Low Size, Weight, and Power Neuromorphic Computing to Improve Combustion Engine Efficiency

Catherine D. Schuman, Steven R. Young, J. Parker Mitchell, J. Travis Johnston
Computer Science and Math Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
{schumancd,youngsr,mitchelljp1,johnstonjt@ornl.gov}

Derek Rose
Electrical and Electronics Systems Research Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
rosedc@ornl.gov

Bryan P. Maldonado, Brian C. Kaul
Energy and Transportation Science Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
{maldonadopbp,kaulbc}@ornl.gov

***Abstract*—Neuromorphic computing offers one path forward for AI at the edge. However, accessing and effectively utilizing a neuromorphic hardware platform is non-trivial. In this work, we present a complete pipeline for neuromorphic computing at the edge, including a small, inexpensive, low-power, FPGA-based neuromorphic hardware platform, a training algorithm for designing spiking neural networks for neuromorphic hardware, and a software framework for connecting those components. We demonstrate this pipeline on a real-world application, engine control for a spark-ignition internal combustion engine. We illustrate how we connect engine simulations with neuromorphic hardware simulations and training software to produce hardware-compatible spiking neural networks that perform engine control to improve fuel efficiency. We present initial results on the performance of these spiking neural networks and illustrate that they outperform open-loop engine control. We also give size, weight, and power estimates for a deployed solution of this type.**

***Index Terms*—neuromorphic, FPGA, engine control unit, internal combustion engine**

I. Introduction

Neuromorphic computers provide an intriguing platform for low power artificial intelligence (AI) at the edge [1]. However, there are several key hurdles to developing neuromorphic computing solutions to real-world applications, including finding sufficiently low power neuromorphic hardware, implementing the appropriate algorithms used to train a spiking neural network (SNN) to deploy onto the hardware platform, and ensuring that the trained SNN fits within the constraints of the pre-determined neuromorphic hardware system.

In this work, we present a complete pipeline for training and deploying a low size, weight, and power (SWaP) neuromorphic hardware solution for a real-world application. The real-world application of interest is an engine control unit (ECU) of a spark-ignition internal combustion engine. To improve engine efficiency and reduce greenhouse gas emissions, transportation researchers have developed advanced combustion strategies that minimize the amount of fuel needed to run the engine. However, as the engine efficiency is pushed to the practical limit, instabilities in the combustion process cause sporadic misfires and partial burns. Such events increase the cycle-to-cycle variability (CCV) of the combustion process which causes undesired levels of noise, vibration, and harshness (NVH). In order to mitigate combustion instabilities, the ECU should command a higher amount of fuel during combustion cycles where misfires or partial burns would otherwise occur. In this case, we would like to utilize a neuromorphic hardware system that will take as input information about the engine combustion's state at each cycle and provide as output the amount of fuel to inject during the next cycle. The goal is to keep the engine running with few or no engine misfires and partial burns in order to minimize combustion CCV and keep the engine running smoothly, but also to minimize the amount of extra fuel needed to stabilize the combustion events.

We present how we combined several previous works in a complete workflow to produce a low cost, low power neuromorphic engine controller that could be reasonably deployed in an engine as part of the ECU. This workflow includes a low cost FPGA-based neuromorphic hardware system called μCaspian [2], an SNN training methodology called EONS [3], [4] that integrates with the engine simulation code, and a

neuromorphic software workflow based on the TENNLab software framework [5] to connect each of the components together. In the following sections, we discuss each of these individual components and how they are combined into a single workflow. We show how the resulting trained SNNs perform on the ECU task within engine simulations and estimate their size, weight, and power performance in the deployed neuromorphic hardware.

II. Related Work

Though neuromorphic computing has grown in popularity in recent years, there are relatively few complete workflows for training an SNN and then deploying to existing neuromorphic hardware. Nengo [6] provides a complete workflow with multiple neuromorphic hardware backends, including an FPGA implementation [7] and Intel's Loihi. Nengo also includes backpropagation-based training approaches with NengoDL [8], so it is possible to train an SNN in Nengo that can then be deployed to a neuromorphic system on an FPGA or a system like Loihi. This type of workflow has been utilized to train low power SNNs for tasks such as keyword spotting [9]. However, this training approach does not natively allow for training multiple objectives and is not easily extendable to control tasks (without the use of additional algorithms).

Modern ECUs with increasing computational capacity have enabled the use of model-based control strategies in advanced combustion modes such as stoichiometric diluted combustion [10], homogeneous charge compression ignition (HCCI) [11], partially premixed combustion [12], and reactivity controlled compression ignition (RCCI) [13] among others. Such closed-loop combustion control strategies, however, require physics-based control-oriented models accurate enough for model-based control but computationally low cost for real-time implementation. Given the restrictive environment of the combustion chamber and the complex physical phenomena occurring during the process, simplifications in modeling are unavoidable. To overcome these limitations and fully utilize the information contained in empirical engine data, model-free control based on learning has increasingly gained popularity in the engine research community [14].

III. Approach

A. Engine Simulators

The engine simulator is based on the control-oriented cycle-to-cycle combustion model proposed by Daw et al. [15]:

$$\begin{bmatrix} M_{\text{fuel}} \\ M_{\text{air}} \\ M_{\text{inert}} \end{bmatrix}_{k+1} = X_{\text{res}}[k] \begin{bmatrix} 1-\eta_c[k] & 0 & 0 \\ -\text{AFR}_{\text{s}}\eta_c[k] & 1 & 0 \\ (1+\text{AFR}_{\text{s}})\eta_c[k] & 0 & 1 \end{bmatrix} \begin{bmatrix} M_{\text{fuel}} \\ M_{\text{air}} \\ M_{\text{inert}} \end{bmatrix}_k + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} m_{\text{fuel}}[k] + \begin{bmatrix} 0 \\ m_{\text{air}} \\ m_{\text{EGR}} \end{bmatrix} \tag{1}$$

$$Q_{\text{gross}}[k] = \eta_c[k] M_{\text{fuel}}[k] Q_{\text{LHV}} \tag{2}$$

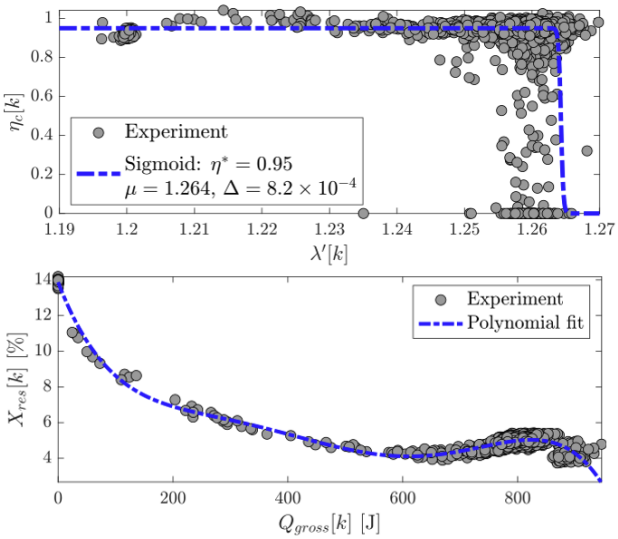

Fig. 1. Top: cycle-to-cycle values of combustion efficiency $\eta_c[k]$ as function of in-cylinder composition $\lambda'[k] = \frac{M_{\text{air}}[k] + M_{\text{inert}}[k]}{M_{\text{fuel}}[k]} \cdot \frac{1}{\text{AFR}_{\text{s}}}$. Bottom: residual gas fraction $X_{\text{res}}[k]$ as function of gross heat release $Q_{\text{gross}}[k]$.

Here, the states of the system M_{fuel}, M_{air}, and M_{inert} are the total amount of fuel, air, and inert gases for cycle k. The model parameters consist of the stoichiometric air-to-fuel ratio $\text{AFR}_{\text{s}} = 14.85$, the lower heating value of the fuel $Q_{\text{LHV}} = 41$ MJ/kg, the combustion efficiency η_c, and the residual gas fraction X_{res}. The inputs are the injected fuel mass (control) m_{fuel} and the admitted masses of air $m_{\text{air}} = 308.37$ mg and recirculated exhaust gas $m_{\text{EGR}} = 58.03$ mg. The target variable for CVV control is the gross heat release Q_{gross}. Low values of Q_{gross} indicate the occurrence of partial burns and misfires. For offline simulations, the parameters η_c and X_{res} are required to be functions of the states. The authors in [16] identified simple parametric functions corresponding to the blue dash-dotted lines in Figure 1. The variability observed in the empirical data was simulated using additive Gaussian noise. We will refer to this approach as the "simple" model.

A more accurate model, however, can be developed using the kernel density estimation (KDE) technique. Here, the model parameters are sampled from two different conditional density functions based on Gaussian kernels:

$$\eta_c[k] \sim p_{w_\eta}(\eta_c \mid \lambda'_k, h_\eta) \tag{3}$$

$$X_{\text{res}}[k] \sim p_{w_X}(X_{\text{res}} \mid Q_{\text{gross}}[k], h_X) \tag{4}$$

where h_η and h_X are the corresponding bandwidths. Such hyperparameters were chosen using maximum likelihood cross-validation. The inverse cumulative distribution function (CDF) sampling technique was used to obtain η_c and X_{res} from the uniform random variables w_η and w_X, respectively. We will refer to this approach as the "hybrid" model.

Assuming complete observation of the system, each of these simulators produce the following information about the current state of the engine at each cycle: M_{fuel}, M_{air}, M_{inert}. The action m_{fuel} is applied to the simulator by changing the amount of

TABLE I
INITIAL AND NOMINAL VALUES FOR ENGINE SIMULATOR

State/Variable	Label	Value
Initial total in-cylider fuel	$M_{\text{fuel}}[0]$	20.68 mg
Initial total in-cylider air	$M_{\text{air}}[0]$	309.25 mg
Initial total in-cylider inert gas	$M_{\text{inert}}[0]$	77.45 mg
Nominal fuel injected	$m_{\text{fuel},0}$	20.632 mg
Nominal heat release	Q_{norm}	806 J

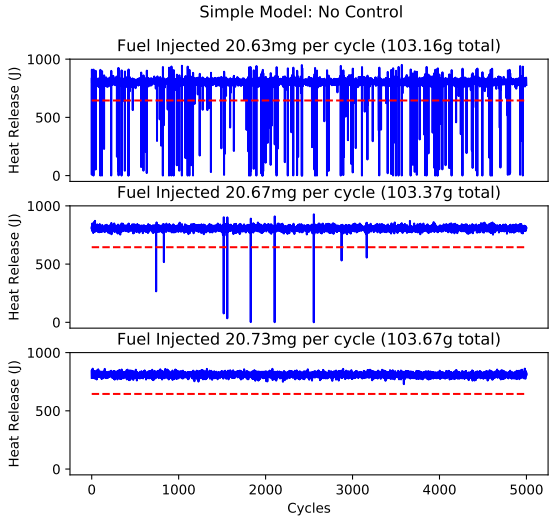

Fig. 2. Heat release values for simulation runs with different fixed fuel injection amounts for the simple model. Values below the red dashed line would likely manifest as partial burns or misfires.

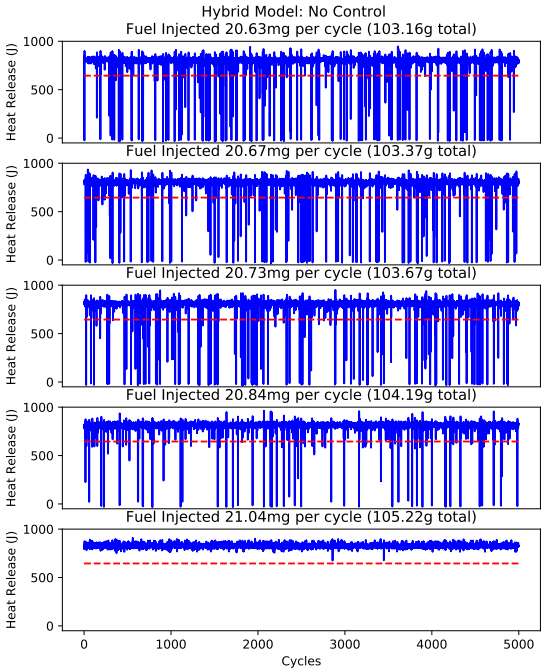

Fig. 3. Heat release values for simulation runs with different fixed fuel injection amounts for the hybrid model. Values below the red dashed line would likely manifest as partial burns or misfires.

fuel injected. The output value of Q_{gross} is used as an indicator of partial burns and misfires. Table I shows the initial values of each state at the start of the simulation as well as the nominal values for the fuel control command and the heat release.

Figures 2 and 3 show examples of the heat release results for different fixed fuel values for the simple and hybrid models. The amount of fuel injected in each cycle is increased from top to bottom. Thus, as can be seen in these figures, by increasing the fixed amount of fuel injected, the number of partial burns and misfires can be reduced and then eliminated. The goal for a closed-loop control system is to decrease or eliminate partial burns and misfires and utilize less fuel than the open-loop, fixed fuel injection approach requires.

B. μCaspian

For both training and deploying a spiking neural network, we use the Caspian neuromorphic platform. Caspian provides both software simulation and hardware implementations using a unified API [2]. In this work, we use the μCaspian hardware architecture which implements a highly efficient event-driven neuromorphic core with 256 integrate-and-fire neurons and 4096 configurable synapses [17].

We have developed a custom board for μCaspian to allow for deployment in real environments. For flexibility, the development board has both USB and direct I/O interfaces. When using the USB interface, the board consumes around 500 mW including I/O and power regulation losses. However, removing USB and extra development circuitry, μCaspian could approach a total power consumption on the order of 10-20 mW. Of that, about 6 mW is necessary for the static and dynamic power of the FPGA logic.

C. Connecting μCaspian and the Engine

To evaluate how an SNN is performing on this engine controller task, we must connect μCaspian with the engine simulator (and, in the future, the engine itself). For this task, the engine simulator (or the engine) will produce a set of observations (M_{fuel}, M_{air}, M_{inert}) that will be used as input to the SNN running on μCaspian. Each of these observations will be numerical values that need to be converted into spikes. We use the TENNLab framework spike encoding methodology to perform this conversion [18]. In this case, we have fixed the spike encoding scheme to utilize 10 bins for each input. All of the observations have a nominal range in which they will appear, $[a, b]$. With 10 bins, these ranges will be split into 10 equal-sized ranges, $[a, a+\frac{(b-a)}{10}]$, $[a+\frac{(b-a)}{10}, a+\frac{2(b-a)}{10}]$, etc. An input neuron will be created for each of these ranges. Then, for a given observation value x, the neuron corresponding to the appropriate range will be spiked once to indicate that value to the network. Because there are three input values, there are a total of 30 input neurons. Neurons in our SNNs are indicated with ID values. Neurons 0-9 correspond to the bins for the amount of fuel observation value, neurons 10-19 correspond to bins for the amount of air observation value, and neurons 20-29 correspond to the inert gas observation value.

TABLE II
FUEL INJECTION ACTIONS PER CYCLE

Output Neuron Index	Amounts	Relationship to $m_{\text{fuel,0}}$
30	20.632 mg	$m_{\text{fuel,0}}$
31	20.642 mg	$m_{\text{fuel,0}}$ + 0.05%
32	20.652 mg	$m_{\text{fuel,0}}$ + 0.1%
33	20.673 mg	$m_{\text{fuel,0}}$ + 0.2%
34	20.735 mg	$m_{\text{fuel,0}}$ + 0.5%
35	20.838 mg	$m_{\text{fuel,0}}$ + 1%
36	21.044 mg	$m_{\text{fuel,0}}$ + 2%
37	21.251 mg	$m_{\text{fuel,0}}$ + 3%
38	21.457 mg	$m_{\text{fuel,0}}$ + 4%

Once the input spikes have been created by the input encoder, they will be sent to μCaspian (either the board itself or the simulator). Then, the SNN on μCaspian will be run for some number of time steps to allow the SNN to decide on an action based on its previous observations. In this case, we run the SNN for 50 time steps, though this value could potentially be changed to improve performance. Similarly to the encoding step, the network on μCaspian will produce spikes as output. These spikes must be converted into a selection of the appropriate action. As noted in the description of the simulator above, the action produced is the amount of fuel to be injected. In this case, we restrict the possible amounts of fuel injected to nine fixed fuel amounts, shown in Table II. These fuel amounts are based on the nominal $m_{\text{fuel,0}}$ value. We use a "winner-take-all" approach to convert output spikes to the corresponding decision. In particular, we create an output neuron for each of the possible actions. Whichever output neuron fires the most corresponds to the chosen action. This conversion of spike counts to actions is performed by an output decoder module, also from the TENNLab software framework. Once the action has been determined, the corresponding action is applied to the engine simulator and a new set of observations are produced that is then provided to the SNN, and so on. This completed system of closed-loop control of the engine using μCaspian is depicted in Figure 4.

Although we currently perform spike encoding and decoding in software, these operations could be integrated into the hardware. The FPGA design can be modified to also include a custom communication interface suitable for sending and receiving real-time engine information. In such a configuration, the decision making process would be self-contained with new data going in and a new action coming out for every time increment.

D. EONS

Evolutionary Optimization for Neuromorphic Systems (EONS) is an evolutionary algorithm-based training approach for SNNs for neuromorphic systems [3], [4]. The EONS algorithm operates by starting with an initial population of potential SNN solutions for the task at hand. Each of the networks are then evaluated using a fitness evaluation to produce a fitness score. The fitness scores are used to drive the selection and reproduction processes in EONS to produce the next population of networks.

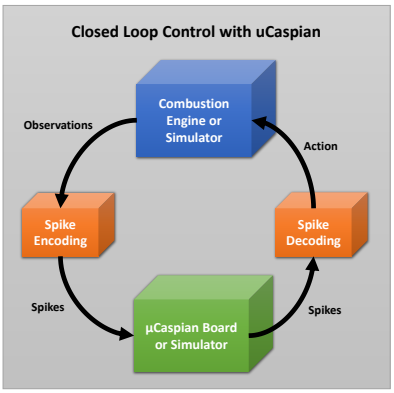

Fig. 4. Closed loop control system, showing how μCaspian and the engine simulator (or engine itself) interact with each other.

In this case, during the fitness evaluation, the candidate SNN solution is simulated on the μCaspian simulator and the car engine is simulated using one of the two engine simulators described above. They are connected to each other via the encoder/decoder modules and the network is simulated controlling the engine for 5000 cycles.

We then use the simulation outputs to calculate a fitness score designed to produce our dual goals of maintaining near nominal engine performance while also minimizing fuel consumption. We heavily penalize misfires and partial burns. To accomplish this, our fitness function is split into three parts. The first is to keep near the normal performance of the engine by maintaining the heat release value close to the nominal value:

$$Q_{error} = \frac{1}{Q_{\text{norm}}^2} \sum_{k=1}^{5000} (Q_{\text{gross}}[k] - Q_{\text{norm}})^2 \tag{5}$$

The second is to minimize fuel usage by keeping the fuel value close to the desired value for the fuel:

$$fuel_{in} = \frac{1}{m_{\text{fuel,0}}^2} \sum_{k=1}^{5000} (m_{\text{fuel}}[k] - m_{\text{fuel,0}})^2 \tag{6}$$

The third is to count the number of cycles where partial burns or misfires occur, where the heat release value is low.

$$N_{bad} = \sum_{k=1}^{5000} g(Q_k) \tag{7}$$

In this case, $g(Q_k)$ refers to:

$$g(Q_k) = \begin{cases} 0 & Q_{\text{gross}}[k] \geq 645 \\ 1 & Q_{\text{gross}}[k] < 645 \end{cases} \tag{8}$$

Finally, since EONS maximizes the fitness function value, and we are interested in minimizing these values, our final fitness function utilizes the negation of each of these. Within this function is a penalty term for each of the three components to weight the importance of those values. In this work, we set

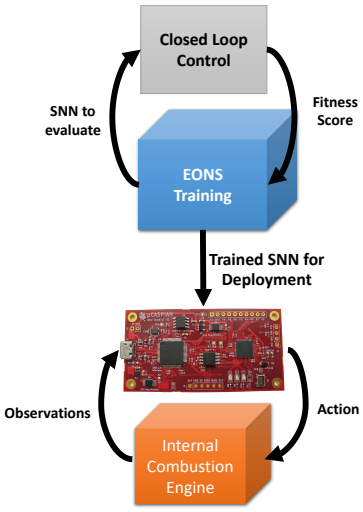

Fig. 5. Full workflow, from training the SNN using EONS to deploying the SNN onto the μCaspian board. The closed loop control box is depicted in more detail in Figure 4.

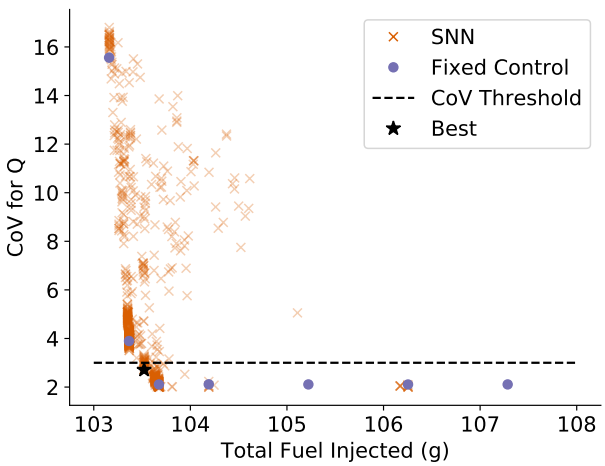

Fig. 6. Coefficient of variation (CoV) and total fuel injected results for the best SNNs for each generation of evolution for all ten runs for the simple engine simulator. The best network is defined as the network with the lowest fuel injected that is also below the CoV threshold of 3 percent (averaged over ten test runs).

$\sigma_Q = 1$, $\sigma_f = 1e4$, and $\sigma_n = 500$. We have determined these values experimentally, but in future work, we intend to investigate the effect of hyperparameter optimization for these penalty values.

$$fitness = -\left(\sigma_Q Q_{error} + \sigma_f fuel_{in} + \sigma_n N_{bad}\right) \quad (9)$$

E. Summary

A diagram showing the complete workflow and all components of the workflow described above is shown in Figure 5. Though we have fixed the approach here, we have previously shown that by tuning the input encoding hyperparameters, one can significantly improve the performance of the SNN [19]. Similarly, by tuning other hyperparameters such as how long the network is evaluated, hardware parameters of the μCaspian board itself, and parameters of the training process we may also significantly improve performance. We intend to perform a hyperparameter optimization process in the future to determine the correct input and output coding parameters.

IV. Results

We ran EONS for 100 generations ten times for both the simple engine model and the hybrid engine model to produce spiking neural networks suitable for deployment on μCaspian for each simulation model. To understand the performance of these networks, we look at two metrics:

- Total fuel injected over all 5000 cycles
- CoV: Coefficient of variation for the heat release value (Q). For this metric, we would like to see values less than three percent (based on industry standards).

As our fitness function is attempting to balance multiple conflicting objectives, we track the best performing networks over the course of evolution. Thus, each run of EONS produces 100 SNNs (the best SNN from each generation). We then evaluate these networks on ten new engine simulation runs. Because the simulations are stochastic, there is some variation in the performance across the ten runs, and we average our two metrics across the ten runs. We compare the performance to approaches with no control system, i.e., those in which the fuel injected at each cycle is fixed (examples of which are shown in Figures 2 and 3 for the simple and hybrid models respectively).

A. Simple Model Results

The results for all 1000 SNNs produced over the course of the ten evolutions for the simple engine simulation model are shown in Figure 6. As can be seen in this figure, many of the networks produced utilize less fuel but reported high CoV values. However, there are also a large cluster of networks towards the lower left corner. We highlight the "best" performing SNN for the simple model. We define "best" as the SNN that utilized the least fuel over the course of simulation and also produced an average CoV (over the ten simulation runs) of at most 3 percent.

A window of 100 cycles of the simulation of the best SNN is shown in Figure 7, illustrating the policy that the SNN utilizes in switching between two fuel injection amounts. We omit showing the full 5000 cycles because the policy is not clear on those plots, but appears to be repeating throughout. The SNN that produces this behavior is shown in Figure 8. As can be seen in this figure, the network model is very small and utilizes a non-traditional structure in which output-to-output, as well as output-to-input connections are allowed. Moreover, there are no hidden neurons required in this network. The network size is 17 neurons and 17 synapses, resulting in a very small, very compact SNN that can allow for an extremely efficient implementation on μCaspian. Additionally, because

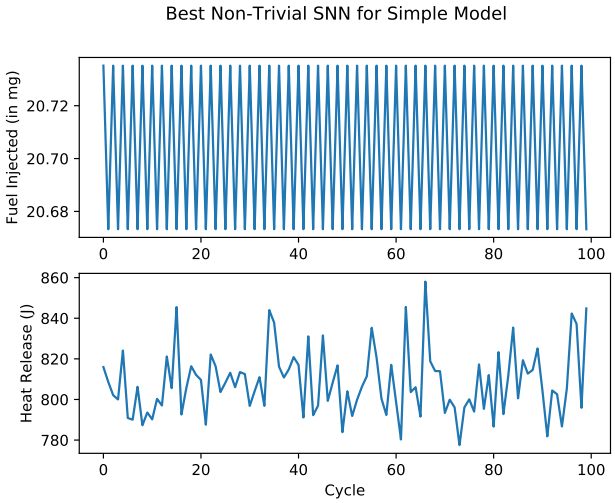

Fig. 7. The first 100 cycles of one simulation as controlled by the best SNN for the simple model. The top plot shows the control policy that the SNN trained to perform.

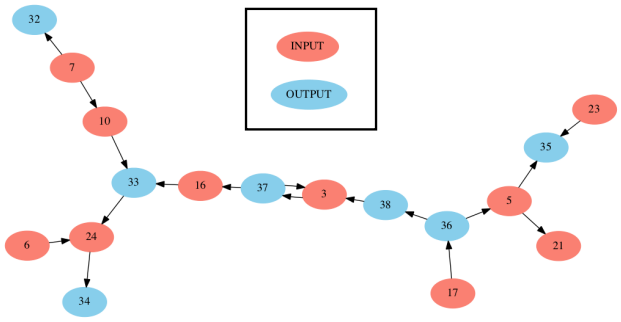

Fig. 8. The best SNN structure evolved by EONS for the simple model. The labels on the neurons indicate their ID in the network. Indices 0-9 encode the fuel observation, indices 10-19 encode the air observation, and indices 20-29 encode the inert gas observation. Output neurons are 30-38 and correspond to the fuel commands shown in Table II.

the μCaspian system is event-driven, the network size roughly corresponds with processing time required. In this case, we would expect a network of this size to require less than 200 microseconds to make a control decision at each engine cycle (not including I/O time).

B. Hybrid Engine Model Results

The results for all 1000 SNNs produced over the course of the ten evolutions for the hybrid engine simulation model are shown in Figure 9. With this more complex simulator, it is clear that it will likely benefit from longer training runs. The networks produced over the course of training do drive towards the Pareto front, but are not as successful in pushing consistently past the performance of the fixed control approach. However, as in the simple model case, there are several networks that utilize less fuel than the best fixed control approach and still perform below three percent for the CoV test. Again, we highlight the best performing network for this task. In this case, the best SNN runs the engine with

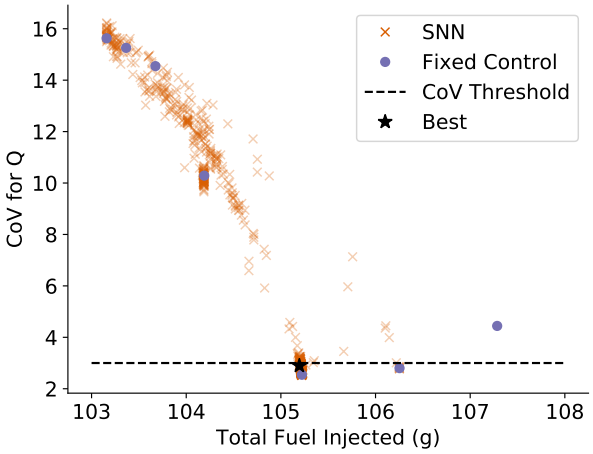

Fig. 9. Coefficient of variation (CoV) and total fuel injected results for the best SNNs for each generation of evolution for all ten runs for the hybrid engine simulator. The best network is defined as the network with the lowest fuel injected that is also below the CoV threshold of 3 percent (averaged over ten test runs).

approximately 26.4 mg less fuel over the course of the 5000 cycles than the fixed control task.

As in the simple case, we show a window of 100 cycles of the simulation of the best SNN produced for the hybrid engine simulator in Figure 10. However, since the policy in this case is not periodic, we show the results for a complete 5000 cycle simulation in Figure 11. As we can see in these figures, there is more variation in the actions that the best network takes over the course of a single simulation and the actions taken are more responsive to the observation values from the engine. In general, we saw that the policies trained for the hybrid model were more complex than those trained for the simple model. This is consistent with what we expected, as the hybrid model itself is more complex and driven by real-world data observations.

The best performing SNN for the hybrid model is shown in Figure 12. Clearly this network is more complex than the best SNN trained for the simple model, but the network still requires no hidden neurons. This network includes a total of 23 neurons and 23 synapses. Similar to the hybrid model, we would expect a network of this size to require less than 200 microseconds to make a control decision at each engine cycle (not including I/O time). Interestingly, in this case we can see that many of the observations integrate together to inform when the higher fuel values should be injected, but the lower fuel values are driven by very little input information. We intend to explore these relationships between observation values further in future work in order to understand how the network is making decisions.

C. Size, Weight, and Power

The SNNs evolved in this work are clearly small enough to fit on the FGPA utilized in the μCaspian implementation. As noted in Section III-B, the full μCaspian development board,

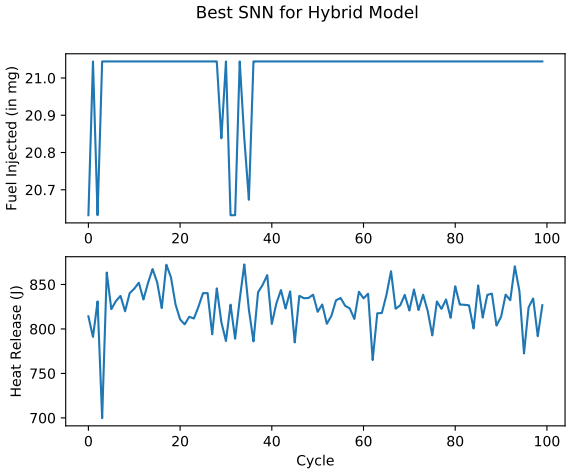

Fig. 10. The first 100 cycles of one simulation as controlled by the best SNN for the hybrid model. The top plot shows the control policy that the SNN trained to perform.

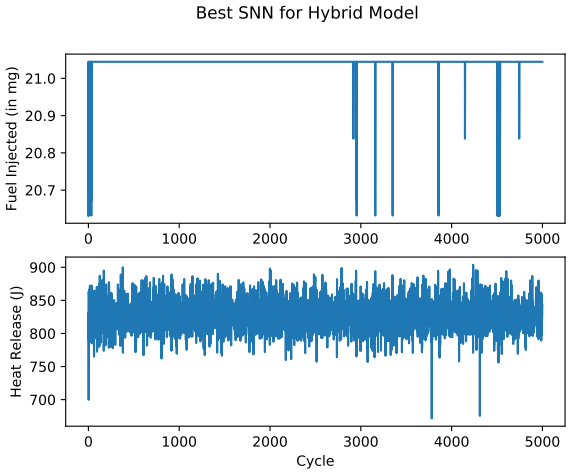

Fig. 11. The full 5000 cycles of one simulation as controlled by the best SNN for the hybrid model. The top plot shows the control policy that the SNN trained to perform.

as shown in Figure 5, utilizes 500 mW, which includes I/O and power regulation losses. This development board forms a "worst-case scenario" for the deployment of this system, as it includes several features for development.

As noted in Section III-B, by removing components associated with the development board, such as USB and extra development circuitry, we estimate that the μCaspian implementation would consume between 10 mW and 20 mW. The FPGA used to implement the μCaspian architecture is the Lattice iCE40 UP5K[1]. The iCE40 UP5K is available in a 2.15 $\times$ 2.55 mm packaging, with a weight of approximately 0.0505 oz. Moreover, these systems are extremely inexpensive and are commercially available, providing a feasible path forward for rapid, low size, weight, and power deployment

[1]https://www.latticesemi.com/en/Products/FPGAandCPLD/iCE40UltraPlus

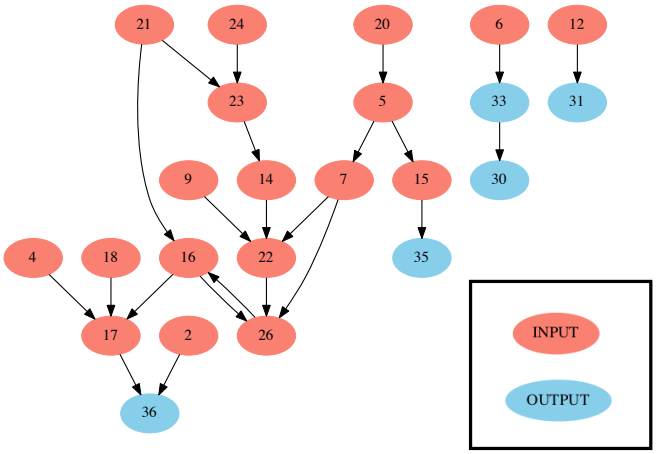

Fig. 12. The best SNN structure evolved by EONS for the simple model. The labels on the neurons indicate their ID in the network. Indices 0-9 encode the fuel observation, indices 10-19 encode the air observation, and indices 20-29 encode the inert gas observation. Output neurons are 30-38 and correspond to the control commands shown in Table II.

of neuromorphic systems on real-world applications.

V. Future Work and Conclusions

In this work, we have demonstrated a complete neuromorphic workflow, from application to hardware, for training and deploying low size, weight, and power neuromorphic solutions for real-world applications. We demonstrated the results of this workflow on a control task to improve fuel efficiency in spark-ignition internal combustion engines. By utilizing low power AI hardware such as neuromorphic systems, we can potentially enable more efficient engines and reduce greenhouse gas emissions. We show that it is feasible to utilize an SNN approach as deployed on neuromorphic hardware to control an engine in simulation. We show that even this preliminary SNN approach can outperform current, open-loop control strategies. Moreover, we demonstrate the resulting SNNs are very small and sparse and can be deployed onto an inexpensive and low size, weight, and power commercial FPGA, providing the opportunity for rapid deployment.

There are several avenues that we intend to pursue for future work. First, the engine simulators we use in this work are based on a single set of engine operating conditions. We intend to apply this same workflow to train SNNs for other engine characteristic settings. We will also investigate whether a single SNN can be optimized to generalize across a variety of engine operating conditions. Additionally, here, we trained and tested on two engine simulators. We intend to utilize these trained SNNs on the μCaspian development board on a real spark-ignition combustion engine at the National Transportation Research Center to evaluate how it performs in a real-world setting.

As noted in Section III, there are several hyperparameters associated with this approach, including hyperparameters for the μCaspian hardware, EONS algorithm, spike encoding

and decoding, and the penalty terms in the fitness function. In this work, we utilize default hyperparameters for EONS and μCaspian, and spike encoding/decoding, and we hand-tuned penalty terms by selecting them from a small set of values. We have previously utilized Bayesian optimization to automatically tune these hyperparameters to improve performance on classification tasks [20]. By optimizing these hyperparameters, we expect that we can significantly improve the performance of the resulting SNNs on the ECU task. We also intend to include this hyperparameter optimization approach in our neuromorphic pipeline in future work to further automate the use and deployment of neuromorphic systems. We have previously investigated the use of multi-objective optimization approaches with EONS to minimize energy or size of SNNs or to improve resiliency [21], as well as utilizing hyperparameter optimization approaches to minimize size, weight, and power [19]. By tuning the hyperparameters, we expect that we may be able to further reduce the size or energy usage of our networks on μCaspian.

Finally, the prototype neuromorphic pipeline we have demonstrated in this work will allow us to quickly develop, evaluate, and deploy low size, weight, and power neuromorphic solutions for real-world applications. We intend to explore other tasks at the National Transportation Research Center, as well as tasks associated with wearable medical devices, such as real-time monitoring and anomaly detection, and tasks associated with smart infrastructure.

ACKNOWLEDGEMENTS

This material is based upon work supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC05-00OR22725, and in part by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC.

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

This research used resources of the Compute and Data Environment for Science (CADES) at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. We would like to thank Chris Layton for his support in our utilization of CADES Cloud.

This research was supported in part by the DOE Office of Energy Efficiency and Renewable Energy (EERE), Vehicle Technologies Office, under the guidance of Gurpreet Singh and Michael Weismiller, and used resources at the National Transportation Research Center, a DOE-EERE User Facility at Oak Ridge National Laboratory.

REFERENCES

[1] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *arXiv preprint arXiv:1705.06963*, 2017.

[2] J. P. Mitchell, C. D. Schuman, R. M. Patton, and T. E. Potok, "Caspian: A neuromorphic development platform," in *Proceedings of the Neuro-inspired Computational Elements Workshop*, 2020, pp. 1–6.

[3] C. D. Schuman, J. S. Plank, A. Disney, and J. Reynolds, "An evolutionary optimization framework for neural networks and neuromorphic architectures," in *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2016, pp. 145–154.

[4] C. D. Schuman, J. P. Mitchell, R. M. Patton, T. E. Potok, and J. S. Plank, "Evolutionary optimization for neuromorphic systems," in *Proceedings of the Neuro-inspired Computational Elements Workshop*, 2020, pp. 1–9.

[5] J. S. Plank, C. D. Schuman, G. Bruer, M. E. Dean, and G. S. Rose, "The tennlab exploratory neuromorphic computing framework," *IEEE Letters of the Computer Society*, vol. 1, no. 2, pp. 17–20, 2018.

[6] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. Voelker, and C. Eliasmith, "Nengo: a python tool for building large-scale functional brain models," *Frontiers in neuroinformatics*, vol. 7, p. 48, 2014.

[7] B. Morcos, "Nengofpga: an fpga backend for the nengo neural simulator," Master's thesis, University of Waterloo, 2019.

[8] D. Rasmussen, "Nengodl: Combining deep learning and neuromorphic modelling methods," *Neuroinformatics*, vol. 17, no. 4, pp. 611–628, 2019.

[9] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith, "Benchmarking keyword spotting efficiency on neuromorphic hardware," in *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, 2019, pp. 1–8.

[10] B. P. Maldonado, K. Zaseck, E. Kitagawa, and A. G. Stefanopoulou, "Closed-Loop Control of Combustion Initiation and Combustion Duration," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 3, pp. 936–950, 2020.

[11] M. Fathi, O. Jahanian, and M. Shahbakhti, "Modeling and controller design architecture for cycle-by-cycle combustion control of homogeneous charge compression ignition (HCCI) engines – A comprehensive review," *Energy Conversion and Management*, vol. 139, pp. 1 – 19, 2017.

[12] L. Yin, G. Turesson, P. Tunestål, and R. Johansson, "Model Predictive Control of an Advanced Multiple Cylinder Engine With Partially Premixed Combustion Concept," *IEEE/ASME Transactions on Mechatronics*, vol. 25, no. 2, pp. 804–814, 2020.

[13] A. Indrajuana, C. Bekdemir, E. Feru, and F. Willems, "Towards Model-Based Control of RCCI-CDF Mode-Switching in Dual Fuel Engines," in *WCX World Congress Experience*. SAE International, apr 2018.

[14] B. P. Maldonado, N. Li, I. Kolmanovsky, and A. G. Stefanopoulou, "Learning reference governor for cycle-to-cycle combustion control with misfire avoidance in spark-ignition engines at high exhaust gas recirculation–diluted conditions," *International Journal of Engine Research*, vol. 0, no. 0, pp. 1–16, 2020.

[15] C. S. Daw, C. E. A. Finney, J. B. Green, M. B. Kennel, J. F. Thomas, and F. T. Connolly, "A Simple Model for Cyclic Variations in a Spark-Ignition Engine," in *1996 SAE International Fall Fuels and Lubricants Meeting and Exhibition*. SAE International, oct 1996.

[16] B. P. Maldonado and B. C. Kaul, "Control-oriented modeling of cycle-to-cycle combustion variability at the misfire limit in si engines," 2020, proceedings of the ASME 2020 Dynamic Systems and Control Conference. To appear.

[17] J. P. Mitchell, C. D. Schuman, and T. E. Potok, "A small, low cost event-driven architecture for spiking neural networks on fpgas," in *International Conference on Neuromorphic Systems 2020*, 2020, pp. 1–4.

[18] C. D. Schuman, J. S. Plank, G. Bruer, and J. Anantharaj, "Non-traditional input encoding schemes for spiking neuromorphic systems," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–10.

[19] M. Parsa, J. P. Mitchell, C. D. Schuman, R. M. Patton, T. E. Potok, and K. Roy, "Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design," *Frontiers in Neuroscience*, vol. 14, p. 667, 2020.

[20] ——, "Bayesian-based hyperparameter optimization for spiking neuromorphic systems," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 4472–4478.

[21] M. Dimovska, T. Johnston, C. D. Schuman, J. P. Mitchell, and T. E. Potok, "Multi-objective optimization for size and resilience of spiking neural networks," in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. IEEE, 2019, pp. 0433–0439.