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 distributed as layer-specific control vectors to the main LLM to influence its computational properties during a forward pass, analogous to how neuromodulators alter neuronal gain, plasticity, and network states across different cortical depths. 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., via precision scaling), 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"—specifically, layer-wise control vectors. These signals are then distributed throughout the base LLM to dynamically adjust the functioning of its core components during the forward pass, allowing for depth-dependent modulation policies (e.g., increasing gain in early layers while tightening precision in deeper layers). Key proposed modulation mechanisms include:

- **Modulating Attention:** Adjusting attention softmax precision (acting as inverse 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 precision, 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].

Beyond individual transmitters, higher-order cortical networks orchestrate the deployment of these resources. Most notably, the Salience Network, anchored by the dorsal anterior cingulate cortex (dACC) and orbital frontoinsula, serves as a critical integration hub [26]. It continuously monitors continuous sensory streams and internal physiological states to identify "homeostatically relevant" events [27]. When a salient event is detected, this network engages executive control systems and disengages default mode processing. The NCN architecture mimics this specific function: acting as a parallel "Salience Network" that filters the input stream to determine when and how to modulate the primary language model's processing depth and precision.

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 [28]), and an output prediction head. Formally, let the Base LLM be a function $F(\mathbf{x}; \boldsymbol{\theta})$, where $\mathbf{x}$ is the input sequence and $\boldsymbol{\theta}$ 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 $F(\mathbf{x}; \boldsymbol{\theta}, \mathcal{Z})$, where $\mathcal{Z}$ modulates the activation dynamics of $\boldsymbol{\theta}$.

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 updated implementation (`model.py`, `ncn.py`), potential inputs include:

- **Global Sequence Representation (Salience Pooling):** Unlike passive pooling operations (e.g., mean pooling) which act as low-pass filters—often averaging out specific, high-frequency "trigger" tokens required for control—we implement a biologically grounded *Salience Pooling* mechanism. Drawing inspiration from the biological Salience Network (anchored by the dACC and orbital frontoinsula), which actively scans sensory streams for homeostatically relevant stimuli [26], the NCN initializes a learnable parameter $\mathbf{q}_{sal} \in \mathbb{R}^{1\times d}$ (the "Salience Query"). This vector acts as a learned "Top-Down Attentional Set," querying the sequence embeddings $\mathbf{E} \in \mathbb{R}^{T\times d}$ to extract a weighted global context $\mathbf{c}_{global}$:

  $$\mathbf{c}_{global} = \text{Attention}(\mathbf{q}_{sal}, \mathbf{E}, \mathbf{E}) \tag{1}$$

  From an information-theoretic perspective, this mechanism allows the controller to minimize the entropy of the context vector by exclusively attending to specific patterns or instructions that necessitate a shift in processing strategy, rather than treating the entire context window with uniform importance.

- **Phasic Integration (Local Context):** To enable dynamic, moment-to-moment adaptation, the NCN combines the global context with the current local hidden state $\mathbf{h}_t$ (the input to the current layer or step). The total input to the NCN control layers is the summation of tonic (global) and phasic (local) signals:

  $$\mathbf{u}_t = \mathbf{c}_{global} + \mathbf{h}_t \tag{2}$$

  This ensures that the modulation signals $\mathcal{Z}_t$ are responsive to both the overall "mood" of the text (Tonic) and the immediate processing requirements of the current token (Phasic).

- **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 (e.g., entropy of attention weights).

- **External Control Signals:** Explicit signals provided by a user or a controlling system can be fed into the NCN to steer LLM behavior.

While the general architecture supports diverse inputs like internal metrics or external controls, our provided reference implementation (`model.py`) focuses specifically on a Dual-Stream (Tonic/Phasic) Embedding Integration. By passing the token embeddings for both the salience pooling and phasic integration steps, the system learns to self-regulate end-to-end without requiring external controllers. 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`).

Implementation Note on Context Horizon Trade-off: While the general architecture defines $\mathbf{c}_{global}$ as a query over the full sequence history $\mathbf{E} \in \mathbb{R}^{T\times d}$, our reference implementation distinguishes between *Training* and *Generation* modes to optimize efficiency. During parallel training, Salience Pooling attends to the full sequence context (T). However, during autoregressive generation, re-scanning the entire history at every step would induce $O(T^2)$ complexity. Therefore, unless a specific historical context cache is maintained, the NCN operates in a **Reactive Mode**, utilizing the immediate token embedding as a low-latency proxy for local state changes.

3.3 NCN Architecture

The NCN module itself is designed to be lightweight relative to the base model. Our reference implementation (`ncn.py`) utilizes a Multi-Layer Perceptron (MLP) acting as a controller. Formally, given the combined tonic-phasic input $\mathbf{u}_t$ (as defined in Eq. 3.2):

$$\mathbf{z}_{flat} = \mathbf{W}_2 \cdot \phi(\mathbf{W}_1\mathbf{u}_t + \mathbf{b}_1) + \mathbf{b}_2 \tag{3}$$

where $\mathbf{W}_1 \in \mathbb{R}^{h\times d}$ maps the input to the NCN hidden dimension h, and $\mathbf{W}_2 \in \mathbb{R}^{(L\cdot S)\times h}$ projects the hidden state to the total number of modulation parameters required for the entire network. Here, L is the number of Transformer layers and S is the number of distinct modulation signal types (e.g., gain, precision, gating).

Crucially, rather than broadcasting a single signal, our implementation enforces Layer-Wise Modulation. The flat output vector $\mathbf{z}_{flat}$ is reshaped into a modulation matrix $\mathbf{Z}_t$:

$$\mathbf{Z}_t = \text{reshape}(\mathbf{z}_{flat}, (L, S)) \in \mathbb{R}^{L \times S} \tag{4}$$

Each row l of matrix $\mathbf{Z}_t$ contains the specific modulation signals for layer l. This allows the NCN to learn distinct control policies for different depths of the network—for example, increasing exploration (low precision) in early feature extraction layers while enforcing strict exploitation (high precision) in deeper reasoning layers—effectively mimicking the heterogeneous receptor densities observed across biological cortical layers.

3.4 Modulation Signals and Transformations

The raw output vector from the NCN, $\mathbf{z}_{raw}$, is typically not used directly. Instead, to ensure numerical stability and align with the biological principles of neuromodulation, different slices of this vector are passed through specific non-linear activation functions. To ensure unambiguous implementation, we provide the complete signal specification in Table 1, derived directly from our reference implementation in `ncn.py` and `transformer_layer.py`.

The NCN computes a flat output vector which is reshaped into a modulation tensor $\mathbf{Z}_t \in \mathbb{R}^{B \times L \times K}$, where B is the batch size, L is the number of LLM layers, and $K = 3$ is the number of signal types. This ensures that every layer l receives a unique set of control parameters.

Table 1: NCN Signal Specification. **Dim** indicates the shape of the signal per-layer (where 1 denotes a scalar broadcast across the hidden dimension d). **Op** denotes the operation type ($\odot$ for element-wise multiplication). Definitions correspond to `ncn.py`.

Signal Name	**Symbol**	**Transformation** $\phi(z)$	**Range**	**Dim**	**Target Component**
Gain	g	$\sigma(z) + 0.5$	$[0.5, 1.5]$	$\mathbb{R}^1$	Residual Stream ($\mathbf{x} \leftarrow \mathbf{x} + g \cdot \text{SubLayer}(\mathbf{x})$)
Precision	β	$\text{softplus}(z) + 0.01$	$(0.01, \infty)$	$\mathbb{R}^1$	Attention Queries ($\mathbf{Q}' \leftarrow \mathbf{Q} \cdot \beta$)
FFN Gate	γ	$\sigma(z)$	$[0, 1]$	$\mathbb{R}^1$	FFN Output ($\mathbf{x}_{ffn} \leftarrow \mathbf{x}_{ffn} \cdot \gamma$)

We define the mathematical rationale for these transformations as follows:

- **Gain Signals (Signal-to-Noise):** To scale layer outputs, it is desirable for the signal to be centered around 1.0 (neutral identity) to preserve gradient flow during early training. A sigmoid function followed by an offset achieves this:

  $$g = \sigma(z_g) + 0.5 \tag{5}$$

  where $\sigma(\cdot)$ is the logistic sigmoid function. In `transformer_layer.py`, this scalar multiplicatively modulates the sub-layer output before it is added to the residual stream.

- **Attention Precision (Entropy Control):** Replacing static temperature, we use Precision (β) to scale the attention logits. This value must be strictly positive to avoid mathematical errors. We utilize the Softplus function with a small epsilon ($\epsilon = 0.01$) for numerical stability, where $\beta \approx 1.0$ represents standard attention:

  $$\beta = \text{softplus}(z_\beta) + \epsilon = \ln(1 + e^{z_\beta}) + 0.01 \tag{6}$$

  As implemented in `attention.py`, this modulation is applied as $\mathbf{Q}' = \mathbf{Q} \cdot \beta$. This scales the dot product $\mathbf{Q}\mathbf{K}^T$, effectively sharpening (if $\beta > 1$) or flattening (if $\beta < 1$) the attention probability distribution.

- **Gating Signals (Metabolic Control):** To multiplicatively gate activations or outputs, we require a value in the range [0, 1], analogous to the open/closed state of an ion channel. A standard sigmoid function is suitable:

  $$\gamma = \sigma(z_{gate}) \tag{7}$$

  This signal scales the output of the Position-wise Feed-Forward Network (FFN), allowing the NCN to suppress or fully activate the metabolic contribution of specific layers based on context.

These transformations ensure that the modulation signals operate within sensible bounds, preventing instability (e.g., negative precision, exploding gains) while providing the optimizer with a smooth, differentiable control manifold.

3.5 Modulation Targets and Mechanisms

The crucial aspect of the architecture 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. We establish a hierarchical control scheme where layer-wide integration rates interact with component-specific gates:

- **Modulating Attention Precision:** Instead of a static softmax temperature, we introduce a dynamic scalar β (Precision) that multiplicatively scales the attention scores. This is implemented within our custom `MultiHeadAttention` module (`attention.py`). Formally, we apply the scaling to the Query vectors $\mathbf{Q}$ before the dot product:

$$\mathbf{Q}' = \mathbf{Q} \cdot \beta \tag{8}$$

$$\text{Attention}(\mathbf{Q}', \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}'\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{9}$$

  Crucially, because the Key vectors $\mathbf{K}$ remain unmodulated, the NCN architecture remains fully compatible with **KV Caching** mechanisms used in modern inference. Previously cached keys do not require re-computation or re-scaling even if the NCN alters the precision β for the current step.

  Physically, β represents the *inverse temperature* in the Boltzmann distribution. It actively regulates the thermodynamic entropy of the attention probability mass function. A high precision ($\beta > 1.0$) acts as a "cooling" agent, minimizing entropy and sharpening the distribution for focused exploitation. Conversely, low precision ($\beta < 1.0$) acts as a "heating" agent, maximizing entropy and broadening the distribution to encourage exploration. This mimics the signal-to-noise regulation of Acetylcholine (ACh) and Norepinephrine (NE).

  Numerical Stability Note: While theoretically unbounded, our experimental analysis reveals that precision values $\beta \gg 4.0$ can drive attention logits beyond the dynamic range of half-precision (FP16) arithmetic. This results in gradient explosion and the collapse of the Automatic Mixed Precision (AMP) scaler during training. To ensure stability, we recommend enforcing a hard upper bound (e.g., clamp$(\beta, \text{max} = 4.0)$) in the implementation.

- **Modulating Layer Gains (Hierarchical Integration):** The g signal serves as a layer-specific integration rate. In our implementation (`transformer_layer.py`), this gain signal modulates the output of *both* the self-attention block and the feed-forward block within a layer before they are added to the residual stream. For layer l:

$$\mathbf{x}'_l = \mathbf{x}_l + g_l \cdot \text{MHA}(\text{LN}(\mathbf{x}_l)) \tag{10}$$

$$\mathbf{x}_{l+1} = \mathbf{x}'_l + g_l \cdot \text{FFN}_{\text{mod}}(\text{LN}(\mathbf{x}'_l)) \tag{11}$$

  From a dynamical systems perspective, the residual stream represents the Euler discretization of a continuous transformation; here, g_l acts as a dynamic step-size controller. By modulating g_l, the NCN controls the Lipschitz constant of the layer's total transformation, effectively regulating the signal propagation depth. This strictly formalizes the "signal-to-noise" modulation attributed to the Locus Coeruleus-Norepinephrine system in the Adaptive Gain Theory [25].

- **Modulating FFN Output (Component Gating):** To allow for specific control over reasoning density versus attention routing, the γ signal is passed to the `PositionwiseFeedForward` module (`feedforward.py`) and applied multiplicatively to its output:

$$\text{FFN}_{\text{mod}}(\mathbf{x}) = (\mathbf{W}_{out}\phi(\mathbf{W}_{in}\mathbf{x})) \odot \gamma \tag{12}$$

  While the FFN contribution to the residual stream is effectively scaled by the product $g_l \cdot \gamma$, these parameters are not redundant. g_l is a shared parameter that scales the update vector of the *entire* layer (preserving the relative magnitude of Attention vs. FFN), whereas γ specifically alters the metabolic contribution of the FFN relative to the Attention block. This ensures parameter identifiability during backpropagation: the gradient for g_l aggregates information from both the Attention and FFN heads, while γ receives gradients solely through the FFN path. This structure allows the NCN to perform "metabolic gating"—selectively disabling the expensive FFN block ($\gamma \approx 0$) while maintaining gradient flow through the Attention mechanism ($g_l > 0$).

3.6 Tonic-Phasic Integration: State-Dependent Reactivity

A critical design choice in neuromodulation is the temporal resolution and interaction of the control signals. While earlier proposals often separated slow (tonic) and fast (phasic) signals into distinct pathways, our current architecture implements a **Unified Integration** mechanism. This allows the global context to dynamically condition the network's responsiveness to local stimuli through a specific broadcasting operation.

The system operates on two distinct physical timescales:

- **Tonic Stream (Global Salience):** The NCN computes a baseline context vector $\mathbf{c}_{global} \in \mathbb{R}^{1\times D}$ from the entire input sequence using Salience Pooling. This sets the "global mood" or background arousal state, which remains adiabatic (constant) across the generation window or sequence window.
- **Phasic Stream (Local Reaction):** Simultaneously, at every timestep t, the NCN ingests the current local hidden state $\mathbf{h}_t \in \mathbb{R}^{1\times D}$, representing the immediate stimulus or token being processed (the "fast" variable).

Efficiency Note: For long-context deployment, we rely on an *Adiabatic Assumption*: global context ($\mathbf{c}_{global}$) shifts significantly slower than local token dynamics. Therefore, $\mathbf{c}_{global}$ can be cached and updated only at specific intervals or upon high-entropy events, rather than re-computed at every step. This decouples the NCN's computational cost from the sequence length, avoiding the $O(T^2)$ complexity scaling associated with full-sequence attention at every decoding step.

In our implementation (`ncn.py`), these signals are integrated *prior* to the non-linear control layers. We leverage the broadcasting semantics of modern tensor operations to project the Tonic vector as a constant bias field across the temporal dimension of the Phasic stream. Formally, for a sequence of length S, the combined input $\mathbf{U}$ is computed by broadcasting $\mathbf{c}_{global}$ onto the sequence of local states $\mathbf{H}_{phasic}$:

$$\mathbf{U} = \text{Broadcast}(\mathbf{c}_{global}, S) + \mathbf{H}_{phasic} \tag{13}$$

The resulting combined state is then projected through the control network's non-linear activation function ϕ:

$$\mathbf{Z} = \mathbf{W}_2 \cdot \phi(\mathbf{W}_1(\mathbf{U}) + \mathbf{b}) \tag{14}$$

Mathematically, this architecture ensures that the modulation is **state-dependent**. Because ϕ is non-linear (e.g., ReLU or GELU), the global context $\mathbf{c}_{global}$ acts as a dynamic bias that shifts the operating point of the activation function. This effectively alters the derivative (gain) of the function with respect to the local input $\mathbf{h}_t$.

Biologically, this mimics the **Adaptive Gain Theory** of the Locus Coeruleus-Norepinephrine (LC-NE) system [25]. Just as the tonic firing rate of LC neurons (arousal) determines the signal-to-noise ratio and gain of the phasic response to incoming stimuli, our Tonic context determines how strongly or weakly the NCN reacts to local surprises. A "high arousal" context might push the activations into a sensitive linear regime, allowing rapid adaptation to new tokens (Phasic responsiveness), while a "low arousal" context might saturate the activations, stabilizing the network against local noise (Tonic dominance).

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, without the need for reinforcement learning or separate meta-gradients. The mechanism relies on the full differentiability of the modulation pathways. The NCN and the main LLM share a unified optimization landscape where the modulation signals $\mathbf{S}_l = [\alpha_{gain}, \beta_{prec}, \gamma_{gate}]_l$ for layer l act as multiplicative hyperparameters that are tuned dynamically per token.

Formally, let θ represent the weights of the main LLM and ψ represent the weights of the NCN. The system minimizes a composite objective function $\mathcal{J}$, which includes the task-specific cross-entropy loss $\mathcal{L}_{task}$ and a homeostatic regularization term $\mathcal{L}_{reg}$ (as defined in the model configuration):

$$\mathcal{J}(\theta, \psi) = \mathcal{L}_{task}(\theta, \psi) + \lambda \sum_{l=1}^{L} ||\mathbf{S}_l - \mathbf{1}||^2 \tag{15}$$

where λ is the homeostatic coefficient. This regularization enforces a principle of "metabolic efficiency," ensuring the NCN only deviates from neutral processing (signal value 1.0) when the reduction in task loss justifies the deviation.

Using the chain rule, the gradient of the task loss with respect to the NCN parameters ψ flows through the modulation signals applied at each layer. For a given hidden state $\mathbf{h}_l$, the NCN modulates the transformation F_l (e.g., via the attention precision β or FFN gate γ). The gradient flow is described by:

$$\frac{\partial \mathcal{L}_{task}}{\partial \psi} = \sum_{l=1}^{L} \left(\frac{\partial \mathcal{L}_{task}}{\partial \mathbf{h}_{output}} \cdot \dots \cdot \frac{\partial \mathbf{h}_{l+1}}{\partial F_l} \cdot \frac{\partial F_l}{\partial \mathbf{S}_l} \cdot \frac{\partial \mathbf{S}_l}{\partial \psi} \right) \tag{16}$$

Crucially, because the interaction between $\mathbf{S}_l$ and the main data stream is multiplicative, the NCN acts as a *gradient gate* for the base model parameters θ. We formalize this as **Gradient Shielding**. Consider a weight matrix $\mathbf{W} \in \theta$ producing a raw feature map $\mathbf{y} = \mathbf{W}\mathbf{x}$, which is then modulated to produce $\tilde{\mathbf{y}} = \mathbf{y} \cdot g$. The effective gradient update for $\mathbf{W}$ is explicitly scaled by the modulation signal:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{y}}} \cdot \frac{\partial \tilde{\mathbf{y}}}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{W}} = \left(\frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{y}}} \right) \cdot g \cdot \left(\frac{\partial \mathbf{y}}{\partial \mathbf{W}} \right) \tag{17}$$

This relationship proves that the NCN implicitly learns a *context-dependent learning rate schedule*. If the NCN identifies a context as irrelevant for a specific expert circuit (e.g., creative writing context for a code-generation circuit) and drives the gain $g \to 0$, the gradient magnitude $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$ vanishes. This effectively "shields" specialized sub-structures from catastrophic interference during multi-task training, updating them only when their specific context is active.

Through this mechanism, if a higher attention precision β (sharpening the softmax distribution) results in lower entropy and lower loss for a specific semantic context (e.g., exact retrieval tasks), the backward pass will reinforce the NCN's tendency to produce high β values when it detects similar embedding patterns in the future via its internal salience pooling. Over millions of updates, the NCN learns the optimal control policy: **for this specific latent context structure, this specific vector of modulation signals minimizes the joint loss $\mathcal{J}$.**

3.8 Homeostatic Regularization

A critical challenge in dynamic control systems is preventing the controller from drifting into unstable regimes or exerting unnecessary influence when the base model is sufficient. While our architectural definitions (Eq. 5, 7) physically bound modulation signals like gain and gating to prevent numerical infinity, unregularized training can still drive these parameters to saturate at their upper or lower bounds, effectively vanishing gradients and locking the network into a rigid processing mode. Furthermore, semi-bounded signals like attention precision (Eq. 6) remain susceptible to divergence. In biological systems, neural activity is metabolically expensive; therefore, neuromodulatory intervention is typically reserved for moments of high homeostatic demand. Drawing on the definition of the Salience Network as a system for "responding to homeostatic demands" [27], we implement this principle directly in the loss function via Homeostatic Regularization.

We interpret the deviation of modulation signals from their neutral state (1.0) as a metabolic cost. To enforce efficiency and ensure training stability independent of input scale, we introduce a regularization term $\mathcal{L}_{reg}$ that penalizes this deviation using the Mean Squared Error (MSE). Unlike a summation-based penalty, which would scale linearly with sequence length and model depth—potentially forcing the NCN to remain dormant on long sequences to save aggregate "energy"—the MSE formulation creates a scale-invariant "energetic floor." The NCN is incentivized to output identity transformations ($g = 1, \beta = 1$) unless the reduction in task loss outweighs the unit metabolic penalty.

Formally, for the set of modulation signal tensors $\mathcal{S}$ (e.g., gain, precision, gating), the regularization loss is calculated as the sum of the means:

$$\mathcal{L}_{reg} = \lambda \sum_{\mathbf{Z} \in \mathcal{S}} \left(\frac{1}{N_{\mathbf{Z}}} \sum_{i=1}^{N_{\mathbf{Z}}} (z_i - 1.0)^2 \right) \tag{18}$$

where $N_{\mathbf{Z}}$ represents the total number of elements in a specific signal tensor (spanning batch, sequence, and layers).

The total loss function minimizes both task error and control effort:

$$\mathcal{L}_{total} = \mathcal{L}_{task} + \mathcal{L}_{reg} \tag{19}$$

This mechanism ensures stability by preventing the divergence of unbounded signals (precision) and the saturation of bounded signals (gain), but more importantly, it forces the NCN to act as a true salience detector: it only modulates the LLM when the input context is sufficiently novel or difficult ("salient") to justify the computational "energy expenditure."

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, a robust training infrastructure, and a custom-developed tokenizer implementation necessary for practical application. This section describes the overall structure and highlights key functional aspects of the repository. The full codebase is 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 modular subdirectories designed to separate architectural logic, data management, and executive control.

4.1.1 Core Architecture and Tokenization

- `ncn_architecture/`: This package contains the core PyTorch implementation of the Neuromodulatory Control Network. It houses the dynamical systems logic, including the `NeuromodulatoryControlNetwork` class (the controller) and the `ModulatedTransformerLayer` (the executive), representing the primary contribution of this work.

  Experimental Optimizations: To validate the architecture's suitability for high-throughput training and low-latency inference, the implementation includes an experimental branch incorporating **five custom Fused CUDA Kernels**:
  - *RMSNorm:* We replace standard Layer Normalization with Root Mean Square Normalization (RMSNorm) [29], implemented via a fused block-reduction kernel to improve numerical stability and speed.
  - *Fused NCN Actuator:* A specialized kernel that fuses the distinct activation functions (Sigmoid, Softplus) and, crucially, enforces the numerical stability clamps (e.g., $\max(\beta) = 4.0$) described in Section 3.5 without separate memory read/write cycles.
  - *Fused Modulation:* We replace standard element-wise addition with a fused kernel that computes $\mathbf{x} \cdot g + \mathbf{residual}$ in a single memory pass, significantly reducing VRAM bandwidth pressure.
  - *Zero-Copy KV Cache:* For autoregressive inference, we utilize a custom CUDA kernel that updates the Key-Value cache in-place without memory allocation overhead.
  - *Online Softmax:* We implement a fused online softmax update kernel to minimize memory footprint during logit computation.
- `gpt2_tokenizer_files/`: Holds standard configuration files (`vocab.json`, `merges.txt`, `tokenizer_config.json`) defining a GPT-2 style Byte-Pair Encoding (BPE) tokenizer.
- `tokenizer/`: Contains a custom Python implementation, developed for this project, to load and utilize the BPE tokenizer defined in the adjacent configuration folder. This module includes logic for normalization, pre-tokenization, and the conversion of raw text into tensor-ready input IDs. This custom tokenizer code is released under the same CC BY-SA 4.0 license as the `ncn_architecture` module.

4.1.2 Executive Control and Analysis

The project includes high-level scripts designed to manage the lifecycle of the model, from training to convergence analysis.

- `train.py`: A robust training executive script designed for scalability and fault tolerance. It supports Distributed Data Parallel (DDP) execution for multi-GPU environments and Automatic Mixed Precision (AMP) for optimized memory throughput. Key features include:
  - **Gradient Accumulation:** Allows for large effective batch sizes even on hardware with limited VRAM.
  - **Stateful Resumption:** The script saves full system states (optimizer, scheduler, and AMP scaler dicts), allowing training to resume seamlessly from interruptions without loss of momentum.
  - **Dynamic Validation:** Utilizes a caching mechanism for pre-tokenized validation sets to accelerate evaluation cycles.
  - **Homeostatic Logging:** Explicitly tracks and logs the regularization penalty ($\mathcal{L}_{reg}$) alongside task loss, ensuring the metabolic cost of control remains within bounded regimes.

- `ppl_analyze.py`: An analytical tool designed to mathematically verify model convergence. It parses the raw `training.log` files and performs a post-hoc analysis of the learning trajectory.
  - **Dual-Span Smoothing:** Applies both short-term and long-term Exponential Weighted Moving Averages (EWMA) to separate local volatility ("noise") from macro-convergence trends.
  - **Log-Loss Transformation:** Converts Perplexity (PPL) back to Log-Loss space to perform linear regression analysis.
  - **Convergence Detection:** Calculates the slope of the loss curve over the final 15% of training steps to mathematically classify the run as "Converged," "Still Improving," or "Diverging," visualizing the results in `convergence_analysis.png`.

4.1.3 Data Management and Artifacts

- `training_data/`: The ingestion point for the text corpus. The system employs a recursive crawler that traverses this directory and all subdirectories to locate and ingest every `.txt` file found. To address the CPU bottleneck inherent in on-the-fly tokenization of large corpora, the experimental branch extends this functionality to support pre-tokenized binary files (`.bin`). This allows the data loader to utilize memory mapping (`mmap`), streaming data directly to the GPU and significantly improving training throughput.
- `ncn_checkpoints/`: The designated storage for model snapshots. These checkpoints store not only model weights but also the complete hyperparameter configuration used during the run, facilitating precise reproducibility and easy inference instantiation.
- `assets/`: A storage directory for post-training artifacts. It contains subfolders such as `training_graphs/` for the visual outputs generated by the analysis script, and `trained_models/` for the final, consolidated model weights ready for deployment.

4.1.4 Initialization Stability Protocol

A critical, often overlooked aspect of hypernetwork-style architectures is the initialization state. Standard weight initialization schemes (e.g., Kaiming or Xavier) result in output projections with a mean distribution near zero ($\mathbb{E}[z] \approx 0$). When passed through the NCN's non-linear activation functions, this neutral initialization can inadvertently force the base LLM into a degraded processing mode at the start of training ($t = 0$).

Specifically, we identified two "Initialization Shocks" that occur if standard initialization is left uncorrected:

- **Entropy Shock:** The precision signal $\beta = \text{softplus}(z) + 0.01$ yields $\beta \approx 0.70$ at $z = 0$. This corresponds to an effective attention temperature of $T \approx 1.42$, forcing the attention mechanism into a high-entropy, overly diffuse state compared to the standard Transformer baseline ($\beta = 1.0$).
- **Metabolic Throttling:** The FFN gating signal $\gamma = \sigma(z)$ yields $\gamma = 0.5$ at $z = 0$. In a residual architecture, significantly dampening the signal branch (the FFN block) at initialization distorts the signal propagation variance and gradient flow compared to a standard Pre-LayerNorm Transformer, where the branch is fully active by default.

To adhere to the principle that the NCN should act as a *sparse interventionist* system (intervening only when necessary), we implement a **Bias Initialization Strategy**. We mathematically invert the activation functions to calculate specific bias offsets for the final NCN projection layer. This forces the system to start as an **Identity Mapping** ($\beta \approx 1, \gamma \approx 1, g \approx 1$), preventing the shocks described above.

The specific bias calculations are as follows:

- **Precision Bias ($\beta \to 1.0$):** We initialize the bias for precision outputs to $b_\beta \approx 0.525$. Since $\text{softplus}(0.525) + 0.01 \approx 1.0$, this ensures standard attention behavior.
- **FFN Gate Bias ($\gamma \to 1.0$):** We initialize the bias for gating outputs to $b_\gamma \approx 3.0$. This ensures $\gamma = \sigma(3.0) \approx 0.95$, keeping the FFN pathway mostly open and preserving gradient flow.
- **Gain Bias ($g \to 1.0$):** The gain signal $g = \sigma(z) + 0.5$ naturally yields 1.0 at $z = 0$, requiring no specific offset.

This protocol ensures that at step zero, the NCN-augmented model behaves nearly identically to a standard Transformer, allowing the NCN to learn deviations smoothly rather than fighting to recover from a degraded initial state.

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 both the base Transformer and the NCN module. It specifically introduces the `ncn_heads` parameter (defaulting to 4), which decouples the attention complexity of the control network from the base model, alongside the definitions for modulation signal types.
- `ncn.py`: Implements the `NeuromodulatoryControlNetwork` class. This module defines the Salience Pooling mechanism—utilizing a learnable `control_query` and a dedicated Multi-Head Attention layer governed by `ncn_heads`—and handles the projection of layer-wise modulation vectors.
- `attention.py`: Provides the custom `MultiHeadAttention` class capable of applying dynamic **precision** modulation (scaling the query vectors) to actively regulate attention entropy.
- `feedforward.py`: Contains the `PositionwiseFeedForward` class with optional output gating/scaling via the `ffn_gate` signal.
- `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 the homeostatic regularization loss (implemented as Mean Squared Error), and managing the dual-stream (tonic/phasic) input flow.

Key implementation choices within this core module, as detailed in Section 3, include the adoption of Attention-based Salience Pooling (mediated by the `control_query` parameter) for context extraction rather than simple average pooling, the generation of vectorized layer-specific outputs to allow depth-dependent control policies, and the use of Pre-LayerNorm structures with 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:

```
# Example Conceptual Usage Reflecting Repository Structure
from ncn_architecture import NCNConfig, ModulatedLLM
# Import the custom tokenizer class from the 'tokenizer' module
from tokenizer import Tokenizer
import torch

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

# --- 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)
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)
```

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, spanning adaptability, efficiency, and controllability:

- **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 increase gain in layers associated with knowledge retrieval. Conversely, for creative writing tasks, it might broaden attention (higher temperature) and increase gain in later, more abstract layers to foster diverse and novel continuations.
- **Zero-Overhead State Switching:** Unlike Mixture-of-Experts (MoE) models which route tokens to distinct static weights (often incurring significant VRAM bandwidth overhead to load experts) or Adapter methods that require swapping memory modules, NCN modulation relies entirely on scalar multiplication. As demonstrated in our implementation, this allows the model to switch processing modes instantly with negligible latency overhead. Furthermore, because modulation is applied via broadcasting or scalar scaling of the Query vectors, the architecture maintains full compatibility with high-throughput inference techniques like Flash Attention and KV-Caching.
- **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 allows the model to utilize the same weights in distinct ways for different tasks, potentially reducing interference.
- **Mitigation of Catastrophic Interference via Gradient Shielding:** In continuous learning scenarios, standard LLMs suffer from catastrophic forgetting, where training on a new task B degrades the weights optimized for task A. The NCN's multiplicative gating offers a solution to the Plasticity-Stability Dilemma. Because the modulation signal g acts as a multiplicative coefficient in the forward pass, it linearly scales the gradient during the backward pass ($\frac{\partial \mathcal{L}}{\partial \theta} \propto g$). If the NCN identifies a context as distinct from previous tasks (e.g., "creative writing" vs. "code generation") and down-regulates ($g \to 0$) the specific layers responsible for the previous task, those weights are effectively "shielded" from gradient updates. This mirrors biological mechanisms where neuromodulators like acetylcholine gate synaptic plasticity windows in specific cortical regions.
- **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. This allows for the implementation of control sliders for attributes like "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 effectively minimized, offering a path toward conditional compute savings.

- **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 of metaplasticity.

5.2 Challenges and Potential Failure Modes

Despite the potential benefits, the NCN architecture presents several challenges, ranging from theoretical risks to practical implementation hurdles observed during experimentation.

- **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.
- **Precision-Induced Gradient Explosion:** Our experimental analysis reveals a specific instability related to attention precision. Without constraints, the NCN can learn to output extremely high precision values ($\beta \gg 4.0$) early in training. Because β acts as a scalar multiplier on the query vectors, it linearly scales the gradients during the backward pass ($\nabla \mathbf{Q} \propto \beta$). In mixed-precision training regimes (AMP), this "thermodynamic overheating" causes the gradient scaler to underflow repeatedly, halting learning. As noted in Section **??**, the introduction of a hard clamp is necessary to mitigate this failure mode.
- **General Stability:** Beyond precision, unconstrained modulation signals pose a general risk to network stability. Layer 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 to strictly penalize metabolic deviations.
- **Training Dynamics and Initialization:** Training the NCN end-to-end with the main LLM is non-trivial. Standard random initialization can force the model into a degraded state. We recommend the specific "Bias Initialization Strategy" detailed in Section 4.1.4 to ensure the NCN outputs "neutral" signals (gains/gates ≈ 1.0, precision ≈ 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. Furthermore, unlike Hypernetworks which often generate weights once per context or task, the NCN is designed to run continuously alongside the base model (per-token). This allows for dynamic reaction to local surprises ("plot twists") or shifting instructions within a single sequence without the prohibitive computational cost of regenerating weight matrices at every step.

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, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[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 Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 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. *arXiv preprint arXiv:2006.16668*, 2020.

[7] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[8] D Hendrycks. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[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. *arXiv preprint arXiv:1701.06538*, 2017.

[10] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

[11] Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *arXiv preprint arXiv:2106.04489*, 2021.

[12] Xiaofeng Zhang, Yikang Shen, Zeyu Huang, Jie Zhou, Wenge Rong, and Zhang Xiong. Mixture of attention heads: Selecting attention heads per token. *arXiv preprint arXiv:2210.05144*, 2022.

[13] Peng Jin, Bo Zhu, Li Yuan, and Shuicheng Yan. Moh: Multi-head attention as mixture-of-head attention. *arXiv preprint arXiv:2410.11842*, 2024.

[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. In *International conference on machine learning*, pages 2790–2799. PMLR, 2019.

[15] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.

[16] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers. *arXiv preprint arXiv:2007.07779*, 2020.

[17] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[18] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[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. *arXiv preprint arXiv:1406.1078*, 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. *arXiv preprint arXiv:1912.02164*, 2019.

[22] Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 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. *arXiv preprint arXiv:1909.08593*, 2019.

[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] William W Seeley, Vinod Menon, Alan F Schatzberg, Jennifer Keller, Gary H Glover, Heather Kenna, Allan L Reiss, and Michael D Greicius. Dissociable intrinsic connectivity networks for salience processing and executive control. *Journal of neuroscience*, 27(9):2349–2356, 2007.

[27] William W Seeley. The salience network: a neural system for perceiving and responding to homeostatic demands. *Journal of Neuroscience*, 39(50):9878–9882, 2019.

[28] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[29] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.