AIHWKIT

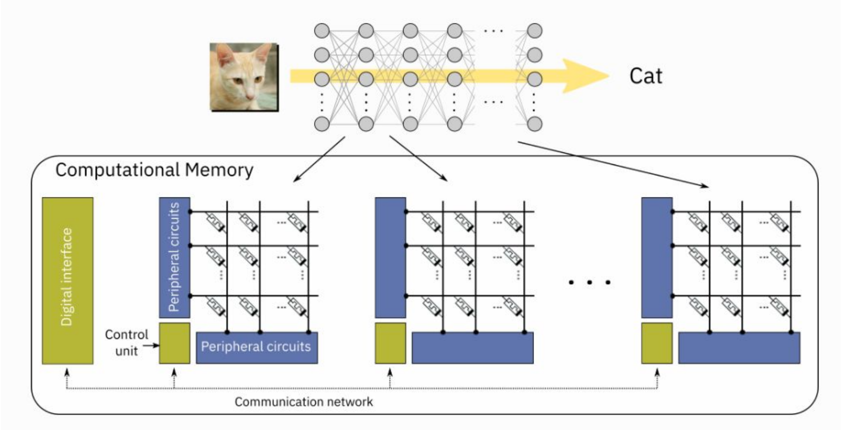
By Henry Ye, Jun Tan, Michael Chen, and Aadit Nagori

Mentor: Dr. Kaoutar El Maghraoui, IBM Research

TABLE OF CONTENTS

Why Analog AI?

- Digital computation creates bottlenecks of memory access speed (called the von Neumann Bottleneck)
- Training a neural network consists of a significant amount of matrix multiplications
- Analog hardware allows for weight values to be stored in an analog conductance state.
- This leads to a constant time complexity as opposed to digital compute which would scale with the size of the matrices

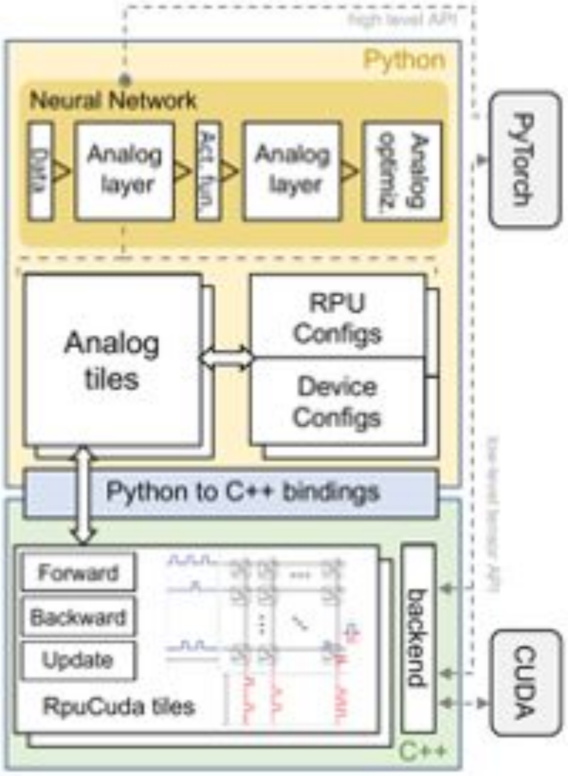


High level: Input activations stored as voltages while weights are stored in conductances

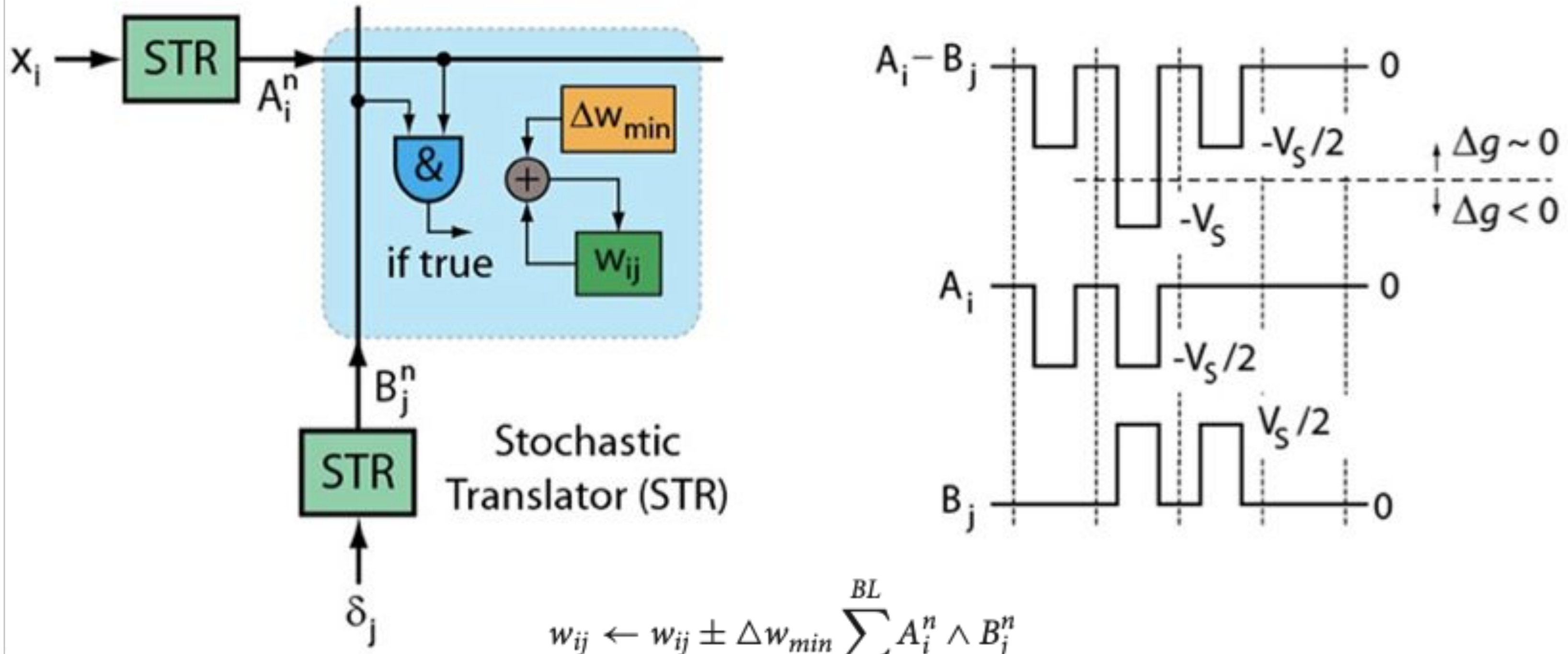
- Allows for multiplication (I = VG) and addition when wires combine via Kirchhoff's junction rule.

What is IBM AIHWKIT?

- An open-source python toolkit for exploring in memory computing capabilities:
  - A series of primitives and features that allows using the toolkit within Pytorch
  - AnalogLinear, AnalogConv2d, …
- A C++, CUDA capable, simulator that enable evaluation of different device and non-idealities:
  - ADC, DAC, out noise scale, programming noise, read noise, drift, drift compensation, device minimum step size, step to step uniformity
- The simulator uses generic Resistive Processing Units (RPUs) to represent each of the elements in the crossbar array
  - These can be configured to simulate the physical properties of various hardware devices like Phase Change Memory



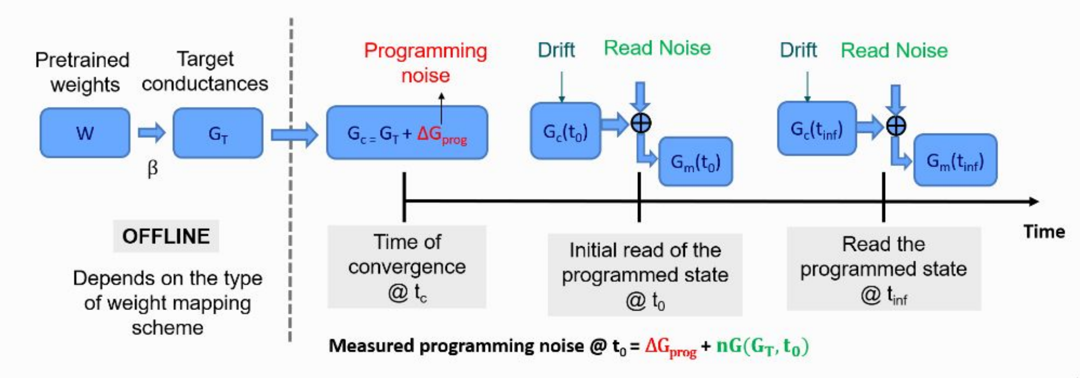
RPU Structure

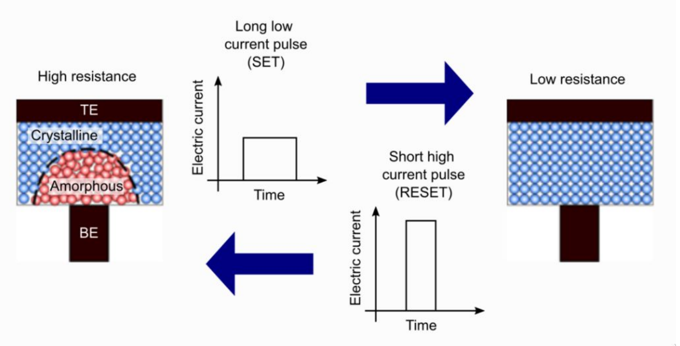


$$w_{ij} \leftarrow w_{ij} \pm \Delta w_{min} \sum_{n=1}^{BL} A_i^n \wedge B_j^n$$

Phase Change Memory

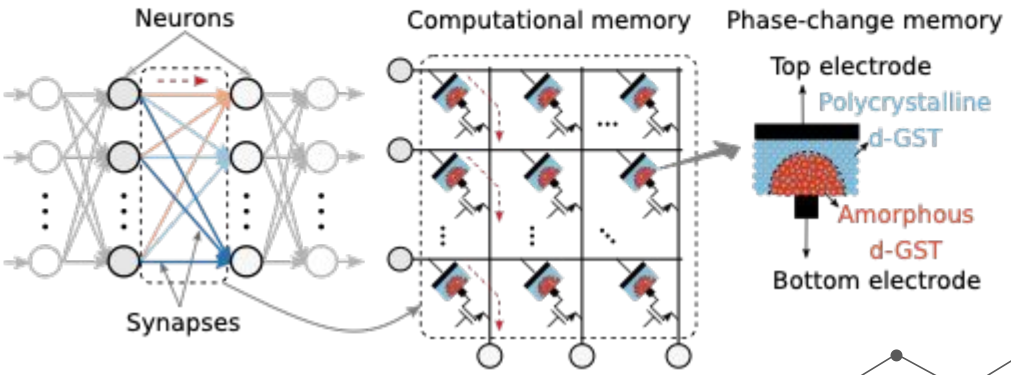
- Phase change memory (PCM) is the most mature c storage-class memory devices
- Consists of phase change material sandwiched between two electrodes
- Resistance/conductance of the material can be controlled via electrical pulses
- Only used for inference, so assumes pretrained weights
- Once they weights have been determined, they are mapped to target conductance values to be programmed onto the device





PCM Material

- As the temperature of the material varies, the state of the material varies from amorphous to crystalline. The change in states of the material will generate electric current increase or decrease, which can be used to store information
  The most commonly used material is Ge2Sb2Te5



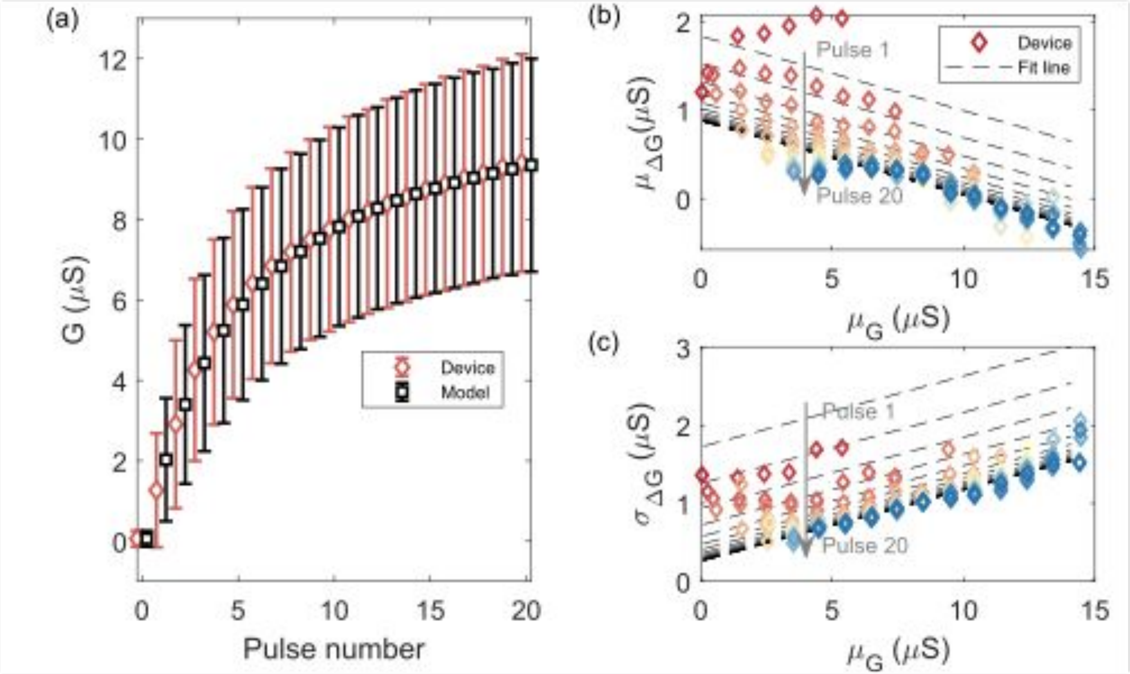
Simulation for the PCM

- The PCM models based on the characterization of a large number of doped-Ge2Sb2Te5 PCM devices.
- The current model is based on measurement of 1 million PCM device that is fabricated at IBM
- The model is able to generate and update corresponding behaviors base on its status Which generalized to the following equation:

$$\mu \Delta G = m1 \mu G + c1 + A1 e^{-p/\alpha}$$

$$\sigma \Delta G = m2 \mu G + c2 + A2 e^{-p/\alpha}$$

Model of PCM



(a)
G (μS)
Pulse number
Device
Model
(b)
$\mu_{\Delta G}$(μS)
μ_G (μS)
Pulse 1
Pulse 20
Device
Fit line
(c)
$\sigma_{\Delta G}$ (μS)
μ_G (μS)
Pulse 1
Pulse 20

PCM Noise + Drift

- ## Programming Noise
  - When programming to the PCM device, there will be some error between the target conductance and the actual value programmed.
  - The noise is calculated via a normal distribution whose standard deviation is based on measurements from hardware

$$g_{\text{prog}} = g_T + \mathcal{N}(0, \sigma_{\text{prog}})$$

$$\sigma_{\text{prog}} = \max\left(-1.1731\, g_T^2 + 1.9650\, g_T + 0.2635, 0\right)$$

- ## Read Noise
  - There also exists intrinsic noise in the PCM device leading to instantaneous fluctuations in the conductances

$$\sigma_{nG}(t) = g_{\text{drift}}(t) Q_s \sqrt{\log \frac{t + t_{\text{read}}}{2t_{\text{read}}}}$$

$$Q_s = \min\left(0.0088/g_T^{0.65}, 0.2\right)$$

$$g(t) = g_{\text{drift}}(t) + \mathcal{N}\left(0, \sigma_{nG}(t)\right)$$

PCM Noise + Drift

- Due to structural relaxations in the amorphous part of the material, the programmed conductances drift over the time since the last programming pulse

- This causes a significant hit to performance

- Can be mitigated with tuning weights with awareness of hardware characteristics of the PCM device, referred to as Hardware-Aware Training

$$g_{\text{drift}}(t) = g_{\text{prog}}\left(\frac{t}{t_c}\right)^{-\nu}$$

Read noise will the subject of experiments showing the effects of drift on inference performance over time

Transformers

- Natural Language Processing algorithm built on neural networks and self-attention mechanisms

- BERT (Bidirectional Encoder Representations from Transformers) is a Transformer pre-trained on finding masked words in a sequence
    - It can then be fine-tuned to perform downstream tasks like sentiment analysis or question answering

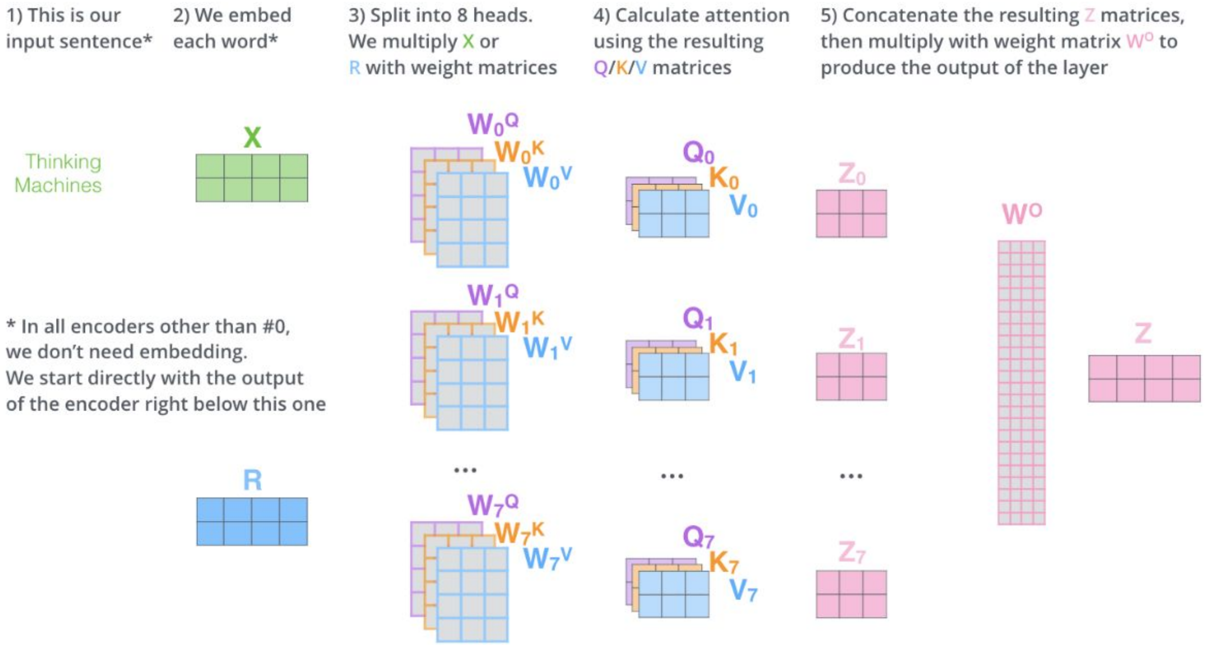


Illustration of Self-Attention mechanism from https://jalammar.github.io/illustrated-transformer/

Question Answering Task

- A reading comprehension dataset consisting of questions from Wikipedia, where answers are fully contained within the context in contiguous blocks of tokens

"5733be284776f41900661181"	"University_of_Notre_Dame"	"Architecturally, the school has a Catholic character. Atop the Main Building's gold dome is a golden statue of the Virgin Mary. Immediately in front of the Main Building and facing it, is a copper statue of Christ with arms upraised with the legend "Venite Ad Me Omnes". Next to the Main Building is the Basilica of the Sacred Heart. Immediately behind the basilica is the Grotto, a Marian place of prayer and reflection. It is a replica of the grotto at Lourdes, France where the Virgin Mary reputedly appeared to Saint Bernadette Soubirous in 1858. At the end of the main drive (and in a direct line that connects through 3 statues and the Gold Dome), is a simple, modern stone statue of Mary."	"What is the Grotto at Notre Dame?"	{ "text": ["a Marian place of prayer and reflection"], "answer_start": [381] }

Tech Stack + Environment setup

- ML Tools:
  - IBM Analog AI Hardware Acceleration Kit
  - PyTorch with CUDA 11.6
  - Hugging Face Libraries:
    - Transformers
    - Tokenizers
    - Datasets
    - Evaluate
- Infrastructure:
  - All experiments run on AiMOS NPL compute nodes
    - SLURM scheduler used to submit jobs to nodes
- Visualization/Tuning Tools:
  - Tensorboard
  - Weights and Biases (wandb)

- Agile software development
  - Miniconda used to manage packages
  - Project Management
    - Slack
    - Github Pull Requests
    - Contributions adhere to PEP8
-

Inference Experiments

- Contribute an example showcasing the use of Transformers in AIHWKIT with drift capabilities
- Took a pre-trained BERT Transformer model and converted it to an analog model
- Configured simulation to mimic the PCM device and observe how drift and different weight noise levels affect the model performance over time
  - Also used configuration mimicking ideal settings to compare to digital baseline
- Metrics tracked are from the Stanford Question and Answering (SQuAD) dataset which include
  - Exact match (percentage of exact answers correct)
  - F1 (harmonic mean of precision and recall)

```python
def do_inference(model, trainer, squad, eval_data, writer, max_inference_time=1e6, n_times=9):
    model.eval()

    metric = load("squad")

    ground_truth = [ { "id": ex["id"], "answers": ex["answers"] } for ex in squad["validation"] ]

    t_inference_list = [0.0] + np.logspace(0, np.log10(float(max_inference_time)), n_times).tolist()

    for t_inference in t_inference_list:
        if not ARGS.digital:
            model.drift_analog_weights(t_inference)

        # Perform inference + evaluate metric here
        raw_predictions = trainer.predict(eval_data)
        predictions = postprocess_predictions(squad["validation"], eval_data, raw_predictions.predictions)

        # Format to list of dicts instead of a large dict
        formatted_preds = [ { "id": k, "prediction_text": v } for k, v in predictions.items() ]

        out_metric = metric.compute(predictions=formatted_preds, references=ground_truth)
        f1, exact_match = out_metric["f1"], out_metric["exact_match"]

        writer.add_scalar("val/f1", f1, t_inference)
        writer.add_scalar("val/exact_match", exact_match, t_inference)

        if ARGS.wandb:
            wandb.log({
                "t_inference": t_inference,
                "f1": f1,
                "exact_match": exact_match,
            })

        exact_match, f1, drift = exact_match, f1, t_inference
        print(f"Exact match: {exact_match: .2f}\t"
              f"F1: {f1: .2f}\t"
              f"Drift: {drift: .2e}")
```

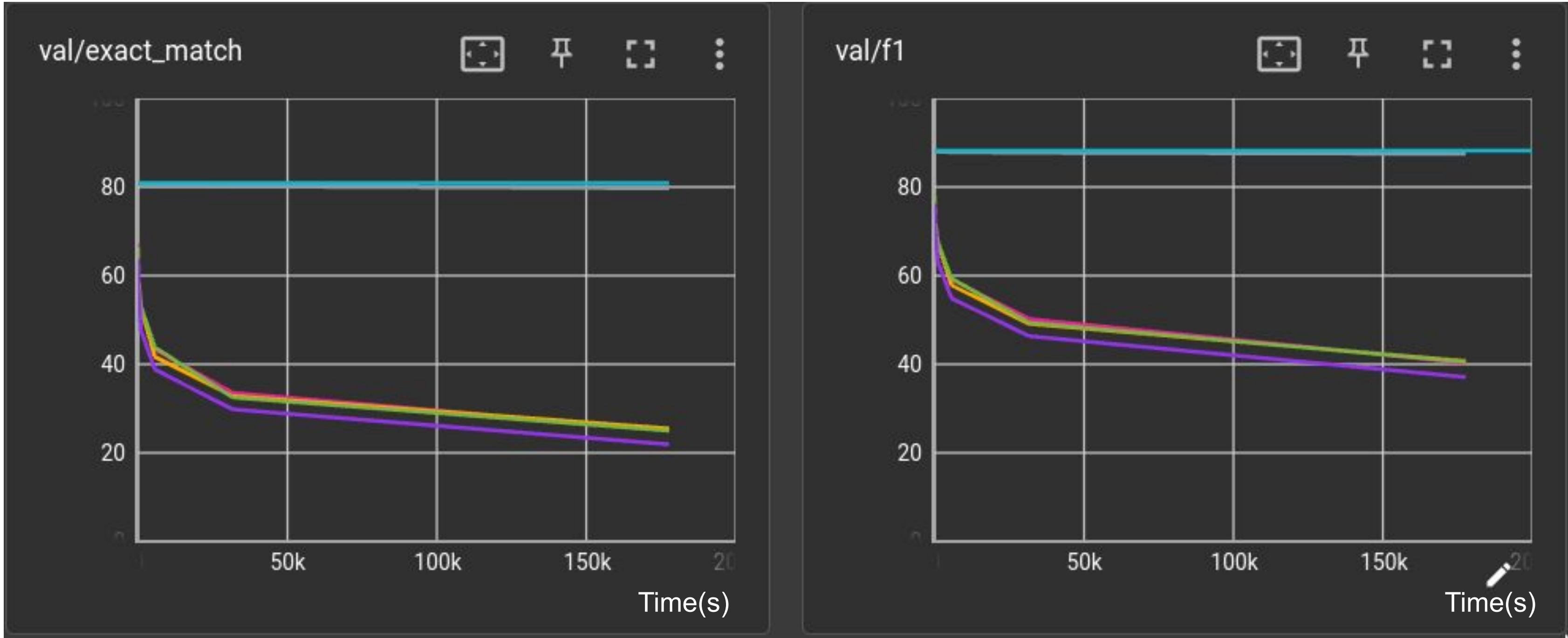
Results

Different noise levels used to measure their impact on drift over time

Blue: Digital
Gray: Ideal

Pink: 0.00875
Green: *0.0175
Orange: 0.035
Purple: 0.07

*Standard PCM weight noise used was 0.0175



Next Steps

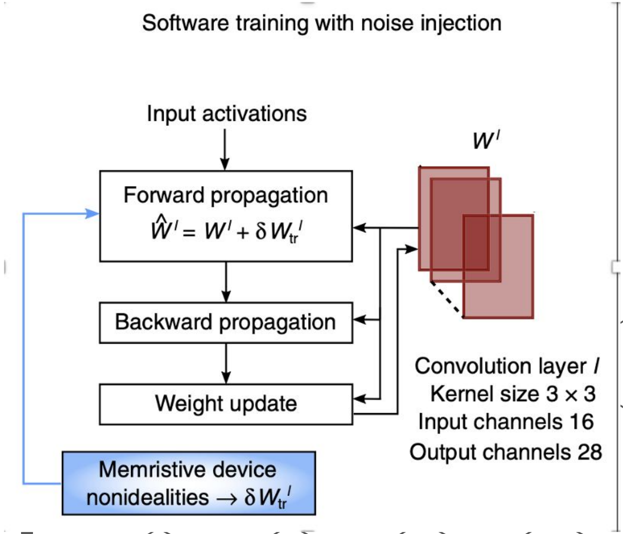
- Increase the robustness of the analog BERT model with Hardware-Aware Training
- Observe how Hardware-Aware Training affects performance over time (should create a more robust model)
    - Perform hyperparameter tuning via Weights & Biases, or through experimentation on AiMOS

- Add distributed compute capabilities in the Analog AIHWKIT simulation to allow for larger batch sizes, larger scale simulations, and speed up of the training and fine-tuning of the large transformer models.

- Contribute documentation to AIHWKIT on how to fine-tune and perform inference using the Analog BERT model

- Clean up code to ensure it adheres to style guidelines
- Add a Jupyter notebook to the list of AIHWKIT notebooks showing how to tune BERT models for analog hardware

In Hardware-Aware Training, hardware induced non-idealities are introduced into the training process (during the forward pass) to improve model inference performance on noisy PCM hardware



What We Learned

- Analog AI concepts
- Phase Change Memory Technology and how it can be used to build AI accelerators
- General AI concepts
- Transformer neural network architecture
- Contributing to an active open source project using Github
  - Team work
  - Pull Requests
  - Git branches
  - Cloning/Forking
  - Etc.
- Performing deep learning training/inference experiments in a supercomputer
  - Working with SLURM scheduler
  - Configuring various environments using conda
  - Working with GPUs
  - Working with Tensorboard to visualize results
  - Hyperparameter tuning with tools such as WandB

References

- https://aihwkit.readthedocs.io/en/latest/index.html

- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. *arXiv*. https://doi.org/10.48550/arXiv.1706.03762