12-2023

Machine Learning for Kalman Filter Tuning Prediction in GPS/INS Trajectory Estimation

Peter Wright

MACHINE LEARNING FOR KALMAN FILTER TUNING PREDICTION IN GPS/INS TRAJECTORY ESTIMATION

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Peter Wright

December 2023

MACHINE LEARNING FOR KALMAN FILTER TUNING PREDICTION IN GPS/INS TRAJECTORY ESTIMATION

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Peter Wright

December 2023

Approved by:

Bilal Khan PhD, Committee Chair, Computer Science

Fadi Muheidat PhD, Committee Member

Jennifer Jin PhD, Committee Member

ABSTRACT

This project is an exploration and implementation of an application using Machine Learning (ML) and Artificial Intelligence (AI) techniques which would be capable of automatically tuning Kalman-Filter parameters used in post-flight trajectory estimation software at Edwards Air Force Base (EAFB), CA. The scope of the work in this paper is to design and develop a skeleton application with modular design, where various AI/ML modules could be developed to plug-in to the application for tuning-switch prediction.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

From the project proposal: "While a finished product is not expected to be produced [during the academic project timeframe] due to time [and scope] constraints, if successful, the academic work will likely be the foundation for the production of an actual software product being produced and commissioned for use in EAFB DT&E platform testing."

The project was to analyze the systems in place at EAFB utilized to produce validated post-flight trajectory data, design a system which could integrate into that process and provide an AI-Assistant to the data analysts, and produce a skeletal prototype of the system that was a sufficiently functional 'proof-of-concept' to further develop it into production-quality software product. Depending on the capability of the system designed, anything from an automated tuning suggestion assistant to a fully-automatic auto-tuning system was a potential solution. We tried to aim high and design a system which could be used in any of those capacities, including having the potential to do the job of the analyst essentially independently, if a high enough performance level in the algorithms were to be achieved.

Design Considerations

We investigated and researched the state of the art in neural network technologies, and other applicable artificial intelligence concepts, and designed

an ensemble system [1] which is based around three main AI/ML components, LSTM neural networks, Genetic Algorithms, and Ensemble Modeling. After the system was designed (and accepted by the EAFB TSPI technical expert), the development was split into two main areas of effort.

The first was the design of the architecture engine/harness that the AI/ML modules for tuning-predictions and tuning-effects-prediction will plug into. This harness also has the Genetic Algorithm component of the design built into it [2]. This portion was built with C# in a windows environment.

The second area of effort was the development of a python codebase that would contain the AI/ML code that will be called by the harness to perform tuning predictions. Also included in this was the development of a console application that assisted us in the data cleaning, preprocessing, building of test/train sets, and performing initial training and testing of the NN components for tuning prediction.

We will describe the theory behind our design, and show the current state of development of these two main system components, and their sub-components, in this paper.

CHAPTER TWO

BACKGROUND

The Problem

Edwards Air Force Base (EAFB) is the United States premier Air Force (AF) range for Developmental Test and Evaluation (DT&E) of major developmental weapons platforms. The Time Space Position Information (TSPI) Engineering Flight (ENRTE) at EAFB uses a state of the art 151-state Carlson-Biermann unitary diagonal Kalman filter for post-flight tuning of trajectory data to achieve 6 Degrees-of-Freedom (6DOF) truth values for platform validation. The software platform which contains the filter as well as a Reich-Tung-Streibel (RTS) back-propagated optimal smoother is called Multi-Optimal Smoother Estimation Software (MOSES). The input data to MOSES is obtained from several families of military grade integrated GPS-INS sensors maintained by ENRTE at EAFB which record GPS and INS in-flight data.

The US Department of Defense (DOD) is increasingly interested in using innovative technologies, including Artificial Intelligence (AI) and Machine Learning (ML) to accelerate efficiency, maximize the power of the workforce, and push capability envelopes forward. Accordingly, accelerated efforts have recently been devoted to automating the tuning of MOSES which currently requires full-time efforts from a team of professionals to operate in optimal conditions.

Kalman filters have many error states being estimated which may require tuning in accordance with the quality of the data collected. The mathematics

contain parameters for process and measurement noise which may be tuned, and some data may be weighted or cut out entirely. Many of these tunings are associated with the physics of GPS and INS measurement data and currently require expert knowledge to tune appropriately.

The Desired Solution

While a human-in-the-loop is desirable for quality assurance, a data driven auto-tuner approach can be proven significantly beneficial over current system with one or both of the following objectives accomplished. The first would be optimization of the tuning process via data driven adaptive auto-tuner module while reducing the amount of manual effort. The second would be provision of a decision support system for an analyst to produce higher quality trajectory solutions.

However, accomplishing the above with a fully functional adaptive auto-tuner could cut significant annual costs currently spent for manual effort (data analysis, validation and system tuning) from trained professionals on a full-time basis. The cost of previous efforts on tuning MOSES has been significant, surpassing $1M spent attempting to produce the above described product, which is indicative of its value to the DOD. The purpose of successful auto-tuning of Kalman filters is to create better real-time or post-processed trajectory data. Such technology would be of interest in many arenas of commercial, military, and civilian autonomous navigation.

Kalman filters are notoriously difficult to tune without expert knowledge about the characteristics of the sensor taking the measurements and the dynamics of the vehicle being tracked. If the filter is tuned too 'tightly', the mathematics can cause the predicted trajectory to 'diverge' forever from the truth, however if the filter is tuned to 'loosely', high accuracy cannot be achieved. For high accuracy applications, the filter must be tuned sufficiently (the measurements must be 'trusted enough') to maintain a tight fix on the position of the vehicle, and data anomalies which might therefore confuse the filter must be hand-tuned away.

Given the rapid advancement in AI and machine learning with the production of voluminous data in multiple areas of interest, specially tuned KF using AI/ML may be possible now, and will be useful to estimate 6DOF positioning of vehicles in many spaces. A data driven approach with the capability to learn and adapt to the real-time dynamics with varying trajectories for a particular vehicle by training on that vehicle's data would be of widespread value.

CHAPTER THREE

SCOPE

The purpose of this project was to develop an architecture, and a skeletal implementation of it, which can support AL/ML-driven modules that work together as an in-house auto-tuner for MOSES. The fundamental technology behind the product is designed to be able to propose tunings for the filter which might ameliorate undesirable conditions, recognize undesirable conditions/features within the data and alert the analyst, and iteratively apply tunings until certain heuristic threshold constraints are satisfied

We investigated the use of many ML technologies for applicability to this task, and the ones we settled on for an initial implementation were Long-Short-Term-Memory (LTSM) – an extension of Recurrent Neural Networks (RNNs) [3], Gated Recurrent Units [4], Genetic Algorithms (GA) [2], and Ensemble Modeling (A hybrid/collaboration between various NN and ML models and a GA). All recurrent neural networks we designed in were implemented with attention mechanisms [5].

We see a lot of potential in implementing Decision Trees – Random Forests, and Convolutional Neural Networks (CNN) in the future.

The desired scope of the project was to develop an application skeleton with a modular design which would be able to tune the data properly provided that the ML/AI modules at particular points in the architecture are capable of

accomplishing the aforementioned tasks to select tunings and/or recognize and ameliorate undesirable data conditions.

The architecture was to be built extensible so that it could support additional future AI/ML modules in a compartmentalized manner, where once those modules were added, they would automatically train on the available archived flight trajectory data possessed by EAFB, and seamlessly integrate with the rest of the system in making automated tuning suggestions.

CHAPTER FOUR

RELATED WORK

EAFB has previously contracted various third party organizations in an attempt to produce a product for this purpose. Despite various commercial contracts, 5 failed attempts have been made in the past by small business organization. Additionally, NASA's Jet Propulsion Laboratory's (JPL) Assistant for Understanding Data through Reasoning, Extraction, and Synthesis (AUDREY) team has been contracted. A phase I attempt which did not employ AI/ML techniques did not yield a desirable solution. A phase II attempt proposing the use of Artificial Neural Networks (NN) in 2020 was aborted due to data availability/requirements, differences in vision, and funding issues which arose a few years into the phase II contract.

CHAPTER FIVE

PROPOSED FRAMEWORK

I designed a C# application, and a python application, which work together to perform the necessary tasks to preprocess data into a form usable by neural networks, create the neural networks, train them, and then use them to make tuning predictions. Those tuning predictions are then combined into a genetic algorithm which combines them and tests them using the MOSES application, and judges their quality. The C# application skeleton was completed, including all modules for feature detection, tuning prediction, genetic algorithm, thread management, and a development GUI. The potions stubbed in but not yet completed are the heuristic algorithms for tuning-set evaluation, and the 'graph-prediction neural networks' module which was designed in to one-day be a more efficient evaluator for the GA than running the actual application (it will try to predict estimates of certain output parameters). The few portions that are merely stubbed in are because they are not critical for the application to work initially as a proof-of-concept, and the project scope in its entirety is several years' worth of professional work, so the portion we are able to complete for the academic culminating work must choose to not implement or only partially-implement certain features in order to meet the desired timeframes and scope-of-work for the academic project.

The application as a whole is called Heuristic Extrapolation Architecture for TSPI Handling of Estimation, 412th Range Squadron (HEATHER). In the rest of this section, I will describe some major considerations taken into account developing HEATHER, and give an overview of how the architecture works.

Major Considerations

HEATHER is designed to run in a secure environment within the DOD, so no external network connectivity is allowed, and to be able to run it on an isolated machine with minimal external network communication is preferred. For this reason, we secured a Lambda high performance workstation to run the application. Securing this machine, we are able to store and process large amounts of flight data on the machine and have the ability to train networks on it in a more reasonable timeframe.

The amount of data and formatting was an issue as well. EAFB has over 20 years of archived flight data, and as programs change over the years that are used to process this data, a lot of massaging had to be done to get sufficient amounts of it into a usable and consistent state. This is an ongoing process but currently we have converted 4100 missions into usable format though the use of some helper programs written in C# to pull data from the archival system, and get it all into a consistent state whereby those missions can be pre-processed, and used for training by the python code.

Usability and maintainability were important considerations, as few people in the professional workforce currently have any expertise with AI / ML technologies, and this is perhaps more true than average within the DOD. For this reason the main engine was written in C#, as the intended location of use for the program has significant expertise in this language, and the C# WinForms library is extensive and powerful for creating stable GUIs with high usability.

Modularity and extensibility were also very important, as the application of ML to this particular task is still an area of ongoing research, and there is not much work done. As we develop ML modules and train them on the data, we may find other solutions/modules would be better, and/or want to change course. So the architecture was designed to easily accommodate whatever ML modules you would want to plug in for tuning predictions.

Speed and performance were also a consideration for two reasons. The smaller reason was the time it will take to preprocess data and train neural networks initially, for the application to use, but as a one-time cost, that is the lesser concern, which was handled sufficiently with the acquisition of the high-performance workstation. The larger performance concern was that the application will be used to process data which is sometimes needed in a quick-turn fashion. There are missions which are critical that the results of the data collecting during flight is needed ASAP to inform the test team for the weapons-platform of what to do next. For this reason, significant work was done in the main application to allow jobs to be broken down into pieces (Gigs) which may be

started or stopped/pre-empted by higher priority jobs, and later resumed without loss of progress.

Training Application

I wrote a console application in python that allows the user create new NN models, specifying hyper-parameters through a menu system. It also provides options for cleaning/wiping of data, resetting the NNs to initial conditions, preprocessing the raw mission data, training the NNs, and testing the NNs. It allows specification through the menu system of system settings, and it organizes all its data and metadata in an organized file structure.

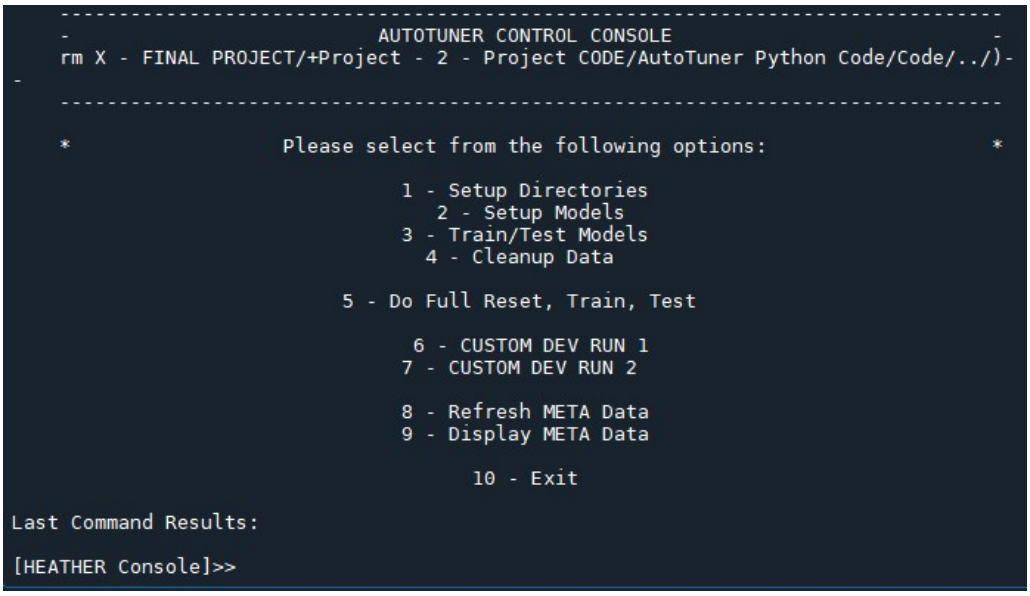

Figure 1. Menu System: Main Menu

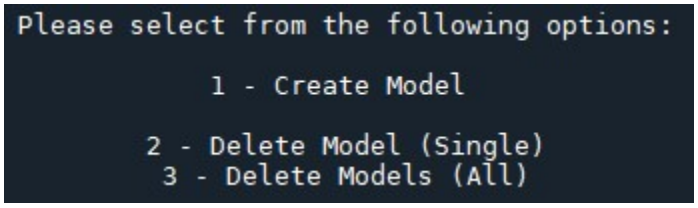

Figure 2. Menu System: Model Setup

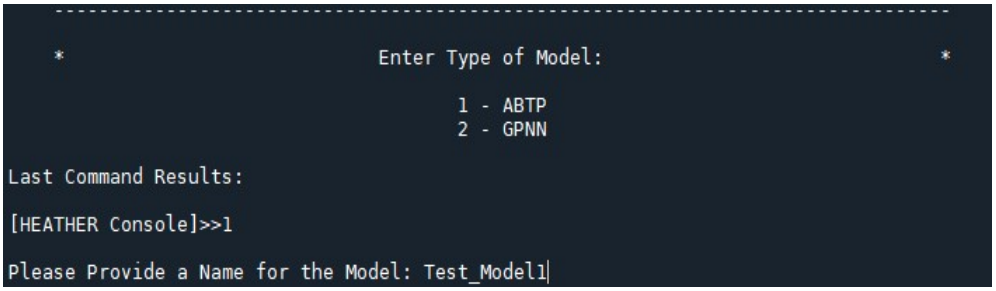

Figure 3. Menu System: Provide Name

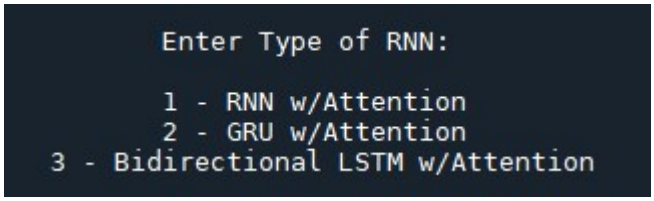

Figure 4. Menu System: Select Model Type

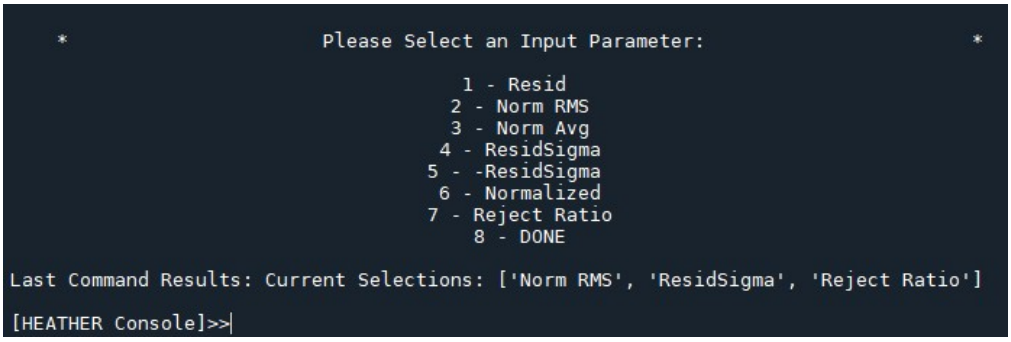

Figure 5. Menu System: Select Input/Output Parameters

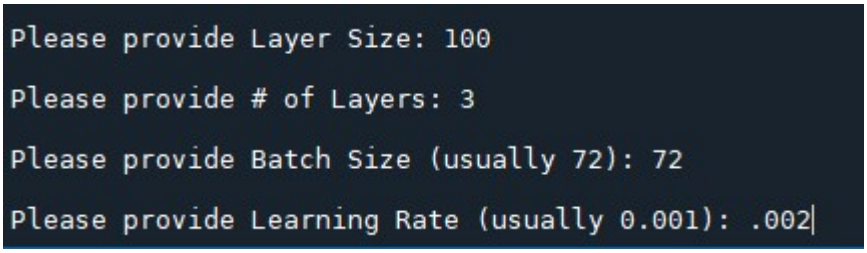

Figure 6. Menu System: Provide Meta-parameters

Once you've set up a model (you can have any number created at a time), you can then see some info about the missions and models in the system. Figure 7 is a screenshot of a small sample where sensitive information about the two sample missions has been omitted

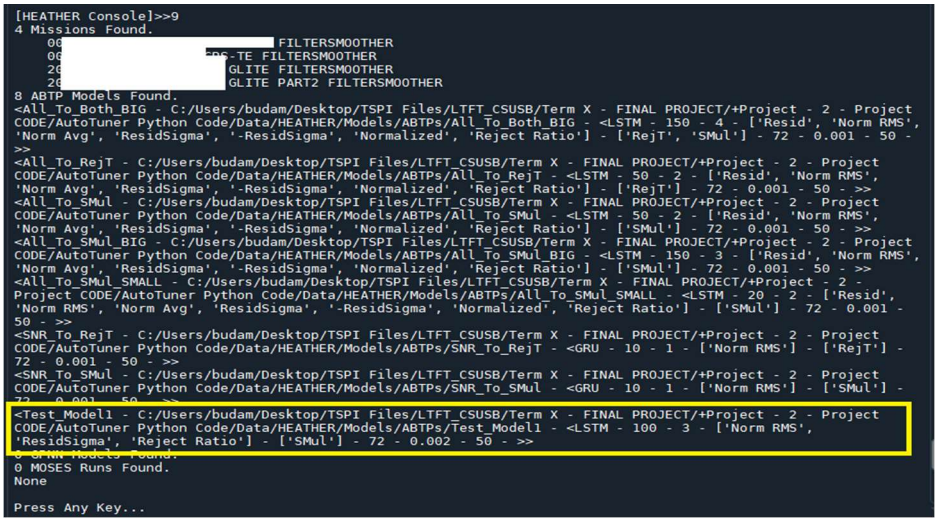

Figure 7. Example Output: Model Meta-Information

Below you'll see the test and training menu, where you may select options to preprocess data, and the system will examine all created models, all missions and what data is present in them, and it will notice whether anything is missing in

the preprocessed data folder. Any missing data is gathered from the raw missions and preprocessed, while data which is already present is ignored.

If already pre-processed data needs to be removed for some reason, the 'Clean Data' menu available from the main menu can do that.

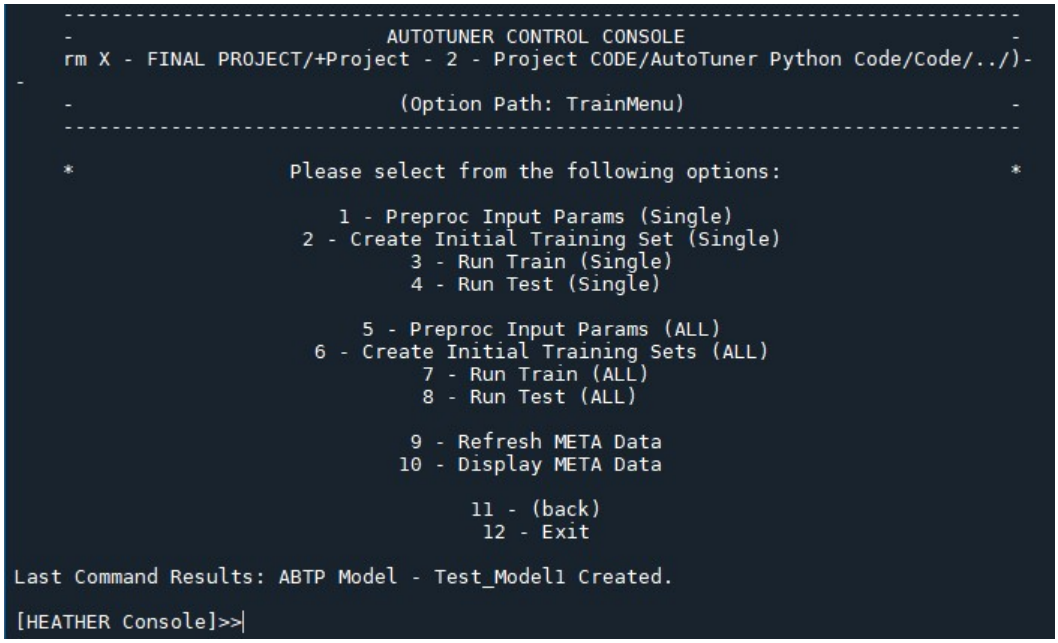

Figure 8. Menu System: Test & Training Menu

In Figure 8, you can see the menu that allows you to select to preprocess all input data. Selecting the preprocess option will show outputs similar to Figure 9 and Figure 10, depending on what data is missing (some data omitted for security reasons):

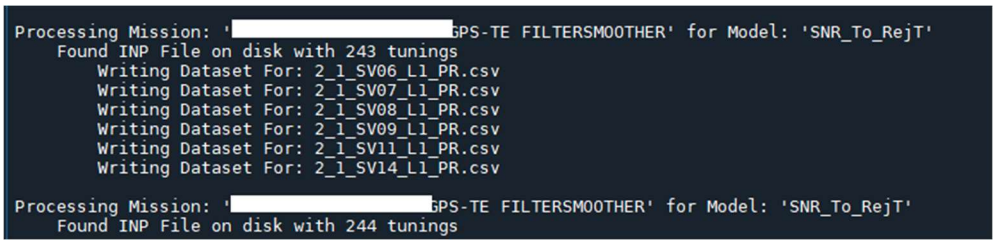

Figure 9. Example Output: Writing Dataset Signals for Each Mission

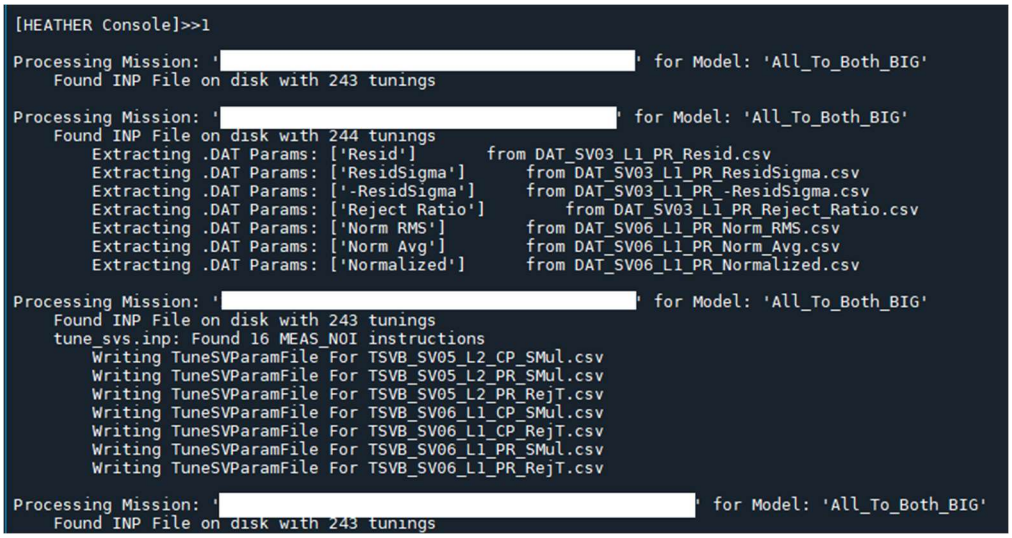

Figure 10. Example Output: Preprocessing MOSES Input Files

Using this console application, which is powered by python code implementing all of the functionality under the hood, a worker should be able to even create new models and train on new data just by dropping new raw missions into the mission folder, or creating a model through the menu system. Once they select 'preprocess all data', or if they clean the data and re-train, using the console, the older models will retrain incorporating all available data at that time.

Real-Time Application

The second part of the program is written in C# and will eventually have a nice GUI, but the GUI implemented currently is just for development purposes, so it's omitted from this report. But over 10k lines of code were created in the main project which implement the following architecture which was planned and diagrammed in MS Visio during the design phases of the project.

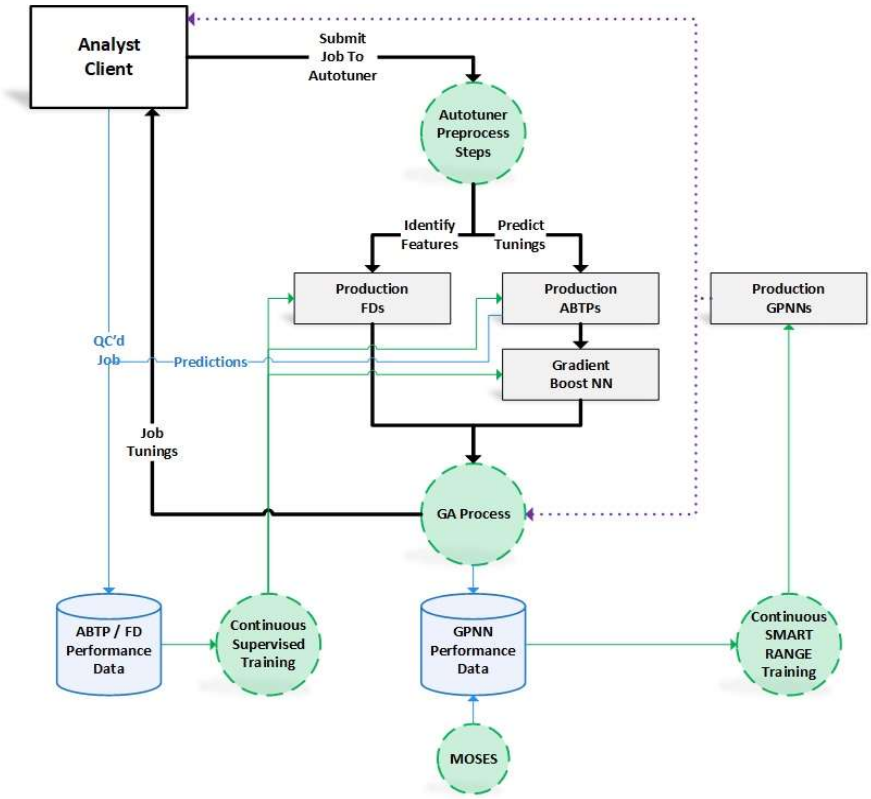

Figure 11. System Design: Logical Component Diagram

A more detailed system diagram and a description of the training loops it uses has been created, but they're too large to show well in this paper. Figure 12 and Figure 13 are the system design documents, of which higher resolution copies may be provided upon request to the author

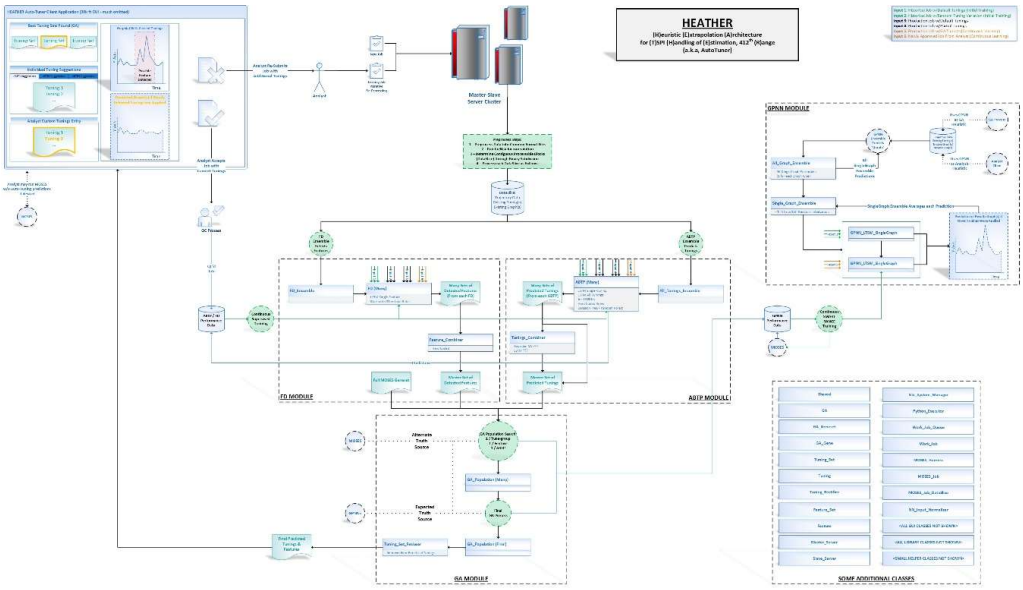

Figure 12. System Design: Complete System Diagram

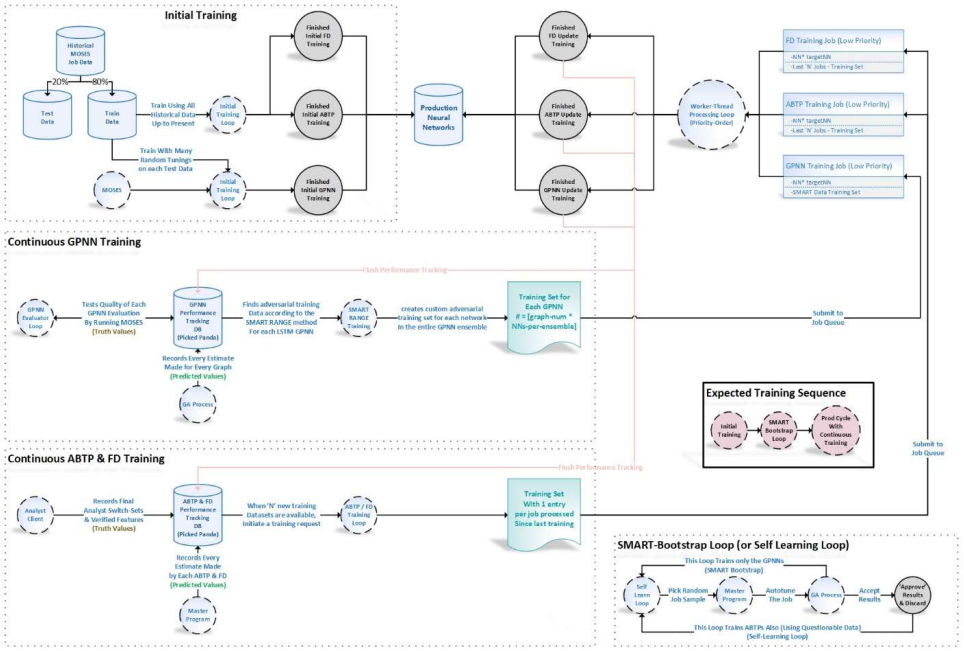

Figure 13. System Design: Available Training-Loops State-Diagram

A Job / Gig system was created so that jobs could be processed bit-by-bit, and if a higher priority job came into the pipeline, the auto-tuning for that higher priority job would pre-empt further execution of the lower priority Jobs in the system. This way, once the Gig objects for the high priority jobs were cleared from the work queue, the lower priority ones would then execute without loss of progress.

The system needs to be able to maintain state through shutdowns, and the Gig system facilitates this also, as these objects can be serialized to disk before and after completion, however the serialization code has not been written yet, but the system is designed to allow it easily in future work.

The modularity of the system should be apparent through the diagrams, whereby if the user creates a new model as shown in the python console application, the HEATHER engine will see that model and incorporate it into the architecture at the right level. Right now, the only models implemented are turning switch predictors, but other ML modules would be seen and used by the engine in a similar manner through future work.

The system also was designed to be tuning-value and estimation-software agnostic, so if a new analysis engine is implemented for trajectory estimation (one is being worked on by an outside contractor at the moment, and may be in use within a few years), all that would need to be rewritten in the HEATHER engine are a few modules that define how the HEATHER system interfaces with

the analysis software, and which describes what tuning switches were available for that analysis engine.

Analyst/Client Integration

There is a third portion to the whole application suite which was not worked on at all during this academic proof-of-concept portion of the development, which is a Client application which will be used by the analyst to interface with the master application. Analysts use desktop PCs which can communicate over the secure network to the master machine. A modification to their Analysis client will need to be made that does this communication, and requests automated tuning suggestions from the HEATHER master application. Work on this client modification was deferred to a later state where the academic work will be turned into a usable/deliverable product for the analysts.

CHAPTER SIX

PRELIMINARY RESULTS

Currently, the system has only been run on a few missions, using the CPU. CUDA enhancements to the code are now complete, and the full dataset is being processed on the high performance workstation, but it will take weeks to months (depending on the size of the networks trained) to train each model on the massive set of data. The current CPU run was performed on just 100 datasets (100 satellite/frequency/code combinations) available from just 2 missions, so the results shown are not indicative of final results, but even from training on 2 missions, I can show the results of the engine starting to show in small ways:

Promising Results

In Figure 14 and Figure 15, you still find a few examples of what we view as promising results of the engine. The neural networks that predicted these tunings were trained on just 100 datasets for 50 epochs.

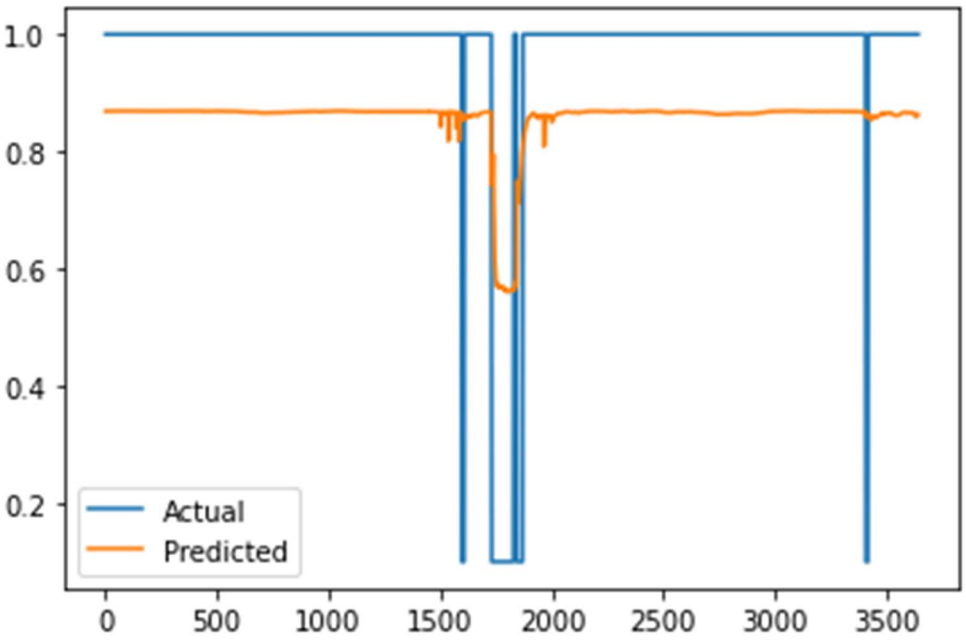

Figure 14. Promising Result 1.

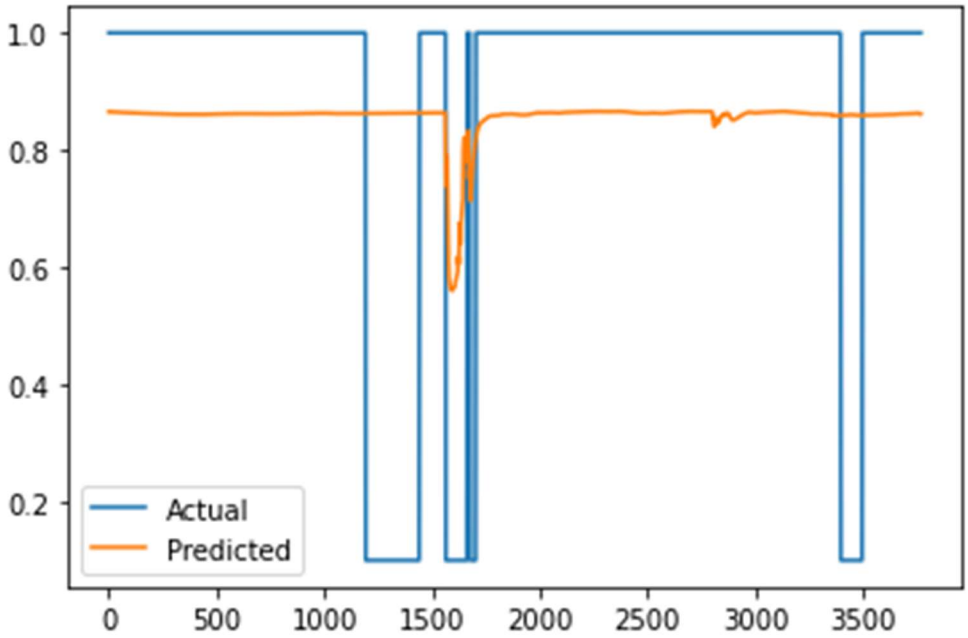

Figure 15. Promising Result 2.

From these results, it's clear that the input parameters we're feeding into the networks have sufficient information for the network to predict when a satellite should be de-weighted. The networks didn't estimate the same de-weighting factors as the analyst did, but changes to meta-parameters, increased training data sizes, and longer training times may yield better results. One important question we had was whether it was even possible for a neural network to be able to determine when a pseudo-range should be de-weighted with the input parameters we're able to obtain for input parameters. These results were just a few of the positive results, and are shown for example.

Less than Promising Results

Figure 16 and Figure 17 are examples of when the network did not recognize that a tuning should be made, or estimated the wrong level of weighting for a satellite measurement.

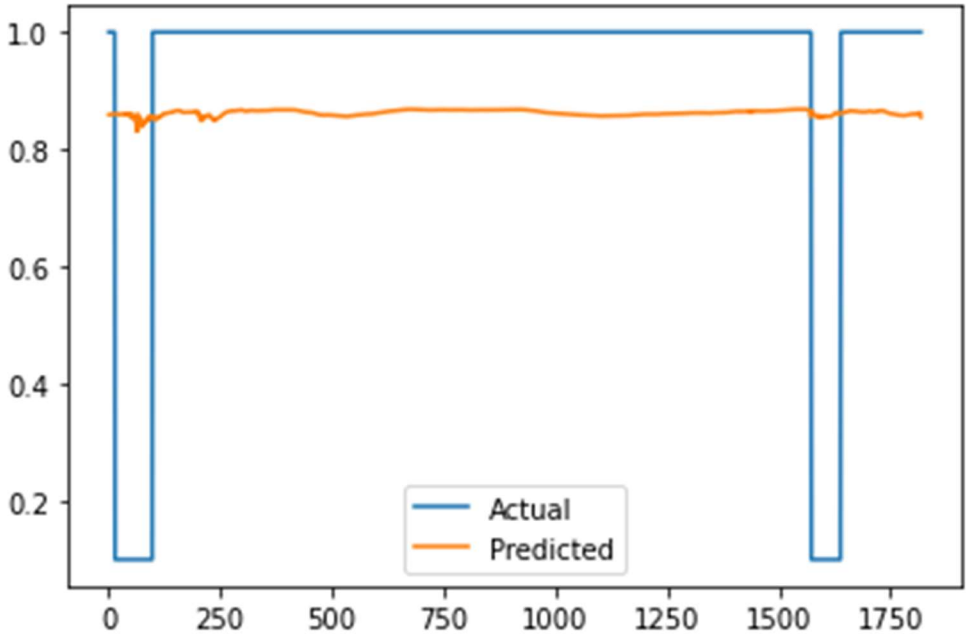

Figure 16. Bad Result 1.

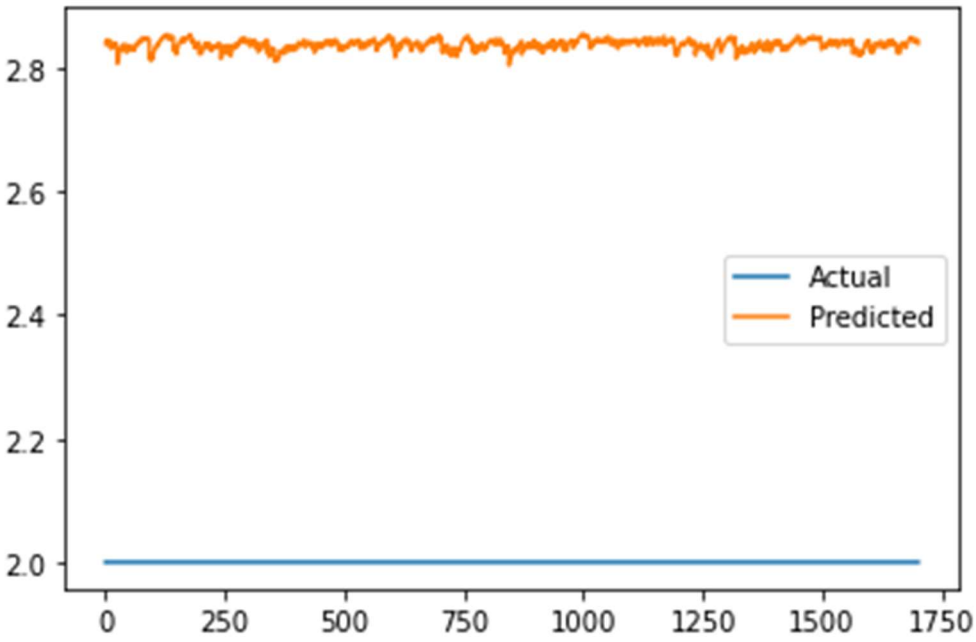

Figure 17. Bad Result 2.

However even when full results are achieved, ‘failure’ cases like these might be expected. Analysts do not always de-weight satellites when their data is poor quality. The characteristics being learned by the neural networks might therefore identify desirable tunings the analyst did not make, based on examples where the analyst did tune.

Another finding of note is that the system is able to predict reasonable numbers only training on 2 missions with 100 satellite/frequency/code datasets. When it's trained on the 100k datasets from 4.1k missions, hopefully the predictions which are off by a steady state will have less loss, and a higher percentage of the tunings which the analyst made were able to be detected and tuned by the networks.

CHAPTER SEVEN

CONCLUSIONS AND FUTURE WORK

Thus far in the system design, a complete proof-of-concept of the core functionality has been accomplished. University policies permitting, this paper may be updated in the future when final results and accuracies from the 4.1k missions available are finished processing on the high-performance workstation.

Future work involves tuning of meta-parameters, implementing more NN models and other ML models like decision trees and perhaps Convolutional Neural Networks (CNN) for faster network training speeds than LSTM networks. Also development of the solution-quality heuristic algorithms, a high quality GUI for the HEATHER engine, a client application and GUI, implementing feature detection, GPNNs, and hooking the HEATHER engine up to the python back-end for iterative-tuning, and automatic continuous training may be performed. Such work represents another 1-2 man years of effort with potential for that to be a high-return investment.

REFERENCES

[1] Robert, E. Schapire, "The Strength of Weak Learnability", Machine Learning, 1990

[2] Tanweer Alam, Shamimul Qamar, Amit Dixit, Mohamed Benaida, "Genetic Algorithm: Reviews, Implementations, and Applications", arXiv, 2020

[3] Sepp Hochreiter, Jürgen Schmidhuber, "Long Short-Term Memory", Neural Computation, 1997

[4] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", arXiv, 2014

[5] A. Vaswani, N. Shazeer, N. Parmar, and J. Uszkoreit, “Attention Is All You Need,” arXiv, 2017.