Language Models Demand New Computer Science

Chang Yang[1], Xinrun Wang[2]*, Shuxin Li[3], Qinggang Zhang[4],
Huachi Zhou[1], Zhuoyi Lin[5], Bo Li[1], Xiao Huang[1]
[1]The Hong Kong Polytechnic University, [2]Singapore Management University
[3]Nanyang Technological University, [4]Jilin University
[5]Institute for Infocomm Research, Agency for Science, Technology and Research (A*STAR)

Abstract

The emergence of large language models (LMs) as versatile problem-solving systems marks a paradigm shift comparable to the advent of digital computers. Traditional computer science provides theoretical foundations for understanding computational processes through complexity theory, algorithms, and formal methods, this position paper argues that **language models require their own distinct theoretical framework, termed as *Computer Science of Language Models* (CSLM)**. We identify seven fundamental differences between traditional computing and language model: 1) the propose-solve-verify triad versus the solving-verifying dichotomy, 2) token-based rather than operation-based computation, 3) probabilistic reliability with hallucinations rather than deterministic correctness, 4) native multimodal understanding beyond pixel-level processing, 5) general-purpose capabilities versus problem-specific algorithms, 6) essential two-stage training-inference processes, and 7) inherent knowledge. We argue these differences necessitate new theoretical frameworks, complexity measures, and evaluation methodologies. Then, we define the **token complexity** as the core concept in CSLM, treating the tokenizer as part of the system, and discuss connections to existing research in scaling laws, memory systems, tool use, and fine-tuning, while identifying critical open questions about LM-complete problems, reliability bounds, emergence phenomena, and multimodal complexity. Drawing parallels to complexity theory's 50-year development, we present a 5-year research roadmap (2026-2030) to build the CSLM for transforming LM development from empirical scaling to principled engineering with rigorous theory and our understanding about the problems, LMs and their relationships.

1 Introduction

Language models have evolved from specialized text processors into versatile computational systems capable of solving increasingly diverse and complex problems [1, 86, 97]. Modern large language models (LLMs) demonstrate remarkable problem-solving capabilities across domains that traditionally require specialized algorithms: they prove mathematical theorems, generate functional code, reason about visual scenes, and even orchestrate complex multi-step workflows [13, 93]. This trajectory suggests that language models are becoming *the new universal computer*, i.e., a general-purpose computational architecture fundamentally different from traditional von Neumann architectures of computers, yet potentially equally powerful and ubiquitous. Despite this transformative empirical success, our theoretical understanding of language model computation remains limited. Traditional computer science, built upon decades of rigorous theory, provides powerful frameworks for analyzing algorithms, proving complexity bounds, and understanding computational limits through classes like P and NP [19, 29, 41]. However, these frameworks were designed for deterministic, operation-based computation with explicit algorithms and verifiable correctness. Language models violate nearly every assumption underlying classical computer science: they operate probabilistically rather than

*Corresponding author (xrwang@smu.edu.sg).

40th Conference on Neural Information Processing Systems (NeurIPS 2026). Position Paper Track.

deterministically, generate tokens rather than execute operations, hallucinate rather than guarantee correctness, understand multiple modalities natively, and encode vast knowledge within their parameters [39, 58] (details in Section 2). This fundamental mismatch between classical theory and modern practices creates a critical gap: we can build powerful language models, but we cannot predict their behavior, understand their limitations, or guide their development through principled theory.

Definition 1 (CSLM)**.** *Computer Science of Language Model (**CSLM**) is a theoretical and empirical framework for understanding computation by language model (LM), encompassing complexity analysis, algorithmic behavior, reliability properties, and the relationship between model architectures, training procedures, and problem-solving capabilities across diverse modalities and tasks.*

Therefore, this paper argues that **language models require their own distinct theoretical framework, termed as *Computer Science of Language Model* (CSLM).** The formal definition of CSLM is presented in Definition 1. First, we identify seven fundamental dimensions along which language model computation differs qualitatively from traditional computer science (Section 2), establishing the need for a distinct theoretical framework. Second, we introduce **token complexity** as the foundational concept in CSLM, providing a unified measure for characterizing both model capabilities and problem difficulty relative to a stated success criterion, with the tokenizer treated as part of the system and each definition instantiated on a running sorting example (Section 3). Third, we connect CSLM to existing research in scaling laws, multi-agent systems, memory management, tool use, and fine-tuning, showing how our framework unifies these diverse areas (Sections 4 and 5). Fourth, We identify critical open questions about LM-complete problems, reliability bounds, emergence phenomena, and multimodal complexity that define the research frontier and present a 5-year research roadmap (2026–2030) inspired by the 50-year development of classical complexity theory, outlining concrete milestones toward establishing CSLM as a mature scientific discipline (Section 6). Finally, we engage the strongest alternative views (Section 7) and the answers to frequently asked questions are collected in Appendix A. CSLM can provide the theoretical foundations for LMs, strengthen our understanding of both the problems we want to solve and the language models we build to solve them, ultimately enabling the development of more capable, reliable, and efficient AI systems.

2 Seven Fundamental Differences

We identify seven fundamental dimensions along which language model computation differs qualitatively from traditional computer science (displayed in Figure 1). We do not set LMs against classical computation. An LM is a computational model, and CSLM extends classical theory, rather than contradicting it, similar to quantum complexity theory. The first three (§1–§3) define the computational model itself, namely what is computed, in what unit, and with what reliability, and they carry the main argument. The remaining four (§4–§7) describe properties of the system that shape token complexity, and we develop them as consequences of the first three.

§1. Propose-Solve-Verify Triad. Traditional complexity theory distinguishes between problems based on the difficulty of solving versus verifying solutions [19, 41]. The class P contains problems solvable in polynomial time, while NP contains problems whose solutions can be verified in polynomial time [29]. This solving-verifying dichotomy has been foundational to computer science for over half a century. Language models introduce a third dimension: *proposal*, by which we mean generating new problems, as distinct from generating solutions or chains of thought. LMs can propose novel problem instances, create variations of existing problems, and even synthesize entirely new problem classes [15, 108]. This claim rests on current practice rather than projection: LMs already put forward conjectures in automated research pipelines, and they generate training problems for themselves and for other models through self-play and co-evolution. This capability enables several research directions unavailable in traditional computing. We can study **proposal complexity**, i.e., how difficult it is for an LM to propose valid, interesting, or challenging problem instances. For example, "How many tokens it will take for LLMs to propose Riemann

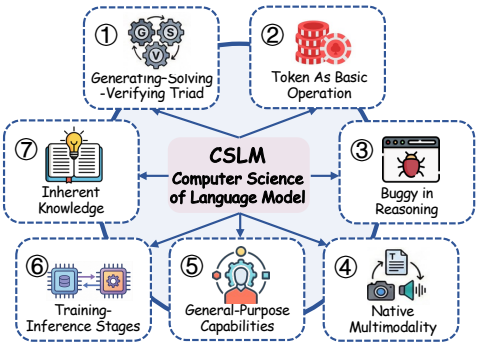


Figure 1: Seven fundamental differences between CSLM and traditional CS.

hypothesis?" and "Which one is hard for LLMs to propose the Riemann hypothesis or the Fermat's Last Theorem?" Furthermore, LMs can explore problem spaces by generating variations, interpolations, and extrapolations, enabling empirical investigation of problem difficulty landscapes in ways impossible with hand-crafted benchmarks. Finally, by generating problems of increasing difficulty, LMs can potentially create optimal training curricula for themselves or other models, leading to recursive self-improvement [24]. The propose-solve-verify triad fundamentally changes how we conceptualize computational tasks: in traditional computer science, problems exist as a priori and novel algorithms are developed to solve them, while in CSLM, LMs operate across all three phases.

§2. Token as Basic Operation. In traditional computing, we measure complexity by counting basic operations [21]: arithmetic operations, comparisons, memory accesses, and other primitive instructions. Time complexity is expressed in terms of the input size n, with $O(n)$, $O(n^2)$, or $O(2^n)$ characterizing algorithm efficiency. For language models, the natural unit of computation is the *token*, i.e., each forward pass processes one token (in autoregressive models) or a sequence of tokens (in parallel architectures). Different models may require varying numbers of tokens to solve identical problems, potentially revealing fundamental differences in their problem-solving mechanisms [54, 98]. Moreover, problems considered trivial in traditional computer science may prove challenging for language models, and conversely, problems deemed complex may be solved efficiently. Consequently, a comprehensive theoretical framework centered on token-based analysis is needed. Understanding token complexity is essential for practical LM deployment, as token consumption directly determines computational cost and latency. This makes developing principled methods for analyzing and optimizing token efficiency a critical priority.

§3. Unreliable Reasoning. Perhaps the most striking difference between traditional and language model computing is reliability. A correctly implemented algorithm produces correct outputs deterministically [43], whereas LMs exhibit probabilistic, error-prone behavior [35, 39]. Unlike classical randomized algorithms (e.g., MCTS, randomized quicksort), which introduce randomness as a *deliberate design choice* with provable correctness guarantees, uncertainty in LM outputs arises from learned distributions over training corpora and we currently lack the theoretical tools to guarantee the correctness of their reasoning. Classical analysis of algorithms does incorporate noise, but typically through analytic models with known form and parameters, such as Gaussian or Poisson noise or bounded adversarial corruption. LM errors are not well described by any fixed analytic family, since they depend on the input, the prompt, and the training distribution. CSLM therefore requires a broader class of noise models, and estimating the parameters of the noise model becomes part of the analysis rather than an assumption. **Hallucination is a pervasive property** [58, 106]: LMs generate plausible but incorrect outputs even when correctly prompted. Unlike implementation bugs, reasoning errors in LMs have no fixed remedy and cannot be eliminated by patching model weights, though architectural improvements and inference-time interventions have demonstrably reduced error rates. We must therefore reason in terms of **probabilistic correctness** rather than deterministic guarantees, raising the question of whether formal scaling laws for reliability can be established. Finally, unreliable reasoning makes **verification both more important and more difficult**: building LLMs that can jointly generate and certify the correctness of their outputs remains an open problem.

§4. Native Multimodality. Traditional computers operate on digital representations, i.e., bits, bytes, numbers, and strings. While they can process images and audio at the signal level, they lack inherent understanding of perceptual content [85]. Specialized algorithms for computer vision and speech recognition exist, but they remain problem-specific tools. Multimodal language models change this fundamentally [36, 86] by embedding text, images, audio, and video into **unified representation spaces**, enabling cross-modal reasoning such as answering text questions about images, generating images from text descriptions, or transcribing speech with visual context. Unlike pixel-level image processing, multimodal LMs extract high-level concepts from perceptual data, recognizing objects, understanding scenes, interpreting diagrams, and reasoning about visual relationships. The **conceptual understanding** is qualitatively different from traditional computer vision. This conceptual handling is tied to token-level processing. Multimodal LMs encode images, audio, and video into token sequences, either through patch encoders that occupy a fixed number of positions per tile or through discrete codebooks that emit tokens into a shared vocabulary, and these tokens enter the same context and consume the same budget as text. Because all modalities are reduced to a shared token stream aligned in a shared representation space, perceptual input becomes available to the same processing as language, and token complexity applies to multimodal problems without a separate unit of cost. With sufficiently advanced multimodal LMs, we approach **universal computation** that

can handle any modality of input and output, beyond the capabilities of traditional computing. The multimodal nature of modern LMs implies that CSLM should be inherently multimodal, as theories for text-only models may not generalize, and the multimodal case remains largely unexplored.

§5. General-Purpose Capabilities. Traditional algorithms are designed for specific problems, e.g., quicksort for sorting [32], Dijkstra's algorithm for shortest paths [22], FFT for signal processing [20]. In contrast, a single LM can perform translation, summarization, question answering, coding, mathematical reasoning, creative writing, and more, all without task-specific programming [1, 12, 17]. LMs develop internal "algorithms" for solving problems, learned from data rather than explicitly programmed, and understanding these **emergent algorithms** through mechanistic interpretability, probing, and analysis is a core CSLM challenge. LMs can compose learned capabilities to solve novel tasks, and understanding **compositional generalization** could lead to theories of how complex problem-solving emerges from simpler primitives. The generalizability of LMs means that CSLM can pursue unified theories of computation across tasks and modalities, something impossible in traditional CS where each domain requires specialized treatment.

§6. Training-Inference Stages. Traditional computer science focuses almost exclusively on algorithm execution (inference). Once an algorithm is designed and implemented, its behavior is fixed and deterministic for given inputs. Language models introduce an essential training stage that fundamentally affects inference behavior [73, 91]. Training an LM is analogous to programming a traditional computer, but the relationship between training data ("program") and inference behavior ("execution") is complex and indirect, requiring theories connecting training procedures to emergent capabilities. Training also consumes tokens, and these training tokens are operations that rewrite the program rather than execute it, so that expending them changes the inference token complexity of a fixed problem (Section 5.5). One success story of CSLM is **scaling laws** [33, 40], which predict model performance based on parameters, data, and compute. Extending scaling laws to cover more phenomena (reasoning, factuality, safety) is critical. LMs exhibit sudden capability jumps at certain scales, i.e., the emergence of in-context learning, chain-of-thought reasoning, and task generalization [93]. Understanding when and why these **phase transitions and emergence** occur is a key scientific question. Recent work on grokking [69] and training dynamics [74] provides insights, but comprehensive theories are lacking regarding how training shapes the solution strategies in LLMs.

§7. Inherent Knowledge. Traditional computers are data-agnostic at runtime where they operate purely on inputs provided with no built-in knowledge about the world [4]. A sorting algorithm knows nothing about what it sorts; a graph algorithm understands only the structure given to it. Language models fundamentally differ by encoding vast amounts of knowledge within their parameters during training [59, 68, 75]. This **parametric knowledge** spans facts, concepts, relationships, procedures, and heuristics accumulated from training data, enabling LMs to answer questions, make inferences, and reason about domains without external information sources. The knowledge is implicit and distributed across billions of parameters rather than explicitly stored in databases or knowledge graphs. This implicit knowledge in LLMs raising questions about **knowledge capacity, organization, and retrieval**: How much information can models store? How is knowledge structured internally? A key challenge is that LMs function simultaneously as both knowledge bases and reasoning engines, making it difficult to disentangle these roles. Distinguishing between memorized knowledge and genuine reasoning remains an open problem. The inherent knowledge in LMs is a double-edged sword: while such knowledge can serve as valuable heuristics for problem-solving, it may also distract LLMs from proper reasoning [42]. Models may blindly follow memorized patterns and produce incorrect answers without engaging in necessary reasoning steps. Recently, Engram [16] also tackle the relationship between memory and reasoning, i.e., MoE and N-gram models, where they find that 75-80% for reasoning and 25-20% for memory is the best ratio.

3 Preliminaries

3.1 Basic Definitions

To establish a rigorous framework for evaluating LMs, we introduce the concept of ***token complexity***, which characterizes the computational cost required to achieve a desired level of performance on a given problem instance. This framework extends classical computational complexity theory to language models by characterizing the token budget required to achieve a specified success rate.

Definition 2 (Problems and Instances)**.** *Let $\mathcal{P}$ denote the set of all considered problems, where $P \in \mathcal{P}$ represents a distinct problem and $\mathcal{I}_P$ denotes the set of all valid instances of problem P.*

Definition 3 (Language Model System)**.** *A language model is denoted M, and it is specified together with its tokenizer $\kappa_M = (V_M, \mathrm{enc}_M, \mathrm{dec}_M)$, where V_M is the vocabulary and enc_M maps inputs of every supported modality to sequences over V_M. An LM system (LMS) $\mathcal{M}$ consists of one or more language models together with their tokenizers and the inference procedure used (e.g., decoding, sampling, and test-time scaling). Tokens used by $M_i \in \mathcal{M}$ are counted in its own vocabulary V_{M_i}.*

We characterize the performance of an LMS on a problem instance via two key measures: the number of tokens $\boldsymbol{\tau}$ and the probability of solving the instance $p \in [0, 1]$ referred to as the *success rate* (SR). For a given instance I, the objective is to minimize τ while maximizing p (Figure 2). Success is defined relative to an *acceptance criterion* Ω, which may be an exact-match check, a formal proof checker such as Lean, a rubric, or an LLM judge, and p is the empirical rate at which the system's outputs are accepted by Ω over repeated independent runs. In particular, p is not the model's next-token probability and does not measure how plausible an output appears.

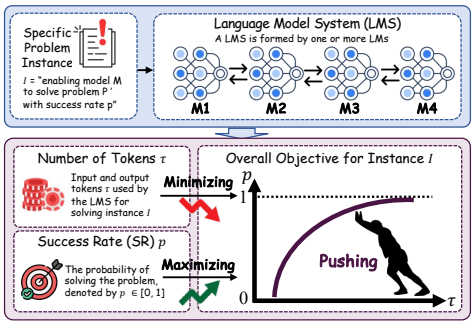


Figure 2: Token vs success rate.

Definition 4 (Token Complexity)**.** *Given an instance I, an LMS $\mathcal{M}$, and an acceptance criterion V (suppressed from the notation), let $\boldsymbol{\tau}^{in}(I, \mathcal{M}, p)$ and $\boldsymbol{\tau}^{out}(I, \mathcal{M}, p)$ denote the minimal total input and output tokens with which $\mathcal{M}$ produces an output accepted by Ω with success rate at least p, respectively, where $\tau_i^*(I, \mathcal{M}, p)$ is the token count attributed to $M_i \in \mathcal{M}$. The* token complexity *of an instance is defined as the family of achievable (τ, p) pairs across all success rates:*

$$\mathsf{TokC}^{in}(I, \mathcal{M}) = \left\{\boldsymbol{\tau}^{in}(I, \mathcal{M}, p) : p \in [0, 1]\right\}, \quad \mathsf{TokC}^{out}(I, \mathcal{M}) = \left\{\boldsymbol{\tau}^{out}(I, \mathcal{M}, p) : p \in [0, 1]\right\},$$

with $\mathsf{TokC}(I, \mathcal{M}) = \left\{\mathsf{TokC}^{in}(I, \mathcal{M}),\ \mathsf{TokC}^{out}(I, \mathcal{M})\right\}$.

Crucially, token complexity is a *curve*, not a scalar, mapping token consumption to success probability $p \in [0, 1]$, analogously to how time complexity in classical theory characterizes full scaling behavior. This representation makes explicit trade-off between computational budget and solution quality. The curve is a stochastic characterization: a given budget yields a success probability, and the framework describes the relation between the two.

Definition 5 (System Profile and Problem Profile)**.** *Let $\mathfrak{M}$ denote a set of LM systems under study. The* system profile *of an LMS $\mathcal{M}$ aggregates its token complexity across all problems:* $\mathsf{SP}(\mathcal{M}) = \left\{(I, \mathsf{TokC}(I, \mathcal{M})) : I \in \mathcal{I}_P,\ P \in \mathcal{P}\right\}$. *The* problem profile *of a problem P aggregates token complexity across all systems:* $\mathsf{PP}(P) = \left\{(I, \mathcal{M}, \mathsf{TokC}(I, \mathcal{M})) : I \in \mathcal{I}_P,\ \mathcal{M} \in \mathfrak{M}\right\}$.

The system profile characterizes one system across tasks, and the problem profile characterizes one problem across systems as a family of curves. Because the problem profile retains each system's curve, heterogeneous systems with different tokenizers or modality encoders are compared without being reduced to a common unit. In the running example, the problem profile of sorting restricted to the two systems is the pair of curves reported in Appendix B.

3.2 Big-O Notation and Formal Complexity Classes

Just as classical algorithms are characterized by their asymptotic operation count as a function of input size n, we characterize LM behavior by the asymptotic token count. Let $n = |I|$ denote the size of problem instance I (e.g., the number of elements for sorting, or the number of variables for SAT).

Definition 6 (Asymptotic Token Complexity)**.** *We say an LMS $\mathcal{M}$ solves problem P with* token complexity $O(f(n))$ *at success rate $p \geq p^*$ if there exists a constant $C > 0$ such that $\tau_{out}(I, \mathcal{M}, p^*) \leq C \cdot f(|I|)$ for all sufficiently large instances I. This asymptotic lens makes token complexity a measurable, comparable quantity across models and problems.*

Asymptotic statements presuppose a parameterized family of instances with a size or difficulty parameter n. For a single instance, such as one open conjecture, token complexity is measurable but no asymptotic claim can be made, which is a scope condition of the framework. Classical problems supply rigorous size parameters, while for other tasks, difficulty can be parameterized by human annotation, and developing such parameterizations is part of the agenda in Section 6.

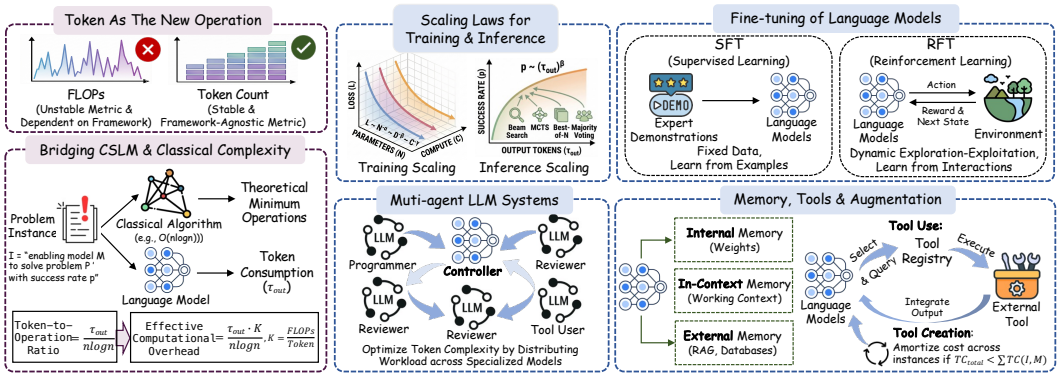


Figure 3: Connections to existing research. Left: CS research. Right: LLM research.

Token counts depend on the tokenizer only up to constants. If every token of κ_A and κ_B encodes between one and L_A (resp. L_B) bytes, then $|\mathrm{enc}_A(s)| \le |s| \le L_B\,|\mathrm{enc}_B(s)|$, and symmetrically, so switching tokenizers changes counts by at most $\max(L_A, L_B)$. Acceptance by Ω is unaffected, and the factor is absorbed into C (Definition 6), so exponents, and hence complexity classes, are tokenizer-invariant. Byte-level BPE and SentencePiece with byte fallback, which cover essentially all deployed text tokenizers, satisfy this assumption, and the empirical factor is small (Appendix M). Retraining a model with a new tokenizer, by contrast, can change the curve itself (Section 2, §2).

Definition 7 (LM-P)**.** *A problem P is in LM-P if there exists an LMS $\mathcal{M}$ solving any instance $I \in \mathcal{I}_P$ with success rate $p \ge p^*$ (for some fixed threshold $p^* > 0$) using $O(\mathrm{poly}(|I|))$ tokens.*

Definition 8 (LM-NP)**.** *A problem P is in LM-NP if a solution can be* verified *using $O(\mathrm{poly}(|I|))$ tokens at success rate p, even generating the solution may require super-polynomial tokens.*

Definition 9 (LM-Complete)**.** *A problem P is LM-Complete for LM-NP if $P \in$ LM-NP and every problem in LM-NP is token-efficiently reducible to P.*

These definitions deliberately mirror their classical counterparts. Whether LM-P = LM-NP is an open question for LMs as the classical P vs. NP question is for algorithm design. The classes are probabilistic: they are defined relative to a fixed success threshold p^*, in the same way that BPP is defined by a fixed error bound, and the threshold does not make the framework deterministic.

Definition 10 (Token-Efficient Reduction)**.** *Problem A is* token-efficiently reducible *to problem B (denoted $A \le_T B$) if there exists $\mathcal{M}$ that transforms any instance I_A into an instance I_B using $O(\mathrm{poly}(|I_A|))$ tokens, such that solving I_B yields a solution to I_A with at least the success rate p.*

4 Connections to Existing CS Research

We count tokens rather than FLOPs for three reasons. First, **access**: FLOPs are measurable only for **open-source models**, whereas token usage is observable for any model through its API. Second, **structure**: a token trace reflects how a model decomposes a problem, backtracks, and elaborates, not just how much arithmetic it performs. Third, **stability**: per-token FLOPs vary with the inference stack, e.g., speculative decoding [44] and quantization [51], while token count does not and directly determines latency and cost. Since each token costs a bounded number of FLOPs, token complexity can still be related to classical complexity bounds, which motivates the following definition.

Definition 11 (Token-to-Operation Ratio and Effective Overhead)**.** *Let P be a problem with classical complexity $T_P(n)$, and let $\mathcal{M}$ solve instances of size n at success rate p^* using $\tau^{out}(n)$ output tokens, each requiring at most K floating-point operations. The* token-to-operation ratio *is $\rho(n) = \tau^{out}(n)/T_P(n)$, and the* effective computational overhead *is $K\rho(n)$.*

For sorting, $T_P(n) = n \log n$, so the token-to-operation ratio $\rho = \tau^{\text{out}}/(n \log n)$ counts tokens per comparison of the optimal classical algorithm, and the effective overhead $K\rho = \tau^{\text{out}} K/(n \log n)$ counts equivalent operations relative to that optimum. ρ is observable for any system, including API-only ones, whereas $K\rho$ requires per-token cost. Neither is needed to define token complexity; they serve as a bridge wherever a classical bound exists, and their growth in n shows whether an LM's gap to the optimal algorithm stays constant or widens with problem size. Unlike sample complexity and VC dimension for learning, token complexity measures the inference cost (Appendix F).

Takeaways

- **Token number is more stable than FLOPs** for comparing language models across different inference frameworks and optimization techniques, once the tokenizer is fixed in the system.
- **Token-to-operation ratio** bridges token complexity and classical algorithmic complexity.
- **Effective computational overhead** quantifies LM inefficiency relative to optimal algorithms.

5 Connections to Existing LLM Research

5.1 Scaling Laws for Training and Inference

Scaling laws represent one of the most successful theoretical frameworks in CSLM. Unlike traditional computer science, where algorithm complexity is determined by design and remains fixed regardless of implementations, LMs fundamentally depend on the amount of compute, data, and parameters.

Scaling Laws for Training. The pioneering work [40] established that language model performance, i.e., the test loss L, follows predictable power-law relationships with three key resources: the number of model parameters N, the size of training dataset D, and the amount of training compute C: $L(N, D, C) \sim N^{-\alpha_N} \sim D^{-\alpha_D} \sim C^{-\alpha_C}$ where $\alpha_N \approx 0.076$, $\alpha_D \approx 0.095$, and $\alpha_C \approx 0.050$ for sufficiently large values. These scaling laws revealed a critical insight: model performance is predictable across multiple orders of magnitude, enabling researchers to forecast the capabilities of future models before training them. Building on this foundation, the concept of *compute-optimal* training is introduced in [33] through the Chinchilla scaling laws, where for a given compute budget C, there exists an optimal allocation between model size N and training data D. Contrary to the prevailing approach of training increasingly large models on fixed datasets, Chinchilla scaling laws showed that model parameters and training tokens should scale proportionally: $N_{\text{opt}} \propto C^{0.5}$ and $D_{\text{opt}} \propto C^{0.5}$. These findings have profound implications for CSLM. We posit that token complexity may provide a more comprehensive characterization of training dynamics than loss alone, as lower training loss does not necessarily imply superior token complexity across all problem classes. We hypothesize that analogous scaling laws exist between token complexity and the triple (N, D, C) [38].

Scaling Laws for Inference. Recent work has revealed analogous scaling laws during inference, where allocating additional computational resources at test time yields predictable performance improvements. The test-time scaling inference strategies include various search methods, e.g., beam search [82] and Monte-Carlo Tree Search (MCTS) [92], Best-of-N (BoN) [50], and majority voting [11]. The inference scaling law discovered in [94] is that the number of the sampled solution and the success rate satisfy a power law, i.e., $p \sim (\tau^{\text{out}})^{\beta}$. This manifests as a fundamental tradeoff between success probability p and output token consumption τ^{out} in our token complexity framework. For a fixed model M and problem instance I, allocating more tokens through multiple sampling, CoT reasoning, or iterative refinement increases the success rate.

Takeaways
- **Token complexity** provides better characterization of training dynamics than loss alone.
- **Analogous scaling laws** may exist between token complexity and the triple (N, D, C).
- **Test-time scaling strategies** enable optimization of the success rate versus token tradeoff.

5.2 Memory and Context Management

Memory systems strongly affect the token complexity of LMSs. LMs use three memory tiers that mirror the classical memory hierarchy. **In-context memory** (the context window [12]) acts like a cache: it is small, and every generated token reads it, so its cost appears in τ^{in}. **External memory** (retrieval systems and document stores [45]) acts like disk: it is large, but each retrieval adds tokens, much like block transfers in the external-memory model [3]. **Internal memory** (knowledge stored in model weights [59, 68, 75]) has no classical counterpart: reading it costs no tokens, but it cannot be written at inference time and is unreliable (Section 2). A token complexity analysis should charge each tier separately, which turns choosing where to store information into an optimization problem.

We focus the remainder of our discussion on external memory systems. A predominant approach is Retrieval-Augmented Generation (RAG) [26, 45], where LMs first retrieve relevant information from external memory before generating responses [99, 104]. RAG affects token complexity through multiple mechanisms: (i) retrieval operations consume tokens for query formulation; (ii) retrieved content occupies context window space, increasing τ^{in}; and (iii) external memory can reduce hallucinations and improve factual accuracy, potentially increasing success rate p. We formalize the impact of memory systems through token complexity as $\mathsf{TokC}(I, \mathcal{M})$ and $\mathsf{TokC}(I, \mathcal{M} \oplus E)$, where $\mathcal{M} \oplus E$ denotes the LM system $\mathcal{M}$ augmented with external memory E. An emerging research direction is self-evolving memory [64, 84], where the LM's external memory E is autonomously updated for self-improvement across problems. This self-evolution introduces additional computational processes such as summarization and reflection [81]. The token complexity framework naturally extends to self-evolving memory by accounting for all tokens consumed during all stages of the system.

Takeaways
- **Three memory tiers form a hierarchy**: in-context memory behaves like registers and cache, external memory like disk, and each tier should be handled separately in token complexity.
- **Token complexity framework extends to external memory** by capturing all tokens consumed during retrieval, generation, and memory update processes.
- **Build optimal memory systems that minimize total token consumption** across retrieval operations, context window usage, and generation, while maximizing success rates.
- **Self-evolving memory** introduces additional computational processes (summarization, reflection) that must be accounted for in token complexity analysis.

5.3 Tool Use and Augmentation

Tool use distinguishes language models from traditional computing systems [71]. LMs learn it from in-context demonstrations [77], fine-tuning on tool-annotated data [66], or reinforcement learning on tool-interaction outcomes [72, 96]. In classical terms, a tool call is a subroutine call, or an oracle query in the sense of oracle machines: the caller pays for preparing the arguments and using the result, while the work inside the subroutine is accounted for separately. Token complexity only considers the orchestration: deciding when and which tool to call, formulating the call, and integrating its output. Delegation pays off when this cost is smaller than the tokens the model would otherwise spend computing the result itself. As with memory, we measure the effect by comparing $\mathsf{TokC}(I, \mathcal{M})$ with $\mathsf{TokC}(I, \mathcal{M} \oplus T)$, where $\mathcal{M} \oplus T$ denotes the LMS $\mathcal{M}$ augmented with tools T.

An emerging research direction is tool creation, where LMs not only use existing tools but generate new tools to solve problems more efficiently [14, 70]. This introduces a meta-level optimization problem: investing tokens upfront to create reusable tools that amortize their creation cost across multiple problem instances. For a tool creation task, the total token complexity across n similar problem instances $\{I_1, I_2, \ldots, I_n\}$ becomes: $\mathsf{TokC}_{\text{total}} = \mathsf{TokC}_{\text{create}}(t) + \sum_{i=1}^{n} \mathsf{TokC}(I_i, \mathcal{M} \oplus \{t\})$, where $\mathsf{TokC}_{\text{create}}(t)$ is the one-time cost of creating tool t. Tool creation is beneficial when this total cost is less than solving all instances without the custom tool: $\mathsf{TokC}_{\text{total}} < \sum_{i=1}^{n} \mathsf{TokC}(I_i, \mathcal{M})$. Therefore, tool creation becomes increasingly advantageous as the number of similar instances grows, thus significantly reduce the token complexity.

Takeaways
- **Tool calls are subroutine calls** that the system can author itself with token consumption
- **Tool use may fundamentally reduce token complexity** by delegating computationally intensive subtasks to specialized external tools with better performance and less token usage.
- **Tool creation amortizes cost across instances**. For n similar problem instances, the total token complexity is: $\mathsf{TokC}_{\text{total}} = \mathsf{TokC}_{\text{create}}(t) + \sum_{i=1}^{n} \mathsf{TokC}(I_i, \mathcal{M} \oplus \{t\})$ where $\mathsf{TokC}_{\text{total}} < \sum_{i=1}^{n} \mathsf{TokC}(I_i, \mathcal{M})$ indicate tool creation is beneficial if with the same success rate p.

5.4 Multi-Agent LLM Systems

Multi-agent LLM systems include multiple language models working collaboratively, which represent an important approach to optimizing token complexity by distributing computational workload across specialized models for solving sub-tasks more efficiently. One of the earliest and most influential multi-agent approaches is multi-agent debate [23, 103], where multiple LMs engage in iterative discussions to solve problems. In this paradigm, different models propose solutions, critique each other's reasoning, and refine their answers through multiple rounds of interaction. This approach leverages the principle that collective reasoning can improve solution quality. From a token complexity perspective, multi-agent debate explores a specific tradeoff: consuming additional tokens through multi-turn dialogue in exchange for higher success rates. Additional to the multi-agent debate, more sophisticate task decomposition and orchestration mechanisms are investigated, HuggingGPT [80] and Chameleon [57] introduced frameworks where a central LM acts as a controller, decomposing complex tasks into subtasks and routing them to specialized models or tools. MetaGPT [34] and GPTSwarm [110] further developed this concept by introducing role-based collaboration, where different LMs assume distinct roles (e.g., programmer, reviewer, project manager) and coordinate through structured workflows. Recent advances such as Chains of Agents (CoA) [107] and LatentMAS [111] explore even more efficient collaboration mechanisms. Rather than explicit communication, these systems enable models to collaborate through shared latent representations or implicit coordination, reducing the communication between agents while maintaining the benefits of multi-agent reasoning.

Takeaways
- **Multi-agent systems optimize token complexity** through task decomposition and specialization.
- **System-level Token complexity** depends on: i) decomposition strategy for sub-tasks, ii) model assignment for completing sub-tasks and iii) communication overhead between agents.

5.5 Fine-Tuning of Language Models

In the final stage, we analyze fine-tuning, i.e., how training tokens change what a model can solve. Training tokens have no classical counterpart. In classical computing, operations are fixed primitives and the algorithm is a free design variable. For an LM with a fixed tokenizer, the algorithm is fixed by its weights and can be changed only through training. Fine-tuning tokens are therefore elementary operations spent to rewrite the algorithm itself, after which the inference token complexity of the same problem changes. We define the problem instance I as "enabling the language model $\mathcal{M}$ to solve problem P' with success rate p," and let $\mathsf{TokC}(I, \mathcal{M})$ be the number of tokens $\mathcal{M}$ needs to learn to do so. This covers acquiring new capabilities in settings such as alignment [63], mathematical reasoning, and robotics [46, 56]. $\mathsf{TokC}(I, \mathcal{M})$ captures the full learning trajectory: the exploration cost of discovering better strategies (sample efficiency) and the rate of convergence to the target performance. The framework is agnostic to the training procedure. Supervised fine-tuning (SFT) [18, 105], reinforcement fine-tuning (RFT) [30, 63], and distillation [2, 48] differ only through the tokens they consume and the inference curves they produce. Their differences show up as differences in token cost: SFT learns from fixed expert demonstrations, whereas RFT learns from trajectories the model generates through environment interaction, so its token complexity also reflects an exploration-exploitation tradeoff. This view unifies learning and inference under one measure. To reach success rate p on P', one can spend training tokens to shift the inference curve, or spend more inference tokens on the original model; this is the allocation question $C = C_{\text{train}} + C_{\text{infer}}$ in Section 6. It also makes token complexity a measurable objective for guiding training and model design.

Takeaways
- **Token complexity captures the complete learning trajectory**, including i) exploration costs, ii) sample efficiency, and iii) convergence dynamics.
- **Fine-tuning** can be formalized as: Problem instance I = "enabling model $\mathcal{M}$ to solve problem P' with success rate p," where $\mathsf{TokC}(I, \mathcal{M})$ quantifies total tokens required for learning.
- **Reinforcement fine-tuning (RFT)** introduces the exploration-exploitation tradeoffs that must be accounted for in token complexity analysis.
- **Training tokens rewrite the algorithm**: unlike classical operations, they change the inference token complexity of a fixed problem, and the framework is agnostic to how they are spent.

6 Open Questions and Roadmap

6.1 Critical Open Questions

Complexity and Hardness. SAT is the canonical complete problem for NP, we need to identify analogous "LM-complete" problems, i.e., the problems that are maximally hard for language models. The key questions are: Do there exist problems that require exponential tokens for LMs to solve, but only polynomial tokens to verify? Can we establish a reduction framework where diverse hard problems can be systematically reduced to these LM-complete problems? Answering these questions would reveal the fundamental computational structure of language models and establish rigorous theoretical foundations analogous to classical complexity theory.

Training Dynamics and Model Improvement. What is the optimal compute allocation between training and inference: $C = C_{\text{train}} + C_{\text{infer}}$? How many tokens are required to acquire new capabilities through supervised or reinforcement fine-tuning? Can we characterize the exploration cost, sample efficiency, and convergence dynamics of RFT in terms of token complexity? Addressing these questions can transform LM development from empirical scaling to principled engineering.

Memory, Tools, and Multi-Agent Systems. How can we accurately measure the impact of memory, tools, and multi-agent systems on token complexity? What strategies enable optimal token complexity through these augmentation mechanisms? How can systems achieve self-improvement through iterative memory updates, dynamic tool creation and utilization, and collaborative multi-agent processes? Addressing these system-level questions is crucial for informing architectural decisions about whether to scale model parameters or augment models with external resources.

6.2 Call to Action: Proposed Roadmap (2026-2030)

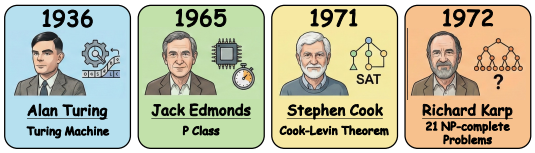


Figure 4: Major milestones of complexity theory

Drawing parallels to complexity theory's 40-year development, we propose a 5-year roadmap to establish CSLM as a rigorous scientific discipline. The first stage (2026) focuses on **foundations and formalization**, establishing rigorous mathematical definitions of complexity classes, efficiency, reduction mechanisms, and problem specifications. The second stage (2027–2028) advances **classification and structural theory**, identifying LM-complete problems across diverse task domains, proving first unconditional lower bounds, and forging connections to traditional complexity theory, information theory, and learning theory. The third stage (2029–2030) brings **maturity and integration**, proving major separation results, developing universal scaling law frameworks, and demonstrating concrete suggestions on LM development through training and inference optimization. The full roadmap is in Appendix K.

7 Alternative Views

We acknowledge that our position, i.e., LMs demand new computer science, is not universally accepted. Several alternative perspectives are discussed below and the full discussion is in Appendix L.

Token Counts Cannot Unify Heterogeneous Tasks. A token spent on prose and a token spent on a step of a symbolic derivation record different activities and two prompts of equal length, such as a request to summarize an article and a request to fact-check it, can demand work that differs significantly. This view may argue that a uniform token count cannot serve as a unified measure. We accept part of this objection. The token complexity describes what a given system spends on a given instance under a given criterion. Tokens and success rate are defined wherever a task can be evaluated, so the cost of a system on sorting and on summarization is expressible in the same two terms even though the tasks are different, similar to the time complexity compares sorting with matrix multiplication. Whether a uniform count is the right abstraction, or whether a count weighted by information content predicts better, is an empirical question that the framework makes testable. Predicting token complexity before a system is run, rather than measuring it afterward, remains open (Section 6), and we regard it as a central goal of CSLM rather than an accomplished result.

Traditional CS Already Encompasses LMs. The most common objection is that traditional CS already encompasses LMs because they are simply probabilistic programs or learned algorithms, and existing frameworks suffice. We acknowledge this perspective but maintain that the seven differences are substantial enough to demand specialized treatment. Quantum computing [62] develops its own subfield due to distinctive properties. As with quantum complexity, the new CSLM framework extends classical theory rather than contradicting it: where a classical bound exists, token complexity is related to it through the token-to-operation ratio (Definition 11), and what token complexity adds are the quantities classical complexity does not express for language models.

Physics of LMs Provides Sufficient Framework. Recent work on the "physics of language models" [5–8, 100, 101] offers valuable insights into training dynamics, knowledge storage, and learning mechanisms. We view CSLM as complementary rather than competing with this perspective. Physics of LMs focuses primarily on understanding model internals and training dynamics, the "how" of language model computation. CSLM addresses the relationship between problems and models, complexity characterization, and computational capabilities of LMs.

Cognitive Science Provides Adequate Frameworks. Some argue that cognitive science and neuroscience provide adequate frameworks for understanding LMs, particularly given parallels between human and machine intelligence. While these fields offer valuable insights, particularly regarding memory, reasoning, and knowledge representation, they focus on biological systems with fundamentally different constraints. Human brains operate with slow analog neurons and extreme energy constraints and LMs use fast digital computation with different scaling properties. Moreover, CSLM addresses computational properties specific to AI systems at scales beyond biological models.

We believe these alternative views, while raising important considerations, ultimately underestimate both the distinctiveness of LMs and the value of developing specialized theoretical frameworks. The field is at an inflection point where theoretical foundations will determine whether LM research remains empirical or becomes a rigorous science with predictive power and principled understanding. Further questions about the scope and use of the framework are answered in Appendix A.

Acknowledgments and Disclosure of Funding

The work described in this paper was substantially supported by a grant from the Innovation and Technology Commission of the Hong Kong Special Administrative Region, China (Project No. ITS/263/24FP). Xinrun Wang gratefully acknowledges support by the Lee Kong Chian (LKC) Fellowship awarded by Singapore Management University. This research is partially supported by the National Research Foundation, Singapore, under its SMART NATION & DIGITAL GOVERNMENT TRANSLATIONAL R&D GRANT (TRANS) PROPOSAL – AWARD NO. TRANS2023-TGC03. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore. Bo Li is supported in part by the Hong Kong SAR Research Grants Council (Grant No. PolyU 15224823) and the Guangdong Basic and Applied Basic Research Foundation (Grant No. 2024A1515011524).

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *ICLR*, 2024.

[3] Alok Aggarwal and S Vitter, Jeffrey. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[4] Alfred V Aho, John E Hopcroft, and Jeffrey D Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[5] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 1, learning hierarchical language structures. *arXiv preprint arXiv:2305.13673*, 2023.

[6] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.1, knowledge storage and extraction. *arXiv preprint arXiv:2309.14316*, 2023.

[7] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.2, knowledge manipulation. *arXiv preprint arXiv:2309.14402*, 2023.

[8] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.3, knowledge capacity scaling laws. *arXiv preprint arXiv:2404.05405*, 2024.

[9] Emmanuel Ameisen, Jack Lindsey, Adam Pearce, Wes Gurnee, Nicholas L Turner, Brian Chen, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, et al. Circuit tracing: Revealing computational graphs in language models. *Transformer Circuits Thread*, 6, 2025.

[10] Loïc Barrault, Paul-Ambroise Duquenne, Maha Elbayad, Artyom Kozhevnikov, Belen Alastruey, Pierre Andrews, Mariano Coria, Guillaume Couairon, Marta R Costa-jussà, David Dale, et al. Large concept models: Language modeling in a sentence representation space. *arXiv preprint arXiv:2412.08821*, 2024.

[11] Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

[12] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

[13] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv preprint arXiv:2303.12712*, 2023.

[14] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *ICLR*, 2024.

[15] Yixing Chen, Yiding Wang, Siqi Zhu, Haofei Yu, Tao Feng, Muhan Zhang, Mostofa Patwary, and Jiaxuan You. Multi-agent evolve: LLM self-improve through co-evolution. *arXiv preprint arXiv:2510.23595*, 2025.

[16] Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Yukun Li, et al. Conditional memory via scalable lookup: A new axis of sparsity for large language models. *arXiv preprint arXiv:2601.07372*, 2026.

[17] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[18] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *JMLR*, 25(70):1–53, 2024.

[19] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[20] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[21] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, 3rd edition, 2009.

[22] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[23] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *ICML*, pages 11733–11763, 2024.

[24] Yi Duan, Ying Liu, Zirui Tang, Haodong Chen, Jun Zhou, Yumou Liu, Bangrui Xu, Yukai Wu, Sidi Chen, Yuhan Zhou, et al. The last AI built by humans: Toward genuine recursive self-improvement. *arXiv preprint arXiv:2609.11873*, 2026.

[25] Oliver Eberle, Thomas McGee, Hamza Giaffar, Taylor Webb, and Ida Momennejad. Position: We need an algorithmic understanding of generative AI. *arXiv preprint arXiv:2507.07544*, 2025.

[26] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[27] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

[28] Lizhou Fan, Wenyue Hua, Lingyao Li, Haoyang Ling, and Yongfeng Zhang. NPHardEval: Dynamic benchmark on reasoning ability of large language models via complexity classes. In *ACL*, pages 4092–4114, 2024.

[29] Michael R Garey and David S Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[30] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[31] Juris Hartmanis and Richard E Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[32] Charles AR Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.

[33] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[34] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *ICLR*, 2024.

[35] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232*, 2023.

[36] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[37] Sukjun Hwang, Brandon Wang, and Albert Gu. Dynamic chunking for end-to-end hierarchical sequence modeling. *arXiv preprint arXiv:2507.07955*, 2025.

[38] Berivan Isik, Natalia Ponomareva, Hussein Hazimeh, Dimitris Paparas, Sergei Vassilvitskii, and Sanmi Koyejo. Scaling laws for downstream task performance in machine translation. In *ICLR*, 2025.

[39] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023.

[40] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[41] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[42] Jisu Kim, Youngwoo Shin, Uiji Hwang, Jihun Choi, Richeng Xuan, and Taeuk Kim. Memorization or reasoning? exploring the idiom understanding of LLMs. In *EMNLP*, pages 21689–21710, 2025.

[43] Donald E Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, 3rd edition, 1997.

[44] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *ICML*, pages 19274–19286, 2023.

[45] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *NeurIPS*, pages 9459–9474, 2020.

[46] Haozhan Li, Yuxin Zuo, Jiale Yu, Yuhao Zhang, Zhaohui Yang, Kaiyan Zhang, Xuekai Zhu, Yuchen Zhang, Tianxing Chen, Ganqu Cui, et al. SimpleVLA-RL: Scaling vla training via reinforcement learning. *arXiv preprint arXiv:2509.09674*, 2025.

[47] Xiaozhe Li, Xinyu Fang, Shengyuan Ding, Linyang Li, Haodong Duan, Qingwen Liu, and Kai Chen. NP-Engine: Empowering optimization reasoning in large language models with verifiable synthetic NP problems. *arXiv preprint arXiv:2510.16476*, 2025.

[48] Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huan-ang Gao, Wenkai Yang, Zhiyuan Liu, et al. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe. *arXiv preprint arXiv:2604.13016*, 2026.

[49] Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. In *ICLR*, 2024.

[50] Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. ZebraLogic: On the scaling limits of LLMs for logical reasoning. *arXiv preprint arXiv:2502.01100*, 2025.

[51] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100, 2024.

[52] Samuel Lippl, Thomas McGee, Kimberly Lopez, Ziwen Pan, Pierce Zhang, Salma Ziadi, Oliver Eberle, and Ida Momennejad. Algorithmic primitives and compositional geometry of reasoning in language models. *arXiv preprint arXiv:2510.15987*, 2025.

[53] Huanyu Liu, Jia Li, Hao Zhu, Kechi Zhang, Yihong Dong, and Ge Li. SATURN: SAT-based reinforcement learning to unleash language model reasoning. *arXiv preprint arXiv:2505.16368*, 2025.

[54] Junnan Liu, Hongwei Liu, Songyang Zhang, and Kai Chen. Rectifying LLM thought from lens of optimization. *arXiv preprint arXiv:2512.01925*, 2025.

[55] Junteng Liu, Yuanxiang Fan, Zhuo Jiang, Han Ding, Yongyi Hu, Chi Zhang, Yiqi Shi, Shitong Weng, Aili Chen, Shiqi Chen, et al. SynLogic: Synthesizing verifiable reasoning data at scale for learning logical reasoning and beyond. *arXiv preprint arXiv:2505.19641*, 2025.

[56] Guanxing Lu, Wenkai Guo, Chubin Zhang, Yuheng Zhou, Haonan Jiang, Zifeng Gao, Yansong Tang, and Ziwei Wang. VLA-RL: Towards masterful and general robotic manipulation with scalable reinforcement learning. *arXiv preprint arXiv:2505.18719*, 2025.

[57] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. In *NeurIPS*, pages 43447–43478, 2023.

[58] Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. On faithfulness and factuality in abstractive summarization. *arXiv preprint arXiv:2005.00661*, 2020.

[59] John X Morris, Chawin Sitawarin, Chuan Guo, Narine Kokhlikyan, G Edward Suh, Alexander M Rush, Kamalika Chaudhuri, and Saeed Mahloujifar. How much do language models memorize? *arXiv preprint arXiv:2505.24832*, 2025.

[60] Jinjie Ni, Qian Liu, Chao Du, Longxu Dou, Hang Yan, Zili Wang, Tianyu Pang, and Michael Qizhe Shieh. Training optimal large diffusion language models. *arXiv preprint arXiv:2510.03280*, 2025.

[61] Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

[62] Michael A Nielsen and Isaac L Chuang. *Quantum Computation and Quantum Information*. Cambridge university press, 2010.

[63] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *NeurIPS*, pages 27730–27744, 2022.

[64] Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T Le, Samira Daruki, Xiangru Tang, et al. ReasoningBank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025.

[65] Artidoro Pagnoni, Ramakanth Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason E Weston, Luke Zettlemoyer, et al. Byte latent transformer: Patches scale better than tokens. In *ACL*, pages 9238–9258, 2025.

[66] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive APIs. In *NeurIPS*, pages 126544–126565, 2024.

[67] Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is turing-complete. *Journal of Machine Learning Research*, 22(75):1–35, 2021.

[68] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? *arXiv preprint arXiv:1909.01066*, 2019.

[69] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

[70] Cheng Qian, Chi Han, Yi Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. Creator: Tool creation for disentangling abstract and concrete reasoning of large language models. In *Findings of EMNLP*, pages 6922–6939, 2023.

[71] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, et al. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40, 2024.

[72] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *ICLR*, 2024.

[73] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[74] Yi Ren and Danica J. Sutherland. Learning dynamics of LLM finetuning. In *ICLR*, 2025.

[75] Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? *arXiv preprint arXiv:2002.08910*, 2020.

[76] Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *NeurIPS*, pages 130136–130184, 2024.

[77] Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: language models can teach themselves to use tools. In *NeurIPS*, pages 68539–68551, 2023.

[78] Jeffrey Seely, Yuki Imajuku, Tianyu Zhao, Edoardo Cetin, and Llion Jones. Sudoku-Bench. `https://github.com/SakanaAI/Sudoku-Bench`, 2025.

[79] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge university press, 2014.

[80] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI tasks with ChatGPT and its friends in hugging face. In *NeurIPS*, pages 38154–38180, 2023.

[81] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *NeurIPS*, pages 8634–8652, 2023.

[82] Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *ICLR*, 2025.

[83] Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. REASONING GYM: Reasoning environments for reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2505.24760*, 2025.

[84] Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025.

[85] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2nd edition, 2022.

[86] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[87] Curt Tigges, Michael Hanna, Qinan Yu, and Stella Biderman. LLM circuit analyses are consistent across training and scale. In *NeurIPS*, pages 40699–40731, 2024.

[88] Kiran Tomlinson, Tobias Schnabel, Adith Swaminathan, and Jennifer Neville. Reasoning about reasoning: BAPO bounds on chain-of-thought token complexity in LLMs. *arXiv preprint arXiv:2602.02909*, 2026.

[89] Alan Turing. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

[90] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer Science & Business Media, 2013.

[91] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 30, 2017.

[92] Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. In *ICML*, pages 49890–49920, 2024.

[93] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022.

[94] Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for LLM problem-solving. In *ICLR*, 2025.

[95] Caijun Xu, Changyi Xiao, Zhongyuan Peng, Xinrun Wang, and Yixin Cao. SCALER: Synthetic scalable adaptive learning environment for reasoning. *arXiv preprint arXiv:2601.04809*, 2026.

[96] Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. SimpleTIR: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. *arXiv preprint arXiv:2509.02479*, 2025.

[97] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[98] Chang Yang, Ruiyu Wang, Junzhe Jiang, Qi Jiang, Qinggang Zhang, Yanchen Deng, Shuxin Li, Shuyue Hu, Bo Li, Florian T Pokorny, et al. Nondeterministic polynomial-time problem challenge: An ever-scaling reasoning benchmark for LLMs. *arXiv preprint arXiv:2504.11239*, 2025.

[99] Chang Yang, Xinrun Wang, Qinggang Zhang, Qi Jiang, and Xiao Huang. Efficient integration of external knowledge to llm-based world models via retrieval-augmented generation and reinforcement learning. In *Findings of EMNLP*, pages 9484–9501, 2025.

[100] Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of language models: Part 2.1, grade-school math and the hidden reasoning process. *arXiv preprint arXiv:2407.20311*, 2024.

[101] Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of language models: Part 2.2, how to learn from mistakes on grade-school math problems. *arXiv preprint arXiv:2408.16293*, 2024.

[102] Zhiyuan Zeng, Hamish Ivison, Yiping Wang, Lifan Yuan, Shuyue Stella Li, Zhuorui Ye, Siting Li, Jacqueline He, Runlong Zhou, Tong Chen, et al. RLVE: Scaling up reinforcement learning for language models with adaptive verifiable environments. *arXiv preprint arXiv:2511.07317*, 2025.

[103] Hangfan Zhang, Zhiyao Cui, Jianhao Chen, Xinrun Wang, Qiaosheng Zhang, Zhen Wang, Dinghao Wu, and Shuyue Hu. Stop overvaluing multi-agent debate – we must rethink evaluation and embrace model heterogeneity. *arXiv preprint arXiv:2502.08788*, 2025.

[104] Qinggang Zhang, Zhishang Xiang, Yilin Xiao, Le Wang, Junhui Li, Xinrun Wang, and Jinsong Su. FaithfulRAG: Fact-level conflict modeling for context-faithful retrieval-augmented generation. *arXiv preprint arXiv:2506.08938*, 2025.

[105] Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Guoyin Wang, et al. Instruction tuning for large language models: A survey. *ACM Computing Surveys*, 2023.

[106] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. Siren's song in the AI ocean: A survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*, 2023.

[107] Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Ö Arık. Chain of agents: large language models collaborating on long-context tasks. In *NeurIPS*, pages 132208–132237, 2024.

[108] Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335*, 2025.

[109] Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, et al. LLaDA 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*, 2025.

[110] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *ICML*, 2024.

[111] Jiaru Zou, Xiyuan Yang, Ruizhong Qiu, Gaotang Li, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, et al. Latent collaboration in multi-agent systems. *arXiv preprint arXiv:2511.20639*, 2025.

A Frequently Asked Questions (FAQs)

A.1 Will CSLM Replace Traditional CS?

NO. CSLM is a natural extension of traditional CS that addresses the fundamental differences between language models and conventional computers, not a replacement for it. The relationship is analogous to that between quantum computing theory and classical complexity theory: the former extends and enriches the latter without rendering it obsolete. Traditional CS remains essential for understanding deterministic computation on von Neumann architectures, algorithm design, and classical complexity. CSLM addresses the unique characteristics of language models that fall outside the conceptual vocabulary of classical frameworks. The two are synergistic: traditional CS informs CSLM's foundational definitions and proof techniques, while CSLM can guide next-generation LM system design with principled theoretical backing.

A.2 CSLM is Built on Tokens, Maybe Token is Unnecessary?

While most frontier LLMs employ token-based architectures that represent text as sequences of discrete tokens, recent research has explored alternative paradigms. Notable examples include Byte Latent Transformer (BLT) [65], which operates directly on byte-level representations, Large Concept Models (LCMs) [10], which process semantic concepts as fundamental units, and H-Net [37], which employs dynamic chunking to model sequences without traditional tokenization. While these approaches show promise for advancing language model research, they remain in nascent stages of development and cannot achieve competitive performance compared with token-based architectures. Consequently, we anticipate that token-based LLMs will continue to serve as the primary paradigm for LM research in the future, ensuring the relevance of our token complexity framework.

Moreover, our framework exhibits inherent generalizability: regardless of whether future models operate on bytes, tokens, concepts, or other linguistic units, the core principle remains applicable. As long as language plays the central role of study in LMs, we can define **minimal computational units** at the appropriate level of granularity, and our token complexity framework can be naturally adapted to accommodate these diverse architectures.

A.3 Can CSLM Extend to Diffusion LLMs (dLLMs)?

Yes. While autoregressive models remain dominant in the current LLM landscape, diffusion LLMs (dLLMs) have emerged as a promising generative paradigm that frames text generation as an iterative denoising process over discrete token sequences [60, 61, 76, 109]. Because dLLMs continue to operate on token-based representations, the token complexity framework extends to them naturally. The key adaptation concerns how tokens are counted: in autoregressive models, each forward pass generates one token, while in dLLMs, the generation process involves multiple denoising iterations over the full sequence. Token complexity for dLLMs should therefore account for the total number of tokens processed across all denoising steps, weighted by the number of iterations. This generalization preserves the core trade-off between token consumption and success rate, and opens new questions about how the number of denoising steps constitutes its own axis of computational cost, a direction we leave for future work.

A.4 BAPO

BAPO (Bounded Approximate Polynomial-time Oracle) bounds [88] provide a formal upper-bound framework for chain-of-thought token complexity in LLMs. BAPO characterizes the relationship between the length of a chain-of-thought reasoning trace and the probability of arriving at a correct answer, showing that for certain problem classes, the success rate achievable within a bounded token budget follows predictable decay laws as problem difficulty scales. This connects directly to the token complexity curve $\mathsf{TC}(I, M)$. Specifically, BAPO bounds instantiate the upper-boundary of the TC curve: they characterize the maximum success rate achievable at a given token budget, given the structural difficulty of the problem.

While both frameworks address the fundamental question of how much computation LLMs require, the CSLM token complexity framework offers several advantages over BAPO. BAPO models transformer computation through prefix bandwidth a and attention bandwidth b, proving tight lower

bounds such as $\Omega(n)$ CoT tokens for MAJORITY, MATCH3_n, and REACHABILITY. However, BAPO is tied to the internal information-flow structure of causal transformers and requires white-box access to the model. By contrast, token complexity

$$\mathsf{TC}(I, \mathcal{M}) = \big\{(\tau, p) : p \in [0, 1]\big\}$$

treats the LM system $\mathcal{M}$ as a black box, making it applicable uniformly to both open-source and proprietary models. Crucially, BAPO operates under a worst-case correctness criterion, whereas token complexity is a curve explicitly trading off token budget τ against instance-specific success rate $p \in [0, 1]$, which is more faithful to the probabilistic nature of LLMs and naturally captures inference-time scaling phenomena of the form $p \sim (\tau^{\text{out}})^{\beta}$. Furthermore, CSLM generalizes beyond single-model CoT to full LM systems augmented with external memory E and tools T, formalizing augmentation economics such as $\mathrm{TC}(I, \mathcal{M} \oplus T) < \mathrm{TC}(I, \mathcal{M})$ and tool creation amortization

$$\mathsf{TC}_{\text{create}}(t) + \sum_{i=1}^{n} \mathsf{TC}(I_i, \mathcal{M} \oplus \{t\}) < \sum_{i=1}^{n} \mathsf{TC}(I_i, \mathcal{M}),$$

while also connecting token complexity to training dynamics and scaling laws over the triple (N, D, C), which are the dimensions entirely outside BAPO's scope. The two frameworks are nonetheless complementary: BAPO results serve as concrete instantiations of the information-theoretic lower bounds that CSLM predicts must exist, but CSLM provides the broader theoretical vocabulary for a full complexity science of language models.

A.5 Why Should CSLM Emerge as a New Subfield?

The emergence of a new theoretical subfield in computer science follows a recognizable pattern. Computability theory, established by Turing's formalization of the Turing machine [89], gave theorists a model of what could be computed, but said nothing about efficiency. That gap motivated Hartmanis and Stearns [31] to introduce time-complexity classes and the time hierarchy theorem in 1965. Cook [19] and Karp [41] then identified NP-complete problems defining the boundary of tractability.

The same pattern recurred with quantum computing: despite being technically realizable on a Turing machine via simulation, quantum computation differs significantly from classical computation (superposition, entanglement, interference-based algorithms). These differences spawned dedicated theoretical fields, i.e., quantum complexity classes, quantum error correction, and the BQP vs. NP question [62], which would be incoherent to express solely in classical terms.

Statistical learning theory provides a third example: classical complexity theory can describe whether a learning algorithm runs in polynomial time, but it cannot characterize sample complexity, i.e., how many training examples suffice to guarantee generalization. That gap required the PAC learning framework and VC dimension, tools that classical complexity theory has no analogue for.

In each case, a new subfield was warranted not because the prior framework was wrong, but because its conceptual vocabulary was insufficient to express the questions practitioners were already asking. We argue that CSLM stands in the same relation to classical CS: the questions "How many tokens does this problem require?" and "What makes a problem LM-complete?" cannot be formulated in traditional complexity theory. Thus, a new computer science is demanded.

A.6 Code

We release the anonymous code at `https://github.com/joannacyang/cslm_mini`.

A.7 AI Usage

AI tools (e.g., Claude and ChatGPT) are used to polish the texts. We use Nano Banana to generate the icons in the illustrative figures for better visualization and the full figures are organized manually.

B Preliminary Results on Sorting

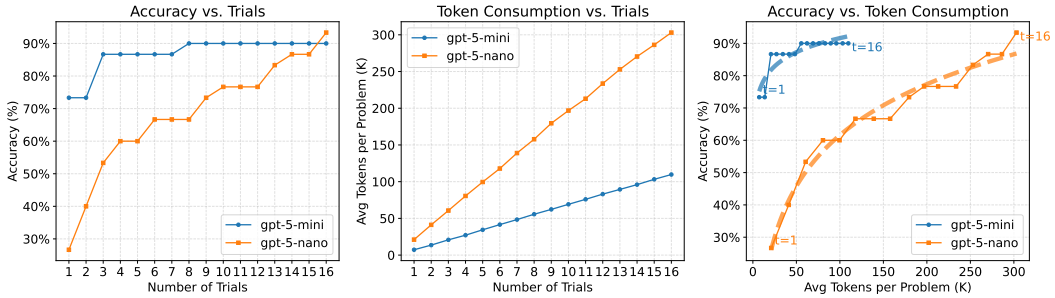


Figure 5: Preliminary results on Sorting with $n = 120$ over 30 instances

This section presents the preliminary results of sorting problems, serving as a concrete instantiation of the token complexity framework in this work.

Experimental Setup. We evaluate two frontier language models, `gpt-5-mini` and `gpt-5-nano`, on sorting instances of $n = 120$ elements. Each model is queried up to $t = 16$ independent trials per instance, with success measured by exact correctness of the output permutation. We report three complementary views of the token complexity curve $\mathrm{TC}(I, \mathcal{M})$: (i) accuracy as a function of the number of trials t, (ii) average token consumption (in thousands) as a function of t, and (iii) the joint accuracy-versus-token-consumption curve, which directly visualizes the (τ, p) trade-off.

Results and Analysis. Figure 5 reveals several findings consistent with the CSLM framework.

Inference scaling follows a predictable curve. Both models exhibit monotonically increasing accuracy as the number of trials grows, consistent with the inference scaling law $p \sim (\tau^{\text{out}})^{\beta}$ discussed in Section 5.1. `gpt-5-mini` achieves higher accuracy in the first trial budgets, reaching approximately 90% at $t = 16$, while `gpt-5-nano` finally catches up and achieves 93.3%. This confirms that the token complexity curve is a meaningful, model-specific characterization: the two models occupy distinct positions in the (τ, p) space.

Token consumption scales linearly with trials. The middle panel shows that average token consumption grows approximately linearly with the number of trials for both models. Notably, `gpt-5-nano` consumes substantially more tokens per trial than `gpt-5-mini` (reaching ≈ 280K vs. ≈ 110K tokens at $t = 16$). This illustrates the tension between model capability and token efficiency: a more capable model with higher prices may consume less tokens and less capable models may achieve the performance with much more tokens.

The accuracy-vs-token frontier differs qualitatively across models. The right panel, which plots the token complexity curve directly, shows that `gpt-5-mini` achieves higher accuracy per token at low budgets (steep initial rise from $t = 1$), suggesting that its first trial is already of high quality. `gpt-5-nano` requires a larger relative increase in token budget to achieve comparable accuracy gains. This qualitative difference is precisely what the system card $\mathrm{SC}(\mathcal{M})$ is designed to capture: a concise characterization of a model's efficiency profile across problem instances.

Connections to Token Complexity Theory. These results provide early empirical support for three theoretical claims. First, the token complexity of sorting under repeated sampling is not a scalar but a curve, validating Definition 4. Second, the power-law relationship between trials and success rate (Section 5.1) is visible in the log-linear structure of the accuracy-vs-trials plot. Third, the gap between `gpt-5-mini` and `gpt-5-nano` illustrates that $\mathrm{TC}(I, \mathcal{M})$ is model-dependent in a structured way, motivating the problem card $\mathrm{PC}(P)$ as a distribution over model-specific curves rather than a single complexity value.

C More Discussion about Token Operations

Tokenization is part of the system. The token is not a unit that exists independently of the model: it is defined by the model's tokenizer, which maps text and, for multimodal models, images, audio, and video to sequences over a vocabulary. We therefore treat the tokenizer as part of the LM system (Definition 3), so that token complexity is well defined for any fixed system by construction. Variable tokenization then becomes an axis of analysis rather than a nuisance to be managed. The vocabulary determines how much of the output is decided at each sampling step: a coarser vocabulary emits more bytes per step, which reduces the number of steps but requires each sampling decision to resolve more uncertainty. How token complexity varies with vocabulary granularity, and how compression trades against the difficulty of each sampling decision, are questions the framework makes well posed. For the purpose of measurement, the variation across mainstream tokenizers is bounded: re-encoding the same output under a different byte-level tokenizer changes the token count by at most a constant factor, and on English text eight widely deployed tokenizers agree to within a factor of 1.13 (Appendix M).

Tokens are heterogeneous. A single token can vary substantially in its arithmetic cost and information content, and a token spent on prose differs from a token spent on a step of a symbolic derivation. A related concern arises in classical complexity theory, where a single "operation" abstracts over vastly different real costs (e.g., a cache hit vs. a cache miss, or an integer addition vs. a floating-point division), while all are counted as $O(1)$ under the RAM model. The power of such abstraction lies not in capturing constant-factor variation but in revealing structural scaling behavior. Token complexity adopts the same philosophy: it characterizes how the number of autoregressive generation steps scales with problem difficulty, which is the quantity most directly governing both success probability and deployment cost, while deliberately abstracting over per-token heterogeneity. The abstraction does not assert that tokens are equal in information content; whether a count weighted by information content predicts behavior better than a uniform count is an empirical question that we return to in Section 7.

Token sequences are non-commutative. Two traces of equal length can perform different computations, just as a specific sequence of FLOPs carries more information than the FLOP count. Token complexity counts generation steps, while the order and structure of the trace, i.e., how a model decomposes a problem, backtracks, and elaborates, remain objects of study in their own right.

D Guidance of Applying CSLM to Your Problems

This appendix provides practical guidance for researchers and practitioners who wish to apply the CSLM framework to a new problem domain. We organize the guidance around three sequential steps: (1) problem specification, (2) token complexity measurement, and (3) system optimization.

D.1 Step 1: Problem Specification

The first step is to formalize the target problem within the CSLM framework.

Define the problem and its instances. Identify the problem $P \in \mathcal{P}$ you wish to study and specify the space of valid instances $\mathcal{I}_P$. For well-studied combinatorial problems (SAT, TSP, graph coloring), canonical instance generators already exist; for novel tasks, the researcher must define a parameterized generator that controls instance size $n = |I|$ and difficulty.

Define success. Specify a verifiable success criterion and a target success rate $p^* \in (0, 1]$. The choice of p^* is application-dependent: for safety-critical deployments, $p^* = 0.99$ may be required; for exploratory tasks, $p^* = 0.8$ may suffice. The token complexity framework is defined relative to this threshold (Definition 6), so the choice should be made explicit and justified.

Specify the LMS. Define the language model system $\mathcal{M}$, including the base model(s), any augmentations (retrieval system E, tool set T, or multi-agent architecture), and the inference strategy (single-pass, chain-of-thought, majority voting, or MCTS). Each configuration defines a distinct point in the (τ, p) space.

D.2 Step 2: Token Complexity Measurement

Once the problem and system are specified, the token complexity curve $\mathrm{TC}(I, \mathcal{M})$ can be estimated empirically.

Sample instances across scales. Generate instances at multiple sizes $n_1 < n_2 < \cdots < n_k$ to enable asymptotic analysis. We recommend at least five size points spanning at least one order of magnitude (e.g., $n \in \{10, 20, 50, 100, 200\}$ for combinatorial problems).

Measure the (τ, p) trade-off. For each instance I and system $\mathcal{M}$, vary the token budget by adjusting the number of trials t (repeated sampling), the chain-of-thought budget, or the beam width. Record both the total token consumption $\tau^{\mathrm{out}}(I, \mathcal{M}, p)$ and the observed success rate $\hat{p}$ across a sufficiently large number of independent runs (we recommend ≥ 30 runs per configuration for reliable estimates).

Fit asymptotic scaling. Plot τ^{out} against n on a log-log scale. A linear fit in log-log space indicates power-law scaling $\tau^{\mathrm{out}} = O(n^{\alpha})$, allowing the asymptotic token complexity class to be estimated. Deviations from linearity may indicate phase transitions or regime changes in model behavior.

Construct the problem card. Aggregate $\mathrm{TC}(I, \mathcal{M})$ curves across multiple systems $\mathcal{M}_1, \mathcal{M}_2, \ldots$ to construct the problem card $\mathrm{PC}(P)$ (Definition 5). This provides a comparative view of how different systems handle the problem and reveals whether the problem is uniformly hard across models or model-dependent in its difficulty structure.

D.3 Step 3: System Optimization

The token complexity framework suggests three concrete optimization directions.

Augmentation analysis. Compare $\mathrm{TC}(I, \mathcal{M})$ against $\mathrm{TC}(I, \mathcal{M} \oplus E)$ and $\mathrm{TC}(I, \mathcal{M} \oplus T)$ to quantify the token-efficiency gain from retrieval and tool use. If the augmented system achieves the same success rate p^* with fewer tokens, augmentation is beneficial. If it requires more tokens for marginal accuracy gains, the overhead may not be warranted.

Multi-agent decomposition. If the problem admits natural sub-task decomposition, evaluate whether a multi-agent architecture reduces total token consumption relative to a single-model baseline. Decomposition is beneficial when the sum of sub-task token complexities is less than the monolithic token complexity: $\sum_k \mathrm{TC}(I_k, \mathcal{M}_k) < \mathrm{TC}(I, \mathcal{M})$.

Tool creation. If the same problem class will be solved many times, evaluate the tool creation economics. Tool creation is warranted when

$$\mathrm{TC}_{\mathrm{create}}(t) + \sum_{i=1}^{n} \mathrm{TC}(I_i, \mathcal{M} \oplus \{t\}) < \sum_{i=1}^{n} \mathrm{TC}(I_i, \mathcal{M}),$$

i.e., the one-time creation cost is amortized over sufficiently many instances.

D.4 Checklist

We provide a concise checklist for applying CSLM to a new problem.

1. **Specify the problem.** Define P, $\mathcal{I}_P$, the instance generator, and the success criterion.
2. **Fix the success threshold.** Choose p^* appropriate for the application context.
3. **Define the baseline LMS.** Specify the base model, inference strategy, and any augmentations.
4. **Generate instances at multiple scales.** Ensure sufficient range for asymptotic analysis.
5. **Measure the token complexity curve.** Record $(\tau, \hat{p})$ pairs across trial budgets.
6. **Fit asymptotic token complexity.** Estimate $O(f(n))$ from empirical scaling plots.
7. **Construct the system card and problem card.** Aggregate across systems and instances.
8. **Evaluate augmentation strategies.** Compare TC with and without retrieval, tools, and multi-agent coordination.
9. **Report the full TC curve**, not just a scalar accuracy, to enable comparison across models and configurations.

D.5 Common Pitfalls

Reporting scalar accuracy instead of the TC curve. A single accuracy number conflates token budget and model capability. Always report the full (τ, p) curve or, at minimum, accuracy at multiple token budgets.

Conflating input and output tokens. $\mathrm{TC}^{\mathrm{in}}$ and $\mathrm{TC}^{\mathrm{out}}$ have different cost structures and scaling behaviors. Report them separately, particularly when comparing RAG-augmented systems (which increase τ^{in} through retrieved context) against tool-augmented systems (which may reduce τ^{out} by delegating computation).

Ignoring the success-rate threshold. Token complexity is defined relative to p^*. A system that achieves $p = 0.95$ in 1000 tokens is not comparable to one that achieves $p = 0.70$ in 500 tokens without specifying the target threshold.

Insufficient instance diversity. Token complexity can vary substantially across instances of the same size due to structural variation (e.g., satisfiable vs. unsatisfiable SAT instances). Average over a representative sample of at least 30 instances per size point, and report the token consumption.

E Summary of Takeaways

We present a summary of the main takeaways of this paper in this section.

- **Token number is more stable than FLOPs** for comparing language models across different inference frameworks and optimization techniques.
- **Token-to-operation ratio** bridges token complexity and classical algorithmic complexity.
- **Effective computational overhead** quantifies LM inefficiency relative to optimal algorithms.
- **Token complexity provides a more comprehensive characterization** of training dynamics than loss alone.
- **Analogous scaling laws** may exist between token complexity and the triple (N, D, C), enabling prediction of model efficiency before training.
- **Test-time scaling strategies** enable optimization of the success rate versus token tradeoff.
- **Multi-agent systems optimize token complexity** through task decomposition and specialization (fewer tokens, higher success rates).
- **System-level Token complexity** depends on: i) decomposition strategy, ii) model assignment and iii) communication overhead.
- **Token complexity framework extends to external memory** by capturing all tokens consumed during retrieval, generation, and memory update processes.
- **Build optimal memory systems that minimize total token consumption** across retrieval operations, context window usage, and generation, while maximizing factual accuracy and reducing hallucinations.
- **Self-evolving memory** introduces additional computational processes (summarization, reflection) that must be accounted for in token complexity analysis.
- **Token complexity framework extends to external memory** by capturing all tokens consumed during retrieval, generation, and memory update processes.
- **Build optimal memory systems that minimize total token consumption** across retrieval operations, context window usage, and generation, while maximizing factual accuracy and reducing hallucinations.
- **Self-evolving memory** introduces additional computational processes (summarization, reflection) that must be accounted for in token complexity analysis.
- **Tool use may fundamentally reduce token complexity** by delegating computationally intensive subtasks to specialized external tools, minimizing token generation for tasks better handled procedurally.
- **Tool creation amortizes cost across instances**. For n similar problem instances, the total token complexity is: $\text{TC}_{\text{total}} = \text{TC}_{\text{create}}(t) + \sum_{i=1}^{n} \text{TC}(I_i, \mathcal{M} \oplus \{t\})$ where $\text{TC}_{\text{total}} < \sum_{i=1}^{n} \text{TC}(I_i, \mathcal{M})$ indicate tool creation is beneficial if with the same success rate p.
- **Token complexity captures the complete learning trajectory**, including i) exploration costs, ii) sample efficiency, and iii) convergence dynamics.
- **Fine-tuning for capability acquisition** can be formalized as: Problem instance I = "enabling model $\mathcal{M}$ to solve problem P' with success rate p," where $\text{TC}(I, \mathcal{M})$ quantifies total tokens required for learning.
- **Reinforcement fine-tuning (RFT)** introduces the exploration-exploitation tradeoffs that must be accounted for in token complexity analysis.

F Relationships to Established Theoretical Fields

We now articulate precise relationships between CSLM and three established theoretical fields, clarifying why token complexity is *complementary* rather than subsumed by these frameworks.

F.1 Statistical Learning Theory

Statistical learning theory (SLT) [79, 90] studies learnability: given a hypothesis class and data distribution, what can be learned and at what sample complexity? SLT asks: "How many training examples suffice to guarantee generalization?" Token complexity asks a fundamentally different set of questions: given a *trained* model operating as a token-generating system, how does the token cost of solving a problem scale with instance structure? How do augmentations (tools, memory, multi-agent coordination) alter this cost?

To illustrate concretely: SLT can tell us that a transformer trained on enough sorting examples will generalize (sample complexity). It does not tell us how many output tokens GPT-4o needs to sort a list of size 50 vs. 500, whether that scaling is $O(n)$ or $O(n^2)$ in tokens, or whether giving the model a code interpreter changes this scaling. These are the questions token complexity is designed to answer.

Furthermore, recent *test-time scaling*, i.e., solving problems without changing model parameters by allocating more inference tokens, is naturally described by the token complexity framework but falls outside the scope of SLT. The closest classical analogue to token complexity is *query complexity* in the oracle computation model: token complexity measures the number of "queries" (token generation steps) the model issues to produce a solution, analogous to how query complexity counts oracle calls rather than internal computation steps.

F.2 Information Theory

Shannon's information theory provides a natural *lower-bounding framework* for token complexity. The entropy $H(S \mid I)$ of a solution S given problem instance I characterizes the irreducible uncertainty an agent must resolve to answer correctly. Since each output token from a vocabulary of size V carries at most $\log_2 V$ bits, this immediately yields an information-theoretic lower bound on the output token count:

$$\tau_{out}(I, \mathcal{M}, p) \geq \frac{H(S \mid I)}{\log_2 V} \cdot c(p)$$

where $c(p)$ is a correction factor that increases as $p \to 1$ (communicating a solution correctly with high probability requires more information than with moderate probability). This connection formalizes the intuition that *token complexity is bounded below by the information content of the problem*, and suggests that token efficiency is intimately related to how compactly an LM can represent and communicate solution information.

F.3 Linguistics and Compositionality

Linguistics offers a complementary lens on why token complexity varies across problem classes. The principle of *compositionality*—the meaning of a complex expression is determined by the meanings of its parts and the rules of their combination—predicts that problems with hierarchically compositional structure should be more tractable for LMs trained on natural language.

This suggests that *compositional token complexity*—the minimum token cost when the LM is allowed to decompose a problem into linguistically natural sub-problems—may be significantly lower than the flat token complexity for a broad class of structured reasoning tasks. Understanding when and why LMs exploit compositional structure is therefore a key question for CSLM.

Furthermore, tokens are not merely computational primitives but units with varying information content across languages and domains. Tokenizer convergence (BPE-based tokenizers trained on large multilingual corpora, e.g., tiktoken, SentencePiece) reduces the practical severity of cross-lingual variance, much as ASCII and UTF-8 became standards for character encoding. The per-token heterogeneity in information content is analogous to heterogeneity in classical complexity models (e.g., cache-hit vs. cache-miss both counting as $O(1)$), and does not undermine the scaling abstraction.

G Comparison between Traditional CS and CSLM

Table 1: Comprehensive Comparison between Traditional Computer Science and Computer Science of Language Models.

	Traditional CS	**CSLM**
Core Paradigm	Solving-verifying dichotomy: distinction between solving problems and verifying solutions	Generating-solving-verifying triad: addition of problem generation as a fundamental computational operation
Basic Operation	Computational operations (arithmetic, comparisons, memory access)	Token operations (generation of discrete symbols from vocabulary)
Reliability Model	Deterministic correctness (correct if programmed correctly, bugs can be eliminated)	Probabilistic reliability (can hallucinate or make errors even when correctly prompted; errors are fundamental)
Data Modality	Primarily text and numerical data; pixel-level image processing without conceptual understanding	Native multimodal understanding (text, audio, image, video) with conceptual comprehension across modalities
Problem Scope	Problem-specific algorithms designed for particular tasks	General-purpose models capable of solving diverse problems without task-specific programming
Stages	Inference stage (algorithm execution)	Two stages: training (learning from data) and inference (problem solving)
Architecture	Hardware evolution (transistors, integrated circuits, parallel processors) as distinct from algorithms	Recurrent Neural Networks (RNNs), transformers, diffusion models
Training Analogy	N/A, or chip design, FPGA configuration, programming language design	Scaling laws, emergence, optimization dynamics
Memory Model	Explicit memory hierarchy (registers, cache, RAM, disk) with deterministic access	Parameters (long-term), context window (working memory), external retrieval (augmented memory)
Tool Usage	Programs are tools, where composition through explicit interfaces	Native tool use and orchestration, and meta-reasoning about when and how to use tools
Knowledge Source	Algorithms encode knowledge explicitly through code	Knowledge emerges from training data and is encoded implicitly in parameters
Generalization	Algorithms for specific problems and generalization through algorithm design	Models generalize across diverse tasks and compositional reasoning

H Related Work

Benchmarking LLMs on NP(C) Problems. Traditional computer science provides principled guidance for LLM development through both benchmarking and training methodologies. NPHardEval [28] evaluates LLMs across three computational complexity classes, i.e., P, NP-complete, and NP-hard with representative problems from each class. ZebraLogic [50] employs the Zebra puzzle, an NP-complete logic problem, to assess language model reasoning capabilities as problem complexity scales. Sudoku-Bench [78] provides 2,765 procedurally generated Sudoku instances across varying difficulty levels for systematic evaluation. Reasoning Gym [83] is an ongoing initiative that collects procedural generators and algorithmic verifiers to produce infinite training data with adjustable complexity. NPPC [98] encompasses 25 NP-complete problems with functionality to generate problem instances of increasing complexity and scale.

NP(C) Problems for Improving LLMs. Recent work explores NP(C) problems as training environments to improve the general reasoning capabilities of LLMs [47, 53, 102]. SATURN [53] utilizes SAT problems to generate training data that enhances LLM reasoning abilities. NP-Engine [47] curates 10 NP-hard tasks from five domains with controllable instance generators and heuristic solvers, which are employed to strengthen LLM performance on NP-hard tasks. SynLogic [55] leverages logical tasks, e.g., Sudoku, to improve general logical and mathematical reasoning capabilities in LLMs. Both RLVE [102] and SCALER [95] incorporate NP(C) problems for advancing LLM reasoning through adaptive difficulty calibration.

Algorithmic Understanding of LLMs. Researchers have pursued algorithmic understandings of LLM capabilities. Theoretical analyses have established that attention mechanisms are Turing complete [67] and chain-of-thought (CoT) reasoning can substantially enhance model expressivity through circuit complexity [49]. AlgEval [25] provides a systematic framework for investigating which algorithms LLMs can learn and deploy. The connection between reasoning traces and internal activations has been explored [52], where researchers identify algorithmic primitives and their composition in complex reasoning tasks. Circuit analysis techniques [9, 87] have also been employed to trace LLMs' internal computational behaviors.

I Computational Complexity: P, NP, NP-complete and NP-hard

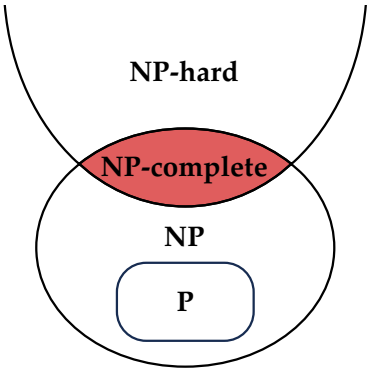


Figure 6: The relation between P, NP and NP-complete

P. The class P consists of decision problems that can be solved by a deterministic Turing machine in polynomial time. In practical terms, these are problems for which efficient algorithms exist. The time required to solve these problems grows polynomially with the input size (n), such as $O(n)$, $O(n^2)$, or $O(n^3)$. Examples include sorting, searching in a sorted array, and determining if a number is prime.

NP. NP contains all decision problems for which a solution can be verified in polynomial time. Every problem in P is also in NP, but NP may contain problems that are not in P. The key characteristic is that if someone gives you a potential solution, you can quickly check whether it's correct, even if finding that solution might be difficult. Examples include the Boolean satisfiability problem and the Traveling Salesman decision problem.

NP-complete (NPC). NP-complete problems are the "hardest" problems in NP. A problem is NP-complete if: i) It belongs to NP, ii) Every other problem in NP can be reduced to it in polynomial time. This means that if an efficient (polynomial-time) algorithm were found for any NP-complete problem, it could be used to solve all problems in NP efficiently. The first proven NP-complete problem was the Boolean satisfiability problem (SAT). Other examples include the Traveling Salesman Problem, Graph Coloring, and the Knapsack Problem. The question of whether P=NP (whether every problem with efficiently verifiable solutions also has efficiently computable solutions) remains one of the most important open questions in computer science.

NP-hard Problems. A problem is NP-hard if every problem in NP can be reduced to it in polynomial time, but unlike NPC problems, NP-hard problems need not be in NP themselves (i.e., they need not be decision problems or have efficiently verifiable solutions). This makes NP-hard a broader class that includes optimization versions of NPC problems (e.g., finding the minimum vertex cover rather than deciding if one of size k exists), as well as problems strictly harder than NP (e.g., certain problems in PSPACE or EXPTIME). In practice, many real-world applications involve NP-hard optimization problems where the goal is to find optimal or near-optimal solutions rather than simply verify their existence.

J Brief History of Complexity Theory

Computational complexity theory is the systematic study of the computational resources for solving computational problems. Emerging in the 1960s as a formal framework for the fundamental limits of computation, it has become one of the central areas of theoretical computer science.

Early Foundations (1930s–1950s). The foundations of complexity theory can be tracked to seminal work on computability in the 1930s. Turing machine is introduced as a formal model of computation in 1936 [89], establishing the Church-Turing thesis: any effectively calculable function can be computed by a Turing machine. This provided the essential mathematical framework for subsequent complexity-theoretic investigations. During this foundational period, research focused primarily on computability, i.e., determining which functions could be computed, rather than computational efficiency. These work established the fundamentals for understanding computation as a resource-bounded process.

Formalization of Complexity Theory (1960s). The systematic study of computational complexity began in the 1960s with several landmark contributions. Juris Hartmanis and Richard E. Stearns [31] published their foundational paper "On the Computational Complexity of Algorithms", which introduces time complexity classes and proved the time hierarchy theorem, demonstrating that additional computational time strictly expands the class of solvable problems. Concurrently, Jack Edmonds [27] proposed that polynomial-time solvability characterizes "efficient" or "tractable" computation. Edmonds also notably argued that Problems that can be solved in polynomial time are considered to have "good algorithms", thereby establishing the fundamental importance of the complexity class P.

Emergence of Complexity Classes (1970s). The 1970s witnessed an explosion of complexity classes and the elucidation of relationships among them. In 1971, Stephen Cook [19] established that the Boolean satisfiability problem (SAT) is NP-complete, i.e., the maximally difficult problem within NP. Independently, Leonid Levin obtained analogous results in the Soviet Union in 1973. This groundbreaking work introduced the theory of NP-completeness and revealed that numerous practical computational problems share identical fundamental complexity. Richard M. Karp [41] subsequently demonstrated that 21 diverse problems spanning graph theory, combinatorial optimization, and other domains are NP-complete, establishing both the universality and practical significance of NP-completeness.

The development of complexity theory offers valuable insights for advancing the CSLM. This progression suggests a three-stage research agenda: first, we must establish a rigorous definition of "efficiency" within the CSLM framework; second, we need to characterize the class of computationally hard problems for language models, and finally we should investigate the structural relationships between problem classes to achieve a comprehensive understanding of the computational landscape of language model capabilities and the intrinsic structures of problems.

K Details of Proposed Roadmap (2026-2030)

We elaborate on the three-stage roadmap proposed in Section 6.2, drawing parallels to the development of classical complexity theory from Turing's foundational model (1936) through Cook–Levin (1971) and Karp's 21 NP-complete problems (1972).

Stage 1 (2026): Foundations and Formalization. The first stage establishes the mathematical bedrock of CSLM, analogous to Hartmanis and Stearns [31] formalizing time complexity classes before any specific hard problem had been identified. The theoretical priorities are: finalizing the token complexity framework as a community standard, including conventions for counting tokens across system prompts, retrieved context, and chain-of-thought traces; stabilizing the definitions of LM-P, LM-NP, and LM-Complete and their sensitivity to the success-rate threshold p^*; and proving the first information-theoretic lower bounds on $\tau^{\text{out}}(I, \mathcal{M}, p)$ for natural problem classes. Empirically, an initial benchmark suite should cover canonical problem classes with asymptotic instance generators (e.g., sorting, SAT, graph coloring, TSP, and knapsack) with baseline token complexity measurements for frontier models.

Stage 2 (2027–2028): Classification and Structural Theory. The second stage identifies LM-complete problems across diverse domains, develops the token-efficient reduction framework into a working proof methodology, and forges connections to established theory. Analogous to Cook's theorem [19] and Karp's 21 reductions [41], the central goal is to identify a problem P^* to which every problem in LM-NP is token-efficiently reducible, or to prove that no such problem exists, which is itself a foundational result. Beyond LM-completeness, Stage 2 should prove unconditional lower bounds reflecting the structural constraints of autoregressive generation, extend token complexity to multimodal instances (Difference §4) and reinforcement fine-tuning dynamics, and formalize the connection between token complexity and sample complexity from statistical learning theory. A systematic reduction framework covering 25 or more problem classes would establish the universality and practical significance of the LM-complete concept.

Stage 3 (2029–2030): Maturity and Integration. The third stage proves major separation results or establishes barrier theorems, develops universal scaling law frameworks, and demonstrates concrete engineering impact. The primary theoretical goals are: resolving or formally characterizing the difficulty of the LM-P vs. LM-NP question; characterizing hardness of approximation and the reliability cliff (Difference §3), i.e., problems for which achieving $p = 0.99$ requires exponentially more tokens than $p = 0.5$; and developing a universal token complexity scaling law over $(|I|, N, D, C, p^*)$, the CSLM analogue of the Chinchilla law [33], enabling principled forecasting of deployment cost before experimentation. Concretely, Stage 3 should demonstrate at least three cases where token complexity theory directly guides architecture or system design decisions—for instance, proving that hierarchical multi-agent architectures are provably more token-efficient than single-model architectures for compositionally structured problem classes. A long-lived benchmark (CSLM-Bench) and open repository of known token complexity results should be established as durable infrastructure for the field.

L Detailed Discussion about Alternative Views

In Section 7, we briefly acknowledged three alternative perspectives on whether language models require a distinct theoretical framework. Here we provide a more detailed engagement with each view, including the strongest versions of the objections and our most thorough responses.

L.1 Traditional CS Already Encompasses Language Models

Objection. The most technically serious challenge to CSLM is that it is unnecessary: language models are probabilistic programs, and probabilistic computation is already well-studied within classical computer science. Randomized complexity classes such as BPP, RP, and ZPP handle probabilistic correctness formally, learning theory (PAC learning, VC dimension) handles generalization from data, and circuit complexity handles bounded-depth computation. If transformers are just large neural networks implemented on deterministic hardware, then CSLM reduces to a specific application of existing frameworks rather than a new science.

Our Response. It is true that transformer inference can be simulated on a Turing machine, just as quantum circuits are. Yet the quantum computing community does not regard this as a reason to forgo dedicated complexity classes such as BQP. The reason is that the simulation is exponentially expensive and destroys the conceptual structure that makes the system scientifically interesting. Similarly, modeling an LLM as a boolean circuit requires a circuit whose size equals the number of floating-point operations in a forward pass, i.e., billions of gates per token, growing with context length. Such a circuit-theoretic description is formally correct but practically incoherent as an analytical lens. CSLM's central questions, including "How does output token count scale with instance size for problem P?", "Is $\mathsf{LM\text{-}P} = \mathsf{LM\text{-}NP}$?", "What is the token overhead of augmenting model $\mathcal{M}$ with retrieval system E?", are not questions that classical complexity theory has the vocabulary to formulate, for the same reason that sample complexity cannot be expressed in time-complexity notation. Theoretical frameworks are valuable not only for what they prove, but for what questions they make well-posed. CSLM makes a class of questions well-posed that are currently expressed only informally.

L.2 The Physics of Language Models Provides a Sufficient Framework

Objection. A growing body of work under the banner "physics of language models" [5–8, 100, 101] investigates knowledge storage, knowledge extraction, and reasoning mechanisms through the lens of statistical physics and mathematical analysis of training dynamics. One might argue that this program, together with mechanistic interpretability [9, 87], already constitutes a rigorous science of language models, rendering CSLM redundant.

Our Response. The physics of LMs operates at the level of model internals: how knowledge is stored in weights, how circuits implement specific computations, how training dynamics lead to grokking or phase transitions. This is a "microscopic" or mechanistic analysis. CSLM operates at a "macroscopic" level: given a (possibly opaque) model and a problem instance, how does token cost scale with instance difficulty? These two levels are complementary, just as thermodynamics and statistical mechanics are complementary to atomic physics. Neither subsumes the other. Token complexity $\mathsf{TC}(I, \mathcal{M})$ is defined over problem instances I and treats the model $\mathcal{M}$ as a black box. This black-box perspective is essential for the framework to apply to closed proprietary systems, which represent the most capable deployed LMs. The physics-of-LMs program, by contrast, typically requires access to weights, activations, and training trajectories. CSLM's framework is therefore the only candidate for a complexity theory that applies uniformly to both open-source and proprietary systems, which is a necessary property for any foundational theory of LM computation.

L.3 Cognitive Science and Neuroscience Provide Adequate Frameworks

Objection. Language models exhibit cognitive phenomena, memory, reasoning, attention, compositionality, that cognitive science has studied for decades. One might argue that adopting cognitive-scientific frameworks (e.g., dual-process theory, working memory models, Bayesian brain hypotheses) is more productive than constructing a new computational theory, particularly given the empirical parallels between human and machine language processing.

Our Response. Cognitive science is fundamentally constrained by the biological substrate it models: energy budgets measured in watts, serial bottlenecks in working memory, the speed of neural firing at tens of milliseconds per spike. These constraints shape every major theoretical construct in cognitive science, from the 7 ± 2 limit on working memory to the Bayesian brain's prior distributions shaped by evolutionary pressures. Language models operate under radically different constraints: a multi-billion-parameter transformer can process thousands of tokens in parallel, has no fixed working memory limit (beyond context window budget), and is not subject to biological noise. A theory shaped around one set of constraints is not straightforwardly transferable to a system governed by a different set. One of CSLM's most promising contributions is the identification of scaling laws relating token complexity to model scale and training compute. These quantitative, predictive relationships have no analogue in cognitive science, which studies fixed-capacity systems that do not improve by orders of magnitude through the addition of more neurons or training data. The empirical discovery that LM capabilities follow power laws across many orders of magnitude is a genuinely new scientific phenomenon that demands a new theoretical vocabulary.

L.4 CSLM Definitions Are Empirically Intractable

Objection. Even granting that CSLM identifies interesting questions, one might worry that the definitions are not empirically tractable. Token complexity $\mathrm{TC}(I, \mathcal{M})$ is a curve over all success rates $p \in [0, 1]$, requiring in principle infinitely many evaluations per instance. Asymptotic complexity classes require behavior "for all sufficiently large instances," which is impossible to verify in finite experiments. The analogy to classical complexity theory therefore breaks down exactly where it matters most: classical complexity proofs are mathematical theorems, whereas CSLM claims would always remain empirical approximations.

Our Response. The practical value of classical complexity theory does not rest solely on formal proofs: it rests on the predictive heuristic that polynomial-time algorithms are tractable and super-polynomial ones are not. This heuristic is used empirically by algorithm designers who have no formal proof that their problem is in P but observe polynomial scaling in practice. CSLM adopts the same stance: token complexity curves can be estimated from finite samples, asymptotic behavior can be extrapolated from empirical scaling experiments, and the resulting approximations are scientifically useful even without closed-form proofs. Test-time scaling research has already produced empirical measurements of the relationship between output token count and success rate for specific problems and models [94], precisely the structure that CSLM formalizes as the token complexity curve. As benchmarks like NPPC [98] and evaluation infrastructure mature, systematic empirical measurement of token complexity across problem classes and model families will become standard practice. We also provide a concrete example of CSLM to sorting problems in Appendix B and a general guidance for applying CSLM to new problems in Appendix D.

M Token Counts Across Mainstream Tokenizers

To quantify the differences of tokenizers in practice, we encode the test split of wikitext-2-raw-v1 (2,000 documents; 931,981 characters; 175,517 words) with eight widely deployed tokenizers. All eight are byte-level BPE or SentencePiece with byte fallback, so the proposition applies to each pair. Table 2 reports the results. Seven of the eight tokenizers produce token counts within 4% of `o200k_base`, and the largest deviation, 13%, comes from Mistral-7B-v0.3, which has the smallest vocabulary. Across vocabulary sizes ranging from 32K to 256K, the tokenizers therefore differ by a small multiplicative constant rather than by order, so tokenizer choice affects the constants that we report and not the exponents on which complexity classes depend. These measurements cover English text; other languages, code, and non-text modalities may show larger variation, which is part of the tokenization question raised in Section 6.

Table 2: Token counts of eight tokenizers on the test split of wikitext-2-raw-v1. "Rel." reports total tokens relative to `o200k_base` (GPT-4o).

Tokenizer	Vocab	Tokens	Tok/char	Tok/word	Bytes/tok	Rel.
GPT-2 (`gpt2`)	50,257	205,561	0.2206	1.171	4.54	0.98×
GPT-4o (`o200k_base`)	200,019	208,855	0.2241	1.190	4.47	1.00×
DeepSeek-V3	128,815	209,157	0.2244	1.192	4.46	1.00×
Gemma-2-9B	256,000	209,743	0.2251	1.195	4.45	1.00×
Llama-3.1-8B	128,256	209,911	0.2252	1.196	4.45	1.01×
GPT-3.5/4 (`cl100k_base`)	100,277	210,067	0.2254	1.197	4.44	1.01×
Qwen2.5-7B	151,665	216,940	0.2328	1.236	4.30	1.04×
Mistral-7B-v0.3	32,768	236,345	0.2536	1.347	3.95	1.13×