
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." We further introduce formal stability mechanisms, including homeostatic regularization, to prevent control manifold collapse. 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 · 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 formal mathematical definition of specific mechanisms by which NCN-generated signals can modulate core Transformer components (attention temperature, residual gains, and activation gating).
- An analysis of the **implicit learning mechanism** enabled by end-to-end backpropagation, including the introduction of a **Homeostatic Regularization** term to ensure stability.
- 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 [24]. 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 (high signal-to-noise ratio).
- **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 and enhance signal-to-noise ratios in neural processing [5]. The **Adaptive Gain Theory** [25] posits that the Locus Coeruleus-Norepinephrine (LC-NE) system optimizes task performance by modulating the gain of downstream cortical units, explicitly regulating the trade-off between focused exploitation (high phasic NE) and disengaged exploration (high tonic NE).
- **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 the Adaptive Gain effects of LC-NE [25]).
- **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 [26]), and an output prediction head. Formally, let the Base LLM be a function $F(\mathbf{x}; \theta)$, where \mathbf{x} is the input sequence and θ are fixed parameters.
- **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 \mathcal{Z} .

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. The combined output is $F(\mathbf{x}; \theta, \mathcal{Z})$, where \mathcal{Z} modulates the activation dynamics of θ .

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 $\mathbf{E} \in \mathbb{R}^{T \times d}$, ensuring padding tokens are excluded via mask $\mathbf{M} \in \{0, 1\}^T$:

$$\mathbf{c} = \frac{\sum_{t=1}^T (\mathbf{e}_t \cdot m_t)}{\sum_{t=1}^T m_t} \quad (1)$$

where \mathbf{e}_t is the embedding at timestep t . This provides the NCN with a summary vector $\mathbf{c} \in \mathbb{R}^d$ 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 LLM’s internal activations.
- **External Control Signals:** Explicit signals provided by a user or a controlling system can be fed into the NCN to steer LLM behavior.

Our reference implementation primarily utilizes the global sequence representation as the NCN input. 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`. Formally, given the context vector \mathbf{c} :

$$\mathbf{z}_{raw} = \mathbf{W}_2 \cdot \phi(\mathbf{W}_1 \mathbf{c} + \mathbf{b}_1) + \mathbf{b}_2 \quad (2)$$

where $\mathbf{W}_1 \in \mathbb{R}^{h \times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{S \times h}$ are trainable weights, \mathbf{b} are biases, and ϕ is a non-linearity (e.g., ReLU, GeLU). The output $\mathbf{z}_{raw} \in \mathbb{R}^S$ contains the raw logits for S modulation signals.

While the current implementation broadcasts a single set of signals, the architecture supports **Layer-Wise Modulation**, where the NCN outputs a matrix $\mathbf{Z} \in \mathbb{R}^{L \times S}$ for L layers, allowing early layers to be modulated differently from deep layers (mimicking heterogeneous receptor densities in biological cortex).

3.4 Modulation Signals and Transformations

The raw output vector from the NCN \mathbf{z}_{raw} is typically not used directly. Instead, different slices or elements of this vector are transformed to produce specific modulation signals constrained to appropriate ranges. We define the mathematical transformations for signals intended to mimic biological effects:

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

$$g = \sigma(z_g) + 0.5 \quad (\text{Range: } [0.5, 1.5]) \quad (3)$$

where $\sigma(\cdot)$ is the sigmoid function.

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

$$\tau = \text{softplus}(z_\tau) + \epsilon = \ln(1 + e^{z_\tau}) + \epsilon \quad (\text{Range: } (\epsilon, +\infty)) \quad (4)$$

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

$$\gamma = \sigma(z_{gate}) \quad (\text{Range: } [0, 1]) \quad (5)$$

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.

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

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \tau) = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k} \cdot \tau} \right) \mathbf{V} \quad (6)$$

Lower temperatures ($\tau < 1.0$) lead to sharper, more focused attention distributions (exploitation), while higher temperatures ($\tau > 1.0$) broaden the distribution (exploration), mimicking attentional effects of ACh/NE.

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

$$\mathbf{x}'_l = \mathbf{x}_l + g_l \cdot \text{MHA}(\text{LN}(\mathbf{x}_l)) \quad (7)$$

$$\mathbf{x}_{l+1} = \mathbf{x}'_l + g_l \cdot \text{FFN}(\text{LN}(\mathbf{x}'_l)) \quad (8)$$

This increases or decreases the overall influence of the layer’s computations based on context. Effectively, this acts as a dynamic step-size controller for the residual stream update, mimicking the signal-to-noise modulation attributed to the Locus Coeruleus-Norepinephrine system in Adaptive Gain Theory [25].

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

$$\text{FFN}(\mathbf{x}) = (\mathbf{W}_{out}\phi(\mathbf{W}_{in}\mathbf{x})) \odot \gamma \quad (9)$$

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.

3.6 Scope and Frequency: Tonic vs. Phasic Modulation

A design choice involves how often modulation signals are computed and how broadly they apply.

- **Tonic Modulation (Implemented):** The NCN computes a single state \mathbf{c} for the entire sequence (e.g., based on pooled embeddings). This sets a "global mood" for the generation. This is computationally efficient and implemented in our codebase.
- **Phasic Modulation (Proposed):** The NCN computes signals at every timestep t , using the current hidden state \mathbf{h}_t as input ($z_t = \text{NCN}(\mathbf{h}_t)$). This mimics the rapid dopaminergic spikes in response to reward prediction errors, allowing the model to change strategy mid-sentence (e.g., upon encountering a high-entropy "surprise" token).

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. Using the chain rule, the gradient of the loss \mathcal{L} with respect to the NCN parameters ψ is:

$$\frac{\partial \mathcal{L}}{\partial \psi} = \sum_{l=1}^L \left(\frac{\partial \mathcal{L}}{\partial F_l} \cdot \frac{\partial F_l}{\partial \mathbf{z}_l} \cdot \frac{\partial \mathbf{z}_l}{\partial \psi} \right) \quad (10)$$

If a slightly higher temperature τ 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 ψ to produce a slightly higher temperature the next time it encounters a similar context. Through millions of such updates, the NCN learns the optimal mapping: **for this type of input context, this set of modulation signals tends to produce the lowest loss.**

3.8 Homeostatic Regularization

A critical risk in dynamic gain control is instability (exploding gradients) or mode collapse (where the NCN sets all gains to zero or infinity). Biological systems maintain homeostasis. To ensure stability, we introduce a regularization term \mathcal{L}_{reg} to the total loss function \mathcal{L}_{total} . This penalty forces the NCN to intervene only when necessary, defaulting to the standard Transformer behavior ($g = 1, \tau = 1$) otherwise:

$$\mathcal{L}_{reg} = \lambda_1 \sum_{l=1}^L \|g_l - 1.0\|^2 + \lambda_2 \sum_{l=1}^L \|\tau_l - 1.0\|^2 \quad (11)$$

$$\mathcal{L}_{total} = \mathcal{L}_{task} + \mathcal{L}_{reg} \quad (12)$$

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 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:

```

1 # Example Conceptual Usage Reflecting Repository Structure
2 from ncn_architecture import NCNConfig, ModulatedLLM
3 # Import the custom tokenizer class from the 'tokenizer' module
4 from tokenizer import NCNTokenizer # Assuming NCNTokenizer is the main class name
5 import torch
6
7 # --- Tokenizer Loading ---
8 # Instantiate the custom tokenizer, potentially pointing it to the data files
9 tokenizer_data_dir = "path/to/ncn_project/gpt2_tokenizer_files"
10 tokenizer = NCNTokenizer(data_dir=tokenizer_data_dir) # Conceptual initialization
11
12 # --- Model Configuration and Instantiation ---
13 # Configure the model, ensuring vocab size matches the loaded tokenizer
14 config = NCNConfig(
15     vocab_size=tokenizer.get_vocab_size(), # Get vocab size from tokenizer
16     d_model=768,
17     nhead=12,
18     num_layers=12,
19 )
20 # Instantiate the model
21 model = ModulatedLLM(config)
22
23 # --- Input Preparation ---
24 text = "Example input text to be processed."
25 # Use the custom tokenizer's method to encode text
26 inputs = tokenizer.encode(text) # Conceptual encoding method
27 input_ids = inputs["input_ids"].unsqueeze(0) # Add batch dimension
28 attention_mask = inputs["attention_mask"].unsqueeze(0) # Add batch dimension
29
30 # --- Forward Pass ---
31 # Model requires input_ids and optional attention_mask
32 logits = model(input_ids=input_ids, attention_mask=attention_mask)
33
34 # logits shape: (1, sequence_length, config.vocab_size)

```

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

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.

- **Improved Multi-Task Learning:** Standard multi-task learning often relies solely on parameter sharing or static adaptation methods. 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. This could lead to less interference between tasks.
- **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. Imagine adjusting sliders for "creativity," "formality," or "safety level" that directly influence NCN inputs, which in turn shape the internal processing dynamics.
- **Dynamic Resource Allocation:** Modulation signals, particularly gain or gating factors, could potentially learn to reduce or effectively nullify the contribution of specific components (e.g., attention heads, FFN 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.
- **More Biologically Plausible Learning:** If NCNs are configured to modulate learning dynamics (e.g., layer-specific learning rates) during training, this could enable dynamic, context-sensitive learning rules that align better with biological principles.

5.2 Challenges and Potential Failure Modes

Despite the potential benefits, the NCN architecture presents several challenges.

- **Risk of Contextual Stereotyping (Manifold Collapse):** 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*.
- **Potential for Bias Amplification and Adversarial Vulnerability:**
 - *Bias Amplification:* The NCN could latch onto and amplify biases from the training data. If certain topics are consistently associated with simpler text, the NCN might learn to process those topics with "simpler" parameters (e.g., lower gain).
 - *Adversarial Contexts:* Conflicting signals in a prompt could confuse the NCN. For example, a prompt requesting high-temperature creativity for a low-temperature task might produce an average, nonsensical set of modulation signals.
- **Stability:** Unconstrained modulation signals pose a significant risk to network stability. Gains that grow too large could cause activation explosion. This is why we introduce the homeostatic regularization term \mathcal{L}_{reg} in Section 3.8.
- **Training Dynamics and Initialization:** Training the NCN end-to-end with the main LLM might be challenging. 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.

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.

Conditioning via Prompts Standard prompting modifies the input sequence to guide LLM behavior [3]. Control is achieved implicitly. NCNs, conversely, introduce an explicit internal mechanism for adaptation. The NCN directly computes modulation signals based on broader context, which then actively alter the internal processing dynamics of the fixed base model parameters.

Mixture-of-Experts (MoE) and Dynamic Attention Heads MoE models [9, 10] and variants like MoA/MoH [13] use a gating network to **route** input tokens to a subset of experts. This results in conditional computation where different tokens activate different weights. NCNs are fundamentally distinct: instead of routing tokens to different static experts, NCNs **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.

Adapters and Parameter-Efficient Fine-Tuning (PEFT) Adapters [14] insert small, trainable modules into a fixed model. NCNs, instead, focus on dynamically modulating the behavior of the existing 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.

Hypernetworks Hypernetworks generate weights for a primary network [17]. For LLMs, generating full weight matrices is computationally expensive. NCNs distinguish themselves by generating low-dimensional dynamic modulation signals (scalars) that influence the primary network’s computations *during* the forward pass.

Gating Mechanisms (LSTM/GRU/FiLM) Internal gating in LSTMs [19] controls information flow based on local state. NCNs generalize this idea under a neuromodulatory framework. Key differences include: (1) NCNs typically derive their modulation signals from a more global context (e.g., pooled sequence representation) computed by a dedicated parallel network. (2) NCNs propose a broader range of potential modulation targets inspired by biology, such as attention temperature.

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, acting as dynamic pruning.
 - *Layer Gating (Dynamic Depth):* By making the ‘gain’ signal more explicitly a "gate", the model could dynamically choose its functional depth.
- **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.
 - *Regularization Modulation:* The NCN could dynamically set dropout rates. For precision tasks (math), it could disable dropout; for creative tasks, it could increase it.
- **Exploring Deeper Architectural Integration:**
 - *Normalization Layer Modulation:* Implementing Feature-wise Linear Modulation (FiLM) where the NCN generates gain and bias vectors for LayerNorm layers.
 - *Activation Function Modulation:* For activation functions like Swish/SiLU ($x \cdot \sigma(\beta x)$), the NCN could dynamically output the β parameter.
 - *Synergy with Routing-Based Architectures:* An NCN could control the routing behavior of an MoE, using global context to bias the selection of experts.
- **Phasic Modulation Implementation:** Future work should explore Recurrent NCNs that process context at every timestep, enabling reaction to surprise within a sentence.
- **Systematic Investigation of Modulation Targets:** Rigorous empirical studies are needed to determine the relative effectiveness of modulating different components.

8 Conclusion

Large Language Models have demonstrated remarkable capabilities, yet their predominantly static processing mechanisms limit their ability to dynamically adapt to varying contexts. 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.

We have detailed the conceptual framework of NCNs, formalized the mathematics of attention and gain modulation, and introduced homeostatic regularization to ensure stability. 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. We anticipate that NCNs hold significant potential for enhancing LLM adaptability, multi-task performance, and controllability. 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 Nee-lakantan, 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, HyukJoong Lee, Yanzhong 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 Merriënboer, 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] Wolfram Schultz, Peter Dayan, and P Read Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997.
- [25] Gary Aston-Jones and Jonathan D Cohen. An integrative theory of locus coeruleus-norepinephrine function: adaptive gain and optimal performance. *Annu. Rev. Neurosci.*, 28(1):403–450, 2005.
- [26] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.