Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Fine-tuning versus Prompting: Choosing the Right Approach for Large Language Model Applications

Uma Shankar Koushik Kethamakka

Independent Researcher, USA

ARTICLE INFO

Received: 11 Feb 2026

Accepted: 15 Feb 2026

ABSTRACT

Large language models offer organizations flexibility: they can either tune the weights of the model on their domain data or they can design their prompts to elicit certain behavior from their domain-agnostic models. This article describes a framework for deciding between the tradeoffs, costs, and logistics of weight tuning and prompt design. This discussion highlights that neither approach is superior, and the best decision depends on task complexity, data availability, latency requirements, how much maintenance the system can afford, and the organization. Solutions with fine-tuning give more specialized behavior, but require curated data, compute resources, and monitoring to maintain the model's performance. Prompting is useful for fast model iteration and maintains model generality, but may not be useful in high-complexity applications where system domain knowledge is needed to remain stable. The article discusses systematic evaluation and measurement both for guiding adaptation decisions and for benchmarking, A/B testing, and monitoring. It also highlights the models specifically trained for reasoning, many of which are now defaulting to (extended) chain-of-thought processing. This has implications for whether prompting or fine-tuning methods are more appropriate, depending on the reasoning ability desired. In this way, the framework provides a systematic approach to identifying and employing the appropriate adaptation methods according to the performance and efficiency goals for applications built using large language models.

Keywords: Large Language Models, Prompt Engineering, Fine-Tuning, Parameter-Efficient Adaptation, Low-Rank Adaptation, Chain-of-Thought Reasoning, Reasoning Models, LLM Evaluation, Domain-Specific Optimization

1. Introduction

One important design choice for organizations building LLM-based applications is whether to train on data from their application domain or to use a generalist model via prompt engineering. This choice determines how well suited a model is to the application domain it is being used in, how expensive it is to maintain, and how easy or difficult it is to update. This is a defining decision for the system. The overall LLM market has seen rapid growth, with industrial applications across a variety of sectors, including healthcare, finance, retail, manufacturing, and other technology service sectors. These machine learning models are increasingly being used for automating cognitive tasks, assisting users in decision making, and providing unprecedented experiences to customers [1].

However, this rapid growth in model architectures and adaptation methods has created a much more complex landscape for working with foundation models, which range from billions to hundreds of billions of parameters, and improve in performance at each scaling step. A wide variety of model adaptation techniques have also been proposed, including prompt engineering, an abundance of fine-tuning techniques, and retrieval-augmented generation. Surveys have found that more than half of the companies using LLMs use prompting exclusively or predominantly, and 25% of companies use fine-tuning at least for some of their systems. A further 15% use hybrid approaches, which also use

prompting, but are not fully prompting-based [2]. Organizations that have deployed fine-tuning have reported 15-30 percent accuracy improvements over prompting for domain-specific tasks, but face 3-5 times the cost for data preparation and model management.

The article provides a systematic framework for deciding on the technical design, economic, and organizational considerations to take into account when deciding on adaptation options. The aim is to provide practitioners with a systematic and explicit framework for deciding on when these considerations should be applied in their specific organizational context.

2. Prompt Engineering: Harnessing Foundation Model Capabilities

2.1 Zero-shot and Few-shot Prompting

Prompt engineering is the design of text prompts to elicit the desired behavior from a frozen LM. In zero-shot prompting, task instructions are given without examples, and the LM is expected to interpret the instructions correctly and perform the desired task based on its prior training. An ideal zero-shot prompt contains an output format, domain and task contexts, and optionally constraints. Research papers on large language models had shown that language models are very capable of performing remarkably well across different tasks without fine-tuning, given complete contexts of prompts are available to them [3].

In few-shot prompting, several examples of desired behavior are provided in the prompt. Few-shot prompting has been shown to improve performance as the number of examples is increased, but typically has diminishing returns after around 5-10 examples depending on the difficulty of the task being performed by the model. Example selection extensively influences performance, and diverse and representative examples often outperform random selection from a larger set. Few-shot prompting is useful when the desired outcome cannot be simply described in words, but can be demonstrated in examples. The ability to use prompt engineering to change the function of a foundation model has changed how companies use AI by reducing both the technical barrier and the experimentation life cycle.

2.2 Chain-of-Thought and Advanced Reasoning Techniques

The goal of chain of thought prompting is to get models to output intermediate reasoning steps when completing a complex task. It has been shown to dramatically improve models on some multi-step reasoning tasks that require multi-step explanations. Chain of thought prompting is applicable to mathematical reasoning, logical reasoning, and complex question answering tasks [4]. This can be done by prompting the model with examples of step-by-step reasoning or by prompting the model to think through a problem before answering. Because the reasoning capabilities of large language models are derived from their foundation models during pre-training, chain of thought prompting is a kind of prompt engineering.

More advanced prompting strategies explicitly structure model reasoning to improve reliability. In tree-of-thought (ToT) prompting, the model explores and selects among multiple reasoning paths, which allows it to solve more complex tasks [5]. Self-consistency generates multiple independent chains of reasoning, then combines their outputs using majority voting in order to capture ensemble effects [6]. Verification prompts make models check their own outputs for mistakes before providing a final answer. These methods can add computational cost, but they can lead to large improvements on challenging reasoning tasks, and approaches such as prompt tuning give performance on par with fine-tuning.

One benefit of the use of prompts is that they do not require retraining when the base model is improved. Only the old prompts need to be run against the new model to check that they are producing the same result as they did on the old model. This makes the application more future-proof,

as organizations will be able to adjust system behavior by modifying the prompts to suit particular business or user requirements.

Technique	Strengths	Best Use Cases
Zero-shot	Maximum flexibility, no data needed	General tasks, rapid prototyping
Few-shot	Pattern inference from examples	Tasks hard to specify in instructions
Chain-of-Thought	Improved reasoning, interpretability	Math, logic, complex analysis
Tree-of-Thought	Explores multiple reasoning paths	Complex problem-solving
Self-Consistency	Ensemble accuracy improvement	High-stakes decisions

Table 1: Prompt Engineering Techniques and Characteristics [3, 4, 5]

3. Fine-tuning Approaches: Specialized Model Adaptation

3.1 Full Fine-tuning

Fine-tuning is the process of sequentially updating the parameters of a pre-trained model to create a second model that can accurately perform a different task using a set of training data from a different domain. Full fine-tuning updates all parameters of a neural network. This enables changes to any aspect of the model's functionality, including its output style, domain knowledge, reasoning patterns, and response format. This method requires considerable computational resources, with the capacity required scaling with the size of the model, with large models requiring multiple high-end accelerators.

The primary benefit of fine-tuning is that task- and domain-specific contextual information and language conventions can be encoded using parameters within the model. In such a way, medical domain pre-training on clinical text has produced large performance increases in medical natural language processing tasks compared to models pre-trained on general-domain text [7]. Fine-tuning ideally allows models to learn domain-specific language and reasoning skills not observed in the data used for pre-training a general-purpose model. Fine-tuning requires careful selection of data, hyperparameters and evaluation metrics. The data should be as similar to the deployment domain as possible. Also, the correct set of transitions must be chosen to avoid bias and errors. Regularization techniques such as small learning rates, gradient clipping, and early stopping help to retain pre-trained behavior and learn new behavior.

3.2 Parameter-Efficient Fine-tuning Methods

Parameter-efficient fine-tuning (PEFT) methods, e.g., adapters, prefix-tuning, and prompt-tuning, are cheaper than full fine-tuning as they only modify a small number of model parameters and achieve comparable performance. For instance, low-rank adaptation (LoRA) reduces the number of trainable parameters by 99 percent or more while achieving full fine-tuning accuracy on most tasks [8]. LoRA works by adding trainable decomposition matrices to the pre-trained layers of the model, which allows fine-tuning the model with a fraction of the trainable parameters without losing information from the pre-trained parameters. LoRA can be fine-tuned on a typical consumer-grade accelerator in minutes to hours, rather than the days needed for full model fine-tuning. Additionally, multiple loRA-trained

weights can be trained for different tasks and loaded on-demand at inference time for efficient multi-tasking.

Prefix tuning and prompt tuning learn only a continuous task-specific vector prepended to each input to the model, freezing the base model and optimizing only a small number of parameters [9]. These approaches inherit much of the efficiency and easy model swapping afforded by low-rank adaptation. Adapter modules can be placed between frozen, pre-trained layers to add new, modular, or limited functionality to the network architecture [10]. For more efficient fine-tuning of large language models on limited hardware, low-rank adaptation can be combined with 4-bit quantization (QLoRA) [11]. These parameter-efficient methods have been critical for enabling organizations with limited compute resources to fine-tune pre-trained models and build custom AI systems for their specific needs.

Fine-tuned models can understand and predict relationships that cannot be captured with prompts or commands in natural language. Fine-tuning provides companies with large, high-accuracy datasets with an opportunity to fine-tune models to conform to use case constraints such as maximum response length, stylistic demands, and safety evaluation. This proprietary training data and capability cannot be replicated by other companies using foundation models offered by public cloud services. The process of building the training data and fine-tuning the models creates a barrier to entry, as these data assets are differentiated technology capabilities.

Method	Parameter Reduction	Accuracy Retention	Key Benefit
LoRA	99%+ reduction	85-95%	Multi-task deployment
Prefix Tuning	~0.1% of params	90-95%	Easy base model swap
Adapters	~2-4% of params	95%+	Modular composition
QLoRA	99%+ with 4-bit	85-90%	Consumer GPU training

Table 2: Parameter-Efficient Fine-tuning Methods Comparison [7, 8, 9, 10, 11]

4. Decision Framework: Selecting the Optimal Approach

The choice between prompting methods and fine-tuning depends on the technical requirements, computational resources, and target application. Simple classification, extraction, and formatting tasks are generally best served through prompting methods, while reasoning, domain-specific knowledge tasks, and tasks with strict stylistic requirements are best served through finetuning models. Data availability may also be a factor. Fine-tuning requires a larger number of fine-tuning samples to be curated, and prompting only requires a few demonstrations. Organizations with access to ample domain-specific data for fine-tuning may prefer this method, while those without these data may need to weigh the cost of data gathering against the expected performance benefits.

Performance constraints such as accuracy and latency need to be satisfied. To improve inference performance, the model may be fine-tuned to skip over long prompts or apply optimizations to the model weights for a particular task [12]. Latency requirements of interactive applications with real-time constraints or of batch applications where latency is of lesser concern determine the trade-off of high-latency complex prompts. Latency can be reduced with model fine-tuning, prompt compression, and architectural decisions that trade-off overall effectiveness for speed in the inference process.

In practice, considerations such as ease of maintenance can dominate: It is easier to maintain prompting on base models if they are updated often, while fine-tuned models would require

retraining. Team capabilities: Organizations with machine learning capabilities and personnel can fine-tune a model instead. For those who cannot, prompt engineering is a better fit. Cost: Prompting is usually more costly per inference, as it has many more tokens per request. Fine-tuning can involve a one-time cost and may be more cost-effective at scale. A happy medium is to use prompt engineering for the prototype or the first few production runs, and then fine-tune only where the training infrastructure cost is worthwhile.

Factor	Favors Prompting	Favors Fine-tuning
Task Complexity	Simple, well-defined tasks	Complex reasoning, domain expertise
Data Availability	Limited examples available	Substantial training data
Consistency Needs	Acceptable variation	High consistency required
Development Velocity	Rapid iteration needed	Time for training available
Base Model Updates	Frequent updates expected	Stable base model
Latency Requirements	Flexible latency acceptable	Strict latency constraints
Team Expertise	Software engineering focus	ML infrastructure available

Table 3: Decision Framework - When to Choose Each Approach [12]

5. Evaluation and Measurement: The Foundation of Informed Decisions

5.1 Systematic Evaluation Frameworks

The decision between prompting and fine-tuning is an empirical one, and a systematic evaluation against task-specific metrics and benchmarks can help inform the choice. Organizations need to establish a thorough evaluation process involving defining success criteria, selecting benchmark datasets that represent the production environment, and developing automated evaluation pipelines to rapidly iterate on the choice. Without evaluating their models on a metric, teams may overoptimize on prompts or fine-tuning, neglecting to use the simplest method (fine-tuning or prompt) for their use case, or incorrectly assume that fine-tuning is needed.

Evaluation should be task-specific, and typically should include multiple metrics that are useful for the intended usage of the model, such as classification accuracy, extraction F1 scores, or other evaluation metrics like BLEU, ROUGE, and human scoring. Consistency metrics evaluate the stability of a model's output across semantically identical inputs. Latency and throughput are additionally important performance metrics and define production constraints. Cost metrics include the number of tokens used, compute time, and infrastructure cost to enable total cost of ownership comparisons. Safety and alignment metrics ensure outputs conform to policies and guidelines and do not contain harmful content. LLM-as-a-judge is a scalable evaluation model for open-ended generation tasks where a powerful model judges outputs based on specified evaluation criteria [17].

For production systems, A/B testing and staged rollout provide an alternative source of validation. A model that performs well in held-out test sets may perform poorly when it is moved into production,

particularly if there is a distribution shift between test and production traffic. Candidates can be run in shadow mode, in parallel to production, without impacting users. Gradual rollout and following the system with monitoring can help discover issues before they become common. The monitored metrics can be compared with current data to evaluate whether model performance is still acceptable given changing user behavior and data distributions.

5.2 The Rise of Reasoning Models

Reasoning models themselves, such as o1 and o3 and the GPT-5.2 series from OpenAI, Claude Sonnet 4.5 and Opus 4.5 from Anthropic, DeepSeek R1, and others, represent a shift in the balance of prompting versus fine-tuning. In these models, extended chain-of-thought reasoning is a part of the model's training, rather than an effect of prompting, as was the case with earlier models. These models spend more time on reasoning over difficult problems, exploring many lines of reasoning in parallel before picking an answer, moving capabilities previously achieved with prompt engineering or model fine-tuning into the model.

Reasoning models outperform base models on multi-step reasoning tasks, math problems, coding problems, and complex reasoning tasks. Benchmarks show large improvements on reasoning models over base models in graduate-level science questions, competition-level math problems, and complex code problems. The additional reasoning step allows reasoning models to debug their initial response, consider alternatives and possibilities, and produce more reliable results. For applications requiring a model to be finetuned to achieve sufficient reasoning performance levels, reasoning models allow similar performance to be obtained with only prompting, simplifying development and shortening the timeline.

However, reasoning models introduce additional trade-offs for systems designers, as they tend to have a longer inference time and a high per-request cost (10x or more compared to non-reasoning models) due to the overhead of reasoning traces. These additional costs can be undesirable in applications where latency constraints are tight or request volumes are high. Reasoning models may not respond to certain prompting methods that work well with customary models. For instance, chain-of-thought prompting may not be necessary in reasoning models because they internally reason. Organizations may choose to trade-off the increased accuracy for a higher cost, so the ability to measure performance systematically is important for specific use cases.

Another risk of fine-tuning is the rapid pace of base model development; about every two months, a new base model is released, generally with substantially improved performance over its predecessor. In this case, if a fine-tuned model were developed from an old base model, it is possible that prompting a base model from a recent generation would outperform the fine-tuned model. This creates a limited incentive to fine-tune. Organizations need to consider if fine-tuning makes sense for their task, given the high likelihood of a better base model being released. This can be reduced by establishing a strong operational infrastructure: a data pipeline for generating and annotating training data, a reproducible training pipeline adaptable to new base models, and an evaluation pipeline that is fast to iterate over. This would turn fine-tuning into a fast-moving capability that subsequently uses better base models (as they are released over time), and this, in turn, may make fine-tuning cost-effective over time, rather than a sunk cost.

6. Applications Across Domains

Customer service automation illustrates this. For example, routing customer queries and answering frequently asked questions are well-suited to prompting to constrain the model with semi-structured system prompts and few-shot examples. In a more complex domain that requires multi-turn conversations, fine-tuning on tickets with a resolution in the past may be a worthwhile investment. Customer service is one of the most common applications studied for LLM-based systems, and has

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

seen improvements in quality and effectiveness (e.g., resolution rate or customer satisfaction) post-deployment [13]. Most production systems are hybrid, using prompting to answer generic questions, while passing more complex product-related queries to fine-tuned LLMs specialized in certain product categories.

Code assistance represents a strong argument for hybrid approaches: when provided with code context in the prompt together with specific instructions, many common coding tasks can be solved. Fine-tuning can help the model adapt to organization-specific code style guides, proprietary libraries, or internal API conventions. Studies of developer productivity using AI-supported code editors have reported results such as reductions in task completion time and improvements in developer satisfaction [14]. Parameter-efficient methods also enable the use of multiple fine-tunings on different codebases that share the same base model, making it easier to adapt to the quickly evolving field of code models.

Domain-specific knowledge systems, such as legal documents analysis, medical records analysis, and financial compliance checking, present perhaps the strongest case for fine-tuning. This is because they require a strong understanding of the domain, usually not available in general-purpose models. Fine-tuning can result in better performance on domain-words, domain-reasoning, and domain-structure (format). Alternatively, retrieval-augmented generation injects knowledge during inference and thus has a lower requirement for fine-tuning while maintaining domain accuracy [15]. The best choice between prompting with retrieval or fine-tuning is use case dependent.

Domain	Recommended Approach	Key Considerations
Customer Service	Hybrid: Prompting + Fine-tuned specialists	Query diversity, escalation paths
Code Assistance	PEFT with multiple adapters	Codebase-specific conventions
Healthcare/Legal	Fine-tuning + RAG	Accuracy requirements, compliance
Financial Services	Fine-tuning for core, prompting for edge	Regulatory auditability
Education	Prompting with curriculum context	Pedagogical alignment, privacy

Table 4: Domain-Specific Application Strategies [13, 14, 15]

7. Broader Implications and Future Directions

A concern is the carbon footprint of training and deployment. Full fine-tuning of large models incurs a much higher electricity and carbon cost than prompting, although this cost is lowered by orders of magnitude using parameter-efficient transfer learning, which makes fine-tuning environmentally sustainable compared to other approaches. Prompting may have no effect on training costs but could incur additional costs during inference, e.g., through longer prompts or additional reasoning steps. The costs of training and inference must be taken into account when considering an adaptation scheme, particularly for applications with high-throughput and inference costs.

The distinction between prompting and fine-tuning is increasingly blurred, and instruction tuning conditions models to follow diverse instructions, improving zero-shot performance and reducing the need for task-specific fine-tuning [16]. Constitutional methods train models to satisfy a Constitution rather than a specific task, allowing the model to exhibit different behavior based on the chosen set of principles during training. Different from prompting or fine-tuning, agent architectures further

introduce the ability to compose model capabilities in addition to tools, memory, and planners, providing additional design dimensions to steer behavior. Fine-tuning modifies the underlying capabilities of the agent, whereas prompting configures agents and tools for the behavior and tool usage of the agent.

New base models with better capabilities continue to be released periodically, which may make existing fine-tuned models obsolete or change the relative strength of prompting strategies. Organizations balance the cost of modifying the current model against the benefit of increased flexibility when upgrading to a newer version. Parameter-efficient methods are helpful because they increase flexibility while enabling customization. Understanding the historical progression of the trade-offs conceptually prepares practitioners to adapt to specific methods as they proliferate in this highly active area.

8. Conclusion

Whether to fine-tune or rely on prompt engineering is influenced by technicalities, cost considerations, and desired outcome, and the two processes are not mutually exclusive and can be combined. Prompt engineering makes it easier for researchers to evaluate and iterate over large sets of tasks. This low barrier makes it worthwhile to use prompt engineering for applications that use arbitrary input and output formats or that work across large collections of disparate tasks. However, more advanced techniques such as chain-of-thought reasoning, tree-of-thought reasoning, and self-consistency can achieve meaningful improvements without changing the model's parameters.

Fine-tune when the performance, stability, or exclusivity gain is worth the extra complexity, usually when the performance and request volume are high. Recent parameter-efficient fine-tuning methods like LoRA, prefix tuning, and adapters can make fine-tuning cheaper (90-95% savings) while still allowing upgrades to future models to be straightforward. This has broadened the applicability of fine-tuning to organizations that previously employed only prompting techniques. Applications in the healthcare, legal, or financial industries may prefer fine-tuning for its more predictable and auditable nature.

Organizations building applications need to develop principled approaches to reasoning about these tradeoffs, as well as being able to respond as conditions change. The future will likely be a combination of prompt engineering and fine-tuning, piecing together different threads in ways that will allow for the continued development of reliable, adaptive, specialized AI. The organizations that develop a mindset to treat prompting as any other technology choice with an evolving life cycle will maximize their AI technology return on investment and remain flexible in responding to the fast-paced evolution of AI technology.

References

[1] Grand View Research, "Large Language Models Market (2025 - 2030)." Available: https://www.grandviewresearch.com/industry-analysis/large-language-model-llm-market-report

[2] Michael Chui et al., "The state of AI in 2023: Generative AI's breakout year," McKinsey & Company, 2023. Available: https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2023-generative-ais-breakout-year

[3] Tom B. Brown et al., "Language Models are Few-Shot Learners," arXiv > cs > arXiv:2005.14165, 2020. Available: https://arxiv.org/abs/2005.14165

[4] Jason Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," NIPS'22: Proceedings of the 36th International Conference on Neural Information Processing Systems, 2022. Available: https://dl.acm.org/doi/10.5555/3600270.3602070

[5] Shunyu Yao et al., "Tree of thoughts: deliberate problem solving with large language models," NIPS '23: Proceedings of the 37th International Conference on Neural Information Processing Systems, 2023. Available: https://dl.acm.org/doi/abs/10.5555/3666122.3666639

[6] Xuezhi Wang et al., "Self-Consistency Improves Chain of Thought Reasoning in Language Models," 2022, ResearchGate, 2023. Available: https://www.researchgate.net/publication/359390115_Self-Consistency_Improves_Chain_of_Thought_Reasoning_in_Language_Models

[7] Kexin Huang et al., "ClinicalBERT: Modeling Clinical Notes and Predicting Hospital Readmission," ResearchGate, 2019. Available: https://www.researchgate.net/publication/332342631_ClinicalBERT_Modeling_Clinical_Notes_and_Predicting_Hospital_Readmission

[8] Edward J. Hu et al., "LoRA: Low-Rank Adaptation of Large Language Models," arXiv > cs > arXiv:2106.09685, 2021. Available: https://arxiv.org/abs/2106.09685

[9] Xiang Lisa Li and Percy Liang, "Prefix-Tuning: Optimizing Continuous Prompts for Generation," Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, 2021. Available: https://aclanthology.org/2021.acl-long.353.pdf

[10] Neil Houlsby et al., "Parameter-Efficient Transfer Learning for NLP," Proceedings of the 36th International Conference on Machine Learning, PMLR 97:2790-2799, 2019. Available: https://proceedings.mlr.press/v97/houlsby19a.html

[11] Tim Dettmers et al., "QLORA: efficient finetuning of quantized LLMs," NIPS '23: Proceedings of the 37th International Conference on Neural Information Processing Systems, 2023. Available: https://dl.acm.org/doi/10.5555/3666122.3666563

[12] Sebastian Ruder, "Recent Advances in Language Model Fine-tuning," 2021. Available: https://www.ruder.io/recent-advances-lm-fine-tuning/

[13] IBM Institute for Business Value, "From cost center to value creator," 2023. Available: https://www.ibm.com/thought-leadership/institute-business-value/en-us/report/ceo-generative-ai/customer-service

[14] Albert Ziegler, "Measuring GitHub Copilot's Impact on Productivity" Communication of the ACM, 2024. Available: https://cacm.acm.org/research/measuring-github-copilots-impact-on-productivity/

[15] Patrick Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," 34th Conference on Neural Information Processing Systems (NeurIPS 2020), 2020. https://proceedings.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

[16] Long Ouyang et al., "Training language models to follow instructions with human feedback," Computation and Language, 2022. https://arxiv.org/abs/2203.02155

[17] Lianmin Zheng et al., "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena," arxiv > cs > arXiv: 2306.05685, 2023. https://arxiv.org/abs/2306.05685

[18] OpenAI, "Learning to Reason with LLMs," 2024. Available: https://openai.com/index/learning-to-reason-with-llms/