NEUROMODULATORY CONTROL NETWORKS (NCNS): A BIOLOGICALLY INSPIRED ARCHITECTURE FOR DYNAMIC LLM PROCESSING

A PREPRINT

Michael Christian Morgan
Independent Researcher
https://github.com/Mmorgan-ML
mmorgankorea@gmail.com

April 25, 2025

ABSTRACT

Large Language Models (LLMs) based on the Transformer architecture have achieved remarkable success, yet their core processing mechanisms remain largely static after training. While powerful, this static nature limits their ability to dynamically adapt their processing strategy based on nuanced contextual cues, task demands, or desired operational modes (e.g., shifting between exploration and exploitation). We propose Neuromodulatory Control Networks (NCNs), a novel architectural modification inspired by the neuromodulatory systems in the vertebrate brain (e.g., those utilizing dopamine, acetylcholine, norepinephrine). NCNs are small, parallel networks that receive contextual input, summarizing the global state, task information, or external control signals, and compute dynamic "modulatory signals". These signals are broadcast to the main LLM to influence its computational properties during a forward pass, analogous to how neuromodulators alter neuronal gain, plasticity, and network states. Instead of merely routing information, NCNs aim to change *how* information is processed throughout the base model by modulating key components like attention mechanisms (e.g., temperature), layer gains, and activation functions. Crucially, the architecture allows the model to *implicitly learn* to self-regulate these parameters via backpropagation, effectively becoming its own "tuning expert." This paper introduces the NCN architecture, details its components and implicit learning mechanism, discusses its conceptual advantages and potential failure modes (such as contextual stereotyping), and provides an open-source PyTorch implementation to facilitate community exploration and future empirical validation.

Keywords Neuromodulatory Control Networks (NCNs) · Dynamic LLM processing · Biologically inspired AI architecture · Large Language Model adaptability · Context-dependent processing LLM · LLM modulation signals · Transformer architecture dynamic adaptation · AI neuromodulation dopamine acetylcholine · LLM attention temperature modulation · Mixture-of-Experts vs NCNs · Open-source PyTorch NCN implementation · Parameter-efficient fine-tuning NCN · Controllable text generation NCN · Large Language Models (LLMs) · Transformer architecture · Neuromodulation · Reinforcement learning from human feedback (RLHF) · Mixture-of-Experts (MoE) · Adapters (PEFT) · Hypernetworks · PyTorch implementation

1 Introduction

Large Language Models (LLMs), predominantly based on the Transformer architecture [1], have revolutionized natural language processing, demonstrating impressive capabilities in generation, understanding, and reasoning. Foundational models like GPT [2, 3] and BERT [4] exemplify this success. However, despite their scale and power, the fundamental processing pathway within these models is typically fixed after pre-training or fine-tuning. Each input token, regardless of the broader context or specific task requirements, undergoes a largely identical sequence of computations dictated by static model parameters. This inherent staticity contrasts sharply with the dynamic adaptability observed in biological

brains, which leverage neuromodulatory systems to flexibly alter processing modes based on internal states and external cues [5]. Current LLMs often struggle with fine-grained contextual adaptation within a single processing sequence, exhibit interference in multi-task scenarios, and offer limited avenues for precise behavioral control beyond modifications to the input prompt [3, 6].

Inspired by the principles of biological neuromodulation, we propose Neuromodulatory Control Networks (NCNs) as an architectural enhancement for LLMs. We propose NCNs not merely as a method for static control, but as a means to enable LLMs to **implicitly learn an optimal processing strategy for any given context**. Trained end-to-end, the NCN learns to dynamically adjust the computational properties of the main model, shifting from creative and exploratory modes to precise and exploitative modes on the fly, based on the input it receives.

Neuromodulatory systems, originating from small nuclei but projecting broadly, release neurotransmitters like dopamine (DA), acetylcholine (ACh), norepinephrine (NE), and serotonin (5-HT). These modulators orchestrate global changes in brain state, influencing attention, learning rates, arousal, memory encoding versus retrieval, and the balance between exploration and exploitation [5, 7]. They dynamically alter how neural circuits process information, rather than solely carrying the primary information content themselves.

Our proposed NCN architecture mirrors this principle. It comprises a standard base LLM (e.g., a Transformer [1]) augmented with one or more small, parallel NCN modules. These NCNs receive input summarizing the current context, which could include:

- Aggregate representations of the input sequence (e.g., pooled hidden states).
- Task identifiers or instruction embeddings (for multi-task settings).
- Statistics derived from the LLM's internal activations.
- Explicit external control signals specifying desired behavior (e.g., style, focus).

Based on this context, the NCN computes a set of "modulatory signals"—low-dimensional vectors or scalars. These signals are then broadcast throughout the base LLM to dynamically adjust the functioning of its core components during the forward pass. Key proposed modulation mechanisms include:

- **Modulating Attention:** Adjusting attention softmax temperature to control focus (sharp vs. broad) or dynamically gating/mixing attention head outputs.
- **Modulating Layer Gains:** Scaling the output of specific layers or components (e.g., attention blocks, feed-forward networks) to increase or decrease their influence.
- **Modulating Activation Functions:** Altering the shape or gating parameters of activation functions (e.g., within FFNs like SwiGLU [7] or GELU [8]).
- **(Potentially) Modulating Learning Dynamics:** During training or fine-tuning, using NCN outputs to set context-dependent learning rates or regularization strengths.

This approach fundamentally differs from existing methods for conditional computation. Unlike Mixture-of-Experts (MoE) which routes tokens to distinct, static experts [9, 10] (e.g., in FFNs [11] or recently extended to attention heads as in Mixture of Attention Heads (MoA) [12] and Mixture-of-Head Attention (MoH) [13] which select a subset of heads per token), NCNs aim to modulate the processing within shared components. Instead of selecting *which* fixed computational path to take, NCNs influence *how* information is processed throughout the base model by dynamically adjusting key components. Compared to Adapters [14, 15, 16] or Hypernetworks [17, 18] that typically generate static, task-specific parameters or modules, NCNs provide dynamic modulation signals that can potentially vary throughout the processing of a single input sequence based on evolving context. While incorporating gating mechanisms similar to those in LSTMs [19], GRUs [20], or FiLM layers [18], NCNs compute these gates based on a potentially more global context captured by the dedicated NCN module, rather than solely local state. Some prior work has also explored dynamically adjusting representations based on context, but NCNs propose a broader, neuromodulation-inspired framework for controlling multiple aspects of the LLM processing itself.

We hypothesize that NCNs offer several potential benefits, including enhanced contextual adaptability (e.g., switching between factual recall and creative generation modes), improved multi-task learning through dynamic parameter sharing, more integrated controllability via exposed NCN inputs (potentially interfacing with methods like PPLM [21], CTRL [22], or RLHF [23]), and potential for dynamic resource allocation.

The primary contributions of this work are:

- The proposal of the Neuromodulatory Control Network (NCN) architecture, grounded in principles of biological neuromodulation [5, 7], for dynamically adapting LLM processing.

- The definition and formulation of specific mechanisms by which NCN-generated signals can modulate core Transformer components (attention, FFNs [using activations like GELU [8] or SwiGLU [7]], gains, Layer Normalization [24]).
- An analysis of the **implicit learning mechanism** enabled by end-to-end backpropagation, and a critical discussion of potential failure modes such as contextual stereotyping.
- An open-source PyTorch implementation of the NCN architecture to serve as a foundation for future research and experimentation by the community.
- A conceptual analysis positioning NCNs relative to existing dynamic and conditional computation techniques in LLMs [1, 14, 17, 16].

This paper details the NCN architecture, its biological motivation, potential mechanisms, and implementation considerations. While empirical validation remains future work, we believe NCNs represent a promising direction for developing more flexible, context-aware, and controllable language models. We hope this proposal and the accompanying codebase will stimulate further investigation into neuromodulation-inspired AI architectures.

The remainder of this paper is structured as follows: Section 2 discusses biological neuromodulation and related work in dynamic LLM architectures. Section 3 details the proposed NCN architecture and modulation mechanisms. Section 4 provides implementation notes based on our codebase. Section 5 outlines potential benefits and challenges. Section 6 compares NCNs to existing approaches. Section 7 suggests directions for future research, and Section 8 concludes.

2 Background and Related Work

To contextualize the Neuromodulatory Control Network (NCN) proposal, we first review the principles of biological neuromodulation that inspire its design. We then discuss existing approaches within machine learning, particularly for Large Language Models (LLMs), that aim to achieve dynamic adaptation, conditioning, or control, highlighting the conceptual distinctions from NCNs.

2.1 Biological Neuromodulation: Dynamic Brain State Control

Neural computation in the brain relies on both fast, targeted synaptic transmission (primarily mediated by glutamate and GABA) and slower, broader neuromodulation. Neuromodulatory systems typically involve relatively small clusters of neurons, often located in the brainstem or basal forebrain, that project diffusely to vast areas of the central nervous system, including the cortex, hippocampus, thalamus, and basal ganglia [5]. Instead of directly transmitting specific informational content, neuromodulators like dopamine (DA), acetylcholine (ACh), norepinephrine (NE), and serotonin (5-HT) act more like "broadcast signals", altering the computational properties and response dynamics of the circuits they innervate [5, 7].

Key neuromodulators and their associated functions provide inspiration for NCN mechanisms:

- **Dopamine (DA):** Strongly associated with reward processing, motivation, and reinforcement learning [25]. Phasic DA signals often encode "reward prediction error"—the difference between expected and received rewards—serving as a crucial learning signal, particularly in the basal ganglia and prefrontal cortex, to reinforce beneficial actions or state transitions. DA also plays roles in modulating working memory gating and attentional control.
- **Acetylcholine (ACh):** Critical for attention, learning, and memory formation. ACh release, often triggered by novel or behaviorally salient stimuli, enhances sensory processing, increases neuronal responsiveness (gain modulation), and promotes synaptic plasticity, particularly within the hippocampus and cortex [5]. It is thought to shift network states to favor encoding new information.
- **Norepinephrine (NE) / Noradrenaline (NA):** Involved in arousal, alertness, vigilance, and the "fight-or-flight" response. NE release, often during stress or salient events, can sharpen attentional focus, enhance signal-to-noise ratios in neural processing, and modulate memory consolidation. It is linked to shifts between exploratory behavior (seeking new information) and exploitative behavior (using known strategies) [5].
- **Serotonin (5-HT):** Possesses perhaps the most diverse and widespread effects, implicated in mood regulation, sleep-wake cycles, patience, and social behavior. Its cognitive roles are complex, potentially influencing behavioral inhibition, temporal discounting, and modulating the impact of negative feedback or punishment, thereby affecting the overall stability and flexibility of network states [5].

From these biological systems, we extract several key principles relevant for designing adaptive artificial networks:

- **Context-Dependency:** Neuromodulatory effects are not fixed; they depend heavily on the current behavioral state, task demands, and environmental context. The same neuron or circuit can respond differently under varying neuromodulatory conditions.
- **Diffuse Influence:** Modulators often affect large populations of neurons simultaneously, changing the overall "tone" or processing mode of entire brain regions or distributed networks.
- **Gain Modulation:** A common effect is altering the gain of neuronal responses (the slope of the input-output function), making neurons more or less sensitive or responsive to their inputs (akin to ACh/NE effects on attention/arousal).
- **Plasticity Modulation:** Neuromodulators act as critical "gates" for synaptic plasticity, determining when and how strongly connections should change based on neural activity and outcomes (akin to DA's role in learning).
- **State Switching:** They facilitate transitions between distinct functional states (e.g., exploration vs. exploitation, focused attention vs. broad monitoring, learning vs. recall).

NCNs aim to capture these principles computationally, providing a mechanism for LLMs to dynamically adjust their processing based on context, analogous to how these biological systems shape brain function.

2.2 Related Work in Dynamic and Conditional LLMs

Several existing techniques aim to make LLMs more adaptive, controllable, or efficient. NCNs share the goal of context-dependent processing but differ mechanistically.

Prompting and In-Context Learning Prepending specific instructions or examples (prompts) to the input is the standard way to condition the behavior of large pre-trained models [3]. This relies on the model's ability to recognize patterns in the prompt and adapt its subsequent generation through its standard forward pass, without any architectural changes or explicit dynamic modulation during processing. While effective, especially at scale, control is implicit and depends heavily on model size and prompt formulation. NCNs propose an explicit, internal architectural mechanism for adaptation based on context signals derived potentially beyond the immediate prompt.

Mixture-of-Experts (MoE) and Dynamic Attention Heads MoE models [9, 10] increase model capacity and enable conditional computation by employing multiple expert subnetworks, typically within Feed-Forward Network (FFN) layers. A gating network routes each input token to a small subset of these experts. This paradigm has recently been extended to the multi-head attention mechanism. Mixture of Attention Heads (MoA) [12] proposes a system where a router dynamically selects a subset of attention heads (treated as experts) for each token, improving efficiency and specialization. Building on this, Mixture-of-Head Attention (MoH) [13] further refines this concept by introducing a weighted summation of selected head outputs and incorporating "shared heads" that are always active alongside dynamically "routed heads". Both MoA and MoH enhance attention mechanisms by routing tokens to different, specialized attention heads. NCNs differ fundamentally from these routing-based approaches. While MoE, MoA, and MoH focus on selecting *which* static expert component (FFN or attention head) processes a token, NCNs compute signals that dynamically *modulate the computational properties* of shared, existing components (e.g., altering attention focus, layer gains, activation functions). The primary distinction lies between the architectural choice of *routing to distinct experts* versus *modulating the behavior of shared parameters* to adapt processing on the fly. Furthermore, while MoA/MoH typically operate with per-token, layer-specific routing decisions, NCNs, in our current proposal, generate more global, broadcast signals influencing the entire network's processing mode.

Adapters and Parameter-Efficient Fine-Tuning (PEFT) Adapters involve inserting small, trainable modules into the layers of a fixed pre-trained model [14, 2]. Only the adapter parameters are trained for downstream tasks, allowing efficient specialization. Various PEFT methods exist [15]. These approaches typically adapt the model statically for a given task or dataset. NCNs, in contrast, aim for dynamic adaptation within a single forward pass based on the immediate context, modulating the behavior of the (potentially fixed) base model parameters rather than inserting separate static modules. NCNs could potentially be combined with adapters, perhaps by having NCNs modulate adapter behavior or having adapters fine-tune an NCN-augmented base model.

Hypernetworks Hypernetworks are networks that generate the weights (or parameters) for another primary network [17]. In the context of LLMs, they have been used, for example, to generate task-specific adapter parameters [11]. While context-dependent, hypernetworks typically generate a static set of weights for the primary network based on a given task or input instance before the main processing begins. NCNs differ by generating dynamic modulation signals that continuously influence the primary network's processing throughout its forward pass, potentially changing based on evolving internal state or external inputs.

Conditional Parameters and Gating The idea of dynamically computed parameters influencing network behavior has precedents. Recurrent Neural Networks like LSTMs [19] and GRUs [20] use internal gating mechanisms computed from the local input and hidden state to control information flow. FiLM (Feature-wise Linear Modulation) layers use conditioning information to compute per-feature scaling (gain) and shifting (bias) parameters, often applied in vision-language tasks [18]. Other related work also explores dynamically constructing representations based on context. NCNs build on this concept but frame it within the broader context of neuromodulation, typically computing modulation signals based on a more global context (pooled states, task IDs, external controls) via a dedicated NCN module and applying these signals potentially to a wider range of targets (attention temperature, activation shapes) beyond simple gain/bias or LSTM/GRU-style local gating.

Controllable Generation Techniques exist to steer LLM outputs towards desired attributes, such as specific topics, styles, or toxicity levels. Methods include using control codes prepended to the input [22], modifying activations during generation [21], or employing reinforcement learning from human feedback (RLHF) [23]. NCNs offer a potential architectural substrate for such control; external signals representing desired attributes could be fed directly into the NCN, allowing it to generate corresponding modulation signals that shape the LLM's internal processing to achieve the desired output characteristics in a potentially more integrated manner.

In summary, while sharing goals with existing methods, the proposed NCN architecture offers a distinct, biologically inspired approach. It focuses on dynamically altering the processing mode of shared LLM components using broadcast signals computed by a dedicated contextual network, aiming for a flexible and integrated form of adaptation and control.

3 Neuromodulatory Control Network Architecture

We propose augmenting a standard Large Language Model (LLM), typically a Transformer [1], with one or more parallel Neuromodulatory Control Networks (NCNs). The NCN is designed to be significantly smaller and computationally cheaper than the main LLM. Its role is to dynamically compute context-dependent modulation signals that influence the processing within the main LLM layers during a single forward pass.

3.1 Overall Framework

The core architecture consists of two main components operating in parallel:

- **Base LLM:** A standard Transformer network composed of embedding layers, multiple Transformer blocks (each containing self-attention and feed-forward sub-layers with residual connections and normalization [24]), and an output prediction head.
- **Neuromodulatory Control Network (NCN):** A smaller neural network that receives contextual input derived from the main LLM's state or external sources. It outputs a set of modulation signals.

The NCN computes its modulation signals based on its input, and these signals are then used within each layer of the base LLM to modify specific computational steps, such as attention scoring or sub-layer outputs. This modulation aims to dynamically alter how information is processed by the fixed weights of the base LLM.

3.2 NCN Input Formulation

The NCN requires access to information that signals the current context or desired processing state. The specific input representation can be varied depending on the application. Based on our proposal and implementation (`model.py`), potential inputs include:

- **Global Sequence Representation:** An aggregate representation of the input sequence, computed from the initial embedding layer or an early Transformer layer. A common approach, implemented in our codebase, is to use masked average pooling over the token embeddings, ensuring padding tokens are excluded:

  $$\texttt{ControlInput} = \texttt{Pool}(\texttt{Embeddings} \times \texttt{AttentionMask})$$

  where `Pool` can be mean or max pooling. This provides the NCN with a summary of the entire input sequence content.
- **Task Information:** For multi-task scenarios, an embedding representing the current task ID or natural language instructions describing the task can be concatenated with or used as input to the NCN. This allows the NCN to tailor modulation signals specifically for the task at hand.

- **Internal State Metrics:** Statistics computed from the main LLM's activations during processing could serve as input. Examples include average activation magnitudes in certain layers, the entropy or variance of attention distributions (indicating focus vs. diffusion), uncertainty estimates from the prediction head, or gradient norms during training. These inputs would allow the NCN to react to the LLM's internal processing dynamics.
- **External Control Signals:** Explicit signals provided by a user or a controlling system can be fed into the NCN to steer LLM behavior. Examples include desired sentiment, style (e.g., formal vs. creative), topic focus, or safety level parameters.

Our reference implementation primarily utilizes the global sequence representation (masked average pooling of embeddings) as the NCN input (see `model.py`, `forward` method). The input dimension expected by the NCN (`ncn_input_dim`) is configurable via `NCNConfig` and defaults to the base LLM's hidden dimension (`d_model`).

3.3 NCN Architecture

The NCN module itself should be lightweight. Our reference implementation (`ncn.py`) utilizes a simple Multi-Layer Perceptron (MLP) as configured by `NCNConfig`:

$$\texttt{RawSignals} = \texttt{MLP}(\texttt{ControlInput})$$

$$\texttt{MLP}(x) = \texttt{Linear\_2}(\texttt{Activation}(\texttt{Linear\_1}(x)))$$

where `Linear_1` maps from `ncn_input_dim` to `ncn_hidden_dim`, `Activation` is a non-linearity (e.g., ReLU, GeLU, specified by `ncn_activation_fn` in `NCNConfig`), and `Linear_2` maps from `ncn_hidden_dim` to `num_mod_signals` (the total number of raw outputs). Alternative NCN architectures could include Recurrent Neural Networks (RNNs, like LSTMs [19] or GRUs [20]) if temporal integration of control inputs across a sequence or dialogue is needed, or even a very small Transformer.

3.4 Modulation Signals and Transformations

The raw output vector from the NCN (`RawSignals`, shape (`batch_size, num_mod_signals`)) is typically not used directly. Instead, different slices or elements of this vector are transformed to produce specific modulation signals constrained to appropriate ranges for their intended function. The mapping from raw outputs to named signals and their transformations is handled within the `NeuromodulatoryControlNetwork` module (`ncn.py`), guided by the `modulation_signal_names` list in `NCNConfig`.

Our implementation defines transformations for signals intended to mimic biological effects:

- **Gain Signals:** To scale layer outputs, often desired to be centered around 1.0. A sigmoid function followed by an offset achieves this:

$$\text{gain} = \text{sigmoid}(\text{raw\_slice}) + 0.5 \quad (\text{Range: } [0.5, 1.5])$$

- **Attention Temperature:** To scale attention logits; must be positive. A softplus function ensures positivity, with a small epsilon for numerical stability:

$$\text{attention\_temp} = \text{softplus}(\text{raw\_slice}) + \epsilon \quad (\text{Range: } (\epsilon, +\infty),\ \text{e.g., } \epsilon = 10^{-4})$$

- **Gating Signals:** To multiplicatively gate activations or outputs, typically in the range [0, 1]. A sigmoid function is suitable:

$$\text{ffn\_gate} = \text{sigmoid}(\text{raw\_slice}) \quad (\text{Range: } [0, 1])$$

These transformations ensure that the modulation signals operate within sensible bounds, preventing instability (e.g., negative temperature, exploding gains) and aligning with their intended purpose. The processed signals are returned as a dictionary mapping signal names (e.g., "gain") to their corresponding tensors (e.g., shape (`batch_size, 1`)).

3.5 Modulation Targets and Mechanisms

The crucial part is how the processed modulation signals dynamically influence the base LLM's computation. These mechanisms must be differentiable to allow for end-to-end training. Our implementation (`ModulatedTransformerLayer.py`, `attention.py`, `feedforward.py`) focuses on modulating standard Transformer components within a Pre-LayerNorm structure:

- **Modulating Attention Temperature:** The `attention_temp` signal is used to divide the pre-softmax attention scores ($QK^T/\sqrt{d_k}$), implemented within our custom `MultiHeadAttention` module (`attention.py`):

$$\texttt{ScaledScores} = \texttt{Scores}/\texttt{attention\_temp.clamp}(\min = \epsilon)$$

  Lower temperatures lead to sharper, more focused attention distributions, while higher temperatures broaden the distribution, mimicking attentional effects of ACh/NE. While approaches like MoA [12] and MoH [13] achieve dynamic attention by routing tokens to a subset of specialized heads, NCNs can provide signals that alter the fundamental properties of *all* attention heads, or dynamically compute mixing weights for their outputs based on a global context.

- **Modulating Layer Gains:** The `gain` signal is used to multiplicatively scale the output of sub-layers before the residual connection. In our implementation (`transformer_layer.py`), the same gain signal modulates the output of both the self-attention block and the feed-forward block within a layer:

$$\texttt{AttnOutput\_Modulated} = \texttt{AttnOutput} \times \texttt{gain}$$

$$\texttt{FFNOutput\_Modulated} = \texttt{FFNOutput} \times \texttt{gain}$$

  This increases or decreases the overall influence of the layer's computations based on context, analogous to ACh/NE gain modulation.

- **Modulating FFN Output:** The `ffn_gate` signal is passed to the `PositionwiseFeedForward` module (`feedforward.py`) and, in the current implementation, applied multiplicatively to its output:

$$\texttt{FinalFFNOutput} = \texttt{BaseFFNOutput} \times \texttt{ffn\_gate} \quad (\text{Potentially combined with gain})$$

  This allows the NCN to control the contribution of the FFN based on context. Alternatively, if using architectures like SwiGLU [7], the NCN could modulate the internal gate activation directly.

While not implemented in the current codebase, other potential targets include modulating activation function parameters (e.g., slope of LeakyReLU), adding learned biases to Query/Key/Value projections in attention, dynamically mixing attention head outputs, or modulating learning rates/regularization during training.

3.6 Scope and Frequency of Modulation

A design choice involves how often modulation signals are computed and how broadly they apply. Our reference implementation adopts a simple and efficient approach:

- **Frequency:** The NCN is computed once per input sequence, based on the initial NCN input (e.g., pooled embeddings).
- **Scope:** The same dictionary of modulation signals is passed to all Transformer layers in the base LLM.

This provides a global, sequence-level modulation. More complex variants could involve:

- Recomputing NCN signals at each layer, potentially using intermediate LLM states as input.
- Computing signals on a per-token basis (computationally expensive).
- Generating layer-specific signals (e.g., different gain values for early vs. late layers).

The current approach balances simplicity and computational cost with providing a global contextual adaptation signal.

3.7 The Learning Mechanism: Implicit Control via Backpropagation

The central hypothesis of the NCN architecture is that a mapping from context to optimal processing strategy can be learned implicitly through standard end-to-end training. The mechanism relies on the differentiability of the entire system, allowing loss gradients to flow back not only through the base LLM's parameters but also through the NCN's parameters. The learning process for a given training instance unfolds as follows:

1. **Context Summarization:** A contextual representation (e.g., pooled input embeddings) is computed and fed into the NCN.
2. **Signal Generation:** The NCN generates a set of modulation signals (e.g., `attention_temp`, `gain`) based on its current weights. Initially, these signals will be random.

3. **Modulated Forward Pass:** The base LLM performs a forward pass, with its operations at each layer being modulated by these specific signals. For example, it might use an `attention_temp` of 1.3 and a `gain` of 0.8 for a specific pass.
4. **Prediction and Loss Calculation:** A prediction is made, and a loss (e.g., cross-entropy) is calculated against the ground truth.
5. **The Critical Feedback Loop:** The gradient of this loss is backpropagated through the entire model. The gradients flow back through the main LLM's weights, but crucially, they also flow back through the modulation points. The system calculates how the loss would have changed if the `attention_temp` had been slightly different. These gradients continue all the way back into the weights of the NCN itself.

If a slightly higher temperature would have resulted in a lower loss for a specific type of prompt (e.g., a creative writing prompt), the gradients will nudge the NCN's weights to produce a slightly higher temperature the next time it encounters a similar context. Through millions of such updates on a diverse corpus, the NCN learns the optimal mapping: **for this type of input context, this set of modulation signals tends to produce the lowest loss.** The NCN effectively becomes a dynamic, internal "tuning expert."

3.8 Interaction with Normalization Layers

The placement of modulation relative to normalization layers like Layer Normalization [24] is important. Our implementation uses a standard Pre-Norm structure (`output = residual + Dropout(SubLayer(LayerNorm(input)))`). Modulation (e.g., gain) is applied to the output of the sub-layer (`AttnOutput`, `FFNOutput`) before the dropout and residual addition. This means the subsequent LayerNorm at the input of the next sub-layer will potentially rescale the activations, interacting with the applied gain. Studying this interaction and potentially exploring alternative placements (e.g., modulating after LayerNorm, or developing adapted normalization schemes) is an area for future investigation.

4 Implementation Details

To facilitate understanding, experimentation, and adoption of the proposed architecture, we provide an open-source reference implementation using PyTorch. The codebase, structured within the `ncn_project` directory, includes the core NCN model architecture and a custom-developed tokenizer implementation necessary for practical application. This section describes the overall structure and highlights key aspects of the implementation. The full codebase will soon be available at `https://github.com/Mmorgan-ML/Neuromodulatory-Control-Networks` and is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License, with the exception of the GPT-2 tokenizer data files whose original license may differ.

4.1 Code Repository Structure

The `ncn_project` repository is organized into three main subdirectories:

- `ncn_architecture/`: Contains the core PyTorch implementation of the NCN-modulated LLM, representing the primary architectural contribution detailed in this paper.
- `gpt2_tokenizer_files/`: Holds standard data files (`vocab.json`, `merges.txt`, `tokenizer_config.json`, etc.) defining a GPT-2 style Byte-Pair Encoding (BPE) tokenizer. These files originate from the standard GPT-2 tokenizer configuration.
- `tokenizer/`: Contains a custom Python implementation, developed by the author as part of this project, for loading and utilizing the BPE tokenizer defined by the configuration files in `gpt2_tokenizer_files/`. This includes modules for the tokenizer model logic, normalizers, pre-tokenizers, and processors necessary to convert raw text into model inputs. This custom tokenizer code is released under the same CC BY-SA 4.0 license as the `ncn_architecture` module.

While this paper concentrates on the novel model architecture within `ncn_architecture/`, the inclusion of the original GPT-2 tokenizer data files and the custom Python tokenizer implementation provides a complete, self-contained package for users wishing to experiment with the model.

4.2 Core Architecture Implementation (`ncn_architecture/`)

The `ncn_architecture/` directory contains the Python modules directly implementing the NCN-LLM:

- `config.py`: Defines the `NCNConfig` dataclass, centralizing hyperparameters for the base Transformer and NCN module.
- `ncn.py`: Implements the `NeuromodulatoryControlNetwork` class (MLP-based) and handles the transformation of raw outputs into named, scaled modulation signals.
- `attention.py`: Provides the custom `MultiHeadAttention` class capable of applying `attention_temp` modulation.
- `feedforward.py`: Contains the `PositionwiseFeedForward` class with optional output gating/scaling via `ffn_gate`.
- `transformer_layer.py`: Defines the `ModulatedTransformerLayer`, integrating the modulated components within a Pre-LayerNorm structure.
- `model.py`: Implements the main `ModulatedLLM` class, orchestrating all components, computing NCN input, and preparing masks.

Key implementation choices within this core module, as detailed in Section 3, include the MLP-based NCN, masked average pooling for NCN input, specific modulation mechanisms for attention temperature and layer/FFN gain/gating, Pre-LayerNorm structure, and standard weight initialization and tying practices.

4.3 Custom Tokenizer Implementation (`tokenizer/` & `gpt2_tokenizer_files/`)

To enable practical use, the repository includes components to tokenize raw text:

- The files in `gpt2_tokenizer_files/` provide the necessary vocabulary and merge rules data for a standard GPT-2 BPE tokenizer.
- The code within the `tokenizer/` directory is a custom implementation developed for this project. It provides the Python classes and logic required to load the data from `gpt2_tokenizer_files/` and perform text tokenization (including normalization, pre-tokenization, BPE model application, and processing) to produce `input_ids` and `attention_mask` tensors compatible with the `ModulatedLLM`.

This custom tokenizer allows users to work with the `ModulatedLLM` without external dependencies for this specific GPT-2 style tokenization process and is provided under the CC BY-SA 4.0 license.

4.4 Usage Example (Conceptual)

The following conceptual example illustrates how the components provided in the repository might be used together, highlighting the use of the custom tokenizer:

Listing 1: Conceptual PyTorch usage of the NCN architecture and custom tokenizer.

```
# Example Conceptual Usage Reflecting Repository Structure
from ncn_architecture import NCNConfig, ModulatedLLM
# Import the custom tokenizer class from the 'tokenizer' module
from tokenizer import NCNTokenizer # Assuming NCNTokenizer is the main class name
import torch

# --- Tokenizer Loading ---
# Instantiate the custom tokenizer, potentially pointing it to the data files
tokenizer_data_dir = "path/to/ncn_project/gpt2_tokenizer_files"
tokenizer = NCNTokenizer(data_dir=tokenizer_data_dir) # Conceptual initialization

# --- Model Configuration and Instantiation ---
# Configure the model, ensuring vocab size matches the loaded tokenizer
config = NCNConfig(
    vocab_size=tokenizer.get_vocab_size(), # Get vocab size from tokenizer
    d_model=768,
    nhead=12,
    num_layers=12,
)
# Instantiate the model
model = ModulatedLLM(config)
```

```
# --- Input Preparation ---
text = "Example␣input␣text␣to␣be␣processed."
# Use the custom tokenizer's method to encode text
inputs = tokenizer.encode(text) # Conceptual encoding method
input_ids = inputs["input_ids"].unsqueeze(0) # Add batch dimension
attention_mask = inputs["attention_mask"].unsqueeze(0) # Add batch dimension

# --- Forward Pass ---
# Model requires input_ids and optional attention_mask
logits = model(input_ids=input_ids, attention_mask=attention_mask)

# logits shape: (1, sequence_length, config.vocab_size)
```

This reference implementation, featuring the core NCN architecture and the accompanying custom-developed tokenizer components, provides a robust foundation for future research, empirical evaluation, and community development based on the NCN concept.

5 Potential Benefits and Challenges

The proposed Neuromodulatory Control Network (NCN) architecture, while awaiting empirical validation, offers several potential advantages over static LLM architectures, alongside inherent challenges that warrant consideration for future research and development.

5.1 Potential Benefits

We hypothesize that incorporating NCNs could lead to significant improvements in LLM capabilities:

- **Enhanced Contextual Adaptability:** This is the primary expected benefit. By dynamically modulating processing based on context signals, NCN-augmented LLMs could adjust their computational strategy on the fly. For instance, during factual question answering, the NCN might learn to lower attention temperature (see Section 3.5) to sharpen focus on relevant details and potentially increase gain in layers associated with knowledge retrieval. Conversely, for creative writing tasks, it might broaden attention (higher temperature) and perhaps increase gain in later, more abstract layers to foster diverse and novel continuations. In dialogue systems, modulation could shift based on detected user intent, emotional tone, or dialogue history length, leading to more nuanced and appropriate responses.
- **Improved Multi-Task Learning:** Standard multi-task learning often relies solely on parameter sharing or static adaptation methods like adapters [14, 2]. NCNs offer a mechanism for dynamic parameter sharing, where the shared base model parameters are configured differently for each task via the NCN's modulation signals (potentially derived from task embeddings fed to the NCN, see Section 3.2). This could lead to less interference between tasks and improved overall performance compared to static approaches, as the model can adopt task-specific processing modes using the same underlying weights.
- **Integrated Control and Steerability:** Exposing control signals as direct inputs to the NCN (see Section 3.2) provides a potentially more integrated way to steer LLM behavior compared to solely relying on prompt engineering [3]. Imagine adjusting sliders for "creativity," "formality," or "safety level" that directly influence NCN inputs, which in turn shape the internal processing dynamics (e.g., attention temperature, layer gains) via modulation signals to achieve the desired output style or constraints. This contrasts with methods like PPLM [21] or CTRL [22] by embedding control within the core processing loop via the NCN architecture.
- **Dynamic Resource Allocation:** Modulation signals, particularly gain or gating factors (see Sections 3.4 and 3.5), could potentially learn to reduce or effectively nullify the contribution of specific components (e.g., attention heads, FFN layers, or even entire Transformer layers) when they are not deemed necessary for the current context or task. If a gain signal approaches zero, the computational contribution of that component is minimized. This opens possibilities for conditional computation, potentially leading to significant efficiency gains during inference by dynamically pruning computation paths based on NCN outputs, similar in spirit but different in mechanism to MoE routing [10, 1].
- **More Biologically Plausible Learning:** If NCNs are configured to modulate learning dynamics (e.g., layer-specific learning rates or regularization strengths) during training (see Section 3.5), this could enable dynamic, context-sensitive learning rules that align better with biological principles [5, 7] than fixed schedules. This

might be particularly relevant for continual learning scenarios, where the NCN could adjust plasticity based on task novelty or perceived interference.

- **Adaptive Exploration/Exploitation Balance:** Factors like attention temperature or potentially direct modulation of output sampling parameters could be controlled by the NCN. This would allow the model to dynamically shift between generating diverse, exploratory outputs (e.g., higher temperature, broader attention) and converging on high-probability, exploitative outputs (e.g., lower temperature, focused attention), adapting its generation strategy based on task requirements or explicit control signals.

5.2 Challenges and Potential Failure Modes

Despite the potential benefits, the NCN architecture presents several challenges. Beyond the technical hurdles, there are significant behavioral risks inherent to an architecture that learns to stereotype contexts.

- **Risk of Contextual Stereotyping:** The NCN's greatest strength—learning a mapping from context to processing style—is also its potential Achilles' heel. The process of backpropagation involves finding the path of least resistance to minimize loss. If the vast majority of "math-like" text in the training data consists of textbook examples, the NCN may learn a rigid heuristic: "If the input vector looks like math, clamp the `attention_temp` low." This is a form of overfitting, not on data points, but on the *correlation between context and processing style*. In this scenario, when faced with a novel prompt like "Hypothesize a creative, non-Euclidean geometric proof," the NCN might still activate its rigid "math" heuristic, forcing the main LLM into an exploitative mode and inhibiting the necessary creativity. This risk can be mitigated by ensuring stylistic diversity in the training data (e.g., including creative math essays) and using high-dimensional context vectors that can distinguish "problem-solving" math from "creative" math.
- **Potential for Bias Amplification and Adversarial Vulnerability:**
  - *Bias Amplification:* The NCN could latch onto and amplify biases from the training data. If certain topics or dialects are consistently associated with simpler text in the corpus, the NCN might learn to always process those topics with a "dumber" set of parameters (e.g., lower layer gain, less compute), making it difficult for the model to produce nuanced or intelligent responses on those topics.
  - *Adversarial Contexts:* Conflicting signals in a prompt could confuse the NCN. For example, a prompt like "Write a very precise, formal, low-temperature legal document in the style of a whimsical, high-temperature Dr. Seuss poem" presents a contradiction. The NCN might produce an average, nonsensical set of modulation signals that serves neither goal, resulting in chaotic output.
- **Stability:** Unconstrained modulation signals pose a significant risk to network stability. Gains that grow too large could cause activation explosion, while temperatures approaching zero could lead to numerical issues and overly peaky distributions. Careful design of the NCN's output transformation functions (e.g., using sigmoid, tanh, softplus with appropriate scaling/offsets, as in Section 3.4) is crucial. Furthermore, the interaction between dynamic modulation and existing normalization layers (e.g., LayerNorm [24]) requires careful study.
- **Training Dynamics and Initialization:** Training the NCN end-to-end with the main LLM might be challenging. Gradients flowing back from the task loss through the modulated components into the NCN could become weak or noisy. To prevent the NCN from derailing the learning process early on, initialization strategies are critical. We recommend initializing the NCN such that it outputs "neutral" signals (e.g., gains/gates of 1.0, temperatures of 1.0) at the start of training, allowing the main LLM to establish baseline performance before significant modulation occurs.
- **Interpretability:** While inspired by specific neuromodulators [5, 7], the signals learned by the NCN might not map cleanly onto distinct biological functions. Understanding why the NCN chooses certain modulation patterns will require dedicated analysis tools.
- **Computational Overhead:** Although the NCN itself is designed to be small, computing its output and applying the modulation signals at each layer adds computational cost (FLOPs and potentially latency) to every forward pass. This overhead must be weighed against the performance gains or efficiency benefits.

6 Comparison to Existing Approaches

Neuromodulatory Control Networks (NCNs) aim to enhance LLM adaptability and control through dynamic, context-dependent modulation of core processing components. While sharing goals with several existing techniques, the proposed NCN architecture differs significantly in its mechanism and scope. This section contrasts NCNs with prominent related approaches.

Conditioning via Prompts Standard prompting modifies the input sequence to guide LLM behavior [3]. Control is achieved implicitly through the model's learned ability to condition on preceding tokens. NCNs, conversely, introduce an explicit internal mechanism for adaptation. The NCN directly computes modulation signals based on broader context (potentially including, but not limited to, the prompt), which then actively alter the internal processing dynamics of the fixed base model parameters, rather than relying solely on the model's implicit interpretation of input patterns.

Mixture-of-Experts (MoE) and Dynamic Attention Heads MoE models employ multiple expert subnetworks and a gating network to route input tokens to a subset of these experts [9, 10]. Recent advancements like Mixture of Attention Heads (MoA) [12] and Mixture-of-Head Attention (MoH) [13] extend this principle to the multi-head attention mechanism, dynamically selecting which attention heads process each token. This results in conditional computation where different tokens activate different, specialized pathways. NCNs, in contrast, are fundamentally distinct: instead of routing tokens to different static experts (whether FFNs or attention heads), NCNs compute signals that *modulate the processing behavior of shared components*. The aim is to change *how* information is processed by the base components (e.g., altering attention focus, gain), rather than selecting *which* static component performs the processing. While MoH introduces weighted summation of head outputs and shared/routed heads [13], these remain within the routing paradigm of selecting among discrete, specialized computational units. NCNs, by influencing the internal operations of existing components, offer a more continuous and integrated form of adaptation.

Adapters and Parameter-Efficient Fine-Tuning (PEFT) Adapters [14, 2] and related PEFT methods [15] insert or modify a small number of parameters within a largely frozen pre-trained model, typically creating a specialized static configuration for a specific downstream task. NCNs, instead, focus on dynamically modulating the behavior of the existing (potentially fixed) base model parameters during a single forward pass. The adaptation is driven by the immediate context via the NCN signals, allowing for potential shifts in processing strategy within a sequence or across different inputs, rather than relying on a pre-set, task-specific adapter configuration.

Hypernetworks Hypernetworks generate weights for a primary network based on some input, such as a task embedding [17, 18]. While context-dependent, they typically produce a static set of parameters (e.g., for adapters or specific layers) before the main processing of an instance begins. NCNs distinguish themselves by generating dynamic modulation signals (like gain factors or temperature scalars) that influence the primary network's computations *during* the forward pass. These signals can potentially change based on evolving context or internal state, offering a more continuous form of adaptation compared to the pre-computation of static weights by hypernetworks.

Gating Mechanisms (LSTM/GRU/FiLM) Internal gating, as seen in LSTMs [19] and GRUs [20], controls information flow based on the current input and local hidden state. FiLM layers [18] compute affine transformation parameters (gain/bias) based on conditioning information. NCNs can be seen as generalizing these ideas under a neuromodulatory framework. Key differences include: (1) NCNs typically derive their modulation signals from a more global context (e.g., pooled sequence representation, task ID) computed by a dedicated parallel network, rather than just local state. (2) NCNs propose a broader range of potential modulation targets inspired by biology, including attention temperature and activation function shapes, beyond the gain/bias or flow control typical of these other gating mechanisms.

Controllable Generation Techniques Methods like CTRL [22] use control codes in the prompt, while PPLM [21] modifies activations during decoding. NCNs offer a different approach by providing an architectural substrate for control. Desired attributes (style, safety, topic) could be fed as external inputs to the NCN (see Section 3.2), allowing it to generate internal modulation signals that directly shape the model's processing (e.g., adjusting attention focus, layer gains) to align with the control objective in a potentially more integrated fashion.

In essence, NCNs represent a distinct approach that emphasizes dynamic, context-driven modification of the processing characteristics of shared LLM components, inspired by the global, state-setting role of biological neuromodulators. This contrasts with methods focused on input modification (prompting), routing to static experts (MoE), adding static task-specific modules (adapters), generating static weights (hypernetworks), or purely local gating (LSTM/GRU). NCNs aim to provide a flexible, built-in mechanism for LLMs to adapt how they process information on the fly.

7 Future Research Directions

The Neuromodulatory Control Network (NCN) architecture presented in this paper offers a novel, biologically inspired framework for dynamic LLM processing. We propose a structured research agenda to explore this space:

- **Expanding Modulation Targets for Efficiency:**

  - *Attention Head Gating:* Instead of a single temperature for the whole attention block, the NCN could output a gate value for each attention head. Gating them to zero would act as a form of dynamic attention head pruning, preventing irrelevant heads from adding noise and saving computation.
  - *Layer Gating (Dynamic Depth):* By making the ‘gain‘ signal more explicitly a "gate" (trainable to be zero), the model could dynamically choose its functional depth. For simple inputs, it might skip middle layers, effectively implementing conditional computation for inference speedups.

- **Modulation for Advanced Learning and Controllability:**
  - *Learning Rate Modulation:* Paralleling dopamine's role in plasticity, the NCN could output a per-layer multiplier for the learning rate during training. This could be powerful for continual learning, allowing the model to "freeze" layers with old knowledge while increasing plasticity in others for new tasks.
  - *Regularization Modulation:* The NCN could dynamically set dropout rates. For precision tasks (math), it could disable dropout; for creative tasks, it could increase it to promote diversity.

- **Exploring Deeper Architectural Integration:**
  - *Normalization Layer Modulation:* Implementing Feature-wise Linear Modulation (FiLM) where the NCN generates gain and bias vectors for LayerNorm layers, providing fine-grained control over representation statistics.
  - *Activation Function Modulation:* For activation functions like Swish/SiLU ($x \cdot \sigma(\beta x)$), the NCN could dynamically output the β parameter, fundamentally altering the network's non-linear properties on the fly.
  - *Synergy with Routing-Based Architectures:* A particularly promising direction is combining NCNs with MoE/MoA/MoH. An NCN could control the routing behavior of an MoE, using global context to bias the selection of experts or modulate the mixing weights in MoH.

- **NCN Architectures and Complexity:** Future work should explore more sophisticated NCN architectures. Recurrent models (LSTMs [19], GRUs [20]) could enable the NCN to integrate contextual information over time within longer sequences or dialogues.

- **Systematic Investigation of Modulation Targets:** Rigorous empirical studies are needed to determine the relative effectiveness and potential synergies of modulating different components. Ablation studies disabling specific modulations (e.g., gain-only vs. temperature-only) will be critical.

8 Conclusion

Large Language Models have demonstrated remarkable capabilities, yet their predominantly static processing mechanisms limit their ability to dynamically adapt to varying contexts, tasks, and control objectives in a manner analogous to biological intelligence. This paper introduced Neuromodulatory Control Networks (NCNs), a novel architectural approach inspired by the neuromodulatory systems of the brain. NCNs are proposed as small, parallel networks that compute context-dependent signals to dynamically modulate the core computational components of a base LLM, influencing how information is processed during a forward pass.

We have detailed the conceptual framework of NCNs, outlining potential input sources, network architectures, and specific mechanisms for modulating attention, layer gains, and feed-forward networks. Furthermore, we presented key details of our open-source PyTorch implementation, providing a concrete realization of the architecture, including custom components necessary for features like attention temperature modulation, to facilitate further research and development by the community. By comparing NCNs to existing approaches like MoE, Adapters, and Hypernetworks, we highlighted the unique focus of NCNs on dynamic, broadcast modulation of shared parameters.

While this work primarily presents the architectural proposal and codebase, we anticipate that NCNs hold significant potential for enhancing LLM adaptability, multi-task performance, controllability, and potentially computational efficiency through dynamic resource allocation. However, realizing this potential requires addressing key challenges related to stability, training dynamics, interpretability, and overhead, which form the basis for extensive future research.

In conclusion, Neuromodulatory Control Networks offer a compelling, biologically grounded direction for enhancing LLM flexibility and context-awareness. We believe that exploring principles derived from biological neuromodulation, as embodied in the NCN architecture, is a promising path toward developing more sophisticated, adaptable, and ultimately more capable AI systems. We release our codebase with the hope of stimulating further investigation and empirical validation of this approach within the research community.

Acknowledgments

The author acknowledges the significant role of Google's Gemini 2.5 Pro throughout the development of this work. Gemini 2.5 Pro was instrumental in generating the initial concept for Neuromodulatory Control Networks (NCNs), drafting the core manuscript text, and producing the LaTeX source code for the final typesetting of this document. The author directed the entire process, curated and validated all outputs, and takes full responsibility for the final content and conclusions of this paper.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[5] Kenji Doya. Metalearning and neuromodulation. *Neural networks*, 15(4-6):495–506, 2002.

[6] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding, 2020.

[7] Noam Shazeer. Glu variants improve transformer, 2020.

[8] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023.

[9] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017.

[10] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022.

[11] Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks, 2021.

[12] Xiaofeng Zhang, Yikang Shen, Zeyu Huang, Jie Zhou, Wenge Rong, and Zhang Xiong. Mixture of attention heads: Selecting attention heads per token, 2022.

[13] Peng Jin, Bo Zhu, Li Yuan, and Shuicheng Yan. Moh: Multi-head attention as mixture-of-head attention, 2025.

[14] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019.

[15] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning, 2022.

[16] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers, 2020.

[17] David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks, 2016.

[18] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer, 2017.

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

[20] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[21] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation, 2020.

[22] Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation, 2019.

[23] Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences, 2020.

[24] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[25] Wolfram Schultz, Peter Dayan, and P Read Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997.