o_i|q_i)$$
  where $o_1 o_2 ... o_T$ is a sequence of T observations and $q_1, q_2, ..., q_T$ are T states
- **Transition Matrix**: contains the transition probabilities, i.e. the likelihood of moving from one '.exe' to another '.exe'.
  $$P(\text{'chrome.exe' -> 'explorer.exe'}) = P(\text{'explorer.exe' | 'chrome.exe'})$$
- **Emission Matrix**: contains the emission probabilities, i.e. the likelihood of moving from one executable file to another app/tab.
  $$P(\text{'chrome.exe' -> 'Spotify'}) = P(\text{'Spotify' | 'chrome.exe'})$$
- **Metrics**: The accuracy is calculated based on whether the prediction falls under the top n probabilities of the app.

Recurrent Neural Network (LSTM/RNN)

- **Problem Statement**: Predict the (total) time usage of an app/tab/recorded process using the past *time-series* data

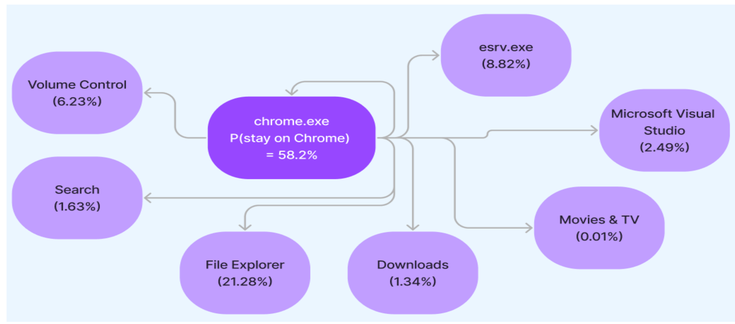


- **Feature Engineering**:
  1. Use time per day - split hourly between 24 columns (labeled 0 - 23)
  2. Lookback 3-5 time steps from the current timestamp
  3. One-hot-encoding (process names or weekdays); np.sin-encoding (hours)
- **Experiments**: implemented by *Keras/Tensorflow*
  1. **Train/Test**: 80/20, no shuffle
  2. **Metrics**: RMSE, TP/TN/FP/FN, Preds==True if within 3 mins of real vals

Predictive Results

Transition Matrix Results
1. n = 1, ACC (User 1) = 47.5%, ACC (User 2) = 40.1%
2. n = 2, ACC (User 1) = 63.9%, ACC (User 2) = 57.5%
3. n = 5, ACC (User 1) = 84.3%, ACC (User 2) = 80.0%
4. n = 10, ACC (User 1) = 95.7%, ACC (User 2) = 92.6%
5. n = 15, ACC (User 1) = 98.8%, ACC (User 2) = 96.6%

Emission Matrix Results: Example: $P(\text{"}app\text{"}|\text{"}chrome.exe\text{"})$



LSTM/RNN Performance

Model	Design (N=nodes)	Eval Bins/Criteria	Performance
Vanilla LSTM > Split Hourly > OH(Process Name)	Input RNN (64N) Hidden Dense (4N) Output Dense (1N)	[0, 0.01] (0.01, 0.02] (0.02, 0.2] (0.2, max]	TP = 691, TN = 0 FP = 65, FN = 0 ACC = ≈ 91.4%
Stacked LSTM > Split Hourly > OH(Process Name)	Input LSTM (16N) Hidden LSTM (32N) Hidden Dense (64N) Output Dense (1N)	[0, 0.01] (0.01, 0.02] (0.02, 0.2] (0.2, max]	TP = 467, TN = 52 FP = 13, FN = 224 ACC = ≈ 68.65%
Stacked LSTM > Split Hourly > Lookback(5 timesteps)	Input LSTM (50N) Hidden LSTM (50N) Output Dense (1N)	Preds==True if abs(diff) <= 3mins	RMSE = 1134.68 ACC = ≈ 27%

Conclusion

- **Data Collection**
  1. *Foreground Window IL* provides the most applicable data for further analysis using EDA, HMM, RNNs
  2. Data should be collected *continuously* and *consistently* to allow detecting the patterns of user behaviors
- **Predictive Tasks**
  1. More data -> better predictive results
  2. HMM: The accuracy increases as n (the number of apps) increases
  3. RNNs: Feature engineering, input RNN libraries, and hyperparameter tuning greatly influence the model performance
  4. We can use these results to infer the daily sequence of app usage and their use times –> devise solutions to reduce the initial user-CPU latency
- **Proposed Solutions**
  Using our predictive results, we can further develop a script to process the background tasks and utilize the *Task Manager* to open the next app 2-3 mins beforehand