

Large Language Models are zero-shot problem solvers – just like Modern Computers

Tim Z. Xiao^{†,◇,*}, Weiyang Liu^{‡,◇}, Robert Bamler[†]

[†] University of Tübingen

[‡] University of Cambridge

[◇] Max Planck Institute for Intelligent Systems

ABSTRACT. Language models (LMs) are trained specifically for one out of many natural language processing (NLP) tasks. By simply scaling them up to large LMs (LLMs), they emerge the ability to solve many other NLP tasks that they have not been trained for. This emergence of zero-shot problem solving ability of LLMs surprised the community. In this survey, we draw an overlooked connection between LLMs and modern computers that also emerge zero-shot problem solving abilities. We discuss their similarities in capability, architecture, and history, and then use these similarities to motivate a different perspective on understanding LLMs and the prompting paradigm. We hope this connection can help us gain a deeper understanding of LLMs, and can spark discussions between the core computer science community and the foundation model community.

Keywords: large language models, universal problem solver, emergence, computer

1. INTRODUCTION

Thanks to increasingly powerful computation infrastructures, we have witnessed many significant breakthroughs delivered by simply scaling up deep learning models in recent years. This is partially due to the empirical findings that for many tasks in natural language processing (NLP) and computer vision, the scale of a model (along with the amount of compute and data used for training) and its test performance are so positively correlated that knowing either, one can even predict the other. Such relationship is described by the *scaling laws* (Ganguli et al., 2022; Hestness et al., 2017; Kaplan et al., 2020). In other words, it is not surprising that a larger language model can achieve a better performance in the task of language modeling.

However, recent development in large language models (LLMs) such as GPT-3 (Brown et al., 2020) and ChatGPT (Schulman et al., 2022) has attracted an unusual amount of interest from both the machine learning community as well as the general public. Apart from the imaginative conversations people had with LLMs, a more fundamental reason is that LLMs are not only showing improvements in language modeling but also showing the abilities to solve a wide range of NLP tasks including question answering, translation, summarization and even algorithmic tasks (Brown

et al., 2020; Chowdhery et al., 2022; Radford et al., 2019; Zhou et al., 2022). This means by training on a single task, the model gains the ability to solve tasks it has never been trained for. Such an emergence of *zero-shot*¹ problem solving abilities in LLMs is surprising and was not predicted from the smaller scale models (Wei et al., 2022). It is still an open question how LLMs emerge with their abilities.

The phenomenon of a system showing spontaneous abilities that cannot be predicted when scaled up is known as *emergence*. Emergence is not a unique phenomenon that happens only in LLMs. It has been observed and discussed in many other subjects including physics and biology at least half a century ago. In the field of machine learning, emergence was also discussed and has inspired pioneer works such as the Hopfield network (Hopfield, 1982). The phenomenon is well explicated in an article by Anderson (1972) with a concise title “*More Is Different*”.

Another example that has emergent ability are the computers we use in our daily life. The elementary components in modern computers are logic gates, which are designed to do simple Boolean operations only, e.g., *AND*, *OR*. But when we wire a large amount of them together, they emerge with the abilities to run various applications such as video games and text editors. If we phrase problems and tasks into programs, then computers can be seen as zero-shot problem solvers just like LLMs, since most of them are not designed or built for any specific program but as general purpose machines. Crucially, computer scientists have developed a host of tools to help understand and further exploit emergent capabilities, such as debuggers, or various high-level programming paradigms.

In this survey, we connect the dots and draw an overlooked connection between these two zero-shot problem solvers, i.e., LLMs and computers. We believe their similarities outline the potential of using LLMs for a much more general problem solving setting, and their differences help us to understand the behaviors of LLMs and how to improve them. In particular, we illustrate how a good language model, which allows a concept to be encoded less ambiguously, is at the core of exploiting this new model of computation. This LLM-based new computation model has already inspired new computing paradigms. For example, Verbalized Machine Learning (VML; Xiao et al. (2025)) introduces the idea of parameterizing machine learning models with natural language, where learning is performed through iterative prompt optimization rather than numerical gradient updates. This bridges traditional machine learning, where models are parameterized numerically and optimized via gradient descent—with a more interpretable, natural language-based paradigm. While this survey focuses on LLMs, a similar story applies to multimodal LLMs (Fei et al., 2022; Li et al., 2024; Liang et al., 2024), which extend the same architecture to other data types by tokenizing inputs like images. We hope that the connections drawn here will spark further discussions on topics such as computation theory, security, distributed computing, and beyond for LLMs.

2. A TALE OF TWO ZERO-SHOT PROBLEM SOLVERS

In this section, we elaborate on the notion that large language models (LLMs) are zero-shot problem solvers in a similar way in which modern computers are zero-shot problem solvers, since

¹We use the term *zero-shot* to highlight that the model is not trained for the tested tasks (similar to its usage in Radford et al. (2019)). This is sometimes also called *universal* or *general* problem solving. In many other LLM papers (Kojima et al., 2022), *zero-shot* means no demonstration is provided during inference time in the prompt. Here we treat both zero-shot and few-shot prompting as examples of zero-shot problem solving.

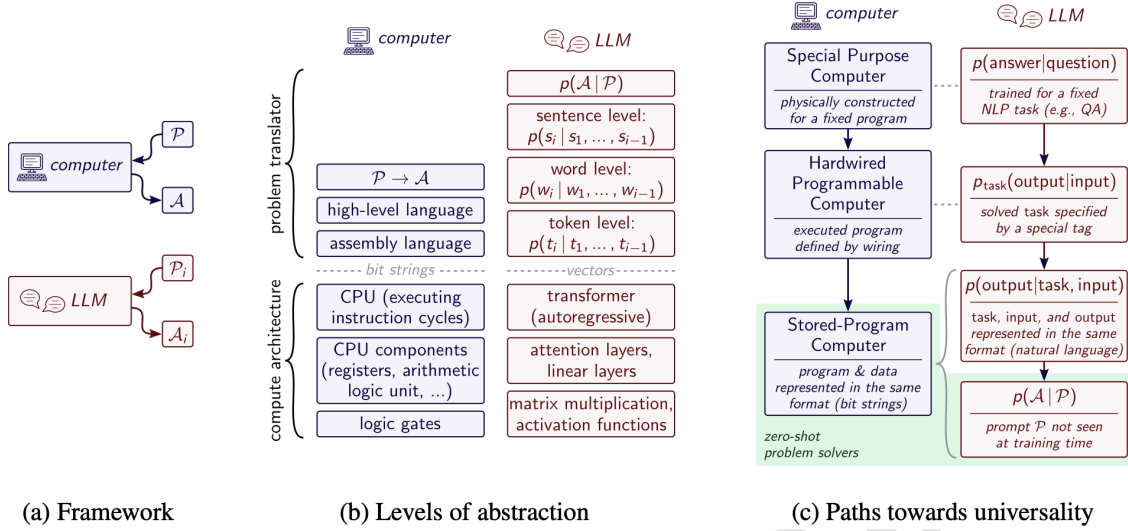


Figure 1. Connections between modern computers and large language models from three different perspectives: (a) they both produce corresponding output after doing computations on the input; (b) they both consist of a compute architecture and a problem translator; (c) they both evolved from a specialized task solver to a general zero-shot problem solver.

they can both solve much more general problems than those they have been trained or hardcoded for. However, we understand much better how such ability emerged in computers than LLMs. Here, we look into the similarities as well as the differences in the formulations and the histories of both models of computation, and try to understand how LLMs emerged with their abilities.

Modern Computers. Computers have undergone several transitions from specialized to universal machines (see Figure 1c). As we will discuss in Section 2.1 below, this transition is similar to the recent history of LLMs. Alan Turing had an early vision (Turing, 1936) of a universal computing machine that was way ahead of its time because the real machines around the time were only constructed for special tasks, such as breaking encrypted communication during wartime. To follow Turing’s vision and to avoid building a different machine from scratch for a new task, early computer scientists, built machines with a large collection of different arithmetic units and switches (e.g., ENIAC (Weik, 1961)). By wiring up these units differently (i.e., programming) and piping the input data through, the machine was able to solve different tasks. While this made it possible to “reprogram” (i.e., rewire) the computer, the program was still part of the hardware wiring, and any changes to it had to be done manually. The later invention of stored-program computers overcame this limitation and led to even more universal computing machines. Here, the central idea, expressed in the von Neumann architecture (von Neumann, 1945), was to represent programs as data rather than as wiring setups. This allowed computers to read and solve arbitrary programs for which they were not set up by wiring. In other words, they become zero-shot problem solvers.

Large Language Models. Written language can be formed by sequences of tokens. A language model learns mappings to the next token t_i from its prefix. Such a mapping can be represented as a conditional probability distribution (see Figure 1b). The joint probability of the sequence $p(\mathbf{t})$ is the product of the conditional probabilities (Bengio et al., 2000), i.e., $p(\mathbf{t}) = p(t_1) \prod_{i=2}^N p(t_i | t_1, \dots, t_{i-1})$. A popular method to model these conditional probabilities is to use Transformers (Vaswani et al.,

2017) and train under self-supervision. With increasing scale of these language models, they turn out to emerge the ability to solve tasks other than language modeling (i.e., they are zero-shot problem solvers), and we call them large language models for distinction.

2.1. Connecting the Dots. Here, we compare the architectures and survey the histories of computers and LLMs, which turned out to be surprisingly similar (see Figure 1). Their likeness might not be a coincidence, and can provide clues for us to understand how LLMs emerge with their abilities.

Similar frameworks. Both computers and LLMs can be seen as mappings from $\mathcal{P}$ to $\mathcal{A}$ (Figure 1a). Given a *program* $\mathcal{P}$, a computer will execute $\mathcal{P}$ and return the answer $\mathcal{A}$ when the program terminates. This mapping is deterministic, which means if $\mathcal{P}$ is a correct implementation of a suitable algorithm, then $\mathcal{A}$ is the correct answer to the problem. Similarly, given a *prompt* $\mathcal{P}$, an LLM will infer and return an answer $\mathcal{A}$. However, this mapping is stochastic and follows the conditional distribution $p(\mathcal{A}|\mathcal{P})$ captured by the LLM. In addition, the interactive feature of LLMs allows us to easily include previous prompts and answers into the new prompt, i.e., $\mathcal{P}_i = [\mathcal{P}_{i-1}, \mathcal{A}_{i-1}, \text{new}]$.

Similar level of abstractions. Both computers and LLMs consist of a *problem translator* on top of a foundational *compute architecture* (Figure 1b). The problem translator turns problems written in a human readable language into some native representation of the compute architecture (bit strings for computers, vectors for LLMs). The compute architecture is formed by a large collection of simple components (logic gates for computers, matrix multiplications and activation functions for LLMs). In both computers and LLMs, the respective compute architecture is able to do arbitrary computations due to both the scale (i.e., the quantity) and the universality of the involved components.

Similar histories. Both computers and LLMs had a similar path of evolution from task-specific to general-purpose problem solvers (Figure 1c). Early approaches to NLP designed and trained a model specifically for a given task, e.g., question answering (QA). To create more general solutions, people then experimented with model architectures that were trained on a fixed set of tasks, specified by a task identification tag (Kaiser et al., 2017; Shazeer et al., 2017). Similar to the von Neumann architecture of computers, the next important development was to represent the task description in the same format as the input and output, i.e., describing a task in natural language rather than using a special tag (McCann et al., 2018). This resulted in a training objective very similar to language modeling. Inspired by this, Radford et al. (2019) trained a pure (but very large) language model (i.e., GPT-2) and showed its ability to solve a range of different NLP tasks beyond language modeling.

2.2. Fundamental Difference. While both computers and LLMs can be seen as general-purpose problem solvers, they differ fundamentally in what they assume about the nature of problems and how those problems are to be expressed. This difference lies in their epistemological commitments, i.e., in what they treat as valid knowledge and representation.

Traditional computers assume a world that is formally structured and logically well-defined. Problems must be specified in a formal language with precise semantics before they can be computed. The act of programming, in this paradigm, is an act of defining the problem space. This worldview aligns with the epistemology of mathematics and formal logic, where truth is absolute and fully specified. As a result, traditional computers are powerful tools for problems that are well-structured, symbolic, and rigorously formalizable.

LLMs, by contrast, are trained on vast corpora of natural language, which embodies ambiguity, cultural context, pragmatic nuance, and incomplete information. They do not assume that a problem must be expressed formally to be computable. Instead, they infer structure and meaning from informal, often underspecified prompts. This reflects a different epistemology, one rooted in interpretation, usage, and statistical regularities in human communication. LLMs are thus adept at solving problems that are difficult to formalize: giving advice, interpreting tone, summarizing arguments, or answering context-dependent questions.

From this perspective, features like ambiguity tolerance and natural language accessibility are not superficial differences between LLMs and computers; they are consequences of deeper commitments about what kinds of problems exist in the world and how those problems are communicated. LLMs allow problems to be posed and solved in the same medium humans use to think and communicate: natural language. Computers demand translation into a symbolic medium, which sacrifices accessibility but gains precision. They differ fundamentally in how they model the world, how they represent problems, and how they reason through solutions. Recognizing this distinction allows us to better understand the types of problems each is best suited to address, and why they complement rather than replace each other.

3. LESSONS TO LEARN

The connections between computers and LLMs identified in Section 2 above reveal two important findings about zero-shot problem solvers. Here, we identify two essential building blocks for them, and we point out a direction to improve LLMs which explains the current focus in the community.

Finding 1. *The zero-shot problem solving ability only emerges when we combine a powerful compute architecture with a suitable problem translator.*

Computational ability is not as rare a property as we might think. There are many, even simple physical systems that can compute, but zero-shot problem solving requires an additional layer of abstraction on top of computation that translates new problems into the type of computations that the system can perform. For example, many problems can be phrased as minimization problems, i.e., finding the lowest point on some surface. A simple compute architecture capable of solving such an optimization problem could drop a ball onto a physical manifestation of a given surface and record where the ball eventually stops. While, in principle, many problems can be phrased as such minimization problems, in practice, the problems humans want to solve are typically described in a more readable form. Without an additional unit that translates the human-readable description into a surface whose lowest point we want to find, the above computation architecture is not capable of zero-shot problem solving.

Both computers and LLMs have very powerful compute architectures. Some Transformer variants, such as the Universal Transformer (Dehghani et al., 2019), are theoretically Turing complete, and modern computers are often modeled as Turing machines despite finite resources. However, the relevance of Turing-completeness differs between the two. In classical computing, Turing-completeness is deeply meaningful: it describes a system’s ability to simulate any formal, symbolic computation given an appropriate program and sufficient resources. It reflects the system’s alignment with a formal epistemology, where problems must be fully specified before they can be solved. LLMs, by contrast, do not require the user to explicitly formalize a problem. Even if they can approximate a Turing machine in theory, their practical power lies in their ability to interpret,

generalize, and reason over natural language, even when the input is ambiguous, fuzzy, or pragmatically structured. From this perspective, both systems are computationally capable, but differ in how problems are expressed and reasoned over. What turns both computers and LLMs into zero-shot problem solvers is the presence of powerful problem translators that bridge high-level human intent with the native operations of the compute architecture, preserving and unlocking their full computational potential.

Finding 2. *Minimizing the gap between the language model of LLMs and the internal language model of a user is at the core of unlocking the full problem solving ability of LLMs.*

As we have argued above, ensuring that a solver understands the problem or instruction is key for solving it. Computers have been excellent problem solvers for the past decades because formal languages are designed to be precise. This means given a piece of code in some formal language, there is only one correct interpretation (illustrated as a delta distribution in Figure 2 bottom left) and good programmers are expected to understand the language fairly well (see sharp peak in Figure 2 top left). In other words, the models that humans and computers have of a formal language are both relatively *precise*, and the two are closely *aligned* with each other. In contrast, natural language is inherently *imprecise* as there are often many correct interpretations given a sentence. Moreover, both humans and LLMs learn natural language from data rather than from a specification, so their two models will typically be somewhat *misaligned* from each other.

Many of the existing works on LLMs as zero-shot problem solvers can be understood as a spectrum of methods for minimizing the communication gap between humans and LLMs. At one end of the spectrum, we humans can adapt our internal language models towards the language models of LLMs (① in Figure 2), e.g., by incorporating certain signalling phrases into our prompts that empirically improve the quality of answers, such as “Let’s think step by step” (Kojima et al., 2022). At the other end of the spectrum, we can adapt LLMs towards our internal language models (② in Figure 2). This can be done during training (e.g., as in InstructGPT and ChatGPT, which were trained with human feedback (Ouyang et al., 2022)), or after training (e.g., personalizing the model by few-shot prompting, i.e., prefixing prompts with examples of similar problems and their solutions or with a conversation history). Most prompting research falls somewhere in between these two extremes, where humans and LLMs adapt their models towards each other (③ in Figure 2) (Liu et al., 2023).

Recent reinforcement-learning-tuned models such as DeepSeek-R1 (Guo et al., 2025) and OpenAI’s o1 (Jaech et al., 2024) can also be seen as better aligned to the user’s internal language model. While they remain within the same computational paradigm, they exhibit stronger reasoning and

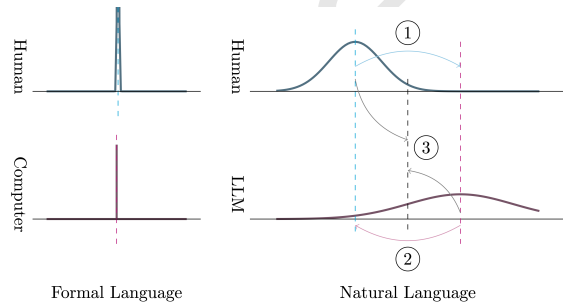


Figure 2. Communication gaps between human and the solver with different languages. Language ambiguity is depicted by variance, model misalignment is depicted by the two separate means.

interaction capabilities through reinforcement learning. Analogous to different instruction set architectures (e.g., ARM v.s. x86) or to built-in runtime systems with enhanced optimizations and debugging, they represent more capable implementations of the same general-purpose solver.

However, as LLMs become more aligned with users and easier to interact with, concerns about over-reliance and cognitive off-loading are increasingly relevant. Emerging research shows that while LLMs can enhance productivity and creativity, frequent dependence may lead to reduced critical thinking, skill degradation, and even psychological dependence (Gerlich, 2025; Huang et al., 2024; Kumar et al., 2025; Macnamara et al., 2024; Sabbah & Li, 2025). These risks suggest that narrowing the communication gap between humans and LLMs must be balanced with design considerations that preserve human agency, oversight, and long-term cognitive resilience.

4. IMAGINING A NATURAL LANGUAGE COMPUTING PARADIGM

The previous sections highlighted how LLMs, like early computers, have evolved from task-specific to general-purpose problem solvers. If this trajectory continues, natural language systems may face challenges similar to those encountered in the history of computing. This section explores how such systems might scale, what principles could guide their development, and what new subdomains may emerge as a result.

4.1. Integrating Algorithms with Natural Language-Based Programs. Prompts can be viewed as programs written in natural language, with LLMs acting as interpreters over a distributional semantic space. While less formally specified than code, prompts define tasks, condition inputs, and elicit structured behaviors, ranging from deterministic functions (e.g., sorting a list) to abstract reasoning (e.g., summarizing an argument).

Reframing prompting as computation opens the door to adapting classical algorithmic techniques to this new medium. In particular, deterministic algorithms can be combined with natural language-defined components. For example, the VML framework (Xiao et al., 2025) mentioned in the introduction retains the structure of traditional learning algorithms while expressing models and optimizers in natural language, executed via LLMs. Prompt design, in this view, becomes an instance of program synthesis, where the goal is to construct language instructions that elicit desired model behavior reliably and efficiently. This hybrid view positions LLMs as semantic engines integrated into algorithmic frameworks, enabling procedural reasoning over language.

4.2. Modularity and Encapsulation for Scalable Reasoning. A key factor in the scalability of classical computing has been modularization, i.e., the ability to build complex systems from smaller, reusable components with well-defined interfaces. The same principle will be essential for scaling natural language-based computation. Here, prompt modules act as encapsulated language functions, each performing a specific subtask such as summarization, classification, or verification.

By defining clear input-output formats (e.g., a paragraph, a boolean answer, a list) through natural language instructions, these modules serve as APIs. This enables the decomposition of complex tasks into sequences of simpler ones, where each module can be tested, reused, and improved independently. Encapsulation ensures that internal reasoning remains isolated, allowing modules to interact through well-specified linguistic contracts without entangling their internal logic.

As in classical systems, such modular, interface-driven design will be essential for scaling LLM-based systems, supporting composability, robustness, and maintainability.

4.3. Language-Native Infrastructure and Emerging Subdomains. Scalable natural language computation will require infrastructure analogous to classical computing stacks, adapted to a linguistic interface. Orchestration frameworks will manage memory, dialogue context, tool usage, and agent coordination. Prompt compilers may translate high-level intent into structured prompt templates, resolving ambiguity and optimizing for task-specific performance. Execution layers will track progress, handle branching and fallback strategies, and ensure semantic consistency over multi-turn or multi-agent workflows.

As this infrastructure matures, we can anticipate the emergence of subdomains structurally similar to those in classical computer science. Security, for example, will involve defending against prompt injection, adversarial language, and unintended emergent behaviors. Distributed systems will manifest as multi-agent collaborations, coordinated through dialogue protocols and turn-taking conventions. Memory and retrieval will rely on hybrid systems that combine embedding-based indexing with contextual synthesis. Networking will involve language-based protocols for structured message exchange among LLMs, tools, and users. These subdomains will not replicate their classical counterparts directly, but they will arise from similar demands for reliability, coordination, and scalability.

4.4. Toward a Theory of Language-Based Computation. The development of natural language computing calls for a new theoretical foundation. Unlike classical computation, which is grounded in formal logic and deterministic semantics, language-based computation is probabilistic, context-dependent, and informally specified. A theory in this space should help us understand which tasks can be solved effectively through prompts, and which are better addressed by traditional methods.

Such a theory may draw on both formal reasoning and empirical benchmarking, with insights often conditioned on model performance over specific tasks. In some cases, the benchmark itself may form an integral part of the theory, i.e., defining the context under which certain behaviors or guarantees are expected to hold.

It should also account for the cost of formalization, a factor often overlooked in classical theory. LLMs offer practical value by operating directly on informal inputs, bypassing the need for fully specified representations. Capturing this advantage is essential for modeling the real-world strengths of language-based systems.

Ultimately, such a theory should offer insight into the kinds of problems LLMs are well-suited to solve, and under what conditions. This understanding would help guide the design of scalable reasoning algorithms that make effective use of language-based computation.

Disclosure Statement. The authors have no conflicts of interest to declare.

REFERENCES

- Anderson, P. W. (1972). More is different. *Science*, 177(4047), 393–396. <https://doi.org/10.1126/science.177.4047.393>
- Bengio, Y., Ducharme, R., & Vincent, P. (2000). A neural probabilistic language model. *Advances in Neural Information Processing Systems*, 13.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877–1901.

- Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., ... Fiedel, N. (2022). Palm: Scaling language modeling with pathways. *arxiv:2204.02311*.
- Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., & Kaiser, Ł. (2019). Universal transformers. *International Conference on Learning Representations*.
- Fei, N., Lu, Z., Gao, Y., Yang, G., Huo, Y., Wen, J., Lu, H., Song, R., Gao, X., Xiang, T., et al. (2022). Towards artificial general intelligence via a multimodal foundation model. *Nature Communications*, 13(1), 3094.
- Ganguli, D., Hernandez, D., Lovitt, L., Askell, A., Bai, Y., Chen, A., Conerly, T., Dassarma, N., Drain, D., Elhage, N., et al. (2022). Predictability and surprise in large generative models. *2022 ACM Conference on Fairness, Accountability, and Transparency*, 1747–1764.
- Gerlich, M. (2025). Ai tools in society: Impacts on cognitive offloading and the future of critical thinking. *Societies*, 15(1), 6.
- Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. (2025). Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., Patwary, M., Ali, M., Yang, Y., & Zhou, Y. (2017). Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*.
- Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8), 2554–2558.
- Huang, S., Lai, X., Ke, L., Li, Y., Wang, H., Zhao, X., Dai, X., & Wang, Y. (2024). Ai technology panic—is ai dependence bad for mental health? a cross-lagged panel model and the mediating roles of motivations for ai use among adolescents. *Psychology Research and Behavior Management*, 1087–1102.
- Jaech, A., Kalai, A., Lerer, A., Richardson, A., El-Kishky, A., Low, A., Helyar, A., Madry, A., Beutel, A., Carney, A., et al. (2024). Openai o1 system card. *arXiv preprint arXiv:2412.16720*.
- Kaiser, L., Gomez, A. N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L., & Uszkoreit, J. (2017). One model to learn them all. *arXiv preprint arXiv:1706.05137*.
- Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.
- Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., & Iwasawa, Y. (2022). Large language models are zero-shot reasoners. *Advances in Neural Information Processing Systems*.
- Kumar, H., Vincentius, J., Jordan, E., & Anderson, A. (2025). Human creativity in the age of llms: Randomized experiments on divergent and convergent thinking. *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 1–18.
- Li, C., Gan, Z., Yang, Z., Yang, J., Li, L., Wang, L., Gao, J., et al. (2024). Multimodal foundation models: From specialists to general-purpose assistants. *Foundations and Trends® in Computer Graphics and Vision*, 16(1-2), 1–214.
- Liang, P. P., Zadeh, A., & Morency, L.-P. (2024). Foundations & trends in multimodal machine learning: Principles, challenges, and open questions. *ACM Computing Surveys*, 56(10), 1–42.

- Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2023). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9), 1–35.
- Macnamara, B. N., Berber, I., Çavuşoğlu, M. C., Krupinski, E. A., Nallapareddy, N., Nelson, N. E., Smith, P. J., Wilson-Delfosse, A. L., & Ray, S. (2024). Does using artificial intelligence assistance accelerate skill decay and hinder skill development without performers’ awareness? *Cognitive Research: Principles and Implications*, 9(1), 46.
- McCann, B., Keskar, N. S., Xiong, C., & Socher, R. (2018). The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*.
- Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Gray, A., et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*.
- Sabbah, J., & Li, F. (2025). When humans and large language models collaborate, problem-finding illuminates. *Innovation*, 1–34.
- Schulman, J., Zoph, B., Kim, C., Hilton, J., Menick, J., Weng, J., Uribe, J. F. C., Fedus, L., Metz, L., Pokorny, M., Lopes, R. G., Zhao, S., Vijayvergiya, A., Sigler, E., Perelman, A., Voss, C., Heaton, M., Parish, J., Cummings, D., . . . Ryder, N. (2022). Chatgpt: Optimizing language models for dialogue. *OpenAI*. Retrieved January 16, 2023, from <https://openai.com/blog/chatgpt/>
- Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., & Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *International Conference on Learning Representations*.
- Turing, A. (1936). On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363), 5.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.
- von Neumann, J. (1945). First draft of a report on the edvac. *Pennsylvania: University of Pennsylvania*.
- Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al. (2022). Emergent abilities of large language models. *Transactions on Machine Learning Research*.
- Weik, M. H. (1961). The eniac story. *Ordnance*, 45(244), 571–575.
- Xiao, T. Z., Bamler, R., Schölkopf, B., & Liu, W. (2025). Verbalized machine learning: Revisiting machine learning with language models. *Transactions on Machine Learning Research (TMLR)*.
- Zhou, H., Nova, A., Larochelle, H., Courville, A., Neyshabur, B., & Sedghi, H. (2022). Teaching algorithmic reasoning via in-context learning. *arXiv preprint arXiv:2211.09066*.