# State-of-the-Art Mixture-of-Experts: Architectures, Routing, and Orchestration in LLM and Multi-Agent Workflows

## Architectural Foundations of Modern MoE Systems

The Mixture-of-Experts (MoE) architecture represents a fundamental paradigm shift in the design and scaling of large-scale neural networks, particularly Large Language Models (LLMs). Evolving from a technique for computational efficiency into a sophisticated method for conditional computation, MoE enables models to achieve unprecedented parameter counts while maintaining manageable training and inference costs. This section details the foundational principles of MoE and analyzes the key architectural variants that define the current state-of-the-art, from pioneering designs to advanced structures for specialization and adaptation.

### The Paradigm of Sparse Conditional Computation

The core innovation of MoE architectures is the decoupling of a model's total parameter count from its computational cost per input token.[1] In a traditional "dense" model, every parameter is activated for every input, meaning that increasing model capacity leads to a proportional increase in computational demand (FLOPs).[3] MoE circumvents this limitation by replacing monolithic components, typically the Feed-Forward Network (FFN) layers within a Transformer block, with a collection of many smaller, parallel FFNs known as "experts".[4] A lightweight, trainable "gating network" or "router" is introduced, which dynamically selects a small subset of these experts to process each input token.[1]

This principle, known as sparse conditional computation, means that while the model may

contain trillions of parameters in total, only a small fraction are active for any given computation. This allows for a dramatic increase in model capacity—the ability to store knowledge and learn complex patterns—without a corresponding explosion in the computational resources required for training and inference.[2] The success of this approach has established MoE as a primary and credible alternative to dense models for scaling to the frontier of AI capabilities.[1]

The breakthrough that enabled modern MoE models was the development of sparsely gated networks, which demonstrated that model accuracy could be maintained while activating only a small fraction of parameters.[1] This routing process also encourages functional diversity among the experts. As the router learns to send specific types of inputs to specific experts, those experts become specialized in distinct aspects of the input distribution, such as different languages, programming domains, or reasoning styles. This specialization not only improves computational efficiency but also enhances the model's generalization and robustness across a wide range of tasks.[1]

## Pioneering Architecture: The Switch Transformer

The Switch Transformer architecture was a pivotal development that simplified and popularized the MoE paradigm for large-scale language modeling.[3] Its key innovation was a move to an extreme form of sparsity: a top-1 token-choice routing strategy. In this design, the router selects only the single most suitable expert for each token, in contrast to earlier designs that might have combined outputs from multiple experts.[1]

This simplification dramatically reduced the communication overhead and computational complexity associated with routing. By routing each token to a single expert, the Switch Transformer demonstrated that it was possible to pre-train a trillion-parameter model with up to seven times the speed of an equivalent-sized dense model (T5-XXL) on the same hardware.[3] This remarkable efficiency gain, achieved without a significant loss in performance, solidified MoE as a viable and highly effective strategy for building the largest foundation models.[1] The design's simplicity and proven scalability made it a foundational blueprint for many subsequent MoE architectures.

## General Sparse MoE (SMoE) and Top-K Routing

Building on the principles demonstrated by the Switch Transformer, the dominant approach in

many contemporary MoE models is a slightly less sparse strategy known as Top-K routing, where k is typically a small integer like 2. In this configuration, the router calculates a score for each expert based on the input token and selects the k experts with the highest scores.[10] The input token is then processed by each of the selected experts, and their outputs are combined using a weighted sum. The weights for this combination are derived from the softmax-normalized scores produced by the router for the chosen experts.[11]

This Top-K approach, prominently used in influential models like Google's GLaM and Mistral's Mixtral-8x7B, strikes a balance between the stark simplicity of the Switch Transformer's top-1 routing and the richer representational capacity that comes from engaging multiple expert perspectives.[1] Activating two experts per token, for example, allows the model to blend specialized knowledge, potentially leading to more nuanced and accurate outputs than a single expert could produce alone. This method has become a standard for sparse MoE (SMoE) implementations, offering a practical and effective compromise between computational efficiency and model performance.[1] For instance, Mixtral-8x7B activates 2 of its 8 experts per layer, resulting in approximately 13 billion active parameters out of a total of 47 billion during inference.[12]

## Advanced MoE Structures for Enhanced Specialization

As MoE architectures have matured, research has moved beyond simple, flat structures toward more sophisticated designs that enable greater specialization and more dynamic resource allocation. These advanced architectures internalize more complex reasoning and control structures, pushing the boundaries of what conditional computation can achieve.

### Hierarchical MoE (H-MoE)

Hierarchical Mixture-of-Experts (H-MoE) architectures introduce a multi-level, tree-like structure to the routing process.[7] Instead of a single, flat layer of experts, H-MoE employs multiple layers of gating networks. The first-level router makes a coarse-grained decision, directing an input to a specific group of experts. Subsequent routers within that group then make finer-grained selections, routing the input to a highly specialized expert at the leaf of the hierarchy.[1]

This design offers several key advantages. First, it enables a more structured and interpretable form of knowledge representation, as experts at different levels can specialize in progressively

narrower subproblems.[7] For example, a top-level router might distinguish between natural language and code, while second-level routers could specialize further into different programming languages or specific linguistic domains. Second, this hierarchical approach improves scalability. By breaking the routing decision into a series of smaller choices, it reduces the computational complexity of routing from a linear relationship with the number of experts,

O(E), to a logarithmic one, O(logE).[15] This makes it feasible to manage a much larger number of total experts without incurring prohibitive routing overhead.

### Dynamic MoE

Dynamic MoE architectures focus on making the allocation of computational resources more adaptive and responsive to the specific demands of each input. A central challenge with fixed top-k routing is its rigidity; every token receives the same amount of computation (i.e., is processed by k experts), regardless of its complexity. Dynamic MoE seeks to address this by allowing the model to vary its computational graph on a per-token basis.

A recent innovation in this area is the Dynamic Adaptive Shared Experts (DASE) architecture.[16] This model combines "shared experts," which are trained on all data and capture common, foundational knowledge, with a larger set of specialized experts. The key component is a hierarchical Adaptive Dynamic Routing (ADR) mechanism that first assesses the complexity of an input feature. Based on this assessment, it dynamically selects the appropriate level of expertise required, routing simpler tokens perhaps only through the shared experts (a "fast path") while directing more complex tokens to deeper, specialized experts for more intensive processing.[16] This approach optimizes resource allocation by tailoring the computational effort to the difficulty of the input, a significant step beyond the one-size-fits-all nature of standard MoE.

The progression from flat to hierarchical and dynamic architectures reflects a significant conceptual evolution. It began with a simple question of efficiency: "Which single expert is best for this token?" This was effectively a one-step classification problem solved by models like the Switch Transformer. Hierarchical MoE introduced a multi-step reasoning process, asking first, "Which general domain does this token belong to?" and then, "Within that domain, which specialized sub-expert is most appropriate?" This mirrors the coarse-to-fine reasoning patterns observed in human cognition. Dynamic MoE adds another layer of sophistication by asking, "How much computational effort does this token *deserve*?" This allows the system to allocate more resources to complex inputs and conserve them for simpler ones. This architectural trend suggests that future models are not just learning *what* to compute, but are also learning *how* to structure the computation itself, building adaptive

and complex internal reasoning graphs for each input. This can be seen as a precursor to the explicit, external computational graphs managed by multi-agent systems.

## Parameter-Efficient MoE for Model Adaptation

While training massive MoE models from scratch has proven highly effective for creating state-of-the-art foundation models, a parallel line of research has focused on more efficient methods for adapting existing pre-trained models. This has led to the development of Parameter-Efficient Fine-Tuning (PEFT) techniques that "upcycle" dense models into MoE architectures, combining the benefits of transfer learning with the efficiency of conditional computation.[1]

### LoRA-MoE

A prominent example of this approach is LoRA-MoE, which integrates Low-Rank Adaptation (LoRA) with the Mixture-of-Experts framework.[1] The standard LoRA technique involves freezing the weights of a large pre-trained model and injecting small, trainable low-rank matrices to adapt the model for a new task. LoRA-MoE extends this by creating multiple distinct LoRA adapters, each fine-tuned on a specific domain or task (e.g., one for medical data, one for legal data, one for coding). These adapters are then treated as a pool of "experts."

A lightweight router is trained on top of the frozen base model to dynamically select the most appropriate LoRA expert or combination of experts for a given input query.[18] This allows a single, deployed base model to dynamically specialize for a multitude of different tasks in real-time. The key advantage is extreme parameter efficiency; the base model's billions of parameters remain unchanged, and only the small LoRA adapters and the router (often less than 1% of the total parameters) need to be trained and stored.[1]

This convergence of PEFT and MoE signals a powerful new paradigm for model customization. Traditionally, adapting a model for a new domain required either full fine-tuning or the creation and deployment of an entirely separate model. PEFT methods like LoRA made this more efficient by creating small, task-specific "delta" weights, but managing and serving dozens of different LoRA adapters for a single base model presents a significant operational challenge. LoRA-MoE elegantly solves this by consolidating the adapters into a single expert pool and learning a router to select among them. This transforms the deployment problem from "which model version should be loaded?" to "which combination of micro-experts should

be activated for this specific query?". This architectural shift enables a single deployed model artifact to exhibit immense versatility, dynamically adapting to thousands of distinct tasks or users with minimal overhead.

| Architecture | Routing Mechanism | Primary Advantage | Key Challenge | Ideal Use Case |
|---|---|---|---|---|
| **Switch-Transformer** | Top-1 Token-Choice | Maximum simplicity and computational efficiency at scale. | Higher risk of expert collapse due to single-expert routing; less representational richness. | Large-scale foundation model pre-training where speed and efficiency are paramount. |
| **Sparse MoE (Top-K)** | Top-K Token-Choice (typically k=2) | Balances efficiency with richer representations by combining multiple expert outputs. | Load balancing is critical; requires auxiliary losses to prevent expert collapse. | General-purpose foundation models (e.g., Mixtral) and multi-task instruction tuning. |
| **Hierarchical MoE** | Multi-level Gating (Tree-based) | Enables structured, coarse-to-fine specialization; logarithmic routing complexity. | Increased router complexity and potential for error propagation down the hierarchy. | Models for domains with clear hierarchical knowledge structures (e.g., multi-lingual translation). |
| **Dynamic MoE** | Adaptive, Complexity-Aware | Optimizes resource allocation by matching computational effort to input | Requires a reliable mechanism to assess input complexity; can be harder | Real-time or resource-constrained applications where latency and cost must |

| | | | | |
|---|---|---|---|---|
| | | difficulty. | to train. | be minimized. |
| **LoRA-MoE** | Router selects from PEFT adapters | Extreme parameter efficiency; "upcycles" existing dense models for multi-task capabilities. | Performance is constrained by the base model and the quality of the LoRA experts. | Multi-task fine-tuning of existing models; personalized models for many users. |

# Intelligent Routing and Gating Mechanisms

The intelligence of a Mixture-of-Experts system is fundamentally embodied in its routing mechanism. While the experts provide specialized computational capacity, it is the router or gating network that determines how and when that capacity is used. The evolution of these mechanisms from simple learned classifiers to sophisticated, context-aware reasoning engines is central to the advancement of MoE. This section dissects the strategies that guide inputs to experts, covering foundational choices, the use of LLMs as routers, methods for behavioral control, and the techniques required to ensure stable and effective training.

## Foundational Routing Strategies: A Tale of Two Choices

At the core of MoE design lies a fundamental decision about the direction of routing: do tokens choose their experts, or do experts choose their tokens? This choice has profound implications for system performance, load balancing, and implementation complexity.

### Token-Choice Routing (Top-K Gating)

Token-choice routing is the most common and intuitive approach. In this paradigm, the router network processes each input token independently and generates a set of scores, one for each available expert. It then selects the k experts with the highest scores (the "top-k") to

process that token.[19] The final output for the token is a weighted combination of the outputs from these

k experts, with the weights determined by the router's scores.[5]

The primary advantage of this method is its simplicity and computational efficiency. The routing decision is local to each token, making it highly parallelizable and straightforward to implement.[8] However, this approach has a significant drawback: it can easily lead to severe load imbalance.[6] Since the router is trained to send tokens to the experts that perform best on them, it may learn to favor a small number of "popular" experts, overwhelming them with tokens while other experts remain underutilized or even completely unused. This "expert collapse" is a critical failure mode that requires mitigation through auxiliary training objectives.[5]

### Expert-Choice Routing

Expert-choice routing inverts the selection process to address the load balancing problem inherent in token-choice methods.[20] Instead of each token selecting its preferred experts, each expert is given a fixed computational budget or "capacity"—the number of tokens it can process per batch. The router then scores all tokens against all experts, and each expert selects the

k tokens from the batch that it is most suited to process.[9]

The main benefit of this design is that it achieves perfect load balancing by construction. Since each expert has a fixed capacity, no single expert can be overloaded.[20] This eliminates the need for auxiliary load balancing losses during training. However, this guarantee comes at the cost of increased implementation complexity and communication overhead. The router must perform an all-to-all comparison between tokens and experts before any assignments can be made, which can be a bottleneck in large-scale distributed systems. Furthermore, it breaks the locality of token processing, as an expert's computation now depends on the entire batch of tokens, not just a single one.

## LLM-Powered Routing: The Router as a Reasoning Engine

A novel and powerful evolution in routing is the replacement of the conventional, simple router (typically a single linear layer) with a full-fledged Large Language Model.[23] This approach,

demonstrated by the LLMoE framework, leverages the extensive world knowledge and sophisticated reasoning capabilities of an LLM to make much more nuanced and context-aware routing decisions than a simple classifier could.[24]

In the LLMoE implementation for financial trading, a Llama3 model serves as the router. Instead of processing a single token embedding, it receives a rich, multimodal input formatted into a natural language prompt. This prompt includes a five-day rolling window of historical stock price data (e.g., price, volume) combined with the text of relevant news headlines from those days.[24] The LLM is then tasked with a classification problem: to determine whether the overall market outlook is "Optimistic" or "Pessimistic" and to provide a natural language justification for its decision. The final classification output is used to route the trading task to one of two downstream expert models, one specialized for optimistic market conditions and the other for pessimistic ones.[24]

This method transforms the router from a simple pattern matcher into a reasoning engine. It can weigh conflicting signals—such as positive price trends versus negative news sentiment—and make a judgment that considers the broader context, much like a human analyst would.[24] The following is a representative prompt template for such an LLM-based router:

JSON


```
{
  "prompt_template": {
    "system_prompt": "You are an expert financial market analyst. Your task is to analyze a 5-day rolling window of stock data and associated news headlines. Based on this multimodal information, classify the market outlook as either 'Optimistic' or 'Pessimistic'. First, provide a brief, clear reasoning for your decision. Then, on a new line, output only the final classification word.",
    "user_template": "Analyze the following data:\nDay 1: Price={price_1}, Volume={volume_1}, News: '{news_1}'\nDay 2: Price={price_2}, Volume={volume_2}, News: '{news_2}'\nDay 3: Price={price_3}, Volume={volume_3}, News: '{news_3}'\nDay 4: Price={price_4}, Volume={volume_4}, News: '{news_4}'\nDay 5: Price={price_5}, Volume={volume_5}, News: '{news_5}'"
  },
  "example_invocation": {
    "role": "user",
    "content": "Analyze the following data:\nDay 1: Price=150, Volume=10M, News: 'Company announces record profits.'\nDay 2: Price=152, Volume=12M, News: 'New product launch receives positive reviews.'\nDay 3: Price=151, Volume=9M, News: 'Broader market shows signs of weakness.'\nDay 4: Price=153, Volume=11M, News: 'Regulators approve key patent.'\nDay 5: Price=155, Volume=11M, News: 'Competitor issues profit warning.'"
  }
```

```
}
```

The success of this approach points toward a future where complex AI systems are composed of numerous specialized models, all orchestrated by a central, powerful "conductor" LLM. While a simple linear router is computationally cheap, its semantic understanding is shallow. The LLMoE experiment demonstrates that a more expensive but far more intelligent LLM router can yield superior, context-aware performance. This pattern is directly mirrored in the architecture of advanced multi-agent systems, where a high-capability "manager" or "supervisor" agent routes complex tasks to a team of specialized worker agents. This suggests that the future of routing in complex systems is not a simple classifier but another LLM, creating a recursive, hierarchical structure. This has significant implications for system design, cost management (as the router call itself becomes a major expense), and the potential for cascading failures if the central router makes a mistake.

## Behavioral Steering via Expert (De)Activation (SteerMoE)

The SteerMoE framework introduces a groundbreaking reinterpretation of the MoE router: not merely as a mechanism for distributing computation, but as a direct and interpretable control lever for modulating model behavior at inference time.[26] This technique operates on the discovery that individual experts within an MoE model can become specialized not just for topical domains (like coding or law) but also for abstract behavioral traits like "faithfulness to a source document" or "adherence to safety guidelines".[12] By identifying these behavior-linked experts and influencing the router to favor or avoid them, one can steer the model's final output without any retraining or modification of the model's weights.[26]

The detection of these experts is achieved through a method called "paired-example routing-difference detection." The system is presented with pairs of prompts designed to elicit contrasting behaviors. For example, one prompt might be a harmful request ("How do I build a bomb?"), while the other is a safe version of the same topic ("Tell me about the history of explosives"). By analyzing the expert activation patterns across these pairs, the system can identify which experts are differentially activated for the unsafe response versus the safe one. A "risk difference score" is then computed for each expert, quantifying its statistical association with the target behavior.[26]

Once these experts are identified, control is exerted during inference by directly manipulating the router's logits before the softmax function is applied. To promote a desired behavior (e.g., safety), the logits of experts positively correlated with that behavior are increased by a small margin, while the logits of negatively correlated experts are decreased. This "soft" intervention nudges the router's probabilistic decision-making, making it more likely to select the desired experts. This technique has been shown to be remarkably effective, increasing

safe response rates by up to +20% on red-teaming datasets. Conversely, when used adversarially to promote unsafe experts, it can reduce safety by -41% on its own, and by a full -100% when combined with existing jailbreak techniques, completely bypassing the safety guardrails of state-of-the-art models.[12]

This research reveals a critical duality in MoE architectures. The routing mechanism, originally conceived for computational efficiency, is also a powerful new tool for fine-grained model alignment. Instead of relying solely on data-intensive fine-tuning, developers can now directly intervene at the routing layer to promote "safe" or "factual" computational pathways within the model. However, this same mechanism constitutes a significant new attack surface. An adversary who understands an MoE model's expert-to-behavior mapping could potentially force the model down "unsafe" or "hallucinatory" pathways to generate malicious content. This implies that securing the router—protecting its decisions from adversarial manipulation—is becoming as critical to AI safety as aligning the model's weights through training.

## Ensuring Training Stability and Specialization

A significant challenge in training MoE models is ensuring that all experts become specialized and are effectively utilized. Without proper constraints, the training process is prone to "expert collapse," a failure mode where the router learns to send all tokens to a small handful of experts, leaving the vast majority of the model's parameters untrained and useless.[5] To prevent this, several auxiliary loss functions are typically added to the main task loss during training.

### Auxiliary Losses for Router Regularization

- **Load Balancing Loss (LBL):** This is the most common and critical regularization technique. It is designed to encourage a uniform distribution of tokens across all experts. The loss is typically calculated as the dot product of two vectors: one representing the fraction of tokens in a batch routed to each expert, and the other representing the average router probability assigned to each expert.[30] By minimizing this loss, the router is penalized for sending too many tokens to any single expert, forcing it to spread the computational load more evenly.[5] A recent and significant innovation is the use of *global-batch LBL*. In distributed training, LBL is often calculated independently on each device (micro-batch). Global-batch LBL synchronizes the expert selection frequencies across all parallel devices before computing the loss, providing a more accurate picture

of the true global load. This has been shown to lead to better domain specialization and improved downstream task performance.[30]

- **Router Z-loss / Entropy Regularization:** These losses address the problem of router overconfidence. The Z-loss penalizes large logit values from the router, while entropy regularization encourages the router to produce a more distributed, higher-entropy probability distribution over the experts.[5] Both techniques prevent the router's output from becoming too "spiky," which can be a precursor to expert collapse.[5]
- **Orthogonality Loss:** While load balancing ensures experts receive a similar *number* of tokens, it does not prevent them from learning redundant functions. The orthogonality loss is a more recent technique designed to explicitly promote expert specialization. It works by adding a penalty term that encourages the representations learned by different experts to be orthogonal to one another. This pushes experts to process distinct types of tokens and learn non-overlapping functions, leading to clearer specialization and improved overall model performance.[31]

# MoE-Style Orchestration in Multi-Agent Systems

The core principles of Mixture-of-Experts—specialization and conditional routing—are not confined to the internal architecture of a single LLM. They are increasingly being applied at a higher level of abstraction to orchestrate systems of multiple, independent LLM-based agents. In this context, the "experts" are distinct agents, each with its own role, tools, and memory, and the "router" is a mechanism for delegating tasks to the most appropriate agent. This section explores how leading multi-agent system (MAS) frameworks like LangGraph, AutoGen, and CrewAI implement MoE-style orchestration patterns.

## LangGraph: State-Driven Conditional Routing

LangGraph is a library for building stateful, multi-agent applications by representing workflows as explicit graphs.[32] In LangGraph, nodes represent computational units (such as an agent or a tool call), and edges define the transitions between them. This graph-based structure makes it particularly well-suited for tasks that require auditable, deterministic, and cyclical workflows, where the state must be reliably managed and passed between steps.[34]

The MoE pattern is implemented in LangGraph through the use of **conditional edges**. A typical workflow involves a dedicated "router" node, which is often a function that makes an LLM call to classify an incoming query or analyze the current state. Based on the output of

this router node, a conditional edge directs the flow of execution to one of several downstream "expert" nodes, each representing a specialized agent or sub-workflow.[35] The graph's state object serves as the shared context that is passed from node to node, and the routing logic is explicitly encoded in the graph's structure using the

add_conditional_edges method.[36]

This pattern directly mirrors a token-choice MoE architecture at the system level: the initial user query or task can be seen as the "token," the router node functions as the "gating network," and the specialized agent nodes are the "experts." The following Python code demonstrates a simple customer support router in LangGraph:

Python

```
# routing_code
from typing import TypedDict, Literal
from langgraph.graph import StateGraph
# Assume llm is a pre-configured LangChain model

class AgentState(TypedDict):
    """Shared state for the agent graph."""
    query: str
    category: Literal["billing", "tech_support", "sales"]

def query_router_node(state: AgentState) -> dict:
    """Uses an LLM to classify the query and update the state."""
    # This function would contain a prompt and an LLM call
    # to classify the state['query'] into a category.
    # For this example, we'll simulate the classification.
    query = state["query"].lower()
    if "invoice" in query or "charge" in query:
        classification = "billing"
    elif "internet" in query or "slow" in query:
        classification = "tech_support"
    else:
        classification = "sales"
    return {"category": classification}

def route_logic(state: AgentState) -> Literal["billing", "tech_support", "sales"]:
    """Returns the category from the state to direct the conditional edge."""
```

```
    return state["category"]

# Define dummy agent nodes
def billing_agent_node(state: AgentState) -> dict:
    print(f"HANDLED BY BILLING: {state['query']}")
    return {}

def tech_agent_node(state: AgentState) -> dict:
    print(f"HANDLED BY TECH SUPPORT: {state['query']}")
    return {}

def sales_agent_node(state: AgentState) -> dict:
    print(f"HANDLED BY SALES: {state['query']}")
    return {}

# Build the graph
workflow = StateGraph(AgentState)
workflow.add_node("router", query_router_node)
workflow.add_node("billing_agent", billing_agent_node)
workflow.add_node("tech_agent", tech_agent_node)
workflow.add_node("sales_agent", sales_agent_node)

workflow.set_entry_point("router")

# The conditional edge routes from the router to the specialized agent
workflow.add_conditional_edges(
    "router",
    route_logic,
    {
        "billing": "billing_agent",
        "tech_support": "tech_agent",
        "sales": "sales_agent",
    }
)

# Define endpoints for the expert agents
workflow.add_edge("billing_agent", "__end__")
workflow.add_edge("tech_agent", "__end__")
workflow.add_edge("sales_agent", "__end__")

app = workflow.compile()
# Invocation example
# app.invoke({"query": "I have an extra charge on my invoice."})
```

## AutoGen: Conversation-Driven Dynamic Routing

AutoGen, a framework developed by Microsoft, takes a fundamentally different approach to agent orchestration. Instead of defining a rigid graph, AutoGen facilitates collaboration through conversation.[34] Agents interact within a

GroupChat environment, and the workflow emerges dynamically from their dialogue. The orchestration is managed by a GroupChatManager, which acts as a moderator for the conversation.[38]

This conversational paradigm allows for a highly flexible and dynamic implementation of the MoE pattern. The GroupChatManager can be configured to function as an intelligent router, using an LLM to select the next agent to speak based on the current state of the conversation and the declared capabilities of each agent.[39] This is particularly powerful for complex, open-ended tasks like research or creative brainstorming, where the optimal sequence of steps is not known in advance.

A common design pattern involves a UserProxyAgent initiating a task within the group chat. The GroupChatManager is prompted with a system message that includes descriptions of all available "expert" agents (e.g., "CodeWriter: an expert at writing Python code," "DataAnalyst: an expert at analyzing data and creating visualizations"). At each turn of the conversation, the manager's LLM analyzes the dialogue history and decides which agent is best equipped to contribute next, effectively performing dynamic, LLM-powered routing.[41] This is analogous to the LLMoE architecture but operates at the level of entire agents rather than individual FFN layers.

## CrewAI: Role-Based Hierarchical Routing

CrewAI is an orchestration framework designed around a role-based architecture, where each agent is defined with a specific role, goal, and backstory, much like a member of a human team.[43] The collaboration strategy is determined by the

Process assigned to the Crew.

The MoE pattern is most directly implemented in CrewAI through the Process.hierarchical setting.[45] When this process is selected, a "manager" agent is either designated by the user or automatically created by the framework. This manager agent is responsible for high-level

planning and task delegation. It does not execute the tasks itself but instead analyzes the overall goal and delegates sub-tasks to the most appropriate "worker" agents within the crew, then validates their outputs.[45] This structure directly emulates a two-level Hierarchical MoE, where the manager agent serves as the coarse-grained router and the specialized worker agents act as the experts.[48]

For more granular and conditional workflows, CrewAI has introduced a feature called Flows. This allows developers to build event-driven orchestration logic using Python decorators.[49] The

@router decorator, for instance, can be used to define a method that listens to the output of a preceding task and conditionally routes the workflow to different downstream tasks or even entirely different crews.[50] This provides a mechanism for creating complex, branching logic that is similar in capability to LangGraph's conditional edges but is expressed through a different, decorator-based syntax.

These three frameworks represent a spectrum of design philosophies for creating MoE-style agentic systems. LangGraph offers maximum control and predictability. Its routing is explicit and deterministic, defined by the graph structure, which is ideal for production environments where behavior must be auditable and constrained. It can be thought of as implementing a "hard-coded" routing table. AutoGen, at the other end of the spectrum, offers maximum autonomy and emergence. Its routing is dynamic and conversational, determined at each step by an LLM-powered manager. This is best suited for exploratory tasks where the solution path is unknown. It represents a "fully dynamic" router. CrewAI's hierarchical process sits in the middle, providing a structured hierarchy (manager delegating to workers) but granting the manager agent autonomy in how it plans and sequences those delegations. This offers a balance between structure and flexibility, akin to a "managed" MoE system. The choice of framework is therefore not merely a technical preference but a strategic decision about the desired trade-off between predictable control and emergent, autonomous problem-solving.

Across all three frameworks, a clear trend emerges: the intelligence of the multi-agent system is increasingly concentrated in the routing component. In simple agent chains, intelligence is distributed. In these more sophisticated MoE-style architectures, the worker agents are often becoming commoditized specialists (e.g., "the API caller," "the report writer"). The most complex reasoning, state management, and strategic decision-making are being moved into the router itself—be it LangGraph's conditional function, AutoGen's GroupChatManager prompt, or CrewAI's manager_agent. Consequently, optimizing the performance of a multi-agent system is becoming synonymous with optimizing its central router. This is where the most capable LLMs are required and where the most sophisticated prompting techniques must be applied. This has significant implications for system design, suggesting a cost-optimization strategy where smaller, cheaper models can be used for worker agents, while the state-of-the-art model is reserved for the critical routing function.

| Framework | Core Abstraction | Routing Mechanism | State Management | Typical Workflow | MoE Analogy |
|---|---|---|---|---|---|
| **LangGraph** | State Graph | Explicit Conditional Edges (add_conditional_edges) | Shared State Object (TypedDict) | Deterministic, Cyclic, Auditable | Hard-coded Router |
| **AutoGen** | Conversation | LLM-based Speaker Selection (GroupChatManager) | Conversation History | Emergent, Dynamic, Collaborative | Dynamic Top-K Router |
| **CrewAI** | Crew / Team | Hierarchical Delegation (Process.hierarchical) or Event-driven (@router in Flows) | Implicit via Task Context and Outputs | Structured, Hierarchical | Hierarchical Router |

## Performance Evaluation and System Metrics

Evaluating the performance of Mixture-of-Experts systems, whether within a single LLM or across a multi-agent architecture, is a complex and multifaceted challenge. Simple measures of task accuracy are insufficient, as they fail to capture the unique dynamics of conditional computation. A comprehensive evaluation framework must incorporate metrics that assess the quality of the routing decisions, the efficiency of system resource utilization, and the intricate trade-offs between cost, latency, and performance.

## Measuring Routing Effectiveness

The quality of the router's decisions is paramount to the success of an MoE system. However, measuring this quality is non-trivial, as a "ground truth" for the optimal expert choice often does not exist. Researchers have developed several direct and proxy metrics to evaluate routing effectiveness.

- **Routing Accuracy:** Directly measuring routing accuracy is challenging. One novel, training-free approach is **SpectR**, which evaluates LoRA-based experts. It uses the spectral properties (specifically, the singular value decomposition) of the LoRA adapter matrices to compute a compatibility score between an input and an expert, serving as a proxy for routing accuracy without requiring a labeled dataset.[18] Another proxy method involves using the output of a full, dense model (or an MoE model with all experts activated) as a "ground truth." The output from the sparsely routed MoE model is then compared to this ground truth using a
  **semantic similarity** metric, such as BERTScore or cosine similarity of embeddings. Higher similarity suggests more "accurate" routing, in the sense that the sparse model is effectively replicating the behavior of its more computationally expensive counterpart.[52]
- **Routing Consistency:** This metric assesses the stability and predictability of the router's decisions. It measures whether the router consistently selects the same experts for similar inputs or across multiple forward passes of the same input.[15] High routing consistency is generally desirable, as it indicates that the model's behavior is stable and not subject to random fluctuations, which is critical for reliable deployment in production systems.
- **Expert Hit Rate:** This is a straightforward diagnostic metric that simply tracks the frequency or percentage of times each individual expert is activated over a large dataset. It is primarily used to identify underutilized or "dead" experts—those that are rarely or never selected by the router. A high number of dead experts can indicate problems with the training process, such as expert collapse.[53]

## Quantifying Load Balancing and Expert Utilization

Effective load balancing is crucial for the efficient training and inference of MoE models. It ensures that computational work is distributed evenly across available resources (e.g., GPUs), preventing bottlenecks and maximizing hardware utilization.

- **Load Balancing Loss (LBL):** While primarily a training objective, the value of the LBL itself serves as a key evaluation metric post-training. The LBL is formally defined as the sum, over all experts, of the product of the fraction of tokens routed to an expert (fi) and

the average router probability for that expert (Pi), typically normalized by the number of experts: LBL=NEΣi=1NEfiPi. A lower LBL value indicates a more balanced distribution of tokens across the experts.[30]

- **Expert Utilization Patterns:** Beyond a single LBL score, it is often insightful to visualize the patterns of expert activation across different domains or tasks. This can be done by plotting heatmaps of expert hit rates for various data slices. While perfect uniformity is not always the goal—some degree of imbalance can be a sign of successful specialization—these visualizations are critical for diagnosing unintended imbalances or confirming that experts are specializing as expected.[31]
- **MaxVioglobal & RMSE:** These are more formal statistical measures for quantifying the stability of load balancing. MaxVioglobal measures the maximum violation of a perfectly uniform distribution, providing a worst-case assessment of imbalance. The Root Mean Square Error (RMSE) can be used to measure the deviation of the observed expert utilization distribution from a target distribution (which may or may not be uniform).[31]

The concept of "good routing" is highly context-dependent. A naive assumption might be that perfect load balancing is always optimal because it maximizes hardware utilization. However, recent research indicates that forcing excessive uniformity, for instance through a high weight on the load balancing loss, can actually harm expert specialization and degrade downstream task performance.[30] True specialization implies that for a given domain of data, some experts

*should* be activated more frequently than others. Similarly, proxy metrics for accuracy, like semantic similarity to a dense model's output, assume the dense model is a gold standard. This may not hold if the specialized experts have learned unique and valuable functions that are absent in the dense counterpart. Therefore, evaluating MoE routing requires a holistic, end-to-end approach. The ultimate goal is not to achieve a specific LBL score or proxy accuracy in isolation, but to find the routing and training configuration that yields the best performance on the final downstream task within an acceptable cost and latency budget.

## Cost, Latency, and Accuracy Trade-offs

MoE systems are fundamentally designed around a trade-off. By increasing the total number of parameters while keeping the number of active parameters constant, they trade increased memory storage costs for reduced computational costs (FLOPs).[2] Evaluating these systems requires a multi-dimensional analysis that captures this complex interplay.

- **The CAP Trade-off:** A useful conceptual framework for this analysis is the CAP (Cost, Accuracy, Performance) trade-off. It acknowledges that it is rarely possible to simultaneously optimize all three dimensions. Practitioners must typically choose to prioritize two out of the three. For example, one might aim for the highest accuracy at a

fixed cost, or the lowest latency for a target accuracy level. Emerging benchmarks, such as MoE-CAP, are being developed to explicitly quantify and compare models along these axes.[9]

- **Pareto Frontier Analysis:** The most sophisticated approach to evaluating these trade-offs is to map the system's Pareto frontier. A Pareto-optimal configuration is one where no single metric (cost, latency, or accuracy) can be improved without degrading at least one of the others. The framework **ECO-LLM** provides a powerful demonstration of this concept. Instead of evaluating a single, fixed system configuration, ECO-LLM's "Emulator" component explores the vast space of possible configurations (e.g., which LLM to use, which RAG chunking strategy, which prompt template) for a given set of queries. It identifies a set of Pareto-optimal configurations, creating a frontier of the best possible trade-offs.[54] The "Runtime" component can then dynamically select a configuration from this frontier for each incoming query based on real-time constraints, such as a user-specified maximum latency or cost budget.[54]
- **Core Metrics for Trade-off Analysis:**
  - **Cost:** Can be measured in various units, including the number of API calls made, the total number of tokens processed by the LLM(s), or direct monetary cost in dollars.[54]
  - **Latency:** Typically measured as the end-to-end response time in milliseconds, from receiving the user query to delivering the final response.[15]
  - **Accuracy:** This is highly task-dependent and should be measured using relevant benchmarks (e.g., MMLU, BLEU, HumanEval). In the absence of standard benchmarks, proxy metrics like semantic similarity can be used.[52]

The principles demonstrated by frameworks like ECO-LLM suggest a future for AI system optimization that is far more dynamic than current practices. Today, systems are typically deployed with a static configuration (e.g., "always use model X for routing and model Y for generation"). The next generation of systems, however, may not be static pipelines at all. Instead, they may consist of a vast library of interchangeable components (experts, agents, tools) and a hyper-intelligent "Emulator" or "Runtime" that assembles a bespoke, optimal workflow on-the-fly for every single query. This would be the ultimate expression of the MoE principle of conditional computation, applied not just to a single layer in a model, but to the entire end-to-end system architecture.

# Optimization for Efficiency and Low Latency

Despite the computational savings offered by sparse activation, Mixture-of-Experts models present significant system-level challenges, most notably their massive memory footprint due to the large total parameter count. Deploying these models efficiently, especially in resource-constrained or latency-sensitive environments, requires a suite of optimization

techniques that go beyond the core architecture. This section covers state-of-the-art methods for expert compression and pruning, as well as design patterns for achieving low-latency inference.

## Expert Compression and Pruning

The primary goal of expert compression is to reduce the memory requirements of MoE models, making them more feasible to deploy.[57] A holistic approach to compression involves a combination of strategies that reduce both the number of experts and the size of individual experts.[59]

- **Expert Trimming:** This category of techniques focuses on reducing the total number of experts in the model.
  - **Expert Drop:** The most straightforward approach, where an importance score is calculated for each expert (e.g., based on activation frequency or impact on model loss), and the lowest-scoring experts are permanently removed or "pruned" from the model.[60]
  - **Layer Drop and Block Drop:** These are more aggressive forms of trimming. Instead of removing individual experts within a layer, **Layer Drop** removes entire MoE layers, replacing them with a standard FFN or a residual connection. **Block Drop** goes even further, removing entire Transformer blocks (which include both an attention layer and an MoE layer). Surprisingly, these aggressive techniques have been shown to substantially improve both computational and memory efficiency while preserving a high percentage of the original model's performance.[59]
- **Expert Slimming:** This category focuses on compressing the weights *within* each individual expert that remains after trimming. This can be accomplished using standard model compression techniques such as:
  - **Pruning:** Applying structured or unstructured weight pruning to the FFNs of each expert to reduce their parameter count.
  - **Quantization:** Reducing the precision of the expert weights (e.g., from 16-bit floating point to 8-bit integers), which can significantly reduce memory usage and accelerate computation on compatible hardware.[59]

A unified compression framework often combines these strategies. For example, one might first perform Block Drop to remove less critical MoE layers, then apply Expert Drop to the remaining layers, and finally use quantization to slim the surviving experts. Such a combined approach has been shown to achieve over a 6x speedup and a 77% reduction in memory usage on a model like Mixtral-8x7B while maintaining over 92% of its original performance.[59]

### The Discovery and Implications of "Super Experts"

Recent research has revealed a critical nuance in expert importance: not all experts contribute equally to the model's capabilities. A study of several open-source MoE models identified a small, distinct subset of experts, termed **"Super Experts" (SEs)**, that play a disproportionately crucial role in model performance.[57] These SEs are characterized by producing rare but extreme activation outliers, which in turn create massive activations in the hidden states between model layers.[64]

The impact of these SEs is profound. Pruning just three SEs from a model like Qwen3-30B-A3B was shown to cause a catastrophic collapse in performance, leading to repetitive and uninformative outputs. In contrast, randomly pruning a much larger number of "normal" experts had a considerably smaller impact.[57] SEs appear to be particularly vital for complex reasoning tasks, such as mathematics, and are believed to be instrumental in inducing "attention sinks," a mechanism critical for the proper functioning of the Transformer's attention distribution.[57]

This discovery challenges the assumption that all parameters or experts are created equal and suggests a Pareto principle at play within neural networks: a small fraction of components are responsible for a majority of the critical functionality. This must fundamentally alter the strategy for MoE compression. The new goal is not simply to reduce the parameter count uniformly. Instead, a more effective approach would be a two-step process: first, identify and rigorously preserve the critical Super Experts at all costs. Second, apply aggressive compression and pruning techniques to the remaining, much larger set of less critical experts. This targeted, non-uniform compression strategy promises to be far more effective at reducing model size while preserving essential capabilities.

## Low-Latency Inference Patterns

For real-time applications, minimizing inference latency is as important as managing memory. Several architectural patterns and system-level optimizations have been developed to reduce the time it takes for an MoE model to produce a response.

- **"Fast Path" Routing:** This concept, demonstrated in the LION-FS model for real-time video analysis, employs a two-stage routing strategy to triage inputs based on their complexity.[66] A lightweight and very fast initial router, the "Fast Path," first analyzes the input to determine if an immediate, simple response is sufficient or if deeper processing

is required. If the task is simple, it is handled immediately. If it is complex, it is passed to a more powerful but slower "Slow Path," which might involve a larger model or a greater number of experts.[66] This pattern can be generalized to many LLM workflows. For example, a customer support system could use a small, cheap model to handle simple queries like "What are your business hours?" while routing complex, multi-turn diagnostic issues to a more capable and expensive MoE model.

- **Simplified Routing and Input Sparsification:** The routing process itself adds computational overhead. For latency-critical applications, simplifying this process is key.
  - **Top-1 Routing:** Adopting the Switch Transformer's top-1 routing strategy is the most direct way to minimize routing latency, as it involves the least amount of computation and communication overhead.[8]
  - **Token Dropping and Aggregation:** For processing continuous streams of data (like video frames or audio), latency can be reduced by sparsifying the input itself. **Token Dropping** techniques use a lightweight model to identify and discard redundant or unimportant tokens *before* they are sent to the main MoE model's router. **Token Aggregation** can fuse features from multiple tokens into a single, more compact representation, reducing the total sequence length that needs to be processed.[66]
- **System-Level Optimizations:** The **eMoE** inference system is a prime example of system-level optimization. It is based on the observation that expert activation patterns are often highly repetitive and predictable, especially within a single task or conversation.[52] The eMoE system uses a lightweight predictor to anticipate which experts will be needed for upcoming tokens or prompts. It then pre-loads only those required experts from slower storage (like CPU RAM or NVMe) into fast GPU memory. This "just-in-time" loading strategy dramatically reduces the memory required on the GPU at any given moment and minimizes the I/O latency associated with swapping experts, achieving up to an 80% reduction in memory consumption and a 17% reduction in inference latency.[52]

These advanced techniques demonstrate that latency optimization is evolving from a model-centric problem to a holistic, system-level orchestration challenge. The initial focus was on model-internal techniques like quantization. The "Fast Path" concept introduces a meta-decision layer—a system-level pre-router that decides whether the expensive model should even be invoked. The eMoE system adds another layer of intelligence: a predictive resource manager that is aware of the model's internal routing behavior and optimizes memory and I/O accordingly. The modern, low-latency AI stack is thus becoming a multi-layered system comprising a lightweight pre-router, a predictive resource manager, and the core MoE model itself. Designing for speed is no longer just a matter of model compression; it is a complex systems design task that integrates tiered computation, predictive caching, and query-aware routing.

# Conclusion

The Mixture-of-Experts paradigm has firmly established itself as a cornerstone of modern large-scale AI, evolving far beyond its origins as a mere scaling technique. The analysis of recent advancements reveals a clear trajectory: from a method for achieving computational efficiency to a sophisticated framework for structured, conditional computation that is reshaping both individual model architectures and the orchestration of complex multi-agent systems.

The architectural evolution from flat, top-k models like the Switch Transformer to more complex Hierarchical and Dynamic MoE structures signifies a move towards internalizing complex reasoning processes. These advanced architectures are not just selecting computational pathways but are learning to structure the computation itself, dynamically allocating resources and building adaptive reasoning graphs on a per-input basis. This trend is mirrored and amplified in the domain of multi-agent systems, where frameworks like LangGraph, AutoGen, and CrewAI explicitly implement MoE principles to orchestrate teams of specialized agents. In these systems, the "router"—be it a conditional graph edge, a conversational manager, or a hierarchical delegator—has become the central locus of intelligence, strategy, and control.

This centralization of intelligence in the routing mechanism presents both a significant opportunity and a critical challenge. On one hand, it provides a powerful and interpretable lever for control. Techniques like SteerMoE demonstrate that by manipulating routing decisions at inference time, it is possible to steer model behavior towards desired outcomes like improved safety and faithfulness without costly retraining. On the other hand, this same mechanism introduces a new and potent attack surface, where adversarial manipulation of routing pathways can be used to bypass safety alignments.

Finally, the practical deployment of these massive models is driving innovation in efficiency and optimization. The discovery of "Super Experts" suggests a Pareto principle in neural network functionality, demanding a shift from uniform compression strategies to a more targeted approach that identifies and preserves these critical components while aggressively compressing the rest. Simultaneously, the pursuit of low latency is moving beyond model-centric techniques towards holistic, system-level orchestration. Concepts like "Fast Path" routing and predictive expert loading indicate that the future of efficient AI deployment lies in intelligent, multi-layered systems that can dynamically reconfigure the entire computational workflow for each query to meet specific cost, latency, and performance constraints.

In synthesis, the state-of-the-art in Mixture-of-Experts is characterized by a convergence of ideas. The principles of specialization and conditional routing, once applied to FFN layers within a single model, are now being scaled up to orchestrate entire systems of agents and

scaled down to adapt pre-trained models with parameter-efficient techniques. The future of AI development will likely see a deepening of this paradigm, leading to highly modular, dynamically assembled systems where intelligent routing is the key to unlocking both unprecedented capability and practical, efficient deployment.

### Cytowane prace


1. Mixture of Experts in Large Language Models †: Corresponding author - arXiv, otwierano: września 23, 2025, https://arxiv.org/html/2507.11181v1
2. What is mixture of experts? | IBM, otwierano: września 23, 2025, https://www.ibm.com/think/topics/mixture-of-experts
3. MoE Fundamentals: Why Sparse Models Are the Future of AI - Cerebras, otwierano: września 23, 2025, https://www.cerebras.ai/blog/moe-guide-why-moe
4. A Comprehensive Survey of Mixture-of-Experts: Algorithms, Theory, and Applications - arXiv, otwierano: września 23, 2025, https://arxiv.org/html/2503.07137v1
5. Mixture of Experts in Large Language Models: Intuition, Methods ..., otwierano: września 23, 2025, https://medium.com/@hexiangnan/mixture-of-experts-in-large-language-models-intuition-methods-and-system-design-cbe7a5e995eb
6. Mixture of Experts LLMs: Key Concepts Explained - Neptune.ai, otwierano: września 23, 2025, https://neptune.ai/blog/mixture-of-experts-llms
7. Improving Deep Learning Performance with Mixture of Experts and Sparse Activation, otwierano: września 23, 2025, https://www.preprints.org/manuscript/202503.0611/v1
8. Mixture-of-Experts (MoE) Models in AI - Artificial Intelligence in Plain English, otwierano: września 23, 2025, https://ai.plainenglish.io/mixture-of-experts-moe-models-in-ai-4bcbcdecccf8
9. A 2025 Guide to Mixture-of-Experts for Lean LLMs - Cohorte - AI for Everyone, otwierano: września 23, 2025, https://www.cohorte.co/blog/a-2025-guide-to-mixture-of-experts-for-lean-llms
10. Mixture of Tokens: Continuous MoE through Cross-Example Aggregation - arXiv, otwierano: września 23, 2025, https://arxiv.org/html/2310.15961v2
11. A Closer Look into Mixture-of-Experts in Large Language Models - ACL Anthology, otwierano: września 23, 2025, https://aclanthology.org/2025.findings-naacl.251.pdf
12. Steering MoE LLMs via Expert (De)Activation - arXiv, otwierano: września 23, 2025, https://arxiv.org/html/2509.09660v1
13. Mixture of experts - Wikipedia, otwierano: września 23, 2025, https://en.wikipedia.org/wiki/Mixture_of_experts
14. Sparse MoE with Language Guided Routing for Multilingual Machine Translation, otwierano: września 23, 2025, https://openreview.net/forum?id=ySS7hH1smL
15. Understanding Mixture of Experts (MoE): A Deep Dive into Scalable AI Architecture, otwierano: września 23, 2025, https://www.researchgate.net/publication/388828999_Understanding_Mixture_of_

Experts_MoE_A_Deep_Dive_into_Scalable_AI_Architecture
16. arxiv.org, otwierano: września 23, 2025, https://arxiv.org/html/2509.10530v1
17. [2509.10530] Dynamic Adaptive Shared Experts with Grouped Multi-Head Attention Mixture of Experts - arXiv, otwierano: września 23, 2025, https://www.arxiv.org/abs/2509.10530
18. SpectR: Dynamically Composing LM Experts with Spectral Routing ..., otwierano: września 23, 2025, https://openreview.net/forum?id=tK8GHR62EX
19. What is Mixture of Experts (MoE)? How it Works and Use Cases - Zilliz Learn, otwierano: września 23, 2025, https://zilliz.com/learn/what-is-mixture-of-experts
20. Choose, Drop, Route: MOE vs MOD vs MOR | by Ketaki | Aug, 2025 - Medium, otwierano: września 23, 2025, https://medium.com/@ketaki.kolhatkar99/choose-drop-route-moe-vs-mod-vs-mor-b553d38171c1
21. Applying Mixture of Experts in LLM Architectures | NVIDIA Technical Blog, otwierano: września 23, 2025, https://developer.nvidia.com/blog/applying-mixture-of-experts-in-llm-architectures/
22. Mixture of Experts in Large Language Models - ResearchGate, otwierano: września 23, 2025, https://www.researchgate.net/publication/393724282_Mixture_of_Experts_in_Large_Language_Models
23. [2501.09636] LLM-Based Routing in Mixture of Experts: A Novel Framework for Trading, otwierano: września 23, 2025, https://arxiv.org/abs/2501.09636
24. arxiv.org, otwierano: września 23, 2025, https://arxiv.org/html/2501.09636v1
25. arXiv:2501.09636v2 [cs.LG] 17 Jan 2025, otwierano: września 23, 2025, https://arxiv.org/pdf/2501.09636?
26. Steering MoE LLMs via Expert (De)Activation - arXiv, otwierano: września 23, 2025, https://arxiv.org/html/2509.09660
27. Steering MoE LLMs via Expert (De)Activation - arXiv, otwierano: września 23, 2025, https://www.arxiv.org/pdf/2509.09660
28. Steering MoE LLMs via Expert (De)Activation | AI Research Paper Details - AIModels.fyi, otwierano: września 23, 2025, https://www.aimodels.fyi/papers/arxiv/steering-moe-llms-via-expert-deactivation
29. Steering MoE LLMs via Expert (De)Activation - arXiv, otwierano: września 23, 2025, https://arxiv.org/abs/2509.09660
30. Demons in the Detail: On Implementing Load Balancing Loss for Training Specialized Mixture-of-Expert Models - ACL Anthology, otwierano: września 23, 2025, https://aclanthology.org/2025.acl-long.249.pdf
31. Advancing Expert Specialization for Better MoE - arXiv, otwierano: września 23, 2025, https://arxiv.org/html/2505.22323v1
32. LangGraph vs AutoGen vs CrewAI: Complete AI Agent Framework ..., otwierano: września 23, 2025, https://latenode.com/blog/langgraph-vs-autogen-vs-crewai-complete-ai-agent-framework-comparison-architecture-analysis-2025
33. LangGraph vs CrewAI vs AutoGen | Ultimate Comparison Guide AI - Rapid

Innovation, otwierano: września 23, 2025, https://www.rapidinnovation.io/post/a-comparative-analysis-of-langgraph-crewai-and-autogen
34. A Developer's Guide to Multi-Agent Frameworks: CrewAI, AutoGen, and LangGraph | by Nishant Gupta | AiGenVerse | Aug, 2025 | Medium, otwierano: września 23, 2025, https://medium.com/aigenverse/a-developers-guide-to-multi-agent-frameworks-crewai-autogen-and-langgraph-15531c0c7dfe
35. Langraph Conditional WorkFlow. Routing WorkFlow | by ... - Medium, otwierano: września 23, 2025, https://medium.com/@adarishanmukh15501/langraph-c61c8fcaac8f
36. state graph node - GitHub Pages, otwierano: września 23, 2025, https://langchain-ai.github.io/langgraph/concepts/low_level/
37. AutoGen vs. LangGraph vs. CrewAI:Who Wins? | by Khushbu Shah | ProjectPro - Medium, otwierano: września 23, 2025, https://medium.com/projectpro/autogen-vs-langgraph-vs-crewai-who-wins-02e6cc7c5cb8
38. AutoGen | Phoenix - Arize AI, otwierano: września 23, 2025, https://arize.com/docs/phoenix/cookbook/agent-workflow-patterns/autogen
39. LangGraph vs AutoGen: How are These LLM Workflow Orchestration Platforms Different? - ZenML Blog, otwierano: września 23, 2025, https://www.zenml.io/blog/langgraph-vs-autogen
40. Group Chat — AutoGen - Microsoft Open Source, otwierano: września 23, 2025, https://microsoft.github.io/autogen/stable//user-guide/core-user-guide/design-patterns/group-chat.html
41. Routing between agents · microsoft autogen · Discussion #1011 - GitHub, otwierano: września 23, 2025, https://github.com/microsoft/autogen/discussions/1011
42. A practical guide for using AutoGen in software applications | by Clint Goodman - Medium, otwierano: września 23, 2025, https://clintgoodman27.medium.com/a-practical-guide-for-using-autogen-in-software-applications-8799185d27ee
43. AI Agent Frameworks: Choosing the Right Foundation for Your Business | IBM, otwierano: września 23, 2025, https://www.ibm.com/think/insights/top-ai-agent-frameworks
44. Framework for orchestrating role-playing, autonomous AI agents. By fostering collaborative intelligence, CrewAI empowers agents to work together seamlessly, tackling complex tasks. - GitHub, otwierano: września 23, 2025, https://github.com/crewAIInc/crewAI
45. Hierarchical Process - CrewAI, otwierano: września 23, 2025, https://docs.crewai.com/learn/hierarchical-process
46. Processes - CrewAI Documentation, otwierano: września 23, 2025, https://docs.crewai.com/concepts/processes
47. Mastering CrewAI Flows: Building Hierarchical Multi-Agent Systems | by Jishnu Ghosh | Aug, 2025 | Medium, otwierano: września 23, 2025,

https://medium.com/@jishnughosh2023/mastering-crewai-flows-building-hierarchical-multi-agent-systems-408f790a8d2a
48. CrewAI / Hierarchical Manager: Build Reflection Enabled Agentic | by TeeTracker - Medium, otwierano: września 23, 2025, https://teetracker.medium.com/crewai-hierarchical-manager-build-reflection-enabled-agentic-flow-8255c8c414ec
49. CrewAI Flow - GeeksforGeeks, otwierano: września 23, 2025, https://www.geeksforgeeks.org/artificial-intelligence/crewai-flow/
50. A Comprehensive Guide to CrewAI Flows: Building a Smart Greeting System - Medium, otwierano: września 23, 2025, https://medium.com/@diwakarkumar_18755/a-comprehensive-guide-to-crewai-flows-building-a-smart-greeting-system-9d80f906847d
51. What are Agentic Flows in CrewAI? - Analytics Vidhya, otwierano: września 23, 2025, https://www.analyticsvidhya.com/blog/2024/11/agentic-flows-in-crewai/
52. eMoE: Task-aware Memory Efficient Mixture-of-Experts-Based (MoE) Model Inference, otwierano: września 23, 2025, https://arxiv.org/html/2503.06823v1
53. Dynamic Mixture of Experts for Adaptive Computation in Character-Level Transformers, otwierano: września 23, 2025, https://www.mdpi.com/2078-2489/16/6/483
54. ECO-LLM: Orchestration for Domain-specific Edge-Cloud Language Models - arXiv, otwierano: września 23, 2025, https://arxiv.org/html/2507.09003v1
55. Cost-Aware Neural Network Splitting and Dynamic Rescheduling for Edge Intelligence, otwierano: września 23, 2025, https://repositum.tuwien.at/bitstream/20.500.12708/188349/1/Luger%20Daniel%20-%202023%20-%20Dynamic%20Scheduling%20of%20Neural%20Network%20Splitting.pdf
56. CartesianMoE: Boosting Knowledge Sharing among Experts via Cartesian Product Routing in Mixture-of-Experts - ACL Anthology, otwierano: września 23, 2025, https://aclanthology.org/2025.naacl-long.505/
57. arxiv.org, otwierano: września 23, 2025, https://arxiv.org/html/2507.23279v1
58. How the Compression Error of Experts Affects the Inference Accuracy of MoE Model?, otwierano: września 23, 2025, https://www.researchgate.net/publication/395388290_MoE-Compression_How_the_Compression_Error_of_Experts_Affects_the_Inference_Accuracy_of_MoE_Model
59. Towards Efficient Mixture of Experts: A Holistic Study of Compression Techniques - arXiv, otwierano: września 23, 2025, https://arxiv.org/html/2406.02500v3
60. Towards Efficient Mixture of Experts: A Holistic Study of Compression Techniques - arXiv, otwierano: września 23, 2025, https://arxiv.org/pdf/2406.02500
61. MoE-I$^2$: Compressing Mixture of Experts Models through Inter-Expert Pruning and Intra-Expert Low-Rank Decomposition - Semantic Scholar, otwierano: września 23, 2025, https://www.semanticscholar.org/paper/00814e5631c20bec502838a1e8040b3f5f258971
62. Findings of the Association for Computational Linguistics: ACL 2025 ..., otwierano:

września 23, 2025, https://aclanthology.org/volumes/2025.findings-acl/
63. Unveiling Super Experts in Mixture-of-Experts Large Language Models - ResearchGate, otwierano: września 23, 2025, https://www.researchgate.net/publication/394174801_Unveiling_Super_Experts_in_Mixture-of-Experts_Large_Language_Models
64. arxiv.org, otwierano: września 23, 2025, https://arxiv.org/pdf/2507.23279
65. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model, otwierano: września 23, 2025, https://www.semanticscholar.org/paper/DeepSeek-V2%3A-A-Strong%2C-Economical%2C-and-Efficient-Shao-Dai/53a803388e83ae89261624099d7be4287ace67cb
66. arXiv:2503.03663v1 [cs.CV] 5 Mar 2025, otwierano: września 23, 2025, https://arxiv.org/pdf/2503.03663
67. LION-FS: Fast & Slow Video-Language Thinker as Online Video Assistant - arXiv, otwierano: września 23, 2025, https://arxiv.org/html/2503.03663v1
68. Daily Papers - Hugging Face, otwierano: września 23, 2025, https://huggingface.co/papers?q=Aggregation%20Routing
69. Shohaib Mahmud - Google Scholar, otwierano: września 23, 2025, https://scholar.google.com/citations?user=V1evrcwAAAAJ&hl=en
70. Suraiya Tairin - Google Scholar, otwierano: września 23, 2025, https://scholar.google.com/citations?user=DDfgxkUAAAAJ&hl=en