SubQ-1.1-Small Technical Report

Saul Ramirez* Subquadratic AI
Alex Whedon* Subquadratic AI
Ashmal Vayani Subquadratic AI
Phong Vo Subquadratic AI

Abstract

Many high-value AI workloads require reasoning over complete artifacts, including entire code repositories, document collections, and knowledge bases, rather than isolated excerpts. Most systems do not reason over these artifacts directly. They rely on retrieval pipelines, chunking strategies, and agentic orchestration that decompose information into fragments and reconstruct it at inference time. These approaches are often effective, but they exist in large part because dense attention scales quadratically with context length, making direct reasoning over large artifacts increasingly expensive as context windows grow. Much of the modern AI stack is therefore designed to manage context scarcity rather than reason over complete artifacts directly.

We introduce SubQ-1.1-Small, a long-context language model built on Subquadratic Sparse Attention (SSA), a content-dependent sparse attention mechanism with linear compute and memory cost. Relative to dense attention, SSA reduces attention FLOPs by 64.5× at a 1M-token context window. This reduction makes long-context work cheaper not only at inference time but during development: it made million-token-scale pretraining and evaluation practical enough to run more than one hundred long-context experiments and search for a training recipe rather than guess at one.

That experimental regime produced the central result of this work: long-context retrieval generalized far beyond the training window. Trained primarily at 1M tokens, with additional training at 2M, SubQ-1.1-Small achieves 99.12% on the 13-task RULER benchmark, maintains 100% single-needle retrieval through 2M tokens, and sustains 98% retrieval accuracy at 12M tokens, roughly an order of magnitude beyond its primary training window, while attending to only 0.13% of token pairs, nearly a 1,000x reduction. Beyond retrieval, SubQ-1.1-Small approaches frontier-level performance on AutomationBench Finance, a long-horizon agentic benchmark, providing early evidence that the same long-context training regime transfers to long-horizon reasoning. Together, these results suggest that efficient attention matters not only because it lowers inference cost, but because it makes the long-context training experiments needed for whole-artifact reasoning practical.

*Equal contribution.

1 Introduction

1.1 Why Long Context Matters

Large language models have moved rapidly from research systems into production. They write and review code, analyze documents, answer questions over technical material, and increasingly act as agents that perform multi-step work. Where they are most reliable today is on tasks that fit comfortably within a model's context window. The problems that remain most challenging often have a different structure: they require reasoning over a complete artifact rather than an excerpt of one.

A legal agreement may define a term on page 2, qualify it on page 12, carve out an exception on page 46, and amend it again in a schedule. A function may be defined in one file, called from forty others, tested indirectly, and constrained by invariants encoded in the architecture rather than in comments. A research question may require reconciling dozens of papers whose terminology overlaps but whose claims diverge; a financial review may require connecting filings, earnings reports, contracts, and internal records. In these cases, the task is not merely to retrieve a relevant passage. It is to reason across relationships distributed throughout a large artifact.

We refer to these throughout this report as *whole-artifact reasoning tasks*: tasks whose structure requires reasoning across the complete artifact rather than over isolated fragments. As models are increasingly applied to software repositories, document collections, knowledge bases, and other large corpora, the ability to perform whole-artifact reasoning becomes increasingly important.

1.2 Building Around Context Scarcity

Despite the growing importance of whole-artifact reasoning, most AI systems do not reason over complete artifacts directly. Instead, they rely on retrieval pipelines, chunking strategies, summaries, and agentic workflows that partition information into smaller contexts and reconstruct relevant fragments at inference time. Their prevalence reflects a fundamental computational constraint: direct reasoning over large artifacts remains expensive because dense attention scales quadratically with context length.

Viewed through this lens, much of the modern AI stack can be understood as scaffolding built around context scarcity. Information is partitioned, retrieved, summarized, routed, and reconstructed before reasoning occurs. The resulting systems frequently combine learned models with substantial amounts of externally specified information routing and task decomposition.

This pattern bears resemblance to Sutton's Bitter Lesson [35]. Throughout the history of AI, researchers have repeatedly introduced increasingly sophisticated mechanisms to compensate for the limitations of contemporary models, only to see those mechanisms displaced as larger-scale learning and computation became practical. Hand-engineered features gave way to learned representations. Task-specific pipelines gave way to end-to-end training. In each case, much of the apparent complexity reflected the limitations of the underlying learning system rather than the structure of the task itself.

Modern retrieval and orchestration systems may represent a similar phenomenon. Attention is already a learned mechanism for information routing and retrieval. Yet because applying attention over complete artifacts remains prohibitively expensive, modern AI systems frequently rely on additional scaffolding to determine what information a model sees and when it sees it. Their widespread adoption may also reflect the constraints of the current attention regime rather than fundamental requirements of the underlying tasks.

Efficient long-context architectures provide an opportunity to test this possibility. As reasoning over complete artifacts becomes computationally practical, some forms of scaffolding may prove essential, while others may gradually disappear as more of the information-selection problem is absorbed by learned computation.

1.3 SubQ-1.1-Small

We introduce SubQ-1.1-Small, a long-context language model built on Subquadratic Sparse Attention (SSA), a content-dependent sparse attention mechanism that scales linearly in both compute and memory with sequence length. We trained SubQ-1.1-Small at context lengths up to two million tokens and observed strong retrieval performance well beyond the training window, extending to exploratory evaluations at up to twelve million tokens. The two-million-token training limit reflects the scope of the training program rather than a limitation of SSA itself, which has previously been

used to train on sequences up to twelve million tokens. Together, these results demonstrate that content-dependent sparse attention can support strong long-context retrieval and substantial length generalization at multi-million-token scales.

SSA also makes long-context training substantially cheaper. This matters not only for deployment, but also for research. Figure 1 illustrates this relationship. Dense attention makes long-context training expensive, limiting the scale and number of experiments that can be performed. By reducing those costs, SSA expands what researchers can afford to explore.

Using this efficiency, we conduct a broad investigation of long-context training and evaluation at million-token and multi-million-token scales. These experiments examine the relationship between training context length, continued pretraining, retrieval performance, and context-length generalization. Across this study, we find that strong long-context behavior can emerge from relatively small models when trained efficiently at scale.

Together, these results demonstrate that content-dependent sparse attention can support practical long-context retrieval and substantial context-length generalization at multi-million-token scales. They also provide new empirical evidence about how long-context capability develops during training when experimentation at these scales becomes practical.

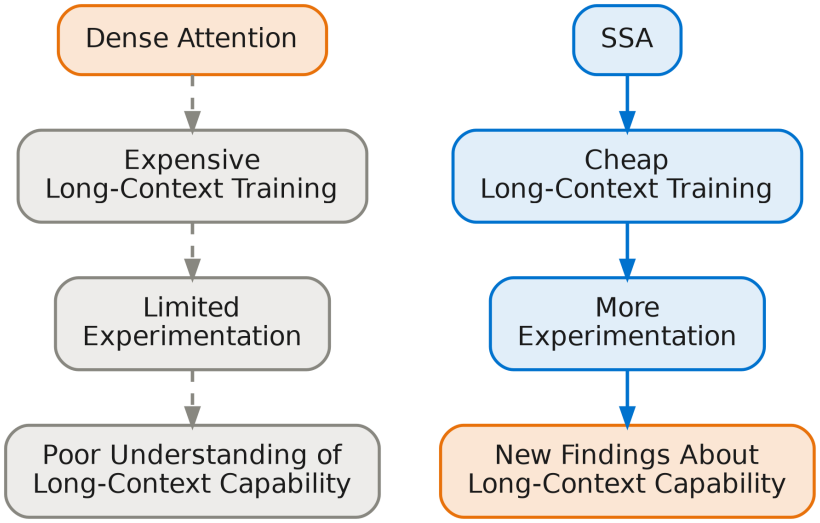

Figure 1: Relationship between the attention regime and experimentation. Dense attention makes long-context training expensive, limiting the number and scale of experiments that can be performed. SSA reduces those costs, enabling broader experimentation and the findings reported in this work.

2 Background

The quadratic cost of attention has motivated years of research into more efficient alternatives. Proposed solutions span sparse attention, linear attention, state-space models, and hybrid architectures, each reducing the cost of sequence processing through a different mechanism. Despite substantial progress, dense attention remains the dominant architecture in frontier models, while most efficient alternatives that appear in production retain some dense-attention components to preserve capabilities lost by the pure approaches.

The remainder of this section examines these approaches and the specific tradeoffs that have prevented efficient sequence models from simultaneously achieving efficient scaling, content-dependent retrieval, and strong long-context capability. Figure 2 maps the families that follow and shows where SSA sits among them.

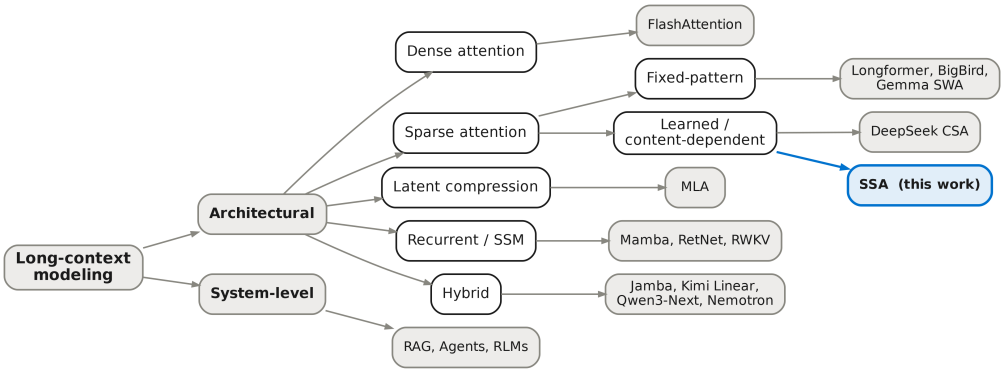

Figure 2: A taxonomy of approaches to long-context modeling. Architectural approaches change how the model itself processes a sequence; system-level approaches leave the model fixed and orchestrate around its context limit. SSA is a learned, content-dependent sparse-attention method, placed alongside the closest published comparison point.

2.1 Flash Attention

Flash Attention [7; 5] is the most important piece of work on attention efficiency of the last several years, and it is not a competitor to anything in this paper. It is the current standard implementation of dense attention, and we use it as our dense-attention baseline throughout.

The contribution of Flash Attention is at the level of the GPU kernel, not the mathematical formulation. Standard attention, computed naively, materializes the full $n \times n$ attention matrix in GPU memory, a cost that becomes prohibitive well before the compute cost does, because memory bandwidth between the GPU's high-bandwidth memory and its on-chip SRAM is the binding constraint on throughput. Flash Attention reorders the computation so that the attention matrix is never fully materialized: it tiles the computation into blocks that fit in SRAM, maintains running softmax statistics as it goes, and writes only the final output back to high-bandwidth memory. The peak memory required for attention drops from $O(n^2)$ to $O(n)$, and throughput rises substantially. The outputs are numerically identical to the naive formulation.

What Flash Attention does not change is the asymptotic compute cost of attention. The number of floating-point operations remains $O(n^2)$, even though those operations are executed with far greater memory efficiency. Doubling the sequence length still quadruples the amount of attention computation required, and the quadratic scaling that dominates million-token workloads remains unchanged (Figure 3). Flash Attention solved the memory problem that made dense attention impractical at long context lengths; it did not solve the compute problem that ultimately limits how far dense attention can scale. In doing so it made longer-context experimentation practical for the first time, but it did not remove the scaling barrier that ultimately bounds how far that experimentation can extend.

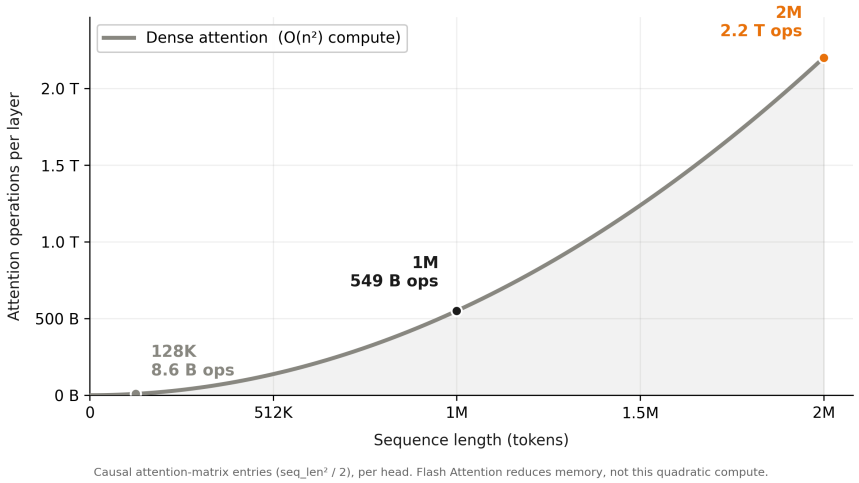

Figure 3: Dense attention operations per layer as a function of sequence length, shown across SubQ-1.1-Small's operating range. At 128K tokens, a length many systems still treat as long context, attention requires only 8.6 billion operations per layer; at 1M, 549 billion; at 2M, 2.2 trillion, a fourfold increase for a doubling of context. Flash Attention preserves this quadratic growth in compute even as it reduces the memory footprint.

2.2 Sparse Attention

Sparse attention is therefore the most direct architectural response to attention's quadratic cost. If only a small fraction of token interactions meaningfully contribute to the output [4; 3], a model should be able to compute only those interactions and avoid the rest. In practice, the challenge has never been sparsity itself. The challenge has been deciding which interactions to keep.

Most sparse-attention systems make that decision through a fixed pattern. Sliding-window attention [2], strided attention [3], and related approaches restrict attention according to position rather than content. These methods achieve favorable scaling properties and remain useful in practice, sliding-window attention ships in production models such as Gemma [12], but they sacrifice the content-dependent routing behavior that makes dense attention effective. A token can only attend where the pattern allows it to attend, regardless of whether the relevant information is actually there.

Other approaches attempt to preserve capability through compression rather than masking. Methods such as Multi-head Latent Attention (MLA) [9] compress keys and values into a learned latent representation, substantially reducing the KV-cache memory and bandwidth that bottleneck autoregressive inference, and they have seen adoption in frontier systems. MLA is best understood as solving a different problem than the one SSA targets: it addresses inference-time memory efficiency rather than the cost of the prefill attention computation. The quadratic work performed when ingesting a long context is unchanged, so MLA on its own does not alter long-context training economics.

The recurring challenge across efficient-attention research is not obtaining sparsity. It is preserving content-dependent retrieval while remaining computationally efficient. Existing approaches typically succeed on one side of this tradeoff but not both: they either achieve favorable scaling by restricting where information can flow, or preserve flexibility while retaining substantial quadratic computation. Ultimately, fixed-pattern sparse attention mechanisms have failed on long-context retrieval benchmarks like RULER. The central problem remains unresolved: how to obtain efficient scaling without sacrificing the retrieval behavior that long-context reasoning depends on.

2.3 Learned Sparse Attention

Learned sparse attention has emerged as a promising alternative to dense attention for scaling transformer models to increasingly long contexts. Rather than requiring every query token to attend across the entire sequence, learned sparse methods attempt to identify and access only the most relevant portions of the context, reducing computational cost while preserving access to important information. Unlike traditional sparse attention approaches that rely on fixed patterns or handcrafted

routing rules, learned sparse methods determine where attention should be allocated directly from data, allowing sparsity patterns to adapt to the content of the sequence itself.

A prominent example is Native Sparse Attention (NSA), also known as DeepSeek Sparse Attention, introduced by DeepSeek-AI [10]. NSA replaces fixed sparsity patterns with a learned routing mechanism known as the Lightning Indexer that predicts which portions of the sequence should receive attention. This represented an important shift in the design of long-context architectures. Rather than manually specifying how information should flow through the model, the routing decisions themselves are learned, allowing attention allocation to adapt dynamically to the content of the sequence. However, the lightning indexer is a distilled transformer still using full attention, and it doesn't avoid the quadratic compute complexity problem, ultimately overtaking the teacher model's sparse attention compute around 52,000 tokens.

DeepSeek V4 [11] extends this paradigm through a hybrid long-context architecture that combines learned sparse retrieval with compressed memory representations. Its Compressed Sparse Attention (CSA) mechanism applies learned retrieval in the same way as NSA, but over compressed representations of the sequence. By performing retrieval over a compressed memory rather than the original token stream, CSA reduces the amount of information that must be searched while preserving content-dependent access to long-range context. However, it still has quadratic compute complexity in the Lightning Indexer, which dominates compute as inputs grow large.

The architecture also introduces Heavily Compressed Attention (HCA), which is conceptually distinct from learned sparse retrieval. Rather than selecting a sparse subset of memory locations, HCA compresses distant portions of the sequence aggressively and performs dense attention directly over the resulting compressed memory. CSA therefore achieves efficiency through learned selection, whereas HCA achieves efficiency through compression. Together, these mechanisms provide access to information at multiple contextual scales while avoiding dense attention over the full sequence. Because HCA applies dense attention over a compressed sequence, its benefit is scalar, with compute scaling quadratically with regards to the sequence length.

V4 is particularly relevant to this report because it demonstrates that learned sparse attention can be deployed successfully at frontier scale. Its results provide strong evidence that content-dependent routing can preserve model quality while substantially reducing the practical cost of long-context processing. At the same time, V4 illustrates an important limitation of current approaches. CSA reduces computation through learned retrieval, while HCA reduces computation through aggressive compression, yet both continue to scale quadratically with context length. As a result, long-context training and experimentation remain expensive at million-token scales. Learned sparse attention therefore represents a significant step toward practical long-context models, but not the end of the scaling challenge. This observation motivates the analysis that follows.

2.4 Linear Recurrence and State Space Models

A second family of approaches abandons attention's all-pairs structure entirely. Rather than comparing every query against every key, these methods process sequences recurrently: each position updates a compressed internal state that carries forward information from everything seen so far, and the output at each position is produced from the current state alone. The complexity is $O(n)$ by construction.

The line begins with linear attention [18], which replaces the softmax with a factorizable kernel that admits a recurrent form. The current frontier is the state space model (SSM) line, Mamba [13] and its successors [6; 20], alongside gated variants such as GDN [38], RetNet [34], and RWKV [28]. The Mamba-2 result established that SSMs and linear attention are, under the appropriate parameterization, two views of the same underlying mathematical object. That unification matters here because it means the limitations of the family are not artifacts of any specific architecture; they are consequences of the shared structure.

That structure, a fixed-size state absorbing an unbounded sequence, is a compression, and compression is lossy. SSMs are strong on tasks where a compressed representation suffices (summarization, classification, locally-structured generation) and weak on tasks that require exact reproduction or precise retrieval of content introduced earlier in the sequence [17; 1]. A state that compresses the past cannot also preserve it verbatim, and long-context retrieval requires the latter. This is the capability gap that has kept pure recurrent methods out of a displacing role at frontier scale, and it is the reason the efficient alternatives that do appear in production are almost always hybrids, recurrent layers

paired with dense attention layers retained specifically to preserve the retrieval capability the recurrent layers cannot provide.

2.5 Hybrid Models

The hybrid response to the capability gap is to combine both mechanisms: a model built primarily from efficient recurrent layers, with a small number of dense attention layers inserted at regular intervals to preserve the retrieval capability the recurrent layers cannot provide. Several recent models ship this way, Jamba [22], Kimi Linear [19], Qwen3 Next and Qwen3.5 [30], and the Nemotron v3 Nano and Super models [27], and on the benchmarks that matter at release time, hybrids perform well at a meaningful fraction of the compute cost of comparable dense models.

The deeper problem with the hybrid approach is the one that the benchmarks at release-time context lengths do not surface: *constant-factor improvements do not change asymptotic scaling behavior.* A hybrid that is three times cheaper than a dense model at 32K tokens is a meaningful improvement, but if the dense attention layers retained in the hybrid still scale quadratically, the ratio between hybrid and dense stays constant as sequences grow. The hybrid has moved the line, but not changed its shape. At hundreds of thousands and millions of tokens, the quadratic component of the hybrid dominates the cost profile, and the efficiency that looked substantial at release time compresses toward the efficiency of the dense baseline. A second issue compounds this: the attention layers in a hybrid are load-bearing in a way the efficient layers are not, and the ratio cannot be pushed down arbitrarily without losing the retrieval capability the hybrid was designed to preserve.

MiniMax provides an informative case study in this tradeoff. MiniMax-M1 adopted a hybrid architecture combining Lightning Attention with full attention, enabling efficient operation at long context [25]. Their subsequent frontier model, M2, returned to full attention across all layers [26]: during development, hybrid variants matched full attention on standard benchmarks but showed clear deficits on higher-order multi-hop reasoning at larger scale, and the supporting infrastructure for efficient attention, low-precision state storage, prefix caching, speculative decoding, remained less mature than for full attention [24]. This does not show that hybrids fail; it shows that reducing asymptotic cost is not sufficient on its own, because retrieval quality, reasoning at scale, and system maturity must be preserved at the same time.

2.6 System-Level Workarounds

Where architectural approaches fall short, system-level approaches compensate. Retrieval-augmented generation (RAG), agentic workflows, and recursive language models all extend the effective context of a model by decomposing a large problem into smaller pieces that fit within a limited context window. These systems are often effective and, for many workloads, remain the appropriate solution. Their common characteristic is that long-context behavior is achieved through orchestration rather than through direct access to the complete artifact.

RAG [21] retrieves a small subset of a larger corpus and places it into context. This approach scales well to corpora far larger than any practical context window and remains indispensable when information changes frequently. Its limitation is that reasoning is performed over retrieved fragments rather than over the complete artifact, making overall performance dependent on the quality of retrieval.

Agentic frameworks decompose a task into multiple context-sized steps connected by orchestration logic. This enables models to operate over larger problems than would fit into a single context window, but introduces dependencies between steps and requires information to be repeatedly summarized, transferred, or reconstructed throughout execution. The human curation in these systems often leads to a decrease in system generalization.

Recursive Language Models (RLMs) [40] push this idea further by allowing models to recursively inspect and process portions of an input that are too large to fit in context directly. They represent one of the most capable forms of long-context scaffolding developed to date. However, as with other decomposition-based approaches, no individual model invocation has direct access to the complete input. The repeated steps can lead to missing context, high latency, and high cost.

These systems solve an important problem and will remain necessary whenever corpora exceed practical context limits. They do not remove the underlying constraint; they work around it. The distinction matters because the central question of this report is whether the constraint itself can be relaxed.

2.7 The Open Problem

The recurring pattern is not that previous approaches failed. It is that each approach relaxed one constraint while retaining another (Figure 4). Some achieve efficient scaling without content-dependent retrieval. Others preserve retrieval while retaining substantial quadratic computation. System-level approaches move retrieval outside the model entirely. The open problem is to combine subquadratic scaling end-to-end, including the selection or indexing stage, not only the attention read, with content-dependent retrieval, arbitrary-position access, and practical ultra-long-context training within a single system.

We are not aware of a widely deployed architecture that provides all four simultaneously. SSA is our attempt at that combination, and SubQ-1.1-Small is a model it produced.

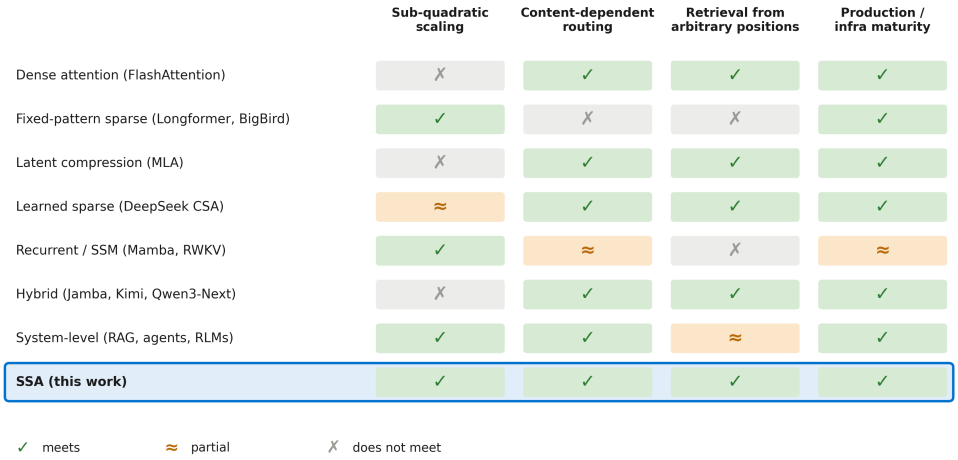

Figure 4: The long-context tradeoff. Each row is a family of approaches; each column is one of the four properties a complete solution must provide. Every prior family leaves at least one column unmet or only partially met, the recurring pattern that motivates this work. SSA targets all four at once.

3 Methods

Rather than training a new model from scratch, we converted an existing open-weight frontier model by replacing its dense attention with Subquadratic Sparse Attention (SSA). Long-context capability was then developed through staged context extension, large-scale continued pretraining, and targeted post-training. This section describes the requirements that motivated SSA, the training process used to develop long-context capability, and the infrastructure required for multi-million-token-scale experimentation.

3.1 Subquadratic Sparse Attention

SSA was designed to satisfy three requirements simultaneously, the combination that, as the Background argued, existing approaches had not achieved in a practical long-context system. The mechanism by which SSA meets these requirements is outside the scope of this report; here we focus on the requirements themselves, because they motivated both the architecture and the experimental campaign that followed.

Requirement 1: dense-attention-level retrieval and reasoning quality. The first requirement was to preserve the retrieval and reasoning behavior that makes dense attention [36] effective. Many efficient sequence models achieve favorable scaling but degrade retrieval and reasoning as context grows, precisely the regime we cared about. The design target was therefore not efficiency for its own sake but dense-attention-like retrieval from arbitrary positions, which requires that routing be **content-dependent**, determined by the tokens themselves rather than by a fixed positional pattern [2; 39].

Requirement 2: subquadratic scaling. Dense attention scales quadratically with sequence length, the cost that makes multi-million-token training and evaluation expensive. The second requirement was a mechanism whose cost grows more slowly while still meeting Requirement 1. SSA is **sparse**, each query attends to a small, selected subset of positions, and its selection, retrieval, and attention steps are each linear in sequence length, so the mechanism is linear end-to-end rather than only within the attention operation.

Requirement 3: full-context training and autoregressive generation. The model needed to optimize over the entire available context during training while retaining standard sequential autoregressive decoding at inference time. State-space and recurrent architectures such as Mamba [13] and RetNet [34] achieve favorable scaling through compressed state representations, but training decisions are mediated through that state rather than direct access to arbitrary positions in the context. Conversely, non-autoregressive approaches can optimize globally but depart from the generation paradigm that underlies contemporary language-model reasoning and tool use. The third requirement was therefore to preserve both: full-context optimization during training and efficient token-by-token generation during inference.

The remainder of this section describes what those capabilities made possible.

3.2 Experimental Design

SubQ-1.1-Small did not emerge from a single training run. Over the course of development, we conducted more than one hundred long-context experiments across six major generations of the model, exploring context-extension schedules, continued-pretraining mixtures, retrieval-focused post-training objectives, capability-preservation techniques, and evaluation methodology. Alongside the formal benchmarks reported later, we maintained a fixed set of deployment-oriented evaluations spanning repository-scale code reasoning, multi-document synthesis, contract analysis, and engineering-documentation retrieval so that progress was measured against realistic whole-artifact tasks rather than synthetic probes alone.

This scale of experimentation was possible because SSA makes multi-million-token training practical. Under dense attention, each long-context experiment incurs quadratically increasing compute costs, limiting the number of hypotheses that can be tested at the context lengths where long-context behavior emerges. SSA reduced that constraint enough that iteration at multi-million-token context remained under a minute per step, allowing us to compare variants across pretraining mixtures, context-extension schedules, recovery techniques, and post-training compositions at scale. The resulting increase in experimental throughput enabled the large-scale empirical study reported in this work.

3.3 Context Extension and Continued Pretraining

Long-context continued pretraining (CPT) was the primary mechanism through which long-context capability was developed. Training emphasized naturally long-form data, including books, long documents, and repository-scale code, to expose the model to genuine long-range dependencies.

Supporting this required progressively extending the model's context window. The donor model supported a 262K-token context. We used YaRN [29] to rescale the positional representation as context length increased and extended the model in stages: 262K $\rightarrow$ 512K $\rightarrow$ 1M $\rightarrow$ 2M (Figure 5). Long-context CPT was performed between extension stages rather than extending directly to the final target length.

The training mixture combined naturally long sequences with shorter documents packed to the target context length using document separators. Similar to UltraLong [37] and DeepSeek-V3 [8], we did not mask cross-document attention boundaries during packing. The model therefore attended over the entire packed sequence, including separator tokens, rather than treating documents as isolated segments.

Most CPT tokens were trained at a 1M-token context. To study how long-context capability develops, we varied both CPT volume and context length across model generations while evaluating retrieval at multiple context lengths.

Figure 5: Staged context-window extension with continued pretraining. The donor model began with a 262K-token context window. YaRN positional scaling was re-applied at each stage (262K, 512K, 1M, and 2M), with long-context continued pretraining performed between extensions. Most CPT tokens were trained at the 1M stage, followed by a final extension and training stage at 2M tokens.

3.4 Post-Training and Capability Balance

Long-context continued pretraining was primarily responsible for developing long-context capability. Post-training served a different role: shaping how that capability was expressed while preserving and enhancing reasoning, coding, and instruction-following abilities.

A central challenge was capability balance. Improvements in retrieval did not reliably transfer to other capabilities, and training choices that strengthened long-context behavior could shift the balance between instruction following, reasoning, retrieval, and knowledge-intensive performance. As a result, post-training was designed to optimize multiple capabilities simultaneously rather than retrieval in isolation.

To reduce the influence of sequence length on optimization, we explored sample-level loss aggregation, which averages cross-entropy at the example level rather than across all tokens. This reduces the extent to which a small number of extremely long examples dominate gradient updates during training.

The post-training corpus combined synthetic retrieval tasks, long-context reasoning data, coding-oriented examples, educational materials, and general instruction-following data. Training was staged and iterative: targeted phases introduced or strengthened specific capabilities, followed by recovery phases intended to preserve performance across the broader capability suite. Mixture composition, loss formulation, and evaluation cadence were tuned jointly throughout development rather than treated as independent design choices.

3.5 Experimental Infrastructure

Training on multi-million-token sequences is a memory problem as much as a compute problem. Linear-cost attention removes the quadratic attention computation, but activations, optimizer state, and the raw length of each example must still fit within the available memory budget. A typical long-context run used a batch of roughly one-to-two million tokens per node, and at the primary training lengths used throughout development, iteration remained under a minute per step. This made large-scale experimentation at multi-million-token context practical rather than exceptional.

The memory budget was managed through incremental escalation, with each context-length stage using only the combination of parallelism and offloading techniques required for that stage (Figure 6). Rather than adopting the most expensive distributed configuration from the outset, training progressed up a memory-scaling ladder as context length increased: from a single node, to intra-node sequence parallelism, to CPU offloading, and eventually to multi-node execution for the longest contexts. This allowed the majority of development to run with the simplest configuration capable of supporting the target context length while reserving more expensive strategies for later stages. The foundation was hybrid-sharded data parallelism [41], which shards parameters and optimizer state within a node while replicating across nodes. Sequence parallelism [15] shards each long example across the GPUs of a node, while ZeRO-style partitioning with CPU offload [31] moves optimizer state and activations off-device when necessary.

As the context length increased above 2M tokens, additional layers of the stack were engaged, including multi-node sequence parallelism, nested CPU offloading, and Ring Attention [23] for distributing individual examples across nodes. These techniques are individually well established, but none operated efficiently with SSA out of the box. Each required adaptation to accommodate SSA's content-dependent selection and retrieval operations, which introduce memory-access patterns and synchronization requirements absent from standard dense attention. The resulting system enabled routine multi-million-token training while preserving the rapid iteration rates that made the broader experimental study possible.

Extending training above a 2M-token context used the same infrastructure at correspondingly higher memory cost. At extreme evaluation lengths beyond the training target, approximately 8M tokens and above, BF16 underflow and related numerical-stability issues became practical constraints.

Figure 6: Memory-scaling ladder used during development. As context length increased, training progressively escalated from a single-node configuration to sequence parallelism, CPU offloading, multi-node sequence parallelism, nested offloading, and finally Ring Attention. Most experiments were run on the lowest rung capable of supporting the target context length, minimizing system complexity while preserving rapid iteration.

4 Results

We evaluate SubQ-1.1-Small across six dimensions: long-context retrieval, context-length generalization, knowledge, coding, long-horizon agentic tasks, and efficiency. The central result is that retrieval behavior developed primarily at a one-million-token context generalizes substantially beyond the training window, extending to twelve million tokens in held-out evaluation. We then evaluate whether this capability transfers to realistic multi-step tasks, whether it preserves the broader reasoning and coding abilities of the donor model, and whether SSA is efficient enough to make experimentation at these scales practical.

4.1 Long-Context Retrieval

RULER [14] is a 13-task long-context evaluation suite developed by NVIDIA to measure retrieval capability beyond single-fact lookup. Its tasks span four axes: single-key retrieval; multi-key retrieval (locating and returning multiple independent facts from the context); common-word and frequent-word extraction (aggregating distributional statistics over the full context rather than retrieving a single passage); and multi-hop variable tracing (where the answer requires chaining evidence across positions that are individually insufficient). This breadth is why we treat RULER as our primary retrieval benchmark: frequency extraction and multi-hop tracing in particular exercise broad attention coverage and cross-position composition that single-fact benchmarks do not. A model that retrieves individual facts perfectly but cannot aggregate or compose across positions will score well on single-key tasks and fail on these.

We report RULER at 128K tokens, the longest standardized evaluation length in the original suite. SubQ-1.1-Small achieves **99.12** on the full 13-task average. Performance is effectively saturated on the retrieval-oriented tasks, with the remaining errors concentrated in aggregation-style tasks such as common-word and frequent-word extraction, where success depends on broad coverage of the context rather than retrieval of a specific target.

Needle-in-a-Haystack (NIAH) places a single retrievable statement, such as a UUID-keyed passage, at a controlled depth within a long context of unrelated natural-language text, then prompts the model to locate and return it exactly. We use NIAH as a controlled probe for context-length generalization: because each sample has exactly one target and a well defined correct answer, it produces a clean binary signal that is unambiguous to score at lengths where more complex multi-task evaluation becomes expensive to run. We evaluate at 1M and 2M tokens (within the training window) and at 6M and 12M tokens (held-out). The held-out evaluation consists of 50 single-needle UUID samples packed to approximately 12M tokens in the style of the niah_single_1 task. The dataset was prepared for third-party verification. SubQ-1.1-Small achieves 100% at 1M and 2M and 98% at 6M and 12M (Figure 7). At 12M tokens, the model is attending to only 0.13% of token pairs.

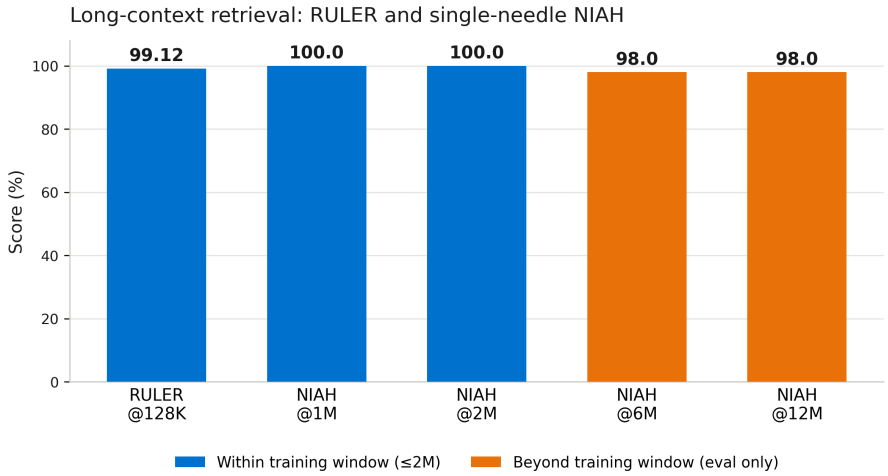

Figure 7: SubQ-1.1-Small long-context retrieval. Single-needle NIAH accuracy is perfect (100%) at 1M, 2M, 6M, and 12M tokens; RULER at 128K is near-saturated at 99.12%.

4.2 Knowledge Capability

A long-context model that retrieves precisely but reasons poorly over what it retrieves does not solve complete-artifact tasks. Contract analysis requires understanding legal definitions, not just locating them; repository-scale code reasoning requires understanding control flow, not just finding the relevant function. Knowledge capability, the ability to apply factual understanding and multi-step reasoning to domain-specific problems, is therefore not a secondary axis. It is a prerequisite for the retrieval capability reported above to translate into useful downstream behavior.

This is also where long-context training creates risk. The continued pretraining we performed can displace the knowledge and reasoning behavior acquired during the donor model's original training. Earlier checkpoints in our campaign showed exactly this pattern: retrieval improved while knowledge-intensive evaluations regressed. The capability-balancing stages described in Section 3.4 were developed in direct response.

GPQA Diamond [32] is a graduate-level science benchmark of expert-written questions in physics, chemistry, and biology, validated to be difficult for non-specialists. Domain PhD holders achieve approximately 65% accuracy, making it one of the more rigorous tests of whether a model retains deep factual knowledge and multi-step scientific reasoning.

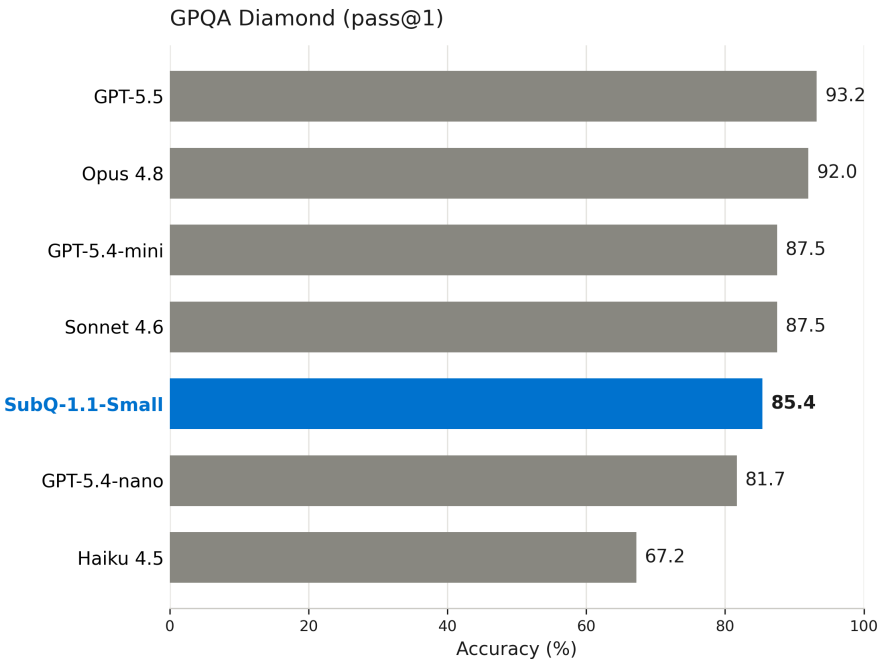

Figure 8: Knowledge capability on GPQA Diamond (pass@1). SubQ-1.1-Small (85.4%) sits just below the mid-tier frontier models (Sonnet 4.6 and GPT-5.4-mini at 87.5%) and well above the smaller tiers (GPT-5.4-nano 81.7%, Haiku 4.5 67.2%).

SubQ-1.1-Small achieves 85.4 pass@1. SubQ-1.1-Small lands between the small and mid frontier tiers, ahead of the smallest frontier models and close to the mid-sized ones. The intended interpretation is not state of the art across all short-context benchmarks; it is that a small model optimized for long-context behavior can also have strong knowledge and reasoning performance, which confirms that long-context optimization does not inherently come at the cost of reasoning quality. The result reflects the effectiveness of the capability-balancing stages described in Section 3.4.

4.3 Coding Capability

Long-context models have many applications in coding. Planning, review, and long-horizon memory and coherence are possibly the highest-value opportunities in the coding space today. Rather than general agentic programming, we see SubQ-1.1-Small being used for these specialized but critical tasks within broader coding systems.

LiveCodeBench [16] is a continuously updated competitive-programming benchmark that draws problems from platforms including LeetCode, Codeforces, and AtCoder. Problems are released after model training cutoffs, which limits memorization and tests genuine algorithmic reasoning. We evaluate LiveCodeBench v6 and report pass@4 following the evaluation method used in the original LiveCodeBench paper.

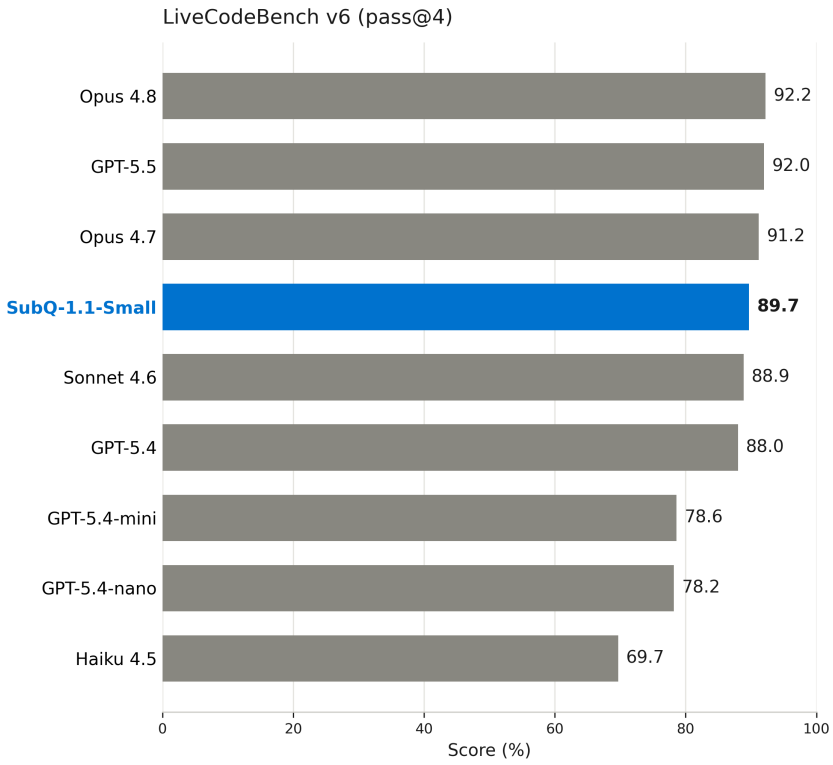

Figure 9: Coding capability. SubQ-1.1-Small reaches 89.7% on LiveCodeBench (pass@4, original-paper protocol), confirming that coding behavior was reintroduced after long-context optimization.

SubQ-1.1-Small achieves 89.7 pass@4, confirming that our long-context model can also perform strongly on coding tasks. As discussed in Section 5.3, coding data also served a dual role during training: it improved non-code long-context retrieval, likely because code is dense with the kind of cross-position dependencies that train general routing behavior.

4.4 Long-Horizon Agentic Tasks

The benchmarks in the preceding sections evaluate individual capabilities: retrieval, knowledge, and coding. In deployment, these capabilities must work together. A finance workflow might require an agent to read structured filings and unstructured correspondence, discover available API endpoints across multiple business applications, apply organizational policies, and produce actions whose correctness depends on evidence gathered across many prior steps. Errors compound across steps, and the agent must maintain coherent state throughout.

AutomationBench [33] evaluates exactly this setting. It presents agents with natural-language business tasks that require orchestrating actions across interconnected applications (CRMs, email, calendars, spreadsheets, messaging platforms) via REST APIs. Agents must autonomously discover the right endpoints from approximately 500 available across 47 applications, make sequential and interdependent API calls, and adhere to layered business rules drawn from policy documents. The environment is seeded with irrelevant and misleading records, so the agent must distinguish relevant context from noise across steps. Grading is binary end-state correctness, whether the right data reached the right systems, with no partial credit. We evaluate the Finance vertical, which applies this structure to finance-oriented workflows, because it tests whether long-context capability translates into sustained multi-step reasoning under realistic conditions. The results place SubQ-1.1-Small close to the strongest comparison model and ahead of smaller frontier baselines:

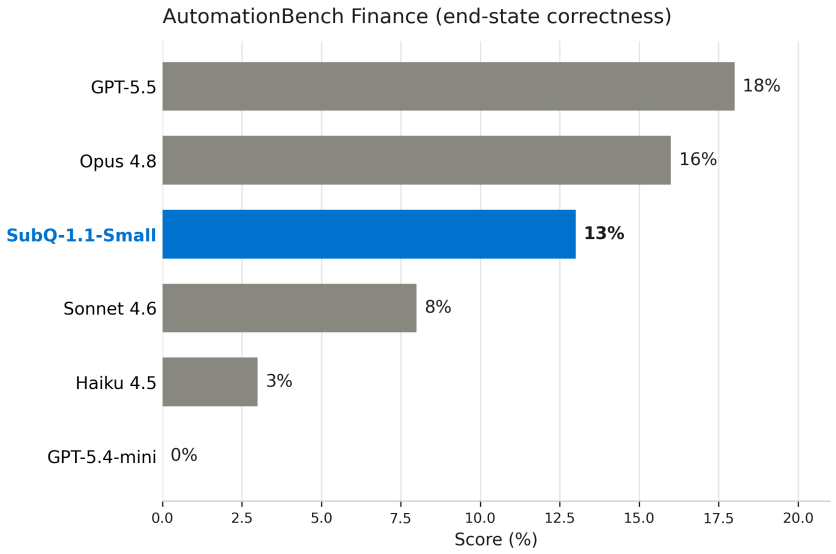

Figure 10: AutomationBench Finance scores on long-horizon agentic finance tasks. SubQ-1.1-Small (13%) sits close to the absolute frontier (Opus 4.8 16%, GPT-5.5 18.0%) and ahead of the medium and small models from Anthropic and OpenAI (Sonnet 4.6 8%, Haiku 4.5 3%, GPT-5.4-Mini 0.0%).

4.5 Efficiency

The efficiency measurements evaluate one attention layer on a B200 against a dense baseline on the same backbone. Settings seen here are extremely conservative. Given our time constraints, we have not ablated how aggressive we can be with sparsity and instead set a value we figured would be extremely safe in any setting, which proved to be true up to 12M tokens. The focus was not to maximize sparsity but rather to maximize context length. However, future work will include modeling with up to many times greater sparsity than today. Limited experiments were completed successfully with 4x the sparsity below with extremely postive results, and the floor could be even lower.

Table 1: Attention-mechanism FLOPs per forward pass for one attention layer, SSA vs. the same backbone with dense attention on every layer.

Context Length	Dense attention (PFLOP)	SSA (PFLOP)	Reduction
32K	0.25	0.12	2.1×
64K	0.99	0.25	4.0×
128K	3.9	0.49	8.0×
256K	15.8	0.99	16×
512K	63.0	2.0	31.5×
1M	252	3.9	64.5×

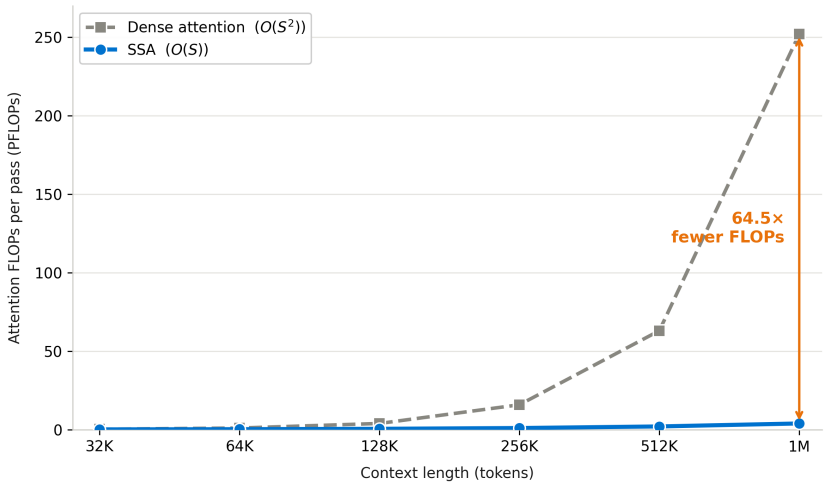

Figure 11: SSA compute on a single B200. Attention-mechanism FLOPs per forward pass: dense attention grows quadratically while SSA grows linearly, a 64× reduction at 1M tokens.

We also measure the attention mechanism in isolation against FlashAttention-2 on a single attention layer. This isolates the scaling behavior of SSA from MLPs, normalization, and other model-level costs.

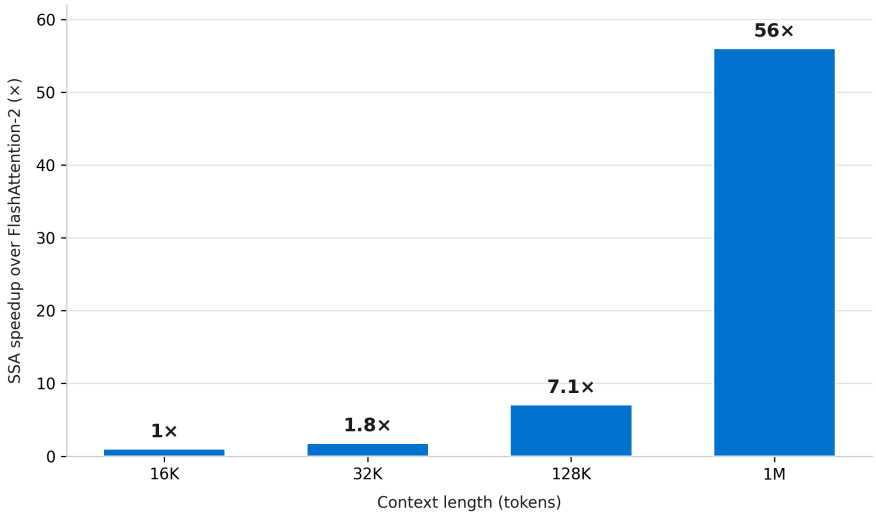

Figure 12: Single-attention-layer SSA speedup over FlashAttention-2. SSA reaches parity near 16K tokens and pulls away as context grows, to 56× at 1M tokens, where SSA runs in 966 ms versus 54,164 ms for FlashAttention-2. Measured on an H100.

Both measurements show the same shape: SSA's advantage over dense attention is not a uniform constant-factor speedup but a scaling-law win that becomes more valuable as sequence length grows.

5 Discussion

5.1 Context-Length Generalization

The central long-context result of this report is not a score at any single length. It is the algorithm's ability to retain retrieval accuracy with extremely high sparsity and linear scaling and generalize retrieval beyond the length on which the model was trained. SubQ-1.1-Small's long-context continued pretraining occurred overwhelmingly at 1M tokens, with a small fraction at 2M and none beyond it.

We evaluate single-needle retrieval out to 12M tokens, six times the maximum trained length and roughly twelve times the primary training window.

Single-needle retrieval generalizes strongly. On 50 held-out UUID NIAH samples per length, SubQ-1.1-Small achieves 100% recall at 1M and 2M, and 98% recall at both 6M and 12M. This was not a design target of the training run; it emerged after long-context CPT. Previously, we had achieved this result via multi-million-token SFT, but an initial run of long-context CPT enabled the model to generalize this result with less super-long-context SFT. At 12M tokens, SSA was attending to only 0.13% of tokens in the attention layer, making this evaluation practical without additional compression.

We believe this behavior is consistent with SSA's content-dependent routing. Because retrieval is driven by content relevance rather than fixed positional patterns, the mechanism may not impose an obvious context-length boundary once the relevant routing behavior has been learned.

5.2 Efficient Attention as a Research Accelerator

A dense long-context campaign is expensive enough that most teams get a handful of attempts. Under SSA we ran more than a hundred. That difference in experimental throughput, not the inference speedup, is what produced the findings below. The causal chain is concrete: SSA's linear cost not only made long-context experimentation not only economically tractable, but also enabled faster and deeper iteration.

Iteration stayed under a minute per step at million-token context, which was enough to run repeated variants across CPT mixtures, context-extension schedules, capability-balancing techniques, and post-training compositions, and to do so at the context lengths where long-context behaviors actually emerge.

The observations in this discussion are the kind that emerge from a high-variant-count regime: each was visible only because we could afford the variants that surfaced it, many CPT-volume settings, NIAH evaluations across many lengths and checkpoints, more checkpoints than a typical pipeline produces. The more variants a team can run, the more confidently outcomes can be attributed to specific changes, and capability becomes responsive to iteration velocity rather than to raw scale alone.

If long-context capability is bottlenecked on experimental throughput rather than on raw scale, then algorithmic efficiency becomes a first-class scaling variable, comparable in importance to model size and dataset size. The frontier on this axis is, in our reading, not yet well-explored.

5.3 Long-Context Pretraining as the Unlock

Across the experiments behind SubQ-1.1-Small, long-context CPT volume was the most consistent predictor of long-context retrieval gains. We are careful about how strongly to read this. We have not run controlled ablations in which the same CPT recipe is paired with architecturally distinct backbones, so we cannot rule out that a different architecture would respond differently to the same training regime. We are also early in our experimentation with reinforcement learning for long-context reasoning and retrieval. The version of the finding we hold with confidence is the weaker one: within the set of experiments we ran, long-context CPT was the most consistent lever. In our runs, the post-training variants we tried did not substitute for exposure to long contexts during continued pretraining, and our long-context post-training benefitted significantly from more long-context pre-training.

5.4 Balancing Short- and Long-Context Capability

A recurring pattern during SubQ-1.1-Small development was how tightly short- and long-context capability had to be balanced: gains in long-context capability frequently came at the expense of short-context capability unless the training was managed for both. At other times, long- and short-context training were complementary, where a long-context training run could improve the model's performance on short-context tasks or vice-versa. Research into which capabilities were most sensitive, which were most complementary, and which data was most effective at bolstering both was a key part of our work.

5.5 Evaluation and Measurement

Benchmark score and deployment-shaped behavior diverged more than we expected, and the gap changed how we selected checkpoints. The clearest case was MRCR v2. Early SubQ-1.1-Small

checkpoints scored strongly on MRCR, and we initially treated it as an important long-context signal. As development progressed, however, MRCR movement diverged from the behaviors we were trying to improve: repository-scale code reasoning, multi-document synthesis, and contract-style analysis. The checkpoints that looked best by MRCR were not consistently the checkpoints that behaved best on those tasks.

We identified this gap through early workflow testing and a fixed set of qualitative spot-checks maintained alongside the formal benchmarks. The first signal was qualitative: MRCR-optimized checkpoints often felt worse in use, even when the benchmark moved in the right direction. To make that signal less anecdotal, we tracked it through fixed spot-checks covering repository-scale code reasoning, multi-document synthesis, and retrieval against real contract and engineering-documentation corpora. These were not standardized benchmark results, and we do not report them as such. They were development diagnostics, scored by team members against a consistent rubric, and used to test whether a checkpoint was improving on the workflows users were actually trying to run.

The tradeoff showed up during checkpoint selection. Pushing MRCR higher did not reliably make the model better on the tasks we cared about, and in some runs it made the model worse on the fixed spot-checks.

RULER became the more useful development signal. Improvements and regressions on RULER more often tracked the qualitative spot-checks. Our interpretation is that RULER's multi-task structure better overlaps with whole-artifact reasoning: aggregation, composition, retrieval under distractors, and multi-hop reasoning.

5.6 DeepSeek Sparse Attention and the Cost of Selection

DeepSeek's recent long-context systems are the closest published comparison point to SSA. The primary architectural innovation in DeepSeek's dynamic sparse attention line, including DeepSeek V3.2 and DeepSeek V4, is the Lightning Indexer: a learned mechanism that dynamically chooses which context positions each query should attend to (DeepSeek-AI 2025, 2026). This is the same problem SSA is designed to solve. The important distinction is where the cost of choosing is paid. We set out to solve this problem without re-introducing the quadratic scaling laws of DeepSeek's Lightning Indexer, so as the create a benefit that is more than scalar.

There are three related mechanisms worth separating. DeepSeek Sparse Attention (DSA) is the dynamic sparse attention mechanism itself: the Lightning Indexer, a full attention model distilled from the teacher model, scores query-key pairs and selects which positions downstream sparse attention should read in the larger teacher model. CSA uses the same selection idea, but runs the Lightning Indexer over a compressed representation of the context. In other words, DSA and CSA perform the same kind of learned dynamic selection, and they differ primarily in the representation over which selection is performed. HCA is different: it does not perform learned selection. It compresses the context heavily and then applies brute-force dense attention over that compressed representation.

SSA directly targets the selection role played by the Lightning Indexer in DSA and CSA. Conceptually, SSA can replace the selector in either setting: over an uncompressed representation, as in DSA, or over a compressed representation, as in CSA. HCA-style compression is also not inherently incompatible with SSA. We have done limited exploration of SSA combined with heavily compressed representations, but we have not yet robustly validated at a model that performs SSA selection over those representations. We view that combination as future work.

In DSA, the indexer is cheaper than the attention it serves only at short context. In DeepSeek v3.2, beyond a crossover near 52,000 tokens, its quadratic scoring overtakes the linear-complexity sparse attention in the teacher model, reaching roughly 16.1 times the teacher-attention cost at 1M tokens and 190.4 times at 12M tokens. A routing mechanism intended to make long context affordable becomes the dominant long-context cost, reintroducing quadratic scaling after providing scalar compute savings.

Table 2: Main-attention (fixed top-k selection) versus Lightning-Indexer FLOPs per layer for a V3.2-style DSA configuration. The main attention reads the released configuration's fixed 2,048 selected tokens and is linear in sequence length; the indexer scores all pairs and is quadratic. The indexer is cheaper below about 52K tokens, then overtakes the attention it feeds: roughly 16× more expensive at 1M tokens and 190× at 12M. T $= 10^{12}$, P $= 10^{15}$ FLOPs.

Seq. length	Indexer	Sparse attn	Index / Attn
≈52K	28.0T	28.0T	1.0× — crossover
128K	156.4T	71.0T	2.2×
256K	594.2T	142.1T	4.2×
512K	2.31P	284.2T	8.1×
1M	9.13P	568.3T	16.1×
2M	36.3P	1.14P	31.9×
4M	144.6P	2.27P	63.6×
8M	577.5P	4.55P	127.0×
12M	1,298.5P	6.82P	190.4×

SSA's selector is dramatically cheaper. The table below compares per-layer prefill FLOPs under a matched selected-position budget:

Table 3: Per-layer prefill FLOPs for DSA versus SSA under a matched selected-position budget. DSA includes Lightning Indexer scoring.

Seq. length	DSA layer	DBSA layer	DSA / DBSA
128K	227.4T	71.0T	3.2×
256K	736.3T	142.1T	5.2×
512K	2.60P	284.2T	9.1×
1M	9.70P	568.3T	17.1×
2M	37.4P	1.14P	32.9×
4M	146.9P	2.27P	64.6×
8M	582.0P	4.55P	128.0×
12M	1,305.4P	6.82P	191.3×

5.7 Implications for Long-Context Applications

The discussion in this section so far has been about model development. The remainder is about deployment.

The most important enterprise implication of long-context models, in our view, is not larger context windows themselves. It is the ability to reason directly over complete or more-complete artifacts and create more generalizable systems that avoid many of the tradeoffs that come through the curation in RAG and agent systems. Many enterprise AI systems today rely on carefully curated and scoped retrieval, re-ranking, chunking, and orchestration infrastructure to reconstruct context that already exists in the underlying data, introducing brittleness and compounding errors along the way. As long-context capability improves, some of the retrieval, re-ranking, and orchestration logic currently required to reconstruct context can move into the model itself. We view this transition, from reasoning over retrieved fragments to reasoning over the artifact directly, as one of the most important shifts enabled by long-context AI, and the rest of this section develops what it means in practice.

5.7.1 Whole-Artifact Reasoning

The practical implication of these results is not simply that larger windows fit more tokens. It is that some tasks currently implemented as retrieval problems are more naturally whole-artifact reasoning problems once the relevant artifact fits in context. In many practical systems, retrieval and chunking are used not because the task is naturally decomposable, but because the relevant artifact does not fit in context. As the reachable context length grows, some decomposition stages become unnecessary rather than merely easier to implement.

The structure recurs across domains: legal work requiring cross-reference resolution across a contract, financial review connecting filings and internal records, research requiring synthesis across a bounded

literature. In each case the difficulty is not locating a passage; it is reasoning over relationships distributed across the artifact. Fragmentation systematically destroys those relationships before the model ever sees them. Holding the whole artifact in context changes the shape of the task rather than only the speed of it.

In many cases, even simply increasing the number of retrieved chunks, increasing the size of chunks, or increasing the number of concurrent retrieval steps can improve a system, and access to higher-intelligence, cheaper, and faster context even within existing windows will enable that transfromation.

The implication is not that retrieval is obsolete. For corpora larger than any plausible context window, knowledge that changes faster than the prompt can be updated, and workflows with genuine multi-stage structure, retrieval and orchestration remain the right tools. The narrower claim is that some scaffolding exists primarily to compensate for context limits. As efficient long-context models extend the reachable window, that class of scaffolding becomes smaller. For tasks whose difficulty comes from relationships distributed across a bounded artifact, preserving the artifact is not an implementation detail; it is part of the capability.

6 Conclusion

We presented SubQ-1.1-Small, a long-context language model built on Subquadratic Sparse Attention (SSA), a content-dependent attention mechanism with linear compute and memory complexity. SSA reduces attention FLOPs by 64.5× at a 1M-token context window, which made multi-million-token training and evaluation practical enough to run more than one hundred long-context experiments at the context lengths where the relevant behavior appears. The value of SSA is therefore not only that it makes long-context inference cheaper. It makes long-context experimentation cheaper.

That experimental regime produced the central empirical result of this work: long-context retrieval generalized far beyond the window used during training. SubQ-1.1-Small was trained primarily at 1M tokens, with additional training at 2M, yet single-needle retrieval held at 98% at both 6M and 12M. In our experiments, long-context capability did not behave like specialization to a fixed context length. It behaved like a capability learned through exposure to long contexts, with long-context continued pretraining emerging as the strongest lever we found.

The same training regime also produced evidence beyond retrieval. On AutomationBench Finance, SubQ-1.1-Small achieved 13%, approaching Opus 4.8 and substantially outperforming Sonnet 4.6 and Haiku 4.5 under the same setup, early evidence that the long-context training regime transfers to long-horizon agentic reasoning.

The broader implication is that efficient attention changes the development loop. If the cost of long-context experiments is too high, teams are forced to guess at the recipe. If the cost falls far enough, they can search for it. In our case, that search mattered as much as the architecture itself: it exposed the importance of long-context continued pretraining, capability balancing, coding data, and evaluation choices that would have been difficult to discover from a small number of runs.

The point of this progress is not longer context windows for their own sake. It is to train models that can reason over complete artifacts directly: repositories, document collections, knowledge bases, and long-running workflows. Models that can do this well should make many current systems simpler: less chunking, less retrieval scaffolding, less orchestration to compensate for missing context. The goal is faster, cheaper, and more capable models whose intelligence is applied to the artifact itself rather than to fragments reconstructed around a context limit.

References

[1] Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. In *Proceedings of ICLR*, 2024.

[2] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[3] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[4] Gonçalo M. Correia, Vlad Niculae, and André F.T. Martins. Adaptively sparse transformers. In *Proceedings of EMNLP-IJCNLP*, 2019.

[5] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[6] Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Proceedings of ICML*, 2024.

[7] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in NeurIPS*, 2022.

[8] DeepSeek-AI. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[9] DeepSeek-AI. Deepseek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[10] DeepSeek-AI. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv preprint arXiv:2502.11089*, 2025.

[11] DeepSeek-AI. DeepSeek-V4-Flash: Compressed sparse attention with a lightning indexer, 2026. Model card and technical report. Available at https://huggingface.co/deepseek-ai/DeepSeek-V4-Flash.

[12] Gemma Team, Google DeepMind. Gemma 4 technical report. 2026.

[13] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[14] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, et al. RULER: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

[15] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. DeepSpeed Ulysses: System optimizations for enabling training of extreme long sequence transformer models. *arXiv preprint arXiv:2309.14509*, 2023.

[16] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

[17] Samy Jelassi, David Brandfonbrener, Sham M. Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. In *Proceedings of ICML*, 2024.

[18] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of ICML*, 2020.

[19] Kimi Team. Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

[20] Aakash Lahoti, Kevin Y. Li, Berlin Chen, Caitlin Wang, Aviv Bick, J. Zico Kolter, Tri Dao, and Albert Gu. Mamba-3: Improved sequence modeling using state space principles. *arXiv preprint arXiv:2603.15569*, 2026.

[21] Patrick Lewis, Ethan Perez, Aleksandra Piktus, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in NeurIPS*, 2020.

[22] Opher Lieber, Barak Lenz, Hofit Bata, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

[23] Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context. *arXiv preprint arXiv:2310.01889*, 2023.

[24] MiniMax. Why did MiniMax-M2 end up as a full attention model?, 2025. MiniMax engineering blog, 30 October 2025. Available at https://huggingface.co/blog/MiniMax-AI/why-did-m2-end-up-as-a-full-attention-model.

[25] MiniMax. MiniMax-M1: Scaling lightning attention to 128k sequences. *arXiv preprint*, 2025. MiniMax subsequently returned to fully dense attention for their frontier model.

[26] MiniMax. The MiniMax-M2 series: Mini activations unleashing max real-world intelligence. *arXiv preprint arXiv:2605.26494*, 2026. M2 returns to full attention across all layers, reporting that no efficient-attention variant reliably matched full-attention quality across production reasoning, coding, and agent tasks.

[27] NVIDIA. NVIDIA nemotron 3 family of models. Technical report, 2025. Available at https://research.nvidia.com/labs/nemotron/Nemotron-3/.

[28] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, et al. RWKV: Reinventing RNNs for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

[29] Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023.

[30] Qwen Team, Alibaba. Qwen3-next technical report. 2026. Available at https://huggingface.co/docs/transformers/main/model_doc/qwen3_next.

[31] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. *arXiv preprint arXiv:1910.02054*, 2020.

[32] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof Q&A benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

[33] Daniel Shepard and Robin Salimans. AutomationBench: Evaluating AI agents on realistic business workflows, 2026. arXiv preprint arXiv:2604.18934.

[34] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

[35] Richard S. Sutton. The bitter lesson. http://www.incompleteideas.net/IncIdeas/BitterLesson.html, 2019. March 13, 2019.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in NeurIPS*, 2017.

[37] Chejian Xu, Wei Ping, Peng Xu, Zihan Liu, Boxin Wang, Mohammad Shoeybi, Bo Li, and Bryan Catanzaro. From 128K to 4M: Efficient training of ultra-long context large language models. *arXiv preprint arXiv:2504.06214*, 2025.

[38] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *Proceedings of ICLR*, 2025.

[39] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In *Advances in NeurIPS*, 2020.

[40] Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models. *arXiv preprint arXiv:2512.24601*, 2025.

[41] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. PyTorch FSDP: Experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.