# Introduction

The process performance of an operating system is a critical aspect that determines the overall efficiency and responsiveness of the system. Processes are essentially the tasks or programs that are executed by the operating system, and their performance can have a significant impact on the overall performance of the system. The operating system needs to efficiently manage the allocation of resources such as CPU time, memory, and I/O operations to ensure that each process runs smoothly without causing delays or crashes. Therefore, predicting the performance of processes can help achieve to provide a more efficient system performance and ensure all programs can be completed efficiently. It can bring several benefits including early detection of performance issues, improved resource utilization, better capacity planning, reduced downtime, improved user experience, cost savings, etc. To make the prediction happen, this involves monitoring various process metrics, such as CPU utilization, memory usage, I/O throughput, and response time, and adjusting the system configuration as needed to optimize performance.

On any operating system, the user can check process information in terminal by some build-in command lines. The information usually includes each task’s unique process ID, username of owner of each task, priority of each task, total virtual memory used by each task, amount of actual physical memory each process is consuming, shared Memory size in kilobytes unit used by each task, the state each process is in, CPU usage for each process, memory usage of each task, CPU time, command that is being run, etc.

By leveraging the vast amounts of data generated by the system, machine learning algorithms can identify patterns and make predictions about process performance with a high degree of accuracy.

In summary, in this project I apply machine learning techniques to learn the performance and behavior of processes in operating systems. I build couple predictive models. Firstly, I develop predictive models using linear regression and random forest regression to estimate the performance of processes in operating system. Secondly, I show that the process performance can be accurately predicted by using data I collect from Linux operating system and mac-OS operating system.

# Predictive Models

In this project, I collect data from Linux operating system and mac-OS operating system and use predictive modeling techniques from machine learning [5, 6] to obtain estimates of performance of process in both operating system respectively. I ~~use a total of nine models. The four linear regression models are described in the next section. Section 3.2 discusses the five neural network based models developed in this work.~~

# 3.1. Linear Regression (LR) Models

Linear regression is a basic and commonly used type of predictive analysis in machine learning algorithms. The overall idea of it is to examine the following questions. Firstly, does a set of predictor variables (also called exogenous variables or regressors) provide accurate result in predicting an outcome variable (also called criterion variable or endogenous variable)? Secondly, which variables are significant predictors of the outcome variable?

~~, and in what way do they–indicated by the magnitude and sign of the beta estimates–impact the outcome variable?~~

~~These regression estimates can explain the relationship between one dependent variable and one or more independent variables. The simplest form of the regression equation with one dependent and one independent variable is defined by the formula y = c + b\*x, where y is estimated dependent variable score, c is constant, b is regression coefficient, and x is score on the independent variable.~~

# 3.2. Random Forest Regression (RFR) Models

# 3.4. Data Collection and Input Values

Data collection is an essential part of in this project. I need large amount of data related to process in different operating systems.

In this project, I write a kernel module which help collects the data in Linux operating system. A kernel module, also known as a driver, is a software component that can be loaded or unloaded dynamically into the kernel of an operating system. It can interact with hardware components and provide services to user-space applications. The kernel module is written in a low-level programming language which is C language. It adheres kernel's programming interface (API) and driver model. I use couple kernel function to help achieve the goal of kernel module which is to log information related to processes into a virtual file. I load the kernel module I build on Linux operating system. When it is loaded, it becomes part of the running kernel and I do not need to do additional steps to execute kernel to collect process-related information. The kernel module does its work as the system is on.

Proclog.c is the file I built for kernel module. The file needs to be compiled and there will be couple files with same name but different file types which are generated in the process of compiling. The one file ending by .ko is the critical file used to install in Linux operating system. When .ko file is loaded successfully, a msg will remind the user. What this kernel module does is create a virtual file called log\_file stored in /proc/ folder. log\_file is writable by kernel-level functions but not user-level applications. As Linux environment is on, log\_file is consistently updated by kernel module to log process-related data. So it will grow big in file size dramatically in a very short time. With help of its attribute as virtual file, it does not take significantly large space on hard disk.

As kernel modules are typically not designed to write actual files because they operate at a low-level in the operating system and are primarily used to interact with hardware components, system resources, and other kernel components, I need to handle writing an actual file by user-space processes or applications that run on top of the operating system. So I write an additional c file which will be executed in user level to export content of virtual file into actual file. And because it is not likely to export the comprehensive data from virtual data due to its consistent update by kernel module for data logging, I create an input step for user to manually input a positive integer which sets up the number of lines to export. If we let the file open for exporting, the read will never reach to the end and the program for data export will never end.

The reason that I choose to build a kernel module to collect process-related data on Linux operating system is that the kernel module can be loaded and unloaded dynamically without rebooting the operating system. This flexibility of managing system resources does not provide a hard work for kernel of Linux operating system to handle.

However, every coin has two sides that building kernel module has various restrictions too. Debugging on kernel module is difficult because there are no detailed debug hits given by operating system to help developers fix issues.

Besides, I build a function in python to collect data on mac-OS operating system. Python has its library called psutil to retrieve process-related data by couple lines of python code.

I set up an input for user to choose own interval for data collection. The python function can retrieve process data every interval that user inputs and store the data in a file in csv format.

One of advantage of building Python on a mac-OS environment is that it is relatively easy because the mac-OS operating system comes with pre-installed development tools such as a C compiler and the XCode development environment. These tools are essential for building Python from source code, and their availability on macOS makes it straightforward to set up and build Python on a MacBook.

~~In the process we have done some data preprocessing by replacing the whitespace into comma which can be recognized by csv format.~~

# Data Preparation

Once I gather the data, I need to preprocess data and use machine learning algorithms to build and train a predictive model that can predict processor performance based on the input data. the data will be divided into trained data and test data. We can use regression algorithms such as linear regression, decision tree regression, or random forest regression to predict the performance. You can also use classification algorithms such as decision trees or neural networks to classify the performance as good or bad based on the input data.

In our experiments, Clementine software automatically scales the input data to the range 0-1 to prevent the effect of scales of different parameters. The linear regression methods expect the input parameters to be numerical. Therefore some of the inputs to Clementine (as they will be presented in the following section) need to be mapped to numeric values. For some other input parameters this kind of transformation is not possible, hence these are omitted by Clementine. However, neural network models can have any type of input (numeric, flag, categorical), and are automatically transformed and scaled to be used in model generation. In this work, we feed all the input available parameters to Clementine. Then the program automatically measures the importance of the parameters, and depending on the methodology adds or removes predictor variables to the model. In some of the chronological design space experiments Clementine omits some predictor variables because these input parameters does not have any variation (e.g. single L2 cache size configuration). Other than this kind of predictor elimination, we don’t discard any input.

# Evaluation