# Improving Neuroevolutionary Transfer Learning of Deep Recurrent Neural Networks through Network-Aware Adaptation

AbdElRahman ElSaid
Rochester Institute of Technology
Rochester, New York
aelsaid@mail.rit.edu

Joshua Karns
Rochester Institute of Technology
Rochester, New York
josh@mail.rit.edu

Zimeng Lyu
Rochester Institute of Technology
Rochester, New York
zimenglyu@mail.rit.edu

Daniel Krutz
Rochester Institute of Technology
Rochester, New York
dxkvse@rit.edu

Alexander Ororbia II
Rochester Institute of Technology
Rochester, New York
ago@cs.rit.edu

Travis Desell
Rochester Institute of Technology
Rochester, New York
tjdvse@rit.edu

## ABSTRACT

Transfer learning entails taking an artificial neural network (ANN) that is trained on a source dataset and adapting it to a new target dataset. While it has been shown to be quite powerful, its use has generally been restricted by architectural constraints. Previously, in order to reuse and adapt an ANN's internal weights and structure, the underlying topology of the ANN being transferred across tasks must remain mostly the same while a new output layer is attached, discarding the old output layer's weights. This work introduces *network-aware adaptive structure transfer learning* (N-ASTL), an advancement over prior efforts to remove this restriction. N-ASTL utilizes statistical information related to the source network's topology and weight distribution in order to inform how new input and output neurons are to be integrated into the existing structure. Results show improvements over prior state-of-the-art, including the ability to transfer in challenging real-world datasets not previously possible and improved generalization over untransferred RNNs.

## CCS CONCEPTS

• **Computer systems organization → Neural networks**; • **Computing methodologies → Transfer learning**; • **Theory of computation → Evolutionary algorithms**;

## KEYWORDS

Neuroevolution, Recurrent Neural Networks, Transfer Learning.

**ACM Reference Format:**
AbdElRahman ElSaid, Joshua Karns, Zimeng Lyu, Daniel Krutz, Alexander Ororbia II, and Travis Desell. 2020. Improving Neuroevolutionary Transfer Learning of Deep Recurrent Neural Networks through Network-Aware Adaptation. In *Genetic and Evolutionary Computation Conference (GECCO '20), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3377930.3390193

ACM ISBN 978-1-4503-7128-5/20/07…$15.00
https://doi.org/10.1145/3377930.3390193

## 1 INTRODUCTION

As predictive analytics becomes increasingly more commonplace, there is a widespread and growing number of real-world systems that stand to benefit from an improved ability to forecast and predict data. In many fields, such as power systems, aviation, transportation, and manufacturing, developing engines for predictive analytics requires time series forecasting algorithms capable of adapting to noisy, constantly changing sensors as well as major system modifications or upgrades. Mechanical systems, such as self-driving cars, robotics, aircraft, and unmanned aerial systems (UAS) in particular would benefit immensely from the ability to more accurately predict potential equipment and systems failure, which is critically important for cost and safety reasons. Developing predictive models for these systems poses a particularly challenging problem given that these systems experience rapid changes in terms of their operation and sensor capabilities.

Transfer learning potentially offers a solution. It has already proven to be a powerful tool to improve the optimization of deep artificial neural networks (ANNs), allowing them to re-use and fine-tune knowledge gained from training on large, prior datasets. However, transfer learning has mostly been limited to problems which require minimal to no topological changes, *i.e.*, changing the number of neurons and/or their synaptic connectivity, so that the previously trained weights and structure can be more easily fine tuned to the new target dataset. This is common in ANN specialization, or "pre-training". Gupta *et al.* specialized a recurrent neural network (RNN) trained to predict 20 different phenotypes from clinical data, by retraining it to predict previously unseen phenotypes with limited data [7]. Zhang *et al.* applied the same principle when predicting the remaining useful life (RUL) of various systems when data was scarce [26]. Other examples include [13, 24–26].

Approaches involving some structural modification include work by Mun *et al.*, which removed an ANN's output layer, replacing it with two additional hidden layers and a new output layer [14]. Partial knowledge transfer has also been done through the use of pre-trained embeddings of (sub)words and phrases [6, 12]. Hinton *et al.* proposed the concept of "knowledge distillation", where an ensemble of teacher models are "compressed" to a single pupil model [8]. Tang *et al.* also conducted the converse of this experiment – they trained a complex pupil model using a simpler teacher model [19]. Their findings demonstrated that knowledge gathered by a simple teacher model can effectively be transferred to a more complex pupil model which will have greater generalization capability. Deo *et al.* also concatenated mid-level features from two datasets as input to a target feed forward network [4].

Neuroevolutionary approaches have mostly focused on utilizing indirect encodings for transfer learning, such as Taylor *et al.* utilizing NeuroEvolution of Augmenting Topologies (NEAT [18]) to evolve task mappings to translate trained neural network policies between a source and target [20]. Yang *et al.* took this concept further to designing networks for cross-domain, cross-application, and cross-lingual transfer settings [23]. Vierbancsics *et al.* used a different approach where an indirect encoding was evolved that could be applied to ANN tasks that required different neural structures [21].

To date, transfer learning research has almost exclusively focused on classification tasks, with most work focusing on feed forward and convolutional neural networks (CNNs), and has mostly avoided any major architectural changes to the ANNs being transferred. While some studies have utilized recurrent neural networks (RNNs), they have not attempted to develop RNN models for time series data forecasting, a very challenging regression problem (see Section 4 for example prediction problems and why traditional statistical auto-regressive forecasting methods like ARIMA are insufficient). This work advances transfer learning to this area, as these capabilities will play a crucial role in developing predictive engines for previously described systems.

Prior work introduced *adaptive structure transfer learning* (ASTL), where input and output nodes were added and removed to adapt the structure of an RNN evolved on a source data set to a target data set [1]. However, this work did not take into account the overall structure and weight distribution of the transferred network, only adding minimal connections for the newly added input and output nodes. This work proposes a significantly improved approach for transferring RNN knowledge across tasks through *network-aware adaptive transfer learning* (N-ASTL). N-ASTL overcomes the limitations of ASTL by utilizing statistical information related to the source RNN's topology and weight distribution to inform the adaptation process. N-ASTL shows significant improvements over previously reported results, including successful transfer learning on the challenging data sets where ASTL previously failed.

## 2 EVOLUTIONARY EXPLORATION OF AUGMENTING MEMORY MODELS

This work utilizes the Evolutionary eXploration of Augmenting Memory Models (EXAMM) algorithm [15] to drive the neuroevolution process. EXAMM evolves progressively larger RNNs through a series of mutation and crossover (reproduction) operations. Mutations can be edge-based: *split edge*, *add edge*, *enable edge*, *add recurrent edge*, and *disable edge* operations, or work as higher-level node-based mutations: *disable node*, *enable node*, *add node*, *split node* and *merge node*. The type of node to be added is selected uniformly at random from a suite of simple neurons and complex memory cells: Δ-RNN units [16], gated recurrent units (GRUs) [2], long short-term memory cells (LSTMs) [9], minimal gated units (MGUs) [27], and update gate RNN cells (UGRNNs) [3]. This allows EXAMM to select for the best performing recurrent memory units. EXAMM also allows for *deep recurrent connections* which enables the RNN to directly use information beyond the previous time step. These deep recurrent connections have proven to offer significant improvements in model generalization, even yielding models that outperform state-of-the-art gated architectures [5]. To the authors' knowledge, these capabilities are not available in other neuroevolution frameworks capable of evolving RNNs, which is the primary reason EXAMM was selected to serve as the basis of this work. Due to space limitations we refer the reader to Ororbia *et al.* [15] for more details on the EXAMM algorithm. The N-ASTL and ASTL implementations have been made freely available and incorporated into EXAMM github repository[1].

To speed up the neuro-evolution process, EXAMM utilizes an asynchronous, distributed computing strategy that incorporates the concept of islands to promote speciation. This mechanism encourages both exploration and exploitation of massive search spaces. A master process maintains the populations for each island and generates new RNN candidate models from the islands in a round-robin manner. Workers receive candidate models and locally train them via back-propagation through time (BPTT), making EXAMM a memetic algorithm. When a worker completes the training of an RNN, that RNN is inserted back into the island that it originated from. Then, if the number of RNNs in an island exceeds the island's maximum population size, the RNN with the worst fitness score, i.e., validation set mean squared error (MSE), is deleted.

This asynchronous approach is particularly important given that the generated RNNs will have different topologies, with each candidate model requiring a different amount of time to train. This strategy allows the workers to complete the training of the generated RNNs at whatever speed they are capable of, yielding an algorithm that is naturally load-balanced. Unlike synchronous parallel evolutionary strategies, EXAMM easily scales up to any number of available processors, allowing population sizes that are independent of processor availability. The EXAMM codebase has a multi-threaded implementation for multi-core CPUs as well as an MPI [11] implementation that allows EXAMM to readily leverage high performance computing resources.

To initialize the island populations, EXAMM "seeds" each island population with the minimal network topology possible for the given inputs and outputs – a topology with no hidden nodes where each input node has a single feed forward connection to each output node. Each island population utilizes this *minimal genome* as a seed network as the first RNN in its population, which is ultimately sent to the first worker requesting an RNN to be trained. Subsequent requests for work from that island create new RNN candidates from mutations of the seed network until the population is full. When a given island population is full, EXAMM will start generating new RNNs from that island utilizing both mutation and intra-island crossover (both parents are selected within that same island). When all island populations are full, EXAMM will begin to generate additional, new RNNs from an inter-island crossover process. This crossover selects the first parent from the island that an RNN is being generated from and matches it with the most fit RNN from a randomly selected other island to serve as the second parent.

[1]https://github.com/travisdesell/exact

**Algorithm 1** Removal of Unused Nodes and Edges

```
function RemoveUnused(SeedNetwork sn, Param[] targetOutputs, Param[] targetInputs)
    ▹ Remove unused input and output nodes
    for all InputNode i in sn.inputNodes do
        if i.param not in targetInputs then
            sn.removeNode(i)
    for all OutputNode o in sn.outputNodes do
        if o.param not in targetOutputs then
            sn.removeNode(o)
    ▹ Mark reachability of edges and hidden nodes
    sn.markForwardReachability()
    sn.markBackwardReachability()
    for all Node n in sn.hiddenNodes do
        if !n.forwardReachable() or !n.backwardReachable() then
            n.setDisabled()
    for all Edge e in sn.edges do
        if !e.forwardReachable() or !e.backwardReachable() then
            e.setDisabled()
```

**Algorithm 2** ASTL Seed Network Adaptation

```
function ASTL(SeedNetwork sn, Param[] targetOutputs, Param[] targetInputs)
    RemoveUnused(sn, targetOutputs, targetInputs)
    ▹ Add new input and output nodes
    for all Param ti in targetInputs do
        if !sn.hasInputForParam(ti) then
            sn.addInputNode(new InputNode(ti))
    for all Param to in targetOutputs do
        if !sn.hasOutputForParam(to) then
            sn.addOutputNode(new OutputNode(to))
    ▹ Connect all new input and output nodes
    for all InputNode i in sn.inputNodes do
        if i.param in targetInputs then
            for all OutputNode o in sn.outputNodes do
                sn.addEdge(i, o, $weight \leftarrow \mathcal{U}(-0.5, 0.5)$)
    for all OutputNode o in sn.outputNodes do
        if o.param in targetOutputs then
            for all InputNode i in sn.inputNodes do
                sn.addEdge(i, o, $weight \leftarrow \mathcal{U}(-0.5, 0.5)$)
```

## 3 NETWORK-AWARE ADAPTIVE STRUCTURE TRANSFER LEARNING

### 3.1 Adaptive Structure Transfer Learning

Prior work on *adaptive structure transfer learning* (ASTL) proposed a simple scheme for transfer learning in EXAMM by hijacking the island seeding process to ultimately modify a previously trained RNN instead of the minimal genome [1]. The previously trained RNN is itself adapted to a new dataset through the following steps: *i)* removing unused outputs, *ii)* connecting new outputs to all inputs, *iii)* removing unused inputs, *iv)* connecting new inputs to all outputs, and *v)* disabling *vestigial* hidden nodes and edges.

The disabling of vestigial structures is necessary as removing the unused inputs and outputs can potentially disconnect parts of the RNN's topology, which, if retained, would lead to wasted computation. To safeguard against this, all edges and nodes are flagged for forward reachability, *i.e.*, there is a path to the edge or node from an enabled input node, and backward reachability, *i.e.*, there is a path from the node or edge to any output. Nodes and edges which are not forward and backward reachable are labeled as vestigial and are disabled. They can, however, later be reconnected and enabled via EXAMM's mutation/crossover operations, essentially boostrapping the learning process by allowing to easily reuse previously learned structure. This process is formalized in Algorithms 1 and 2.

### 3.2 Network-Aware Adaptive Structure Transfer Learning

While ASTL had some preliminary success in transferring RNNs for time series modeling tasks, it only added connections between input and output nodes while ignoring the internal latent structure of the network being transferred. Furthermore, it re-initialized all weights in the network, only retaining the source network's structure. We extend ASTL to *network-aware* adaptive structure transfer learning (N-ASTL), which utilizes information about the seed network to improve the transfer learning process. Our hypothesis is that during the process of evolving and training an RNN being used as a seed network, the RNN already contains useful information about the form of its topology as well as weight distribution information which can be used to inform the transfer learning process. As such, N-ASTL leverages knowledge of a seed network's connectivity and weight distribution to connect new input and output nodes. N-ASTL involves three strategies:

#### 3.2.1 *Epigenetic Weight Initialization.*

In ASTL, new node biases and edge weights were initialized uniformly at random, $\mathcal{U}(-0.5, 0.5)$, similar to how EXAMM initializes weights in the minimal seed genome. In N-ASTL, before adapting any structure, the mean, $\mu_w$, and standard deviation, $\sigma_w$, of the seed network's weights are computed. Afterwards, when new edges are generated during the seed network adaption process, weights are initialized to a dynamic normal (Gaussian) distribution driven by $\mu_w$ and $\sigma_w$, or $\mathcal{N}(\mu_w, \sigma_w)$. This mirrors how EXAMM performs epigenetic/Lamarckian weight and bias initialization when performing mutation and crossover operations.

#### 3.2.2 *Output-Aware Input Connection.*

Algorithm 3 presents the output-aware input connection procedure for N-ASTL. Similar to our dynamic weight initialization scheme, before adapting any structure of the seed network, the mean, $\mu_o$, and standard deviation, $\sigma_o$, of the number of outputs that each input and hidden node has in the network are calculated. Following this, the unused input and output nodes are then removed from the seed network, with any resulting vestigial hidden nodes and edges appropriately disabled. Following this, the new output and input nodes are added to the network. Each new input node is connected to either output nodes or enabled hidden nodes with a number of connections that is randomly selected according to a Gaussian distribution but with the restriction that at least one connection must be made, *i.e.*, $max(1, \mathcal{N}(\mu_o, \sigma_o))$. This ensures that all input nodes are connected

**Algorithm 3** N-ASTL Seed Network Adaptation: Output-Aware Input Connection

```
function NASTL-Inputs(SeedNetwork sn, Param[] targetOutputs, Param[] targetInputs)
    RemoveUnused(sn, targetOutputs, targetInputs)
    $\mu_w \leftarrow$ sn.getWeightMean()
    $\sigma_w \leftarrow$ sn.getWeightStdDev()
    $\mu_o \leftarrow$ sn.getMeanOutputs()
    $\sigma_o \leftarrow$ sn.getStdDevOutputs()
    ▷ Connect the new input nodes
    for all InputNode i in sn.inputNodes do
        if i.param in targetInputs then
            $nInputs \leftarrow max(1, \mathcal{N}(\mu_o, \sigma_o))$
            Node[] nodes $\leftarrow$ sn.getEnabledHiddenNodes() $\cup$ sn.getOutputNodes()
            shuffle(nodes)
            for $j \leftarrow 1$ to $nInputs$ do
                sn.addEdge(i, nodes[j], $weight \leftarrow \mathcal{N}(\mu_w, \sigma_w)$)
```

to the seed network in a functional way that also follows a similar distribution to the RNN's existing structure.

*3.2.3 Input-Aware Output Connection.* Algorithm 4 presents the input-aware output connection procedure for N-ASTL. New output nodes are connected in a similar fashion to the new input nodes. Before any adaptation, the mean, $\mu_i$, and standard deviation, $\sigma_i$, of the number of inputs that each output and hidden node has in the network is calculated. After removing unused input and output nodes (along with disabling vestigial edges and hidden nodes) and then adding the new input and output nodes, output nodes are potentially wired to any input node or enabled hidden node. The number of connections is then sampled from a Gaussian distribution over the number of inputs, with again the restriction that at least one connection is made, i.e., $max(1, \mathcal{N}(\mu_i, \sigma_i))$.

**Algorithm 4** N-ASTL Seed Network Adaptation: Input-Aware Output Connection

```
function NASTL-Outputs(SeedNetwork sn, Param[] targetOutputs, Param[] targetInputs)
    RemoveUnused(sn, targetOutputs, targetInputs)
    $\mu_w \leftarrow$ sn.getWeightMean()
    $\sigma_w \leftarrow$ sn.getWeightStdDev()
    $\mu_i \leftarrow$ sn.getMeanInputs()
    $\sigma_i \leftarrow$ sn.getStdDevInputs()
    ▷ Connect the new output nodes
    for all OutputNode o in sn.outputNodes do
        if o.param in targetOutputs then
            $nOutputs \leftarrow max(1, \mathcal{N}(\mu_i, \sigma_i))$
            Node[] nodes $\leftarrow$ sn.getEnabledHiddenNodes() $\cup$ sn.getInputNodes()
            shuffle(nodes)
            for $j \leftarrow 1$ to $nOutputs$ do
                sn.addEdge(nodes[j], o, $weight \leftarrow \mathcal{N}(\mu_w, \sigma_w)$)
```

## 4 TRANSFER LEARNING TASKS

To compare to prior state of the art results reported for the original ASTL, we utilize the same open source aviation based dataset which has been provided as part of the EXAMM github repository[2]. The source data for this transfer learning problem consists of 36 flights gathered from the National General Aviation Flight Information Database[3]. It includes three different airframes, with 12 flights coming from Cessna 172 Skyhawks (C172s), 12 from Piper PA-28 Cherokees (PA28s), and the last 12 from Piper PA-44 Seminoles (PA44s). Each of the 36 flights came from a different aircraft. The different airframes have significant design differences (see Figure 1). The C172s are "high wing" (wings are on the top) with a single engine, the PA28s are "low wing" (wings are on the bottom) and have a single engine, and the PA44s are low wing with dual engines. The flight data consists of per second readings and the duration of each flight data file ranges from 1 to 3 hours. Each airframe shares 18 common sensor parameters, C172 and PA44 add 7 additional sensor parameter which the PA28 does not have, C172s have 3 additional engine parameters which PA-28s and PA-44s do not have, and PA-44s add an additional 11 parameters, mostly related to its second engine, which C172s and PA-28s do not have. All available parameters were used as inputs and Appendix A provides a detailed tabular description of which sensors each airframe has and which were used as prediction outputs (if available). Figure 2 provides an example of the data being predicted showing the pitch and E1 EGT1 values from PA28 flight 8, illustrating the challenges involved. The data is highly noisy with sudden changes, non-stationary, non-seasonal, and has varying correlations to the other input parameters. Due to this, traditional statistical methods, *e.g.*, from the auto-regressive integrated moving average (ARIMA) family of models are not well suited to the task.

## 5 RESULTS

### 5.1 Hyperparameter Settings

All EXAMM neuro-evolution runs utilized 4 islands, each with a max population size of 10. New RNNs were generated via mutation at a rate of 70%, intra-island crossover at a rate of 20%, and inter-island crossover occurred at a rate of 10%. 10 out of EXAMM's 11 mutation operations were utilized (all except for *split edge*), each chosen with a uniform 10% chance. EXAMM generated new nodes by selecting from simple neurons, Δ-RNN, GRU, LSTM, MGU, and UGRNN memory cells uniformly at random. Recurrent connections could span any time-skip generated between $\mathcal{U}(1, 10)$.

All RNNs were locally trained for 4 epochs via stochastic gradient descent (SGD) using backpropagation through time (BPTT) [22] to compute gradients, all using the same hyperparameters. RNN weights were initialized by EXAMM's Lamarckian strategy (described in [15]), which allows child RNNs to reuse parental weights, significantly reducing the number of epochs required for the neuroevolution's local RNN training steps. SGD was run with a learning rate of $\eta = 0.001$ and used Nesterov momentum with $\mu = 0.9$. For the memory cells with forget gates, the forget gate bias had a value

[2]https://github.com/travisdesell/exact/tree/master/datasets/2019_ngafid_transfer
[3]http://ngafid.org

(a) Cessna 172 Skyhawk

(b) Piper PA-28 Cherokee

(c) Piper PA-44 Seminole

**Figure 1: The data used for transfer learning comes from three different airframes (images under creative commons licenses).**

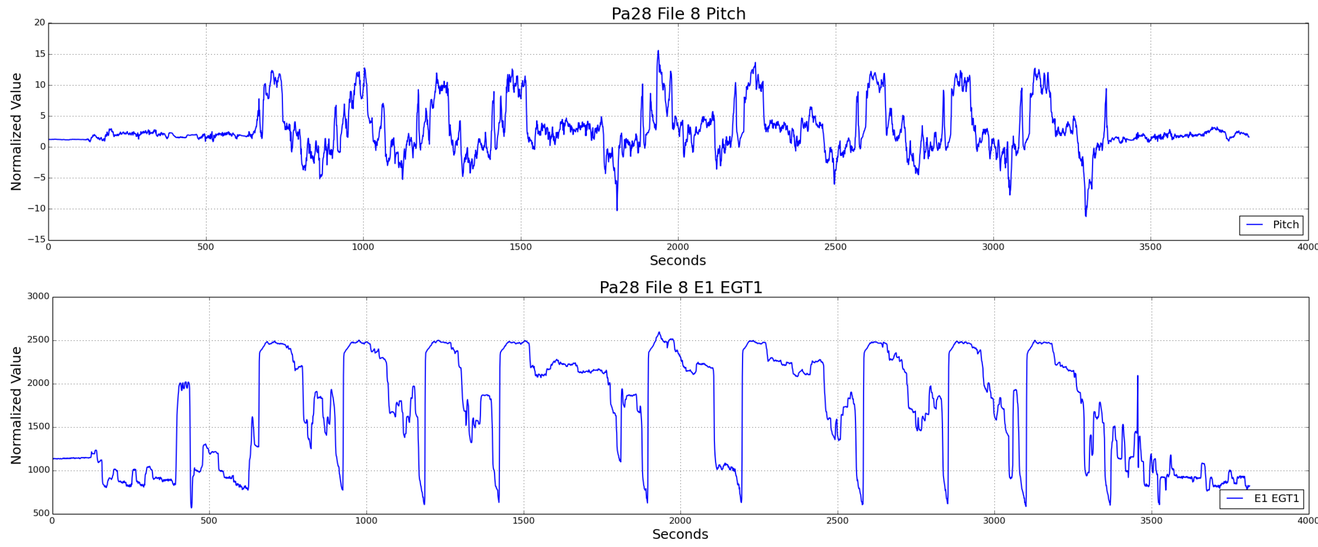


**Figure 2: Example of the pitch and E1 EGT1 parameters from PA28 flight 8 in the NGAFID dataset.**

of 1.0 added to it (motivated by [10]). To prevent exploding gradients, gradient clipping [17] was used when the norm of the gradient exceeded a threshold of 1.0. To combat vanishing gradients, gradient boosting (the opposite of clipping) was used when the gradient norm was below 0.05. These parameters have selected by hand tuning during prior experience with the EXAMM algorithm and this data set.

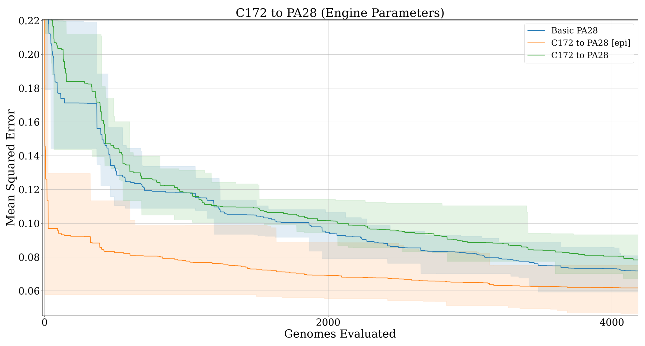
| Task | Inputs Added | Inputs Removed | Outputs Added | Outputs Removed |
|---|---|---|---|---|
| PA28 to PA44 | 13 | 0 | 4 | 0 |
| PA28 to C172 | 8 | 0 | 3 | 0 |
| C172 to PA28 | 0 | 8 | 0 | 3 |
| C172 to PA44 | 10 | 3 | 4 | 0 |
| PA44 to PA28 | 0 | 13 | 0 | 7 |
| PA44 to C172 | 3 | 10 | 0 | 7 |

**Table 1: Transfer Learning Tasks**

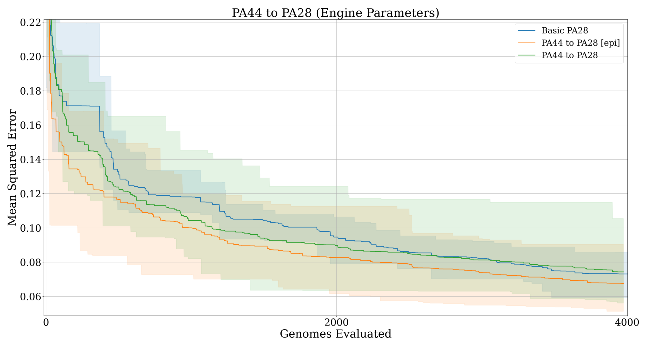
## 5.2 Experimental Design

One major goal of this work is to facilitate and enable the fast development of predictive systems. In the realm of aviation, this could mean that a given organization may operate a fleet of aircraft of certain airframes and employ RNN estimators as part of their predictive systems. Instead of having to train new RNNs from scratch every time existing airframes are modified, new airframes start being utilized, or sensor systems are upgraded, they can instead adapt RNNs trained on their existing airframes and transfer them over for use in these new or modified systems. This transfer process would also require less data compared to the scenario of training estimators/predictors from scratch. To mirror such a scenario, we used EXAMM to evolve and train RNNs on each of the three airframes (C172s, PA28s and PA44s) for 4, 000 generated and trained RNNs, *i.e.* 4, 000 evaluated genomes. This was repeated 10 times for each airframe. The flight data files were split up into training and validation data, with the first 9 used as training data and last 3 used as validation data.

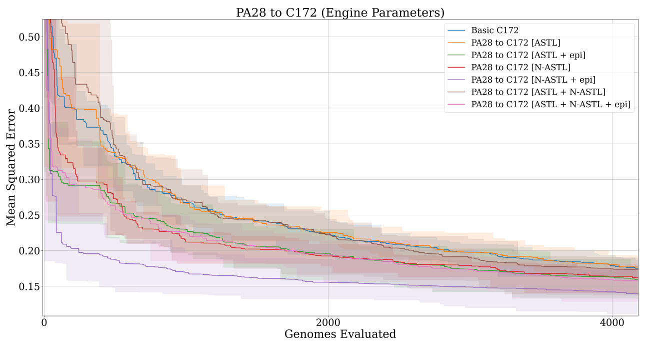
To appropriately evaluate our proposed N-ASTL methodology, we utilized the same prediction task defined in [1], which was to predict the exhaust gas temperature (EGT) engine parameters for the target airframe, as well as a new task predicting non-engine



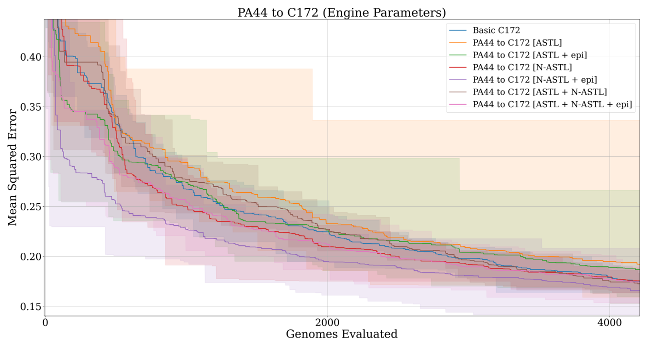
(a) C172 to PA28



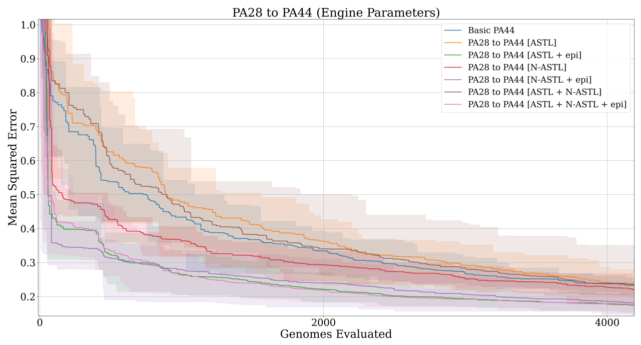
(b) PA44 to PA28



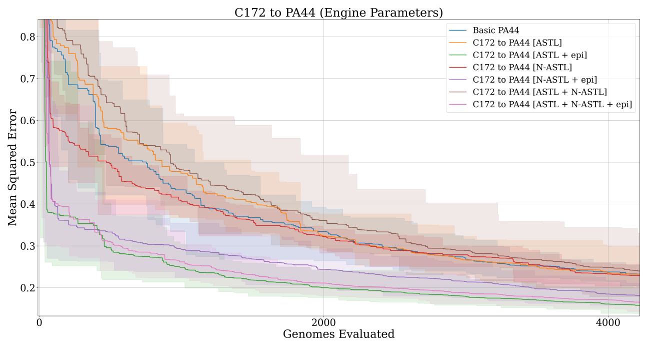
(c) PA28 to C172



(d) PA44 to C172



(e) PA28 to PA44

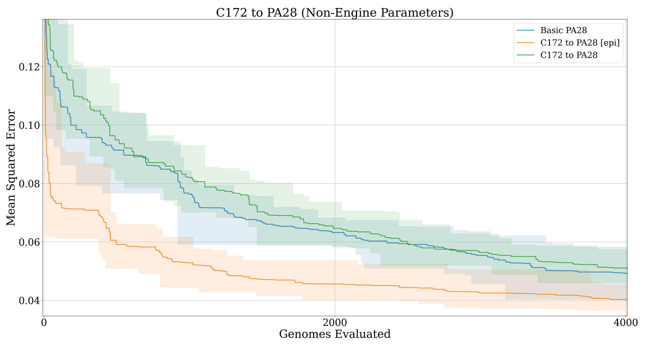


(f) C172 to PA44

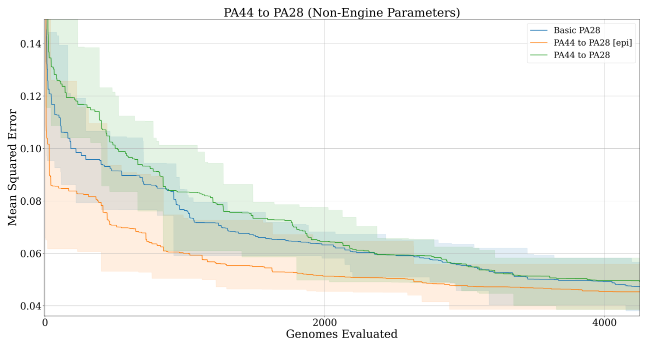
**Figure 3: Convergence rates (in terms of best MSE on validation data) for the EXAMM runs starting from scratch (Basic C172, PA28 or PA44) compared to starting with a seed network transferred from a different airframe when predicting engine exhaust gas temperature (EGT) values.**

parameters. RNNs predicting on PA28 would predict engine 1 (E1) EGT1, those for C172s would predict E1 EGT1-4, and RNNs predicting on PA44 data would predict both E1 EGT1-4 and engine 2 (E2) EGT1-4. The non-engine prediction parameters were Altitude Miles Above Sea Level (AltMSL), Indicated Air Speed (IAS), Lateral Acceleration (LatAc), Normal Acceleration (NormAc), Pitch and Roll. We investigated the transfer learning methods using each airframe as a source and target, resulting in six different transfer learning examples: C172 to PA28, C172 to PA44, PA28 to C172, PA28 to PA44, PA44 to C172, and PA44 to PA28.

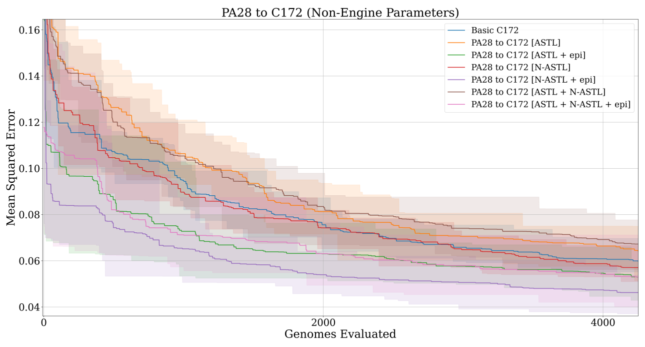
Table 1 presents the different transfer learning tasks examined, along with how many input and output parameters were and removed in each example. Input nodes were added or removed by the previously described strategies to utilize all available sensor inputs for the target data. Likewise, outputs were added or removed to predict all the available EGT parameters. For the non-engine parameters no outputs needed to be added or removed. For each of the 10 repeated runs, the best genome in each set after the 4,000 genome evaluations was selected to be used as a seed network for the transfer learning strategies. For the 3 airframes, the 10 selected seed



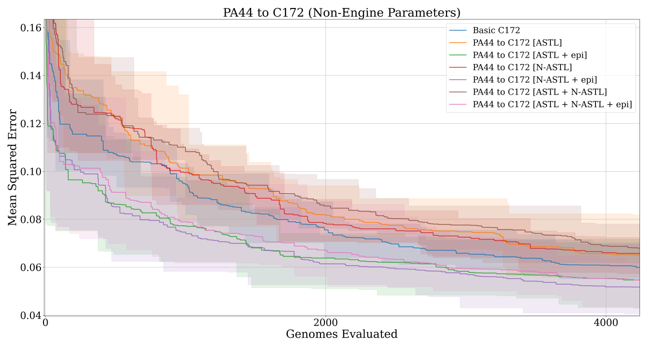
**(a) C172 to PA28**



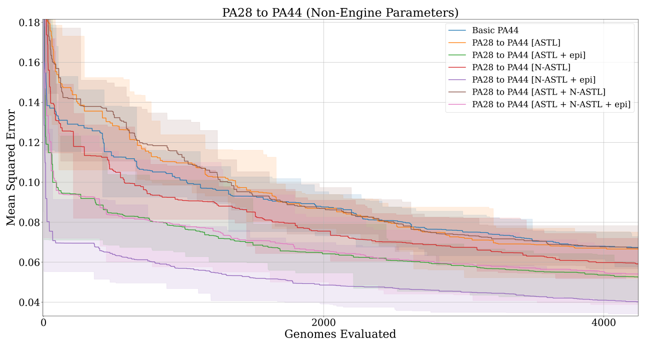
**(b) PA44 to PA28**



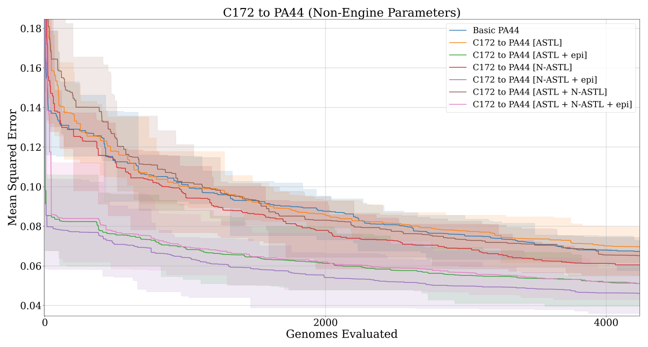
**(c) PA28 to C172**



**(d) PA44 to C172**



**(e) PA28 to PA44**



**(f) C172 to PA44**

**Figure 4: Convergence rates (in terms of best MSE on validation data) for the EXAMM runs starting from scratch (Basic C172, PA28 or PA44) compared to starting with a seed network transferred from a different airframe predicting the non-engine parameters (AltAGL, IAS, LatAc, NormAc, Pitch and Roll).**

networks were then utilized to evaluate the different adaptation strategies. We examined using ASTL and N-ASTL independently, as well as together (denoted as `ASTL + N-ASTL`) for connecting the new input and output nodes. For each of these three strategies, we used either ASTL weight initialization or N-ASTL epigenetic weight initialization (denoted with `+epi`). For the runs where PA28 was the target, as we only removed inputs and outputs, we only examined ASTL with and without epigenetic weight initialization, as there were no new nodes to connect.

## 5.3 Structural Adaptation Evaluation

Figures 3 and 4 compare the different seed network adaptation strategies and their combinations using each of the airframes (C172, PA28 and PA44) as a source to be transferred to each of other airframes (the targets) evolved to predict the engine parameter values and non-engine parameter values, respectively.

For the experiments using the PA28 data as a target, as no input or output nodes were added so transfer learning was tested with and without the N-ASTL epigenetic weight initialization. In all four cases, epigenetic weight initialization had strong error reduction and learned the task much quicker. Additionally, improved results were seen transferring from the PA44 RNNs as a source, where the prior ASTL results had failed.

For the experiments using the C172 data as a target, again large improvements were seen utilizing epigenetic weight initialization. In all cases, `N-ASTL+epi` learned the quickest and resulted in the lowest error, with `N-ASTL+ASTL+epi` providing the next best results followed by `ASTL+epi`. Even without epigenetic weights, N-ASTL performed the strategies involving ASTL connectivity.

For the experiments using the PA44 data as a target, N-ASTL shows its significance very strongly. Prior results with ASTL were unable to improve over utilizing EXAMM from scratch on the PA44 data, while in this case, even without epigenetic weights N-ASTL is still able to improve over EXAMM from scratch. Additionally, the use of epigenetic weights shows a very large improvement, with all three versions showing significant improvement over N-ASTL. In three of the four cases, again `N-ASTL+epi` learned the fastest and found the most accurate results, except in the case of C172 to PA44 on the engine parameter predictions, where `ASTL+epi` and `N-ASTL+ASL+epi` performed slightly, but comparatively better.

### 5.4 Conclusions

Overall, we find these results strongly positive since they show that the N-ASTL strategies provide significantly better performance over all the experiments. Furthermore, in almost every case, the N-ASTL strategies had better performing RNNs after $2,000$ genomes compared to the best found after $4,000$ genomes when evolving RNNs from scratch. Additionally, if we look at the curvature of these plots, the RNNs evolved from scratch on the ASTL and non-epigenetic weight tests were converging to performance which would never reach that found by the N-ASTL runs. This suggests that transferring RNNs trained on other data and seeding them with genomes in a network aware manner is leading to more generalizable RNN architectures.

Epigenetic weight initialization significantly improved the results in all cases, showing that utilizing network aware weight distributions for weight initialization is highly important for the transfer learning process. Additionally, in 6 of the 8 cases which added or removed inputs and outputs (*i.e.*, those transferring to C172 or PA44), N-ASTL without epigenetic weight initialization also outperformed training from scratch, suggesting that utilizing network aware topology information is also important to the process. This is further backed by the fact that in 6 of those 8 cases the `N-ASTL+epi` runs provided the best results. The two cases they did not, PA28 to PA44 and C172 to PA44, were tests where additional outputs were added, which suggests that when additional outputs are added making sure they are connected to all inputs is important, whereas making sure new inputs are connected to all existing outputs is not.

## 6 DISCUSSION

This work investigates the use of a novel network-aware adaptive structure transfer learning strategy (N-ASTL) to further speed transfer learning of deep RNNs. N-ASTL utilizes statistical information about the source RNN's topology and weight distributions to inform how it should be adapted to new data sets which have different input and output parameters, necessitating the use of a different neural architecture. These strategies were evaluated using the challenging real world problem of performing transfer learning of RNNs trained to predict aviation engine parameters between three different airframes with different designs and engines.

N-ASTL provided significant performance improvements over prior state of the art, which did not take into account network topology or weight distributions. Further, N-ASTL was shown to be able to successfully perform transfer learning on tasks where transfer learning was not previously possible. Interestingly enough, this work shows that in many cases the transfer learning strategies are able to evolve RNNs that outperform ones which started from scratch but were evolved and trained on the target dataset for twice as long. In many cases, the curvature of those plots suggest they would never reach the performance of the RNNs seeded by a transferred network. This suggests that the transfer learning strategy is able to evolve more robust and generalizable RNNs, as performance of the non-transferred RNNs levels off at a significantly lower accuracy than the transfer learning evolved RNNs. These results are significant and showcase the use of transfer learning as a means to enhance predictive systems in aviation, with applications to other domains involving time series data of different input/output dimensionalities.

This study also opens up a number of directions for future work. While N-ASTL only seeds EXAMM with a single adapted network structure, the manner in which it connects new inputs and outputs is stochastic. This provides the potential to generate multiple seed network candidates to provide more initial variety to EXAMM's island populations which could lead to improved reliability and robustness when searching for optimal RNN architectures. Additionally, there appears to be a difference between adding inputs to adding outputs. While the tests which only added inputs but not outputs had N-ASTL with epigenetic weights achieving the best results, the tests which added outputs (those transferring to PA44) showed better performance when also adding in ASTL connectivity. It is worth examining if using N-ASTL plus a modified version of ASTL which only fully connects new outputs to inputs may prove better in these cases. Further, N-ASTL has shown that connecting new inputs and outputs to nodes in the hidden layers is important. It is worth further study to see if simply connecting new inputs and outputs to *all* hidden nodes provides any benefit.

Lastly, while this work has focused on the challenging problem of time series prediction with RNNs, future work will involve applying N-ASTL to RNN classification tasks, such as natural language processing, to see if transferring between different language dictionaries or word and character embeddings provides similar improvements, as well as to convolutional neural networks, allowing transfer learning between images and output spaces of different shapes and sizes.

## 7 ACKNOWLEDGEMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Combustion Systems under Award Number #FE0031547 and by the Federal Aviation Administration and MITRE Corporation under the National General Aviation Flight Information Database (NGAFID) award.

## REFERENCES

[1] Zimeng Lyu Daniel Krutz Alexander G. Ororbia AbdElRahman ElSaid, Joshua Karns and Travis Desell. 2020. Neuro-Evolutionary Transfer Learning through Structural Adaptation. In *The 23nd International Conference on the Applications of Evolutionary Computation (EvoStar: EvoApps 2020)*. Seville, Spain.

[2] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[3] Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. 2016. Capacity and Trainability in Recurrent Neural Networks. *arXiv preprint arXiv:1611.09913* (2016).

[4] Ratneel Vikash Deo, Rohitash Chandra, and Anuraganand Sharma. 2017. Stacked transfer learning for tropical cyclone intensity prediction. *arXiv preprint arXiv:1708.06539* (2017).

[5] Travis Desell, AbdElRahman ElSaid, and Alexander G. Ororbia. 2020. An Empirical Exploration of Deep Recurrent Connections Using Neuro-Evolution. In *The 23nd International Conference on the Applications of Evolutionary Computation (EvoStar: EvoApps 2020)*. Seville, Spain.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[7] Priyanka Gupta, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff. 2018. Transfer learning for clinical time series analysis using recurrent neural networks. *arXiv preprint arXiv:1807.01705* (2018).

[8] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[9] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[10] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*. 2342–2350.

[11] Message Passing Interface Forum. 1994. MPI: A Message-Passing Interface Standard. *The International Journal of Supercomputer Applications and High Performance Computing* 8, 3/4 (Fall/Winter 1994), 159–416.

[12] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[13] Nikola Mrkšić, Diarmuid O Séaghdha, Blaise Thomson, Milica Gašić, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2015. Multi-domain dialog state tracking using recurrent neural networks. *arXiv preprint arXiv:1506.07190* (2015).

[14] Seongkyu Mun, Suwon Shon, Wooil Kim, David K Han, and Hanseok Ko. 2017. Deep neural network based learning and transferring mid-level audio features for acoustic scene classification. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 796–800.

[15] Alexander Ororbia, AbdElRahman ElSaid, and Travis Desell. 2019. Investigating Recurrent Neural Network Memory Structures Using Neuro-evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '19)*. ACM, New York, NY, USA, 446–455. https://doi.org/10.1145/3321707.3321795

[16] Alexander G. Ororbia II, Tomas Mikolov, and David Reitter. 2017. Learning Simpler Language Models with the Differential State Framework. *Neural Computation* 0, 0 (2017), 1–26. https://doi.org/10.1162/neco_a_01017 arXiv:https://doi.org/10.1162/neco_a_01017 PMID: 28957029.

[17] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*. 1310–1318.

[18] Kenneth Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.

[19] Zhiyuan Tang, Dong Wang, and Zhiyong Zhang. 2016. Recurrent neural network training with dark knowledge transfer. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 5900–5904.

[20] Matthew E Taylor, Shimon Whiteson, and Peter Stone. 2007. Transfer via intertask mappings in policy search reinforcement learning. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM, 37.

[21] Phillip Verbancsics and Kenneth O Stanley. 2010. Evolving static representations for task transfer. *Journal of Machine Learning Research* 11, May (2010), 1737–1769.

[22] Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78, 10 (1990), 1550–1560.

[23] Zhilin Yang, Ruslan Salakhutdinov, and William W Cohen. 2017. Transfer learning for sequence tagging with hierarchical recurrent networks. *arXiv preprint arXiv:1703.06345* (2017).

[24] Seunghyun Yoon, Hyeongu Yun, Yuna Kim, Gyu-tae Park, and Kyomin Jung. 2017. Efficient transfer learning schemes for personalized language modeling using recurrent neural network. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*.

[25] Guido Zarrella and Amy Marsh. 2016. Mitre at semeval-2016 task 6: Transfer learning for stance detection. *arXiv preprint arXiv:1606.03784* (2016).

[26] Ansi Zhang, Honglei Wang, Shaobo Li, Yuxin Cui, Zhonghao Liu, Guanci Yang, and Jianjun Hu. 2018. Transfer Learning with Deep Recurrent Neural Networks for Remaining Useful Life Estimation. *Applied Sciences* 8, 12 (2018), 2416.

[27] Guo-Bing Zhou, Jianxin Wu, Chen-Lin Zhang, and Zhi-Hua Zhou. 2016. Minimal gated unit for recurrent neural networks. *International Journal of Automation and Computing* 13, 3 (2016), 226–234.

## Appendices

### APPENDIX A AVIATION DATASET PARAMETERS

This paper utilizes 12 flight logs each from aircraft of three different airframes: Cessna 172 Skyhawk (C172s), Piper-Archer 28 Cherokees (PA-28s) and Piper-Archer 44 Seminoles (PA-44s). These aircraft share 18 common sensor parameters, and then each has varying sensor parameters for their engine(s). The data files used in this work are freely available as comma separated value (CSV) format as part of the EXAMM github repository[4]. The following table presents which sensors are present for which airframe and which were used as prediction outputs (if available), all available parameters were used as inputs.:

| | Cessna 172 | Piper-Archer 28 | Piper-Archer 44 | Potential Output | |
|---|---|---|---|---|---|
| Parameter Name | Skyhawk | Cherokee | Seminole | (Engine) | (Non-Engine) |
| Altitude Above Ground Level (AltAGL) | x | x | x | | |
| Barometric Altitude (AltB) | x | x | x | | |
| GPS Altitude (AltGPS) | x | x | x | | |
| Altitude Miles Above Sea Level (AltMSL) | x | x | x | | x |
| Fuel Quantity Left (FQtyL) | x | x | x | | |
| Fuel Quantity Right (FQtyR) | x | x | x | | |
| Ground Speed (GndSpd) | x | x | x | | |
| Indicated Air Speed (IAS) | x | x | x | | x |
| Lateral Acceleration (LatAc) | x | x | x | | x |
| Normal Acceleration (NormAc) | x | x | x | | x |
| Outside Air Temperature (OAT) | x | x | x | | |
| Pitch | x | x | x | | x |
| Roll | x | x | x | | x |
| True Airspeed (TAS) | x | x | x | | |
| Vertical Speed (VSpd) | x | x | x | | |
| Vertical Speed Gs (VSpdG) | x | x | x | | |
| Wind Direction (WndDir) | x | x | x | | |
| Wind Speed (WndSpd) | x | x | x | | |
| Absolute Barometric Pressure (BaroA) | x | | x | | |
| Engine 1 Cylinder Head Temperature 1 (E1 CHT1) | x | | x | | |
| Engine 1 Cylinder Head Temperature 2 (E1 CHT2) | x | | | | |
| Engine 1 Cylinder Head Temperature 3 (E1 CHT3) | x | | | | |
| Engine 1 Cylinder Head Temperature 4 (E1 CHT4) | x | | | | |
| Engine 1 Exhaust Gas Temperature 1 (E1 EGT1) | x | x | x | x | |
| Engine 1 Exhaust Gas Temperature 2 (E1 EGT2) | x | | x | x | |
| Engine 1 Exhaust Gas Temperature 3 (E1 EGT3) | x | | x | x | |
| Engine 1 Exhaust Gas Temperature 4 (E1 EGT4) | x | | x | x | |
| Engine 1 Fuel Flow (E1 FFlow) | x | x | x | | |
| Engine 1 Oil Pressure (E1 OilP | x | x | x | | |
| Engine 1 Oil Temperature (E1 OilT) | x | x | x | | |
| Engine 1 Rotations Per minute (E1 RPM) | x | x | x | | |
| Engine 1 Manifold Absolute Pressure (E1 MAP) | | | x | | |
| Engine 2 Cylinder Head Temperature 1 (E2 CHT1) | | | x | | |
| Engine 2 Exhaust Gas Temperature 1 (E2 EGT1) | | | x | x | |
| Engine 2 Exhaust Gas Temperature 2 (E2 EGT2) | | | x | x | |
| Engine 2 Exhaust Gas Temperature 3 (E2 EGT3) | | | x | x | |
| Engine 2 Exhaust Gas Temperature 4 (E2 EGT4) | | | x | x | |
| Engine 2 Fuel Flow (E2 FFlow) | | | x | | |
| Engine 2 Oil Pressure (E2 OilP) | | | x | | |
| Engine 2 Oil Temperature (E2 OilT) | | | x | | |
| Engine 2 Rotations Per minute (E2 RPM) | | | x | | |
| Engine 2 Manifold Absolute Pressure (E2 MAP) | | | x | | |

[4]https://github.com/travisdesell/exact/tree/master/datasets/2019_ngafid_transfer