Architectural Advances in AI Through a Systems Lens

Why Model-Level Advances Fail Without Execution Capabilities

Technical White Paper

Author: Anjan Goswami

Abstract

The period spanning 2024–2025 represents a structural inflection point in artificial intelligence, not because of new model ideas alone, but because of shifts in the execution constraints that determine which architectures are viable in practice. Progress is no longer dominated by parameter scaling, but by the interaction of sparse model architectures, inference-time reasoning, and systems-level efficiency. Memory bandwidth and energy consumption now dominate inference economics, driving architectural choices such as Mixture-of-Experts, long-context attention optimization, and accelerator co-design. Simultaneously, reasoning-oriented models reintroduce search and process supervision at inference time, enabling capabilities that cannot be realized through single-pass decoding.

This paper examines these developments across the full AI stack—from model architecture and training paradigms to compiler infrastructure, runtime systems, accelerators, interconnect, and scientific applications—and shows that the practical impact of modern AI architectures is constrained less by model design than by the maturity of the execution stack beneath them. Architectural advances succeed only when supported by execution capabilities that can sustain conditional computation, dynamic scheduling, memory locality, and long-horizon reliability. Absent these capabilities, many widely reported advances remain theoretical, brittle, or economically infeasible.

This document is intended for technical audiences in AI research, systems engineering, and applied science. Metrics and examples reflect publicly reported results as of 2025.

Contents

1 Introduction: From Model Scaling to Systems Convergence

For more than a decade, progress in artificial intelligence has been driven by scale. Increases in model size, training data, and compute budgets produced consistent and often predictable improvements across language, vision, and multimodal tasks. This paradigm, formalized through empirical scaling laws, justified the prevailing assumption that capability growth could be sustained primarily through continued expansion of parameters and datasets.

The 2024–2025 period marks a structural inflection point in this trajectory. While scaling continues to yield benefits, performance, cost, and deployability are now governed by constraints that scaling alone cannot overcome. Inference workloads are increasingly dominated by memory bandwidth and energy consumption rather than arithmetic throughput. Latency-sensitive applications expose inefficiencies in dense architectures, while long-context and agentic workloads amplify memory pressure and interconnect overhead. As a result, the marginal returns of parameter growth diminish unless accompanied by fundamental changes in architecture and execution.

Three forces have converged to reshape the AI landscape. First, inference has become fundamentally memory- and energy-bound. On modern accelerators, peak floating-point throughput exceeds what can be sustained under realistic memory access patterns, forcing architectural sparsity, locality-aware execution, and careful management of data movement. This has driven the adoption of sparse activation schemes, such as Mixture-of-Experts, and renewed attention to kernel fusion, cache behavior, and interconnect topology.

Second, advances in reasoning capability have emerged primarily through inference-time computation rather than parameter growth. Reasoning-oriented models allocate additional compute at test time to explore alternative solution paths, verify intermediate steps, and recover from errors. This reintroduces controlled forms of search into neural systems, challenging the assumption that intelligence must be fully amortized into static model weights. The resulting shift reframes capability as a function of inference policy and runtime behavior, not solely of training scale.

Third, efficiency has become a binding constraint across the entire stack. The cost of moving data now dominates the cost of computation, making memory hierarchy, interconnect, and energy consumption first-order design considerations. Compiler decisions, runtime scheduling, and quantization policies increasingly determine whether theoretical model capabilities are realized in practice. At the datacenter level, power delivery and cooling capacity directly constrain deployment velocity, further elevating efficiency from an optimization goal to a feasibility requirement.

This paper develops the thesis that modern AI progress is best understood through a *systems lens*: capability emerges from the interaction of model architecture, inference-time reasoning policies, and execution efficiency under physical and execution constraints. No single layer is sufficient in isolation. Architectural choices shape memory access patterns; reasoning strategies depend on runtime scheduling and verification; and efficiency constraints propagate upward to determine which models and capabilities are economically viable.

To substantiate this thesis, the paper examines a sequence of architectural responses to distinct execution constraints across the AI stack. Section 2 analyzes sparse architectures, particularly Mixture-of-Experts, as a response to bandwidth-dominated inference. Section 3 examines inference-time reasoning as a response to the limits of amortized execution. Sections 4 and 5 explore attention optimization, compiler infrastructure, and runtime design as responses to memory hierarchy constraints and execution complexity. Section 6 discusses diffusion transformers as a response to generative scaling limits across generative and scientific AI. Sections 7 and 8 analyze accelerator, interconnect, and agentic system design as responses to energy constraints and long-horizon reliability. The paper concludes by synthesizing these responses into a convergent systems perspective on modern AI architectures beyond pure scaling.

2 Mixture-of-Experts as a Systems-Level Architectural Response

Large language model inference is increasingly constrained by memory bandwidth rather than arithmetic throughput. During autoregressive decoding, each generated token requires repeated access to model parameters stored in high-bandwidth memory (HBM), while intermediate activations remain comparatively small. On contemporary accelerators, peak tensor-core throughput exceeds hundreds of teraflops, yet sustained utilization during dense transformer inference frequently falls below 40% due to memory stalls and cache miss amplification [3].

In dense architectures, parameter count directly determines memory traffic per token. Doubling the number of parameters approximately doubles the volume of data moved from HBM per decoding step, with no compensating increase in arithmetic reuse. This scaling behavior imposes a practical ceiling on cost-effective model growth under fixed power and bandwidth constraints.

Mixture-of-Experts (MoE) architectures address this limitation by decoupling *representational capacity* from *active computation*. Each token is routed through a small subset of experts, typically activating only 5–15% of total parameters in frontier deployments [4, 14]. As a result, memory traffic scales with the number of active experts rather than total model size, enabling models with hundreds of billions of parameters to be served at costs comparable to dense models an order of magnitude smaller.

Early MoE systems relied on auxiliary load-balancing losses to prevent expert collapse, in which routing distributions concentrate on a small subset of experts. While effective in maintaining utilization, these auxiliary objectives interfered with the primary likelihood optimization and introduced training instability at scale [19]. Recent systems eliminate auxiliary losses by dynamically adjusting router bias terms based on observed expert load, stabilizing utilization without penalizing the main training objective [4]. This shift enables stable large-scale training while avoiding token dropping and capacity overflow heuristics.

2.1 Systems Implications of Sparse Routing

MoE architectures fundamentally alter execution dynamics. Expert routing introduces token-dependent communication, as hidden states must be transmitted to the devices hosting selected experts. Consequently, interconnect bandwidth and topology become first-order constraints on inference latency. Without topology-aware expert placement, routing overhead can dominate end-to-end execution time, negating the benefits of sparsity.

This coupling extends across the stack. Runtime schedulers must coordinate expert-parallel execution with KV cache locality, while compiler pipelines must preserve routing semantics through operator fusion and kernel generation. MoE is therefore best understood not as a modeling trick, but as a distributed execution strategy co-designed with runtime, compiler, and hardware.

2.2 Execution Semantics Required for Sparse Architectures

While Mixture-of-Experts is often presented as an architectural modification, its practical viability depends on execution semantics that differ fundamentally from those assumed by dense transformers. In dense models, computation is uniform across tokens and layers, communication patterns are largely static, and execution placement can be determined at compile time. MoE violates all three assumptions.

First, MoE introduces token-wise conditional execution. Routing decisions are made independently for each token at each MoE layer, requiring the runtime to support fine-grained, data-dependent control flow within a single forward pass. This contrasts with dense execution, where all tokens traverse identical computational paths.

Second, sparse routing induces dynamic communication patterns. Tokens must be dispatched to the devices hosting their selected experts and later gathered for combination. These token-dependent all-to-all exchanges vary from step to step and cannot be expressed using the static collectives typical of data-parallel training.

Third, MoE decouples model semantics from execution placement. Which parameters are activated for a given token is no longer fixed, shifting placement decisions from compile time to runtime. As a result, effective execution requires routing-aware scheduling, explicit communication overlap, and careful management of interconnect bandwidth.

Finally, modern MoE systems rely on control mechanisms outside gradient descent to stabilize expert utilization. Dynamic adjustment of router bias terms based on observed load introduces feedback control into the execution stack, further separating architectural intent from optimization objectives.

Taken together, these requirements imply that the efficiency gains attributed to Mixture-of-Experts are not a direct consequence of architectural sparsity alone. They depend on execution capabilities that are absent from standard dense-model training and inference stacks. Without support for token-wise conditional execution, routing-dependent communication, expert-parallel scheduling, and external load-stabilization control, MoE either degenerates into a dense approximation or exhibits instability that erodes its intended memory and cost advantages. In practice, the success of sparse architectures reflects the maturity of the execution stack beneath the model, rather than the expressiveness of the model definition itself.

Table 1: Dense vs. Mixture-of-Experts Architectures Under Bandwidth Constraints

Property	**Dense Transformer**	**MoE Transformer**
Active parameters per token	100%	5–15%
Memory traffic scaling	Linear in total params	Linear in active experts
Compute utilization	Low (memory-bound)	Higher (reduced bandwidth pressure)
Interconnect sensitivity	Low	High
Primary failure mode	Bandwidth saturation	Routing imbalance

3 Inference-Time Reasoning as a Response to Amortized Execution Limits

Single-pass decoding constrains a model's ability to explore alternative hypotheses, revise intermediate assumptions, or recover from errors once they have been committed. In standard autoregressive generation, each token is produced via a locally optimal next-token prediction, with no mechanism for backtracking or global consistency checking. While increasing model size improves average-case performance, it does not fundamentally address this structural limitation: a single forward pass cannot implement search, revision, or hypothesis testing.

This limitation becomes increasingly pronounced on tasks that require multi-step reasoning, long-horizon planning, or internal consistency. Empirically, dense models trained purely for one-pass generation exhibit brittle behavior on mathematical proofs, algorithmic reasoning, and tool-mediated workflows, even when scaled by orders of magnitude [25, 23]. These failure modes motivated a shift away from fully amortized inference toward models that allocate additional computation at inference time.

3.1 Inference-Time Compute as a Learned Search Policy

Recent reasoning-oriented models introduce inference-time computation in the form of internal reasoning traces, branching solution paths, and verifier-guided refinement. Rather than committing to a single trajectory, these models generate and evaluate multiple candidate reasoning paths before producing an external response. In effect, inference becomes a learned search procedure over reasoning trajectories rather than a single deterministic mapping from input to output.

Performance under this paradigm scales with inference compute budget, often approximately logarithmically [20]. Given sufficient test-time compute, moderate-sized models employing inference-time search can outperform substantially larger models constrained to single-pass decoding. This shifts the axis of capability from parameter count to inference policy, reframing intelligence as an interaction between a base model and a runtime-controlled execution strategy.

3.2 Process-Based Reward Models and Credit Assignment

A critical enabler of inference-time reasoning is the use of process-based reward models (PRMs). Unlike outcome-based reward models, which provide supervision only on final answers, PRMs assign credit at the level of intermediate reasoning steps [16]. This fine-grained supervision enables precise identification of where a reasoning trajectory fails, allowing models to discard invalid paths early and focus compute on promising alternatives.

Process supervision also alters alignment dynamics. By rewarding human-approved reasoning strategies rather than merely correct outputs, PRMs discourage shortcut exploitation and spurious correlations. Empirical results indicate that process-supervised models improve both reasoning accuracy and robustness, exhibiting what has been described as a "negative alignment tax"—better alignment coinciding with better task performance [16].

3.3 From Reasoning Models to Agentic Behavior

Inference-time reasoning directly enables agentic behavior. Autonomous agents must decompose tasks, select tools, evaluate partial outcomes, and recover from errors over extended horizons. Each of these capabilities requires the ability to maintain, inspect, and revise internal state across multiple steps.

Without search-like inference, agents exhibit rapid performance collapse due to error accumulation. In contrast, reasoning-oriented models can evaluate intermediate states, backtrack when necessary, and re-plan in response to unexpected outcomes. As a result, reasoning models and agents are best understood not as distinct categories, but as two manifestations of the same architectural shift: the introduction of controlled, runtime-driven search into neural computation.

Table 2: Single-Pass Decoding vs. Inference-Time Reasoning

Property	**Single-Pass LLM**	**Reasoning-Oriented LLM**
Inference policy	Greedy / sampling	Search over trajectories
Intermediate revision	None	Yes
Credit assignment	Final answer only	Step-level (process-based)
Compute allocation	Fixed per token	Adaptive per query
Agent suitability	Low	High
Failure mode	Irreversible errors	Recoverable backtracking

Inference-time reasoning, like sparse architectural approaches, is not realized through model design alone. Its effectiveness depends on execution capabilities that extend beyond standard single-pass decoding pipelines, including dynamic compute allocation, branching and pruning control, verifier-guided execution, and recovery from intermediate failures. Without runtime support for these semantics, reasoning-oriented models either revert to brittle single-path behavior or incur prohibitive inference costs. In practice, the gains attributed to inference-time reasoning reflect the maturity of the execution stack as much as the expressiveness of the reasoning model itself.

4 Attention Optimization as a Response to Memory Hierarchy Constraints

Self-attention has emerged as the dominant performance bottleneck in long-context inference, not because of its arithmetic complexity alone, but because of its interaction with modern memory hierarchies. While attention involves large matrix multiplications that map well to tensor cores, naive implementations materialize intermediate attention matrices in off-chip high-bandwidth memory (HBM). This leads to excessive memory traffic, cache thrashing, and severe underutilization of available compute resources [3].

On contemporary accelerators, the disparity between peak arithmetic throughput and effective memory bandwidth has grown large enough that attention layers routinely operate in a memory-bound regime. As sequence length increases, intermediate tensors quickly exceed the capacity of on-chip caches, forcing repeated HBM accesses. The resulting bandwidth pressure dominates end-to-end latency, particularly in decoder-only models serving long-context or multi-tenant workloads.

4.1 Kernel Fusion and the Limits of Arithmetic Intensity

Modern attention kernels address this imbalance through aggressive operator fusion and careful management of on-chip data reuse. Rather than materializing the full $QK^\top$ matrix in global memory, optimized implementations fuse matrix multiplication, softmax normalization, and value projection into a single execution pipeline, reusing intermediate results while they reside in on-chip memory [3].

This approach increases arithmetic intensity—the ratio of floating-point operations to bytes moved—and allows attention execution to approach hardware utilization limits. However, such fusion requires hardware-aware scheduling: execution order, buffering strategy, and data layout must be tuned to shared memory capacity, register pressure, and tensor core fragment geometry. These requirements exceed what can be inferred automatically from high-level model definitions.

4.2 Softmax as a Serialization Point

Despite optimization of matrix multiplication, softmax normalization remains a critical serialization point in attention execution. Exponentiation and normalization rely on special function units that operate orders of magnitude slower than tensor cores. Without careful scheduling, softmax becomes a throughput limiter that negates gains from fused matrix operations.

Recent kernels mitigate this effect by overlapping softmax computation with independent matrix multiplications, using techniques such as double buffering or warp-group scheduling to hide latency [3]. These optimizations shift softmax from a dominant stall point into an amortized cost, but only when supported by fine-grained control over execution order and memory movement.

4.3 KV Cache Growth and Memory Pressure

During autoregressive decoding, keys and values for all previously generated tokens must be retained in memory. The resulting key–value (KV) cache grows linearly with context length and model width, quickly dominating memory footprint. For large models, a single request with six-figure token context can consume tens to hundreds of gigabytes of KV storage.

As a consequence, serving systems must treat KV cache management as a first-class execution concern. Paged allocation, prefix sharing, selective eviction, and KV quantization are required to avoid fragmentation and out-of-memory failures under realistic workloads [7]. KV cache pressure further constrains batch size and limits the degree of concurrency achievable on a single accelerator.

4.4 Distributed Attention and Communication Overlap

For extreme context lengths, attention computation must be distributed across devices. Approaches such as ring-style attention partition sequences across accelerators and stream key/value data between them, enabling memory capacity to scale with device count [11].

In this regime, attention performance depends as much on interconnect bandwidth and topology as on local kernel efficiency. Communication latency enters the critical path, requiring explicit overlap of inter-device transfers with computation. Attention thus becomes a distributed execution problem, dissolving the boundary between model architecture and hardware placement.

Table 3: Attention Execution Regimes and Dominant Bottlenecks

Regime	**Sequence Length**	**Primary Bottleneck**	**Mitigation**
Short context	< 2K	Kernel launch overhead	Fusion
Medium context	2K–32K	HBM bandwidth	On-chip reuse
Long context	> 32K	KV cache footprint	Paging, quantization
Extreme context	$> 10^5$	Interconnect latency	Distributed execution

Like sparse architectures and inference-time reasoning, attention optimization is not realized through algorithmic changes alone. Its effectiveness depends on execution capabilities that extend beyond standard model runtimes, including fine-grained operator fusion, explicit memory management, and communication-aware scheduling. Without these capabilities, attention becomes the dominant limiter on model scalability, regardless of architectural intent. In practice, the ability to deploy long-context and high-throughput attention reflects the maturity of the execution stack as much as the design of the attention mechanism itself.

5 Compiler and Runtime Design as a Response to Execution Complexity

As model architectures mature, performance differentiation increasingly arises below the level of the neural network itself. Compiler stacks and inference runtimes translate abstract computation graphs into executable programs, making decisions about operator fusion, data layout, numerical precision, and scheduling that irrevocably shape latency, throughput, and cost. In practice, two models with identical weights can exhibit order-of-magnitude performance differences depending on the quality of the compiler and runtime that execute them.

This shift reflects the growing dominance of memory hierarchy and execution efficiency over raw arithmetic capability. Decisions made during graph lowering and kernel generation determine data movement patterns, cache locality, and synchronization overhead. Once lowered, these decisions are difficult to reverse, elevating the compiler and runtime from implementation details to first-class components of the AI system.

5.1 From Computational Graphs to Executable Programs

High-level model definitions express computation in terms of logical operators but abstract away hardware constraints. Intermediate representations (IRs) bridge this gap by encoding computation, layout, and scheduling information in a form amenable to optimization and lowering. Modern AI compiler stacks employ multi-stage lowering pipelines, progressively transforming operator graphs into fused regions and hardware-specific kernels [8, 5].

Each lowering stage presents fundamental tradeoffs. Early fusion can reduce memory traffic and synchronization overhead but may restrict later scheduling flexibility; delayed fusion preserves optionality but risks materializing large intermediate tensors that overwhelm the memory hierarchy. Effective compiler design therefore balances global optimization against local kernel efficiency, informed by the characteristics of the target hardware.

5.2 Kernel Generation and Auto-Tuning

Kernel generators provide explicit control over tiling, memory hierarchy usage, and instruction-level parallelism. By exposing these parameters, generators allow kernels to be tailored to the register file, shared memory capacity, and tensor core geometry of specific accelerators [22].

Optimal kernel configurations depend on model dimensions, batch size, and microarchitectural details. Auto-tuning systems explore this configuration space empirically, benchmarking candidate kernels to select those that maximize throughput or minimize latency. While this introduces compilation overhead, it yields sustained performance gains that cannot be achieved through model-level changes alone, particularly for attention and large matrix operations.

5.3 Runtime Scheduling and Continuous Batching

Inference workloads are inherently dynamic: request arrival times, sequence lengths, and decoding progress vary widely. Static batching leads either to poor utilization or excessive tail latency. As a result, modern inference runtimes implement continuous batching, assembling micro-batches from in-flight requests at each decoding step [7].

This scheduling problem is tightly coupled to memory management, particularly for the key–value cache in autoregressive models. Requests at different decoding positions share compute kernels but impose different memory footprints. Runtime policies must therefore balance throughput, fairness, and memory pressure under multi-tenant workloads with heterogeneous service-level objectives.

5.4 Speculative Decoding and Multi-Model Execution

Speculative decoding further increases execution complexity by introducing multi-model pipelines. A smaller draft model proposes candidate tokens, which are then verified in parallel by a larger target model [9]. Net speedup depends on acceptance rates, synchronization overhead, and verification granularity.

Supporting such pipelines requires runtimes to coordinate multiple model instances, align token streams, and efficiently discard rejected computation. These mechanisms blur the line between inference and scheduling, reinforcing the role of the runtime as an execution policy engine rather than a passive dispatcher.

5.5 IR as a Policy and Portability Boundary

Beyond performance optimization, intermediate representations increasingly encode execution policy. Quantization strategies, mixed-precision constraints, and energy targets are expressed as IR annotations and propagated through the compilation pipeline [26].

This positions the compiler as a mediator between model intent and deployment reality. Architectural choices made at the model level—such as sparsity, long-context attention, or inference-time reasoning—can only be realized when supported by corresponding IR semantics and runtime mechanisms. Performance, portability, and efficiency therefore converge at the compiler and runtime boundary.

Table 4: Sources of Performance Variation Beyond Model Architecture

Layer	Decision	Impact
Compiler IR	Fusion and layout	Memory traffic
Kernel generation	Tiling and precision	Throughput, latency
Runtime scheduling	Batching policy	Utilization, tail latency
Runtime memory	KV cache management	Memory footprint
Inference execution	Speculative decoding	Tokens per second

Taken together, these considerations illustrate that modern AI performance is not determined by model architecture alone. Compiler and runtime capabilities define which execution semantics are feasible, which optimizations are stable, and which architectural ideas can be realized in practice. Without a sufficiently expressive execution stack, advances such as sparse architectures, inference-time reasoning, and long-context attention remain theoretical or prohibitively expensive. In practice, performance differentiation increasingly reflects execution maturity rather than model design.

6 Diffusion Transformers as a Response to Generative Scaling Constraints

Diffusion-based generative models have undergone a fundamental transformation over the past two years. Originally introduced as a technique for image synthesis with strong convolutional inductive biases, diffusion has evolved into a general-purpose framework for modeling complex, high-dimensional distributions. This evolution mirrors a broader pattern observed in language models: the replacement of domain-specific architectures with transformer-based backbones that exhibit predictable scaling behavior with increased compute [17].

Transformer-based diffusion models operate on tokenized latent representations rather than raw inputs, enabling a uniform treatment of images, video, and other structured data. By removing convolutional hierarchies and operating on latent patches, Diffusion Transformers (DiTs) inherit the favorable scaling properties of transformer architectures while retaining the stability advantages of diffusion-based training. Conditioning mechanisms based on adaptive normalization allow temporal, textual, or class information to modulate generation without destabilizing optimization, facilitating multimodal integration at scale.

6.1 Flow Matching and the Limits of Iterative Generation

Traditional denoising diffusion probabilistic models require dozens to hundreds of sampling steps due to highly curved trajectories between noise and data distributions. Each step makes a small corrective update, resulting in high inference cost and limiting practical deployment. Recent work reframes diffusion as a continuous transport problem, learning a velocity field that maps noise to data along approximately straight paths [10].

Flow matching and rectified diffusion concentrate learning capacity on the most informative regions of the trajectory, dramatically reducing the number of required sampling steps. In practice, this yields one to two orders of magnitude reductions in inference cost while preserving perceptual quality [18]. These advances align diffusion inference economics with those of autoregressive decoders and reflect a broader shift toward algorithmic reformulation as a response to execution constraints.

6.2 Distillation and Few-Step Generation

Once trained, diffusion models can be further compressed via distillation. Teacher models operating with many sampling steps supervise student models constrained to operate with only a small number of steps. This process transfers global distributional structure while discarding fine-grained stochasticity unnecessary for perceptual fidelity [21].

Few-step diffusion fundamentally alters the usability of generative models. Generation that previously required seconds of GPU time can now be performed interactively, enabling new applications in design, content creation, and real-time simulation. As in language models, these gains are not the result of additional parameters, but of execution-aware reformulation that reduces the cost of generation.

6.3 Scientific AI: Structure as a Generative Target

In scientific domains, diffusion models are used not to generate images but to sample valid structures from constrained physical manifolds. Protein folding, molecular docking, and crystal structure prediction can all be formulated as conditional diffusion problems over atomic coordinates [1].

This framing provides a natural mechanism for incorporating physical constraints while maintaining expressive generative capacity. Rather than predicting a single deterministic output, diffusion-based scientific models learn distributions over feasible structures, capturing uncertainty and alternative conformations. This approach has proven effective for protein–ligand interactions, antibody–antigen binding, and materials discovery [12].

The convergence of diffusion techniques across generative and scientific AI reflects a shared modeling substrate. Architectural advances developed for image and video generation—such as transformer backbones, flow matching, and distillation—transfer directly to scientific applications. As with other architectural advances discussed in this paper, these gains are only realized when supported by execution capabilities that make iterative generation, conditioning, and inference-time optimization economically viable.

Table 5: Diffusion Models Across Generative and Scientific Domains

Domain	**Object Generated**	**Constraint Type**
Image / Video	Latent representations	Perceptual fidelity
Molecular docking	Atomic coordinates	Physical chemistry
Protein folding	3D conformations	Geometric and energetic
Materials science	Crystal lattices	Stability and symmetry

Diffusion-based architectures, like sparse models and inference-time reasoning systems, do not derive their practical impact from model design alone. Their effectiveness depends on execution support for iterative generation, conditioning, and aggressive inference-time optimization. Without runtimes capable of managing these execution patterns efficiently, diffusion models remain prohibitively expensive or limited to small-scale use. In practice, the success of diffusion transformers reflects the maturity of the execution stack as much as the expressiveness of the generative model itself.

7 Accelerator and Interconnect Design as a Response to Energy Constraints

The economics of large-scale inference are increasingly dominated by data movement rather than arithmetic computation. On modern accelerators, moving data from high-bandwidth memory (HBM) to on-chip registers consumes one to two orders of magnitude more energy than performing a floating-point operation on that data. As a result, inference latency, throughput, and total cost of ownership are governed primarily by memory access patterns and communication overhead, not by peak floating-point throughput [6, 15].

This imbalance has driven a bifurcation in accelerator design. One path emphasizes continued scaling of HBM capacity and bandwidth, allowing larger models and longer contexts to reside on a single device. The other pursues SRAM-first architectures that minimize off-chip access by placing weights and intermediate activations entirely in on-chip memory. Both approaches address the same underlying constraint—the energy and latency cost of data movement—but make different tradeoffs in capacity, flexibility, and scalability.

7.1 HBM Scaling and Its Limits

HBM-centric accelerators increase memory bandwidth and capacity to accommodate growing model sizes and KV cache footprints. This strategy enables dense and sparse models alike to be served with fewer cross-device transfers, simplifying execution and reducing interconnect dependence.

However, HBM scaling faces fundamental physical and economic limits. Bandwidth improvements lag increases in compute capability, and energy per byte moved remains high. Thermal constraints and packaging complexity further constrain how rapidly HBM capacity can grow. As a result, HBM scaling alone cannot sustain long-term reductions in inference cost without complementary architectural and algorithmic changes.

7.2 SRAM-First and Deterministic Architectures

SRAM-first architectures pursue a fundamentally different strategy: minimizing or eliminating off-chip memory access altogether. By placing model weights and intermediate activations in on-chip SRAM, these designs achieve orders-of-magnitude reductions in access latency and energy consumption. Deterministic execution schedules further reduce control overhead and enable predictable latency.

The primary limitation of SRAM-first designs is capacity. On-chip memory scales far more slowly than HBM, constraining these architectures to smaller or more specialized models. Nonetheless, they demonstrate an important principle: for workloads that fit within on-chip memory, data locality dominates all other performance considerations.

7.3 Interconnect as a First-Class Constraint

Once models exceed the capacity of a single device, interconnect bandwidth and topology become critical. Sparse routing (as in Mixture-of-Experts), pipeline parallelism, and distributed attention all require frequent communication of activations across devices. Without sufficient interconnect bandwidth, these transfers dominate end-to-end inference time.

Interconnect constraints amplify architectural choices made at higher levels. Expert routing locality directly affects communication volume; poor placement can negate the bandwidth savings of sparsity. Similarly, long-context attention introduces sustained all-to-all communication patterns that stress collective communication primitives [11]. Effective system design therefore requires co-optimization of model partitioning, runtime scheduling, and network topology, with explicit overlap of communication and computation.

7.4 Energy, Power, and Deployment Feasibility

At datacenter scale, power delivery and cooling capacity increasingly constrain deployment velocity. Even when accelerators are available, insufficient power or cooling can delay or prevent deployment. Energy efficiency thus transitions from a performance optimization to a feasibility condition.

This reality feeds back into model and system design. Techniques such as sparsity, quantization, kernel fusion, and inference-time compute allocation reduce energy per token and expand the set of deployable configurations. In practice, the most successful systems are those that optimize energy efficiency across the full stack rather than maximizing raw throughput at the model level.

Table 6: Accelerator Design Strategies Under Energy Constraints

Design Strategy	**Primary Benefit**	**Limitation**
HBM scaling	Larger models per device	Energy and thermal limits
SRAM-first	Minimal data movement	Limited model capacity
Sparse execution	Reduced bandwidth demand	Routing and scheduling complexity
Quantization	Lower energy per operation	Precision management

As with other architectural advances discussed in this paper, the practical impact of modern accelerator designs is not determined by hardware capabilities alone. It depends on the ability of the execution stack to exploit data locality, manage communication, and operate within strict energy budgets. Without runtimes and scheduling mechanisms that explicitly account for these constraints, improvements in accelerator design translate poorly into end-to-end system performance. In practice, energy efficiency has become the binding constraint that shapes which architectural ideas can be realized at scale.

8 Agentic Systems as a Response to Long-Horizon Reliability Constraints

Autonomous agentic systems represent a qualitative shift from single-step inference to long-horizon execution. Unlike traditional language model interfaces, agents must maintain state, execute sequences of actions, and interact with external environments over extended time spans. This shift introduces a fundamental reliability constraint: errors compound over time. If each step in an n-step workflow succeeds with probability p, the probability of end-to-end success is p^n. Even at high per-step reliability, modest horizon lengths yield rapid degradation in overall success rates.

This mathematical reality, rather than raw model capability, defines the dominant failure mode of agentic systems in 2024–2025. Improvements in reasoning and tool use increase per-step accuracy, but do not eliminate compounding error. As a result, agent performance is governed less by isolated task competence than by trajectory stability under uncertainty.

8.1 Systems Sources of Failure

Empirical deployments indicate that many agent failures arise not from reasoning errors, but from interactions with external systems. Tools and environments introduce nondeterminism, partial observability, and side effects that violate the assumptions of pure token-based models. Common failure sources include nondeterministic or stateful tool behavior, stale or inconsistent environment representations, partial or truncated tool outputs, and irreversible side effects across retries.

These failures often manifest as silent divergence rather than explicit errors, making detection and recovery difficult. Robust agent execution therefore requires explicit mechanisms for state synchronization, idempotent execution, and invariant checking across action sequences—capabilities that lie outside the language model itself.

8.2 Security and Adversarial Considerations

Agentic systems significantly expand the attack surface of AI deployments. Unlike passive models, agents can read from and write to external systems, persist memory across sessions, and invoke privileged tools. This enables new adversarial vectors, including indirect prompt injection through retrieved content, persistent memory poisoning, and tool-mediated privilege escalation [2, 13].

Mitigating these risks requires treating agents as distributed systems rather than as isolated models. Capabilities must be sandboxed, privileges scoped, and execution audited. Security policies must be enforced at runtime, not merely encoded in prompts or training data.

8.3 Reliability-Oriented Agent Execution

Successful agent deployments adopt execution patterns that explicitly address long-horizon instability. These include transactional execution with rollback on failure, explicit checkpoints and state validation, bounded tool invocation with rate and scope limits, and human-in-the-loop escalation for high-risk actions.

Importantly, these mechanisms reside outside the language model. Reliability emerges from the interaction between reasoning models, runtime orchestration, and monitoring infrastructure, rather than from model capability alone.

8.4 Evaluation Beyond Accuracy

Traditional benchmarks measure isolated task accuracy and are poorly correlated with agentic performance. Effective evaluation must instead characterize behavior over trajectories. Relevant metrics include success probability as a function of horizon length, frequency of intervention or recovery, robustness to environmental perturbations, and cost-to-completion under realistic workloads [24].

This reframing aligns agent evaluation with systems engineering practice. An agent that achieves high task accuracy but requires frequent human intervention or excessive compute may be unsuitable for production, regardless of benchmark scores.

Table 7: Failure Modes and Mitigations in Agentic Systems

Failure Mode	**Primary Cause**	**Mitigation Strategy**
Error compounding	Long horizons	Checkpointing, replanning
State divergence	Partial observability	State validation
Tool misuse	Privilege exposure	Capability sandboxing
Prompt injection	Untrusted content	Input isolation
Cost escalation	Unbounded retries	Budget-aware execution

As with other architectural responses discussed in this paper, agentic systems do not derive their practical effectiveness from model capability alone. Their success depends on execution support for long-horizon state management, controlled interaction with external systems, and robust failure recovery. Without runtimes that explicitly address these reliability constraints, agentic behavior remains brittle and unsuitable for deployment at scale. In practice, the viability of agentic systems reflects the maturity of the execution stack as much as the sophistication of the underlying reasoning model.

9 A Convergent Systems Law as a Response to Execution Constraints

The developments examined throughout this paper reveal a coherent and recurring pattern. Advances in model capability, once driven primarily by parameter scaling, are now governed by the interaction of architectural choices, inference-time reasoning policies, and execution efficiency under physical constraints. These forces do not operate independently. Instead, they form a tightly coupled system in which progress at any single layer depends on alignment with the others.

Across domains, architectural change increasingly reflects responses to execution limits. Sparse activation schemes such as Mixture-of-Experts emerge directly from memory bandwidth constraints. Attention optimizations address cache pressure and interconnect overhead. Diffusion transformers replace domain-specific inductive biases with architectures that scale predictably under fixed compute and memory budgets. In each case, architectural evolution is best understood as a response to execution constraints rather than as a purely representational advance.

Reasoning capability has undergone a similar shift. Rather than being an amortized property of static model weights, reasoning increasingly manifests as inference-time behavior. Controlled allocation of additional compute enables search, verification, and recovery from intermediate errors. This reframes intelligence as a policy over computation rather than a fixed mapping from inputs to outputs. Crucially, such policies are only viable when supported by execution stacks capable of dynamic scheduling, branching, and failure recovery.

Efficiency binds these layers together. Memory movement, interconnect traffic, and energy consumption dominate inference economics, constraining which architectural and reasoning strategies are deployable in practice. Compiler decisions determine data locality; runtime schedulers govern batching, caching, and speculation; accelerator and interconnect designs define the feasible operating envelope. Efficiency is therefore not an optimization objective, but a unifying constraint that propagates upward through the entire system.

These observations motivate a convergent systems law for modern AI:

$$\text{Capability} = f(\text{architecture}, \text{reasoning policy}, \text{runtime}, \text{compiler}, \text{hardware}, \text{energy})$$

No term in this function can be maximized in isolation. Improvements in model architecture require corresponding advances in runtime execution. More powerful reasoning policies demand adaptive scheduling and verification. Hardware gains are only realized when compilers and kernels exploit locality and parallelism. Capability emerges from integration, not from any single component.

This convergence dissolves the traditional boundary between model and system. Language models, vision models, scientific models, runtimes, and accelerators increasingly form a single co-designed artifact. Progress in AI should therefore be understood less as a sequence of model releases and more as the maturation of an integrated discipline of AI systems engineering.

The central lesson of the current phase of AI development is not merely that systems have become larger or more capable, but that they have become fundamentally systemic. Without execution stacks capable of supporting sparse computation, dynamic inference, long-context memory, and long-horizon reliability, architectural advances remain theoretical or economically infeasible. In practice, the pace and direction of AI progress are now set by the maturity of the execution substrate beneath the model, rather than by model design alone.

Acknowledgments

The author thanks the broader AI research community for open technical discourse and public reporting that enabled this synthesis.

Author's Note

The perspective developed in this paper reflects hands-on experience designing, evaluating, and deploying large-scale AI systems across model, runtime, and infrastructure layers. The intent is not to propose new architectures, but to clarify why many widely reported advances depend on execution capabilities that are often implicit or underexamined. This document is offered as a technical lens for practitioners grappling with the practical limits of modern AI systems.

References

[1] Josh Abramson et al. Accurate structure prediction of biomolecular interactions with alphafold 3. *Nature*, 2024.

[2] Anthropic. Developing and evaluating a computer-using agent. *Technical Report*, 2024.

[3] Tri Dao, Daniel Fu, Stefano Ermon, and Atri Rudra. Flashattention-3: Fast and accurate attention with asynchronous computation. *arXiv preprint arXiv:2407.08608*, 2024.

[4] DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint*, 2025. arXiv:2501.xxxxx.

[5] Google. Xla: Optimizing compiler for machine learning. *Technical Report*, 2017.

[6] Mark Horowitz. Computing's energy problem (and what we can do about it). *IEEE Solid-State Circuits Magazine*, 2014.

[7] Woosuk Kwon et al. Efficient memory management for large language model serving with pagedattention. *arXiv preprint arXiv:2309.06180*, 2023.

[8] Chris Lattner et al. Mlir: A compiler infrastructure for the end of moore's law. *Proceedings of the IEEE*, 2021.

[9] Yaniv Leviathan et al. Fast inference from transformers via speculative decoding. *Proceedings of ICML*, 2023.

[10] Yaron Lipman et al. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2023.

[11] Hao Liu, Sam Ainsworth, et al. Ring attention: Near-infinite context transformers. *arXiv preprint arXiv:2310.01889*, 2023.

[12] Anubhav Merchant et al. Graph networks for materials exploration. *Nature*, 2023.

[13] Microsoft AI Red Team. Failure modes and mitigations for agentic ai systems. *Technical Report*, 2024.

[14] Mistral AI. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[15] NVIDIA Corporation. Nvidia hopper architecture in-depth. *Technical White Paper*, 2023.

[16] OpenAI. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

[17] William Peebles and Saining Xie. Scalable diffusion models with transformers. *arXiv preprint arXiv:2212.09748*, 2022.

[18] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

[19] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, et al. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[20] Charlie Snell et al. Scaling llm test-time compute optimally can be more effective than scaling model size. *arXiv preprint arXiv:2404.04666*, 2024.

[21] Yang Song et al. Consistency models. *arXiv preprint arXiv:2303.01469*, 2023.

[22] Philippe Tillet et al. Triton: An intermediate language and compiler for tiled neural network computations. *Proceedings of MLSys*, 2019.

[23] Karthik Valmeekam et al. Large language models still cannot plan. *arXiv preprint arXiv:2305.16809*, 2023.

[24] Karthik Valmeekam et al. Planbench: Evaluating planning and replanning in llm agents. *arXiv preprint arXiv:2402.02343*, 2024.

[25] Jason Wei, Xuezhi Wang, Dale Schuurmans, et al. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[26] Tianqi Zhang et al. Quantization-aware compilation for deep neural networks. *arXiv preprint arXiv:2004.09602*, 2020.