Graph-of-Experts: Trainable Graphs for Hierarchical Multi-Scale Reasoning

Michael Vaden[1,2]

[1] Independent Researcher, Atlanta, USA
[2] College of Computing, Georgia Institute of Technology, Atlanta, USA
*corresponding author: mvaden6@gatech.edu, michaelvaden.mjv@gmail.com

Keywords: Mixture of Experts, Graph Networks, Transformers

Abstract.
-propose a Graph of Experts (GoE) architecture
-general framework for adaptive, compositional problem-solving
-each expert represents a specialized subnetwork
-edges between experts are learnable pathways encoding how info should flow
-gating policy learns which experts to invoke and how to traverse the expert graph
-gating enables dynamic routing, feedback, and recombination of partial results
-single pass -¿ structured reasoning
-blueprint for modern hierarchical intelligence
-efficient computation along graphs
-could coordinate reasoning across multi-modal domains
-need experiment results

1 Introduction

-hierarchical problems require systems that adapt across levels of abstraction
-vesuvius scrolls interesting because of diversity of challenges
-conventional deep networks such as UNets or transformers apply the same operations to every region, lacking mechanisms to dynamically compose specialized behaviors
-reframe modular neural routing as a learnable graph
-each node goes with an expert trained for distinct subtask
-in the vesuvius case, geometry reconstruction, fiber orientation, ink segmentation
-directed nodes encode transition policies governing how information propagates
-GoE learns structured pathways through the expert graph, discovering intermediate representations to solve complex, multi-scale problems
-unifies specialization and coordination
-gating mechanism dynamically selects experts and edge transitions
-allows for self-organization into meaningful workflows
-Add experiment info

2 Background

-Recent progress in adaptive architectures explores dividing neural computation into specialized components
-MoE paradigm trains multiple experts in a specific data regime, while a gating network selects which to activate per input
-Improves efficiency but remains flat due to one pass

-Google's mixture of recursions introduces recursive expert calls, anbling dynamic reasoning chains, these are typically linear or sequential, optimized for symbolic or temporal reasoning tasks

-They lack explicit modeling of relationships between experts themselves, for instance, how information should transition between specialists handling distinct subproblems

-Each edge encodes transition policies

-Gating mechanism operates not as a simple router but as a graph traversal policy

-Trained to discover efficient and semantically meaningful pathways across experts

-Learns compositional workflows rather than isolated specializations

-Mention vesuvius relation

3 Model

Here we formalize GoE as a modular pipeline. We cleanly separate input stems, modality-sepecific adapters that transform raw data into tokens with positions/scale, from a task-agnostic encoder that adds local/global context and emits per-token content embeddings h (for experts) and routing features e (for the graph router). The GoE core then performs sparse traversal over a library of lightweight experts under a learned graph router, allocating computation by difficulty and recording path provenance. Task heads are pluggable and minimal (used only for supervision/inference), so stems and heads change with the domain while the GoE core is identical.

3.1 Architecture

Here we formalize GoE as a modular pipeline. We cleanly separate input stems, modality-sepecific adapters that transform raw data into tokens with positions/scale, from a task-agnostic encoder that adds local/global context and emits per-token content embeddings h (for experts) and routing features e (for the graph router). The GoE core then performs sparse traversal over a library of lightweight experts under a learned graph router, allocating computation by difficulty and recording path provenance. Task heads are pluggable and minimal (used only for supervision/inference), so stems and heads change with the domain while the GoE core is identical.

3.1.1 Input Stem

The input stem acts as a modality adapter for transforming data into standardized latent and tokenized representations. Given an input batch, $x \in \mathbb{R}^{[B,C,\ldots]}$, with batch size B, channel count C, and remaining modality-dependent dimensions, x should be mapped into the latent representation

$$f_0 = \phi(x; \theta_s) \tag{1}$$

where ϕ is a parameterized feature extractor (e.g. convolutional stack for spatial data, temporal encoder for sequences). f_0 is then tokenized by partitioning or sampling the representation into discrete tokens

$$t_i = W_t \psi(f_0, \Omega_i) + b_t, \quad T = \{t_1, \ldots, t_N\} \tag{2}$$

where ψ extracts the data slice (e.g. voxels, patches, temporal sequences) corresponding to the i-th token and W_t projects it into a shared embedding space of dimension d. This process yields a modality-independent token set $T \in \mathbb{R}^{B,N,d}$. By decoupling the feature extraction and tokenization mechanisms from any specific input structure, the input stem acts as a general adapter capable of ingesting multimodal data into a coherent token space.

In addition to token generation, the input stem produces a compact set of auxiliary routing features

$$a_i = g_{\text{aux}}(f_0[\Omega_i]) \tag{3}$$

which summarize routing-relevant characteristics within each token's receptive field and provide the graph router with structural cues for expert selection. g_{aux} should emit a standardized descriptor $a_i \in \mathbb{R}^{d_a}$ for every token. For example, in spatial domains such as the Vesuvius scrolls, g_{aux} may be implemented as a lightweight 3D convolutional projection or a pooled neighborhood encoder that captures local curvature, gradient, or anisotropy patterns. In contrast, for sequential or linguistic data, it could take the form of an attention-based or temporal pooling mechanism that encodes syntactic or positional dependencies.

3.1.2 Modality-Agnostic Encoder

Given input stem output (T, a), the encoder operates on the token set $T = \{t_1, \ldots, t_N\}$ and their associated auxiliary routing features $a = \{a_1, \ldots, a_N\}$. The encoder learns contextual dependencies among tokens while maintaining domain invariance, enabling shared reasoning across heterogeneous modalities. Formally, the encoder applies a sequence of permutation-invariant transformations

$$T' = \Phi(T; \theta_e) \tag{4}$$

where Φ denotes a stack of self-attention and feed-forward blocks of the form

$$\text{LayerNorm} \rightarrow \text{MultiHeadAttention}(Q, K, V) \rightarrow \text{Residual} \rightarrow \text{FeedForward} \rightarrow \text{Residual} \tag{5}$$

Each attention head computes

$$\text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{6}$$

allowing the model to dynamically weight relationships among tokens based on learned similarity.

To ensure routing signals evolve alongside the contextual embeddings, the auxiliary features are also transformed through a lightweight path

$$a' = f_{\text{aux}}(a, T') \tag{7}$$

where f_{aux} may consist of a shallow projection or cross-attention mechanism that refines routing features using the updated token context. This coupling preserves alignment between token semantics and routing behavior, allowing downstream routing modules to operate on context-aware features.

The modality-agnostic encoder thus functions as the system's universal relational core, encoding token and routing dependencies in a manner invariant to the input modality. The resulting (T', a') pair provides a unified, enriched representation for subsequent graph routing and expert specialization.

3.1.3 Graph Router

The graph router defines a directed graph

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}), \qquad |\mathcal{V}| = M \tag{8}$$

where each node $v_m \in \mathcal{V}$ corresponds to an expert module, and each directed edge represents learned dependencies between experts. For each input token t'_i and its auxiliary routing features

a_i' is transformed into combined descriptor $r_i = [t_i'; a_i]$, used to compute an entry distribution over experts

$$p_i = \text{softmax}(W_r r_i + b_r), p_i \in \mathbb{R}^M$$

where $p_{i,m}$ indicates how strongly token i is routed to expert m. Once tokens enter the graph, they propagate through connected experts according to the learned edge weights A_{mn}

$$h_m^{l+1} = \rho(\sum_{n \in \mathcal{N}(m)} A_{mn} W_n h_n^l)$$

where A may be static (learned during training) or dynamically updated via attention over expert embeddings. This enables experts to share context, refine intermediate representations, or coordinate when multiple experts specialize in related subspaces. Routing can thus be interpreted as probabilistic traversal through the expert graph based on three principles:

1. *Auxiliary features* bias the entry points.
2. *Graph topology* defines how expertise flows among related experts.
3. *Router distribution* p_i determines which paths are activated for each token.

This formulation allows computation to be selective, structured, and interpretable: experts form the nodes of a learned relational system, and data tokens dynamically navigate it based on both semantic content and contextual metadata.

3.1.4 Experts

Each node $v_m \in \mathcal{V}$ of the expert graph represents a specialized processing unit trained to perform a distinct transformation on routed token representations. Collectively, these experts form a distributed reasoning system in which specialization emerges from data-driven routing rather than manual assignment. Formally, every expert implements a parameterized function

$$h_m^{l+1} = f_m(h_m^l; \theta_m)$$

where h_m^l is the set (or mean) of token embeddings arriving at node m during layer l. The internal structure of f_m is modality-independent—it can be a small transformer block, an MLP, a convolutional unit, or any lightweight operator suited to the representation space.

Routing probabilities from Section 3.3 determine which tokens reach each expert:

$$\widetilde{h}_m = \sum_i p_{i,m} W_p r_i$$

aggregating routed inputs according to their gating weights $p_{i,m}$. Experts update their local states through a combination of internal processing and graph message passing:

$$h_m^{l+1} = f_m(\widetilde{h}_m; \theta_m) + \sum_{n \in \mathcal{N}(m)} A_{mn} W_c h_n^l$$

where A_{mn} encodes inter-expert communication strength and W_c projects incoming context from neighboring experts.

During training, load-balancing and entropy regularizers encourage even utilization and discourage collapse to a single dominant expert:

$$\mathcal{L}_{\text{balance}} = \lambda_b \sum_m (\frac{1}{B} \sum_i p_{i,m} - \frac{1}{M})^2$$

$$\mathcal{L}_{\text{entropy}} = -\lambda_e \sum_i \sum_m p_{i,m} \log p_{i,m}$$

These terms ensure diverse specialization while maintaining differentiability for end-to-end learning.

The expert layer therefore acts as a distributed knowledge substrate: each expert learns to focus on a coherent subset of representations, yet coordination through the graph preserves global consistency. This structure supports both horizontal specialization (different skills in parallel) and vertical specialization (multi-stage refinement across connected experts).

TODO Token recombination

3.1.5 Recursive Expert Refinement

To enable progressive reasoning and self-correction, the architecture employs a recursive refinement mechanism in which expert activations are iteratively updated based on their own outputs and feedback from neighboring experts. Rather than processing each token once, the system performs multiple refinement cycles

$$H^{k+1} = \mathcal{F}(H^k; \theta_{\text{exp}}; \theta_{\text{router}}), k = 0, \dots K-1$$

where $H^k = \{h_1^k, \dots h_M^k\}$ represent all expert states after iteration k. During each cycle three actions are performed:

1. *Re-routing*: Updated token embeddings are re-evaluated by the graph router using current auxiliary cues to adjust their expert assignments

   $$p_i^{k+1} = \text{softmax}(W_r[r_i^k] + b_r)$$

   allowing the computation graph to evolve dynamically as uncertainty decreases.

2. *Expert update*: Each expert processes its new incoming messages

   $$h_i^{k+1} = f_m(\sum_i p_{i,m}^{k+1} W_p r_i^k, h_{\mathcal{N}(m)}^k; \theta_m)$$

   combining local evidence with context from adjacent experts.

3. *Convergence check*: A lightweight confidence head evaluates per-token uncertainty

   $$u_i^k = \rho(W_u r_i^k)$$

   and recursion halts when $||u^k - u^{k-1}||_2 < \epsilon$ or after a fixed depth K.

This iterative process lets the system focus compute where ambiguity remains—tokens with stable expert assignments converge quickly, while uncertain regions receive additional refinement passes.

The recursive design unifies iterative inference and conditional computation: information flows repeatedly through the expert graph, allowing specialists to exchange context and correct earlier approximations. In practice, this produces smoother, more consistent outputs while maintaining computational efficiency through early stopping.

3.6 Decoder / Output Head

3.7 Training