Powering Cognitive Radio with AI

Paul David

Independent Researcher
paul.david@simplirf.com

Abstract

Large Language Models (LLMs), particularly transformer-based models, are increasingly used as high-level agents in software systems. This paper presents a proof of concept for applying LLMs to GNU Radio by fine-tuning an open-source model (Qwen Code 1.5B Instruct and others) on flowgraph construction tasks. We introduce a modular training pipeline built on Hugging Face libraries, along with custom GNU Radio Companion (GRC) tracing tools that capture flowgraph construction and runtime behavior. This enables domain-specific dataset generation and prototyping of LLM-driven signal processing agents. While effective for orchestration, LLMs are poorly suited to latency-sensitive and resource-constrained environments. To address this, we propose Hebbian Cellular Automata (HCA), a novel model that does not rely on backpropagation. Inspired by local learning rules and modulated feedback, HCA is fully recurrent, synchronous, and well-suited for parallel, low-latency systems. We show preliminary results on simple pattern recognition tasks, positioning HCA as a promising lightweight model. This work traces a research trajectory from LLM-based orchestration toward biologically inspired learning models better aligned with real-time signal processing.

1. Introduction

Over the last few years, Large Language Models (LLMs) have shown remarkable capabilities in code generation, natural language, and software orchestration tasks. Built on transformer-based architectures, these models are increasingly applied as high-level agents across a variety of domains [8]. In the context of communication systems, this raises an intriguing possibility: can LLMs be leveraged and fine-tuned to dynamically construct, modify, or manage signal processing pipelines in environments like GNU Radio?

Our initial exploration focused on this question. We developed a system for integrating open-source LLMs into GNU Radio by fine-tuning models on flowgraph construction tasks. To support this, we built a modular training pipeline using PyTorch and Hugging Face libraries as well a custom instrumentation package for GNU Radio Companion (GRC) to trace flowgraph construction and runtime events. This allowed for rapid and scalable collection of domain-specific datasets and prototyping of several models that can synthesize and manipulate signal processing chains using natural language prompts. This pipeline is built entirely on open-source components and avoids dependence on hosted LLM APIs provided by many AI companies. Although we focus on fine-tuning smaller LLMs, there is also a growing argument that even smaller models, when run locally, can offer practical benefits in efficiency, responsiveness, and control, especially for well-structured tasks like generating JSON. This perspective is supported by recent work suggesting that small language models may be better suited as agents in specialized domains [9].

However, our investigation also revealed limitations to using language models in this context. While powerful for orchestration and generating structured text, LLMs are computationally expensive and ill-suited to latency-sensitive, resource-constrained, or embedded environments, which are conditions often encountered in real-world radio systems and deployments. Deep learning models are typically structured around a two-phase pipeline: A training phase that involves iterative optimization using backpropagation and gradient descent, and an inference phase that performs forward passes through a fixed computational graph to generate predictions or outputs. The inference phase is faster and can be embedded in real-time systems, but it still has non-negligible latency and hardware constraints, especially when GPUs are involved. Although there are examples of modern deep learning networks, such as YOLO [12], that have been heavily optimized and can achieve impressive throughput on embedded hardware, these cases are often the exception to the rule and require quantization, pruning, compiler, and hardware co-design.

As recent work highlights, even lightweight deep learning models require aggressive optimization to operate within the limits of embedded platforms, often with trade offs in accuracy and flexibility [10]. Dataflow systems like GNU Radio are sample-driven and oriented around streaming. They favor low-latency, deterministic, and local operations to maintain timing guarantees across interconnected signal blocks. Deep neural networks, especially language models, cannot easily perform online, sample-by-sample updates. While it is certainly possible to embed inference into real-time communication systems, typically these systems are deployed as monolithic blocks that process whole inputs (entire frames, windows, or images), and most deep learning frameworks are optimized for batched inputs, not streaming samples. Deep learning systems also usually require external

training pipelines and offline training, and those pipelines need significant amounts of data and compute resources. In practice, despite advances in deep learning and ongoing efforts to optimize inference efficiency, there remains compelling questions: are there alternative models that inherently avoid the need for offline training and heavy post processing to meet deployment constraints? Can we find models better suited for solving problems in communications?

The initial investigation motivated a search for alternative approaches to learning and took a path that diverged from traditional deep learning methods. The desire to have a model learn and adapt continuously, rather than rely on a static, offline-trained model was another major motivation. This was simply not practical for deep learning models in general. This led us to investigate non-backpropagation methods for learning, which sidestep the need for global gradient information. In the process, a novel model was discovered: Hebbian Cellular Automata (HCA), which is a neural network architecture that learns using local updates and a feedback signal. HCA aligns well with the structure of signal processing systems: it operates synchronously, supports parallel execution, and adapts in real time, much like components in a streaming flowgraph.

In addition to being biologically inspired by Hebbian learning, which encompasses the common phrase in neuroscience that *"cells that fire together wire together"* [4], HCA also fits within the broader family of energy-based models. These are a class of systems that include Hopfield networks [5] and Boltzmann machines [1]. Like those models, HCA evolves toward stable attractor states, but is structured around entirely local interactions in a cellular automata, making it well-suited for modular and hardware efficient implementations in real-time environments. As far as we know, HCA is the first example of a cellular automata that incorporates learning into the ruleset itself, which are typically fixed for discrete automata and found through manual or automated search strategies.

1.1 Paper Outline

This is essentially a two part paper to describe our research trajectory that is structured as follows:

- Section 2 presents our LLM-based orchestration pipeline, including the training framework, dataset collection tools, for GNU Radio Companion (GRC), and experimental observations of model behavior in flowgraph construction tasks.
- Section 3 introduces Hebbian Cellular Automata (HCA), a novel learning model inspired by Hebbian learning and modulated feedback. We describe some background in cellular automata, this new learning rule, and potential applications.
- Section 4 details early experimental results demonstrating the learning capabilities of HCA in simple pattern recognition for modulation classification.
- Section 5 concludes the paper and discusses future work.

2. LLM Integration with GNU Radio

LLMs have demonstrated powerful capabilities in code generation and orchestration tasks across many domains, especially when utilizing structured text like JSON. To explore their applicability to signal processing workflows, we built a pipeline to fine-tune open-source transformer models, mostly from the Qwen family, on domain-specific data generated from GNU Radio Companion (GRC). This section details the fine-tuning approach using Low-Rank Adaptation of LLMs (LoRA), the architecture of our data collection pipeline, and the tools developed for capturing and curating flowgraph centric datasets.

2.1 Parameter Efficient Fine-Tuning with LoRA

Fine-tuning large models from scratch is computationally expensive and memory intensive. To address this, we used Low-Rank Adaptation (LoRA) [6], a technique that injects a small number of trainable parameters into a pre-trained model while freezing the original weights. LoRA assumes that the weight update matrix ΔW during fine-tuning can be approximated as a product of two low-rank matrices:

$$\Delta W \approx AB, \text{ where } A \in \mathbb{R}^{d\times r} \text{ and } B \in \mathbb{R}^{r\times k}$$

$$r \ll \min(d, k)$$

These matrices are inserted into key transformer submodule, most commonly the attention projections, which are linear projections necessary for self-attention. Self attention in transformers allows each token in a sequence to look at other tokens and weight their importance. We also experimented with targeting the feed-forward network layers.

By injecting LoRA adapters into these layers, we allow the model to more flexibly adapt to the structure of flowgraph-related tasks without needing to retrain the entire model from scratch. This extended targeting has been empirically shown to improve sample efficiency in our datasets.

We also utilize 4-bit quantization with the NormalFloat4 (NF4) format as popularized in QLoRA [2]. QLoRA loads the frozen base model in 4-bit precision which saves a massive amount of VRAM, allowing one to tune much larger models on a single GPU. The LoRA adapters themselves remain in full 16 or 32-bit precision, preserving gradient accuracy and stability.

2.2 Flowgraph Tracing with GRC Dataset Logger

To create a dataset for instruction tuning with enough data, we developed a GRC tracing utility called `grc_dataset_logger` that captures user interactions during flowgraph construction and records block additions, deletions, and parameter modifications. It is also able to capture data during flowgraph runtime assuming its deployed through GRC.

The flowgraph logger works by launching GRC through a patching script that instruments both GUI actions and the flowgraph during runtime. At construction time, each user action is logged and serialized as a structured JSONL trace. At execution time, the tool intercepts the standard run command and substitutes a custom runner, allowing for events like updating parameter values in the GUI to be captured as part of an action trace.

Each trace captures before and after snapshots of the flowgraph so that interactions can be constructed as high-level intentions (e.g. "add an analog signal source block"). This enables the creation of fine-grained datasets that model step-by-step construction of flowgraphs.

2.3 Dataset Construction and Prompt Generation

To convert raw traces into usable training data, a second utility package was developed called `dataset_generation`, which processes these JSONL logs into prompt-context-completion tuples. This tool includes a CLI interface and preprocessing pipeline to format multi-turn conversations, inject flowgraph state history, and standardize completions.

These datasets are compatible with Hugging Face's `datasets` library and are used to fine-tune LLMs with supervised instruction tuning via the `TRL` and `PEFT` libraries. Our training loop includes support for memory-efficient quantized training using `bitsandbytes`, mixed-precision acceleration with `accelerate`, and integration with LoRA-compatible transformer models.

2.4 Pipeline Overview

Figure 1 shows the overall workflow. Users interact with GRC using a patched session launched by `grc_dataset_logger`. Construction and runtime behavior are stored to session records on disk as JSONL files. These JSONL files are post-processed into training datasets using `dataset_generation`, and fine-tuning is carried out within a self-contained Conda environment that includes GNU Radio, PyTorch, and the Hugging Face packages necessary to support the pipeline. Finally, a user facing radio CLI tool is provided to facilitate interaction with the inference engine. The GNU Radio LLM project code is open source and freely available at `github.com/SimpliRF/gnuradio_llm`.

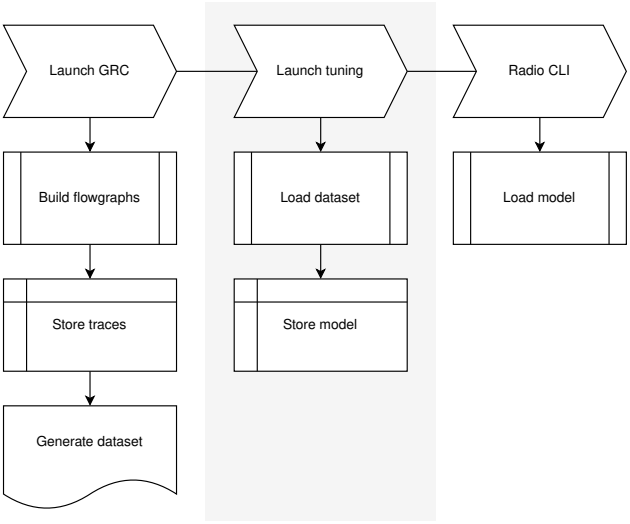

Fig. 1: Overview of the LLM fine-tuning pipeline with GNU Radio.

2.5 Current Limitations

This pipeline is an ongoing work in progress. While tracing and dataset tools support detailed observation of flowgraph construction, current limitations include partial runtime coverage and limited prompt diversity. Future iterations will incorporate more types of traces, additional prompt templates, and evaluation tools to benchmark model flowgraph generation and control.

3. Hebbian Cellular Automata (HCA)

Cellular automata (CA) are discrete, grid or lattice-based models defined by local update rules. Each cell in the grid holds a finite state vector $\mathbf{S}_i$ and evolves based on the state of its neighbors. Originally proposed by John Von Neumann in the 1940s as a framework for self-replicating machines [11], CAs have found applications in modeling physical systems, biological, chemical processes, and computation [7]. CAs are often studied for their emergent properties in dynamical systems, an area of inquiry that has implications for understanding complex systems in various fields including physics, biology, and coding theory through connections to symbolic dynamics. Perhaps the most widely known example of discrete cellular automata is Conway's Game of Life [3], which popularized the idea that simple local rules can lead to surprisingly complex global behavior. Figure 2 illustrates a glider in Conway's Game of Life, which is an initial pattern that returns to its original shape but shifted one cell diagonally.

In this work, we introduce Hebbian Cellular Automata (HCA), a novel learning model that implements mechanisms for local memory, weight adaption, and broadcasting neuromodulated feedback. HCA was developed initially for automatic modulation classification (AMC)

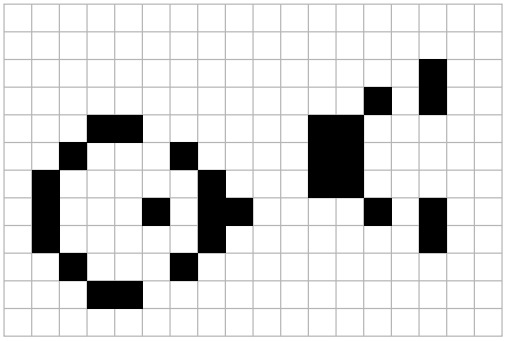

Fig. 2: Glider pattern in Conway's Game of Life.

and to explore whether a better model could be developed for control and latency-sensitive tasks. A learning model with these properties would be much more suitable for integration into DSP and dataflow architectures like GNU Radio. Although the term neuromodulation is often associated with biological nervous systems, the use of feedback signals in HCA was initially inspired by ideas from wireless cellular networks, where feedback from the base station is used to guide handset behavior.

Each HCA cell maintains a structured internal state, composed of discrete channels that include:

- An activation state: $a_i \in \{-1, +1\}$
- Incoming weights to a cell: $w_{ij} \in \{-1, 0, +1\}$
- Local memory for recording a correlation statistic between cell activations: $C_{ij} \in \mathbb{Z}$
- Modulation broadcast parameters that encode global or task-specific feedback: $M_i \in \{-2, -1, 0\}$
- A modulation counter to determine wave direction: $R_i \in \mathbb{N}$

The above description captures the core values in the cell state $\mathbf{S}_i$, and these values are organized as a vector of channels per cell with the full system evolving synchronously at each time step. A cell's next state is computed via an update rule that has access to only the current cell state and its neighbors. In this case we take the Moore neighborhood which consists of the 3×3 grid of cells. This is not too dissimilar from a convolutional layer in a neural network, except that the weight kernels are local to each cell and not shared across the grid. The choice of discrete values was made to simplify the learning dynamics and make the system more interpretable, as we started with continuous-valued weights initially, but found they weren't strictly necessary for a mininmal learning network.

Unlike conventional neural networks, which rely on global loss gradients and backpropagation, HCAs adapt using only local interactions and a reinforcement-style feedback signal. This enables real-time adaptation without requiring an external optimization loop or batch-based updates. We start with a discrete model to explore some fundamental principles and keep the system simple and tractable for analysis and plan to reinvestigate continuous-valued versions in the future.

Typically, the modulation signal is broadcasted from a readout cell, designated to be where a class decision is made. In this implementation, the readout cell is responsible for seeding the modulation signal M_i to be diffused to its neighbors using a flood update. Upon receiving a new wave, each cell updates its modulation counter, which is used to determine the direction of the wave.

3.1 Hebbian Local Update Rule

Each HCA cell updates its state at each time step using only local information: the current state of the cell and its immediate neighbors (Moore neighborhood). The update rule is synchronous across a grid or 2D lattice, and composed of updating activations, modulation diffusion, traces, and weight updates.

The first step in the update rule is to compute activations of each cell:

$$a_i \leftarrow \text{sign}\left(\sum_{j \in \mathcal{N}(i)} w_{ij} a_j\right)$$

The network is fully recurrent and allows for self-connections as well. Afterwards, we update the traces of all the incoming synapses using a raw correlation statistic:

$$C_{ij} = \sum_{n=T-W}^{T-1} a_i[n] \cdot a_j[n]$$

where W is the window size and a_i is the activation at time step n.

After we have the activations and traces computed, the Hebbian update rule is applied to compute the weight update from cell j to i:

$$\Delta w_{ij} = M_i \cdot f_\theta\left(C_{ij}\right) \cdot D_{ij}$$

where M_i is the modulation signal present at cell i and D_{ij} is the local direction indicator.

For each edge $i \leftarrow j$, D_{ij} represents a local direction indicator:

$$D_{ij} = \begin{cases} +1 & \text{if } R_i \geq R_j \\ -1 & \text{if } R_i < R_j \end{cases}$$

This indicator is necessary for breaking symmetry in the network. Without it, the network does not create a directed acyclic graph (DAG) structure, which is important for information flow and dynamics across the grid.

Finally, the weight update is applied according by clipping:

$$w_{ij} \leftarrow \text{clip}(w_{ij} + \Delta w_{ij}, -1, +1)$$

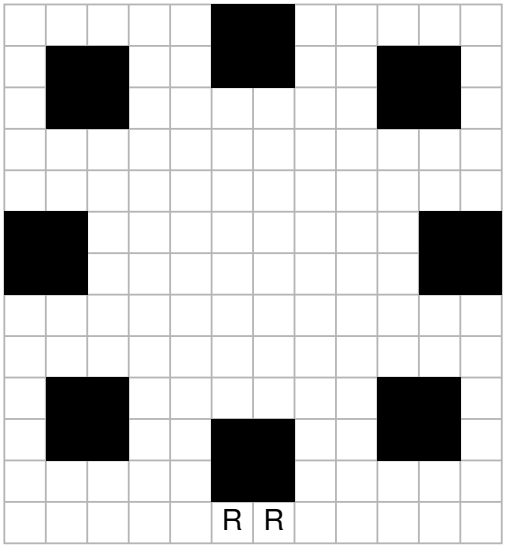

Fig. 3: An ideal 8-PSK constellation implanted on a CA grid. The HCA is trained on this type of data and forms stable attractors, allowing it to classify noisy constellations. Some cells are designated as readout cells.

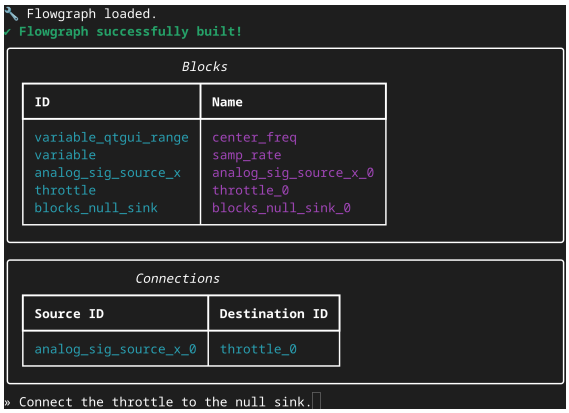

Fig. 4: Example screenshot of the radio CLI generating a flowgraph.

3.2 Applications to Cognitive Radio and Communications

HCA was originally designed for Automatic Modulation Classification (AMC) without requiring a large dataset that would need to capture a variety of noisy examples. It is a well known issue that the training data for a typical deep learning model needs to reflect the variety of possible channel conditions for it to be useful. It would be extremely convenient to have a sample efficient model that could learn from ideal cases and generalize well enough to handle noisy channel conditions, such as an ideal constellation diagram implanted on a CA substrate as shown in Figure 3 during training. The HCA model forms attractors around ideal constellation points and handles noisy constellations. The HCA does not learn to be a channel equalizer in this example, but it is performing pattern recognition. Some cells in the HCA are designated as readout cells trained along with the rest of the network to recognize one class versus the others.

In addition, we believe HCA or a variant of it could be further enhanced to work on lower-level control tasks. Given observations of self-sustained rhythmic activity, a subject of future work is to investigate the possibility of developing a neural oscillator that could learn and adapt to schedule bursts in a time-division duplex (TDD) system.

4. Experimental Results

This section presents preliminary results for both the LLM GNU Radio pipeline and the Hebbian Cellular Automata (HCA) model. While the LLM system is still undergoing active development, we include representative examples to illustrate its behavior. For HCA, we present a proof-of-concept experiment showing its ability to perform pattern recognition and be utilized for simple modulation classification.

4.1 LLM Flowgraph Generation

The LLM system was fine-tuned using the dataset pipeline described in Section 2, including prompt-context-completion tuples generated from GRC usage traces. While the results were promising, we expect the model to improve with a larger dataset. We present a screenshot of the radio CLI generating a flowgraph modification in Figure 4.

4.2 HCA for Pattern Recognition on Constellations

To evaluate the learning capacity of HCA, we constructed a simple classification task where the model must distinguish modulation types based on 2D constellation snapshots. Each input is a binarized image (or grid) representing I/Q sample locations, and the target is indicated by +1 and corresponds to a target class (e.g. BPSK, QPSK, 8-PSK). The training procedure consists of implanting the pattern on the grid and allowing the grid to update for 30 time steps (or ticks), after which the readout cell is used to classify the pattern (+1 for target class or -1 all others). The following results use a grid size of 15×20, with the readout cell placed at the bottom coordinate: $(7, 19)$. We train the network for 50-100 iterations, where at the end of each iteration, we broadcast the modulation signal from the readout cell. This is considered the end of a training episode.

Figure 5 shows the result for BPSK which settles to a white square (+1) at the readout cell after running 30 ticks. Currently, we can only train a classifier to distinguish between one class versus all others, but we have early experiments that show multi-classification is possible by

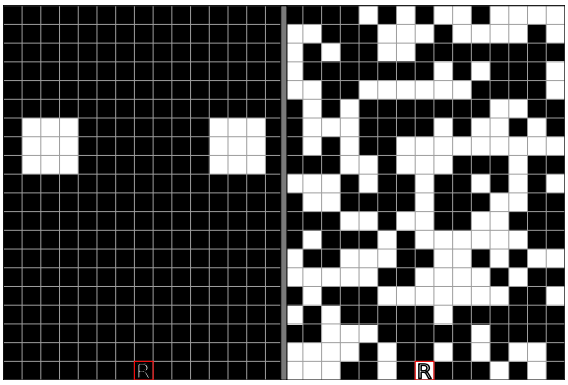

Fig. 5: On the left is the initial BPSK pattern implanted on the grid, while the right shows the grid after 30 ticks. The readout cell has a red border and R with a white square (+1), the target class (BPSK).

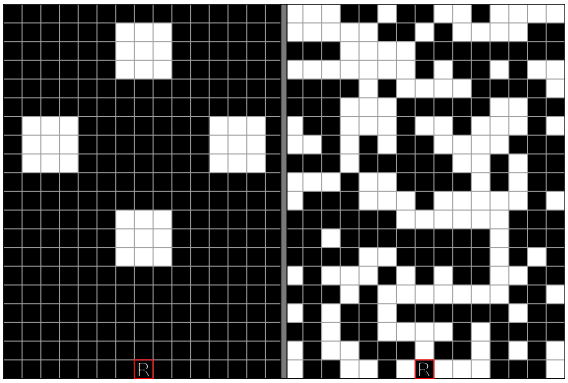

Fig. 6: On the left is the initial QPSK pattern implanted on the grid, while the right shows the grid after 30 ticks. The readout cell has a red border and R with a black square (−1), which indicates it is not the target class (BPSK).

having multiple readout cells and scheduling modulation diffusion correctly.

Figure 6 shows the result for QPSK, which results in a black square (−1), which means its not the target class (BPSK).

To illustrate the capabilities of the network, we also tested it on noisy constellations, where the inputs were flipped with some probability (e.g. 5-10%) after training. The results showed that the HCA could still classify the modulation types (and patterns more generally) effectively, seemingly able to converge despite noise patterns, as shown in Figure 7.

4.3 Current Limitations

This new model needs to be evaluated with standard benchmarks. It is not clear if there is an optimal way to choose readout cells, and so there are still architectural questions that remain. For example, the readout cell location at the bottom of the previously shown grids was arbitrary, and the network seems to adapt to that choice.

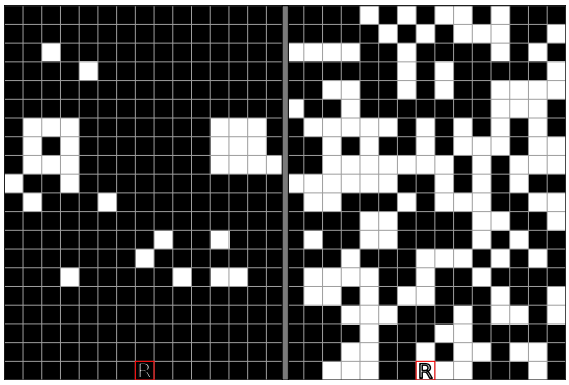

Fig. 7: Same type of result as Figure 5, where BPSK is the predicted class, but with added noise where each cell has a 5% chance of being flipped.

In the above case, 50 iterations and 30 ticks seemed to converge under 20 iterations for randomly initialized weights for different classes (BPSK, QPSK, 8-PSK), indicating the potential that this network is highly sample efficient. Many more experiments need to be conducted and this is the subject of future work.

5. Conclusion and Future Work

This work explored two approaches to applying AI in communication systems: high-level orchestration using fine-tuned LLMs, and low-level learning through a novel Hebbian Cellular Automata (HCA) model. Our LLM pipeline demonstrates how language models can be tuned for flowgraph construction tasks, but future work is needed to expand prompt diversity in training, add more trace coverage, and better capture runtime behavior. Other ideas include exploring vision transformers for flowgraph tasks.

In the second part of our research journey, we introduced a new model that forgoes backpropagation and deep learning in favor of local, feedback modulated updates. Future work includes deeper mathematical analysis, investigating its relationship to energy-based models, and evaluating its ability to perform tasks in pattern and image classification. We also want to investigate if it can be used in other tasks, such as control and synchronization in communication systems.

Taken together, this research highlights a broader point: while LLMs and deep learning dominate the AI landscape, there remains a rich space of alternative models waiting to be discovered, and potentially better aligned with the demands of real-time, modular, and embedded systems. The search for more efficient and locally adaptable learning mechanisms continues to be a promising avenue for AI in signal processing, communications, and more broadly.

References

[1] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. "A learning algorithm for Boltzmann machines". In: *Cognitive Science* 9.1 (1985), pp. 147–169. DOI: 10.1016/S0364-0213(85)80012-4.

[2] Tim Dettmers et al. "QLoRA: Efficient Finetuning of Quantized LLMs". In: *arXiv preprint arXiv:2305.14314* (2023). URL: https://arxiv.org/abs/2305.14314.

[3] Martin Gardner. "Mathematical Games: The fantastic combinations of John Conway's new solitaire game "Life"". In: *Scientific American* 223.4 (1970), pp. 120–123.

[4] Donald O. Hebb. "The Organization of Behavior: A Neuropsychological Theory". In: *Wiley* (1949). Reprinted by Lawrence Erlbaum Associates, 2002.

[5] John J. Hopfield. "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the National Academy of Sciences* 79.8 (1982), pp. 2554–2558. DOI: 10.1073/pnas.79.8.2554.

[6] Edward J. Hu et al. "LoRA: Low-Rank Adaptation of Large Language Models". In: *arXiv preprint arXiv:2106.09685* (2021). URL: https://arxiv.org/abs/2106.09685.

[7] Andrew Ilachinski. *Cellular Automata: A Discrete Universe*. Singapore: World Scientific, 2001.

[8] OpenAI. "GPT-4 Technical Report". In: *arXiv preprint arXiv:2303.08774* (2023). URL: https://arxiv.org/abs/2303.08774.

[9] NVIDIA Research. "Small Language Models are the Future of Agentic AI". In: *arXiv preprint arXiv:2506.02153* (2025). URL: https://arxiv.org/abs/2506.02153.

[10] Shriyank Somvanshi et al. "From Tiny Machine Learning to Tiny Deep Learning: A Survey". In: *arXiv preprint arXiv:2506.18927* (2025). URL: https://arxiv.org/abs/2506.18927.

[11] John Von Neumann. *Theory of Self-Reproducing Automata*. Ed. by Arthur W. Burks. University of Illinois Press, 1966.

[12] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. "YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information". In: *arXiv preprint arXiv:2402.13616* (2024). URL: https://arxiv.org/abs/2402.13616.