Learning to Reason with Transformers via Search Inductive Biases: A Proposal

Carlos Núñez-Molina, Israel Puerta-Merino, Pablo Mesejo, Juan Fernández-Olivares
University of Granada, Spain
Andalusian Institute of Data Science and Computational Intelligence (DaSCI)
ccaarlos@ugr.es, israelpm01@ugr.es, pmesejo@go.ugr.es, faro@decsai.ugr.es

Abstract

Large Language Models have revolutionized the field of AI. Most recently, with the advent of Large Reasoning Models like OpenAI's o1 (Strawberry), they are becoming increasingly proficient at reasoning tasks, such as math, computer programming and Sequential Decision Making (e.g., Automated Planning). In this preliminary work, we present an alternative approach for *learning to reason*: instead of performing reasoning at the *LLM-level*, we propose to do so at the *transformer-level*. To achieve this, we introduce the **Search Transformer**, a novel neural architecture that enhances the transformer model with a *search inductive bias*, thus allowing it to perform variable test-time computation. We formulate search operations (e.g., node selection and successor generation) in terms of differentiable, attention-based computations, in order to learn a search process end-to-end using back-propagation. By learning to search, we believe Search Transformers will adquire promising System-2 capabilities, thus surpassing the performance of standard transformers at reasoning-related tasks.

Introduction

Large Language Models (LLMs) (Brown et al. 2020) have revolutionized the field of AI, enabling applications in areas as diverse as Natural Language Processing (Min et al. 2024), Medicine (Thirunavukarasu et al. 2023), Finance (Li et al. 2023) and Education (Kasneci et al. 2023). Nonetheless, these impressive results do not generalize well to System-2 tasks (Kahneman 2011), i.e., tasks that require reasoning in order to be addressed effectively, such as generating complex code (Du et al. 2024), solving math problems (Mirzadeh et al. 2024) and planning (Valmeekam, Stechly, and Kambhampati 2024). In the past few years, much effort has been devoted to improving the reasoning abilities of LLMs, i.e., to teaching LLMs to reason. As a result, there has been a surge of new approaches such as Chain-of-thought (Wei et al. 2022), Self-refinement (Madaan et al. 2024) and Reflection (Shinn et al. 2024), which have paved the way for the development of Large Reasoning Models (LRMs) like Open AI's recent o1 (codename *Strawberry*) (OpenAI 2024).

In this preliminary work, we choose a different approach for *learning to reason*: instead of performing reasoning at the *LLM-level*, we propose to do so at the *transformer-level*. We build upon the hypothesis that ***any reasoning process can be formulated as search*** and introduce the **Search Transformer**, an enhancement of the popular transformer architecture with a *search inductive bias* and adaptive test-time computation (Snell et al. 2024). Search Transformers learn to carry out an explicit search process in order to solve the task provided as input. This search process is formulated in terms of differentiable, attention-based operations, thus enabling end-to-end training through back-propagation.

Our approach can be regarded as an extension of Universal Transformers (Dehghani et al. 2019). However, instead of recursively processing each token, we propose to incorporate a stronger inductive bias for learning a search procedure. Additionally, there exist other approaches for learning to reason (Veličković and Blundell 2021) (e.g., Neural Turing Machines (Graves, Wayne, and Danihelka 2014) and Neural Logic Machines (Dong et al. 2019)) and, more specifically, for learning to plan (Núñez-Molina, Mesejo, and Fernández-Olivares 2024) (e.g., Value Iteration Networks (Tamar et al. 2016) and Action Schema Networks (Toyer et al. 2018)). Unlike these methods, we propose to leverage transformer-like attention operations, which are powerful enough to tackle a wide range of tasks and can be adapted to many different task encodings (e.g., natural language), not requiring formal descriptions such as PDDL (Haslum et al. 2019).

Alternatively, there exist many approaches for improving the reasoning capabilities of LLMs, either during the pretraining/finetuning process (e.g., STaR (Zelikman et al. 2022), Refiner (Paul et al. 2023) and o1 (OpenAI 2024)), or by performing in-context learning (e.g., Chain-of-thought (Wei et al. 2022), Self-refinement (Madaan et al. 2024) and RAP (Hao et al. 2023)). Nonetheless, most of these techniques only work on LLMs beyond a certain size (Plaat et al. 2024), for which training and inference is computationally expensive. For instance, it is often much more efficient to solve an Automated Planning (AP) task with an ad-hoc planner instead of an LRM (Valmeekam, Stechly, and Kambhampati 2024). Therefore, we propose Search Transformers as a way of exploiting the power of attention without incurring into the computational costs of LLMs.

Finally, recent works (Lehnert et al. 2024; Gandhi et al.

2024; Su et al. 2024; Saha et al. 2024) train transformers to search by mimicking the operation of a search algorithm like A*. Unlike our approach, these methods are trained on search/reasoning traces containing the nodes expanded by the teacher search algorithm to solve each training example. Conversely, we propose to train Search Transformers solely on input-output pairs, not requiring example search traces. This makes our method applicable to tasks where this information is unavailable, e.g., when the world model/system dynamics are unknown so no search algorithm can be run to obtain example search traces.

We devote the rest of this paper to the design of Search Transformers, explaining how attention can be used to learn and carry out a search process. In future work, we will implement our proposal and compare it with other learn-to-reason approaches on several reasoning benchmarks, including AP tasks.

Proposal: Search Transformers

The main goal of our work is to improve the reasoning abilities of transformers. In order to achieve this, we propose two extensions to the transformer architecture. First, the use of adaptive test-time computation, i.e., allowing the model to perform variable amounts of computation depending on the input. The intuition behind this is simple: hard tasks require more computational effort (i.e., *thinking time*) than easy tasks in order to be successfully solved. Second, we propose to incorporate a *search inductive bias* into transformers. We hypothesize that any reasoning process can be formulated in terms of search. Therefore, we believe that teaching transformers to conduct a search process should be beneficial over alternative test-time computation approaches with weaker inductive biases (e.g., simple token-level recurrence as in Universal Transformers).

Search Transformers simultaneously learn an action/-world model and how to plan/search over it. Therefore, they are similar to learn-to-plan approaches such as TreeQN (Farquhar et al. 2018) and DRC (Guez et al. 2019), although these models only tackle Sequential Decision Making problems. The meaning of this action model depends on the task being solved. For instance, in Sequential Decision Making (e.g., AP) tasks, it would encode the actions available to the agent and how they affect the state of the world. When solving a math equation, it could encode a set of operators/transformations (e.g., dividing both sides by the same number) and how they affect the equation. If performing logical deduction, it may encode a set of logical rules to deduce new facts from a set of true premises. In addition to the action model, a search process requires an initial state and goal to achieve. In the case of Search Transformers, both items are encoded in the tokens provided as inputs.

We propose to train our approach in an end-to-end fashion by back-propagating the gradients of a loss function. To do so, we can resort to the same training procedures as standard transformers. For instance, Search Transformers could be trained in a self-supervised fashion using teacher-forcing. As an alternative, we could train them in a supervised manner on a dataset composed of input-output pairs, e.g., comprising questions with their correct answers or task descriptions with their solutions.

Search Transformers receive as inputs a sequence of tokens describing a task to solve, e.g., "Solve the following equation: $3x + 5 = 20$". Then, they carry out a search process (see Figure 1.a1-a3) in order to discover useful information for solving the task. Once this search has concluded, the expanded nodes are provided as additional inputs to a standard transformer, which we refer to as the **Output Transformer (OT)** (see Figure 1.b). The OT is used to autoregressively predict the output tokens from the input tokens and search nodes. Therefore, search nodes incorporate additional information into the context window of the transformer, in a similar fashion to in-context learning approaches like few-shot prompting. In our previous example, nodes could encode the sequence of mathematical derivations needed to solve the equation: $3x + 5 = 20 \rightarrow 3x = 15 \rightarrow x = 5$. Nonetheless, unlike information added by in-context learning methods, search nodes are not represented as tokens but directly as latent-space embeddings, thus following the approach proposed in (Hao et al. 2024).

The search process conducted by Search Transformers starts from an initial node n_1 with a predefined, constant embedding (e.g., full of zeroes). Then, at each step, the following three operations are applied in sequence: 1) node and action selection, 2) node transition, and 3) next node generation. Steps 1 to 3 are applied iteratively, until the halting condition is met (see subsection below). We now explain these operations in detail:

Node and action selection. At each step, the **Expansion Transformer (ET)** receives the input tokens and nodes expanded so far in the search and, for each node n_i, predicts the probability p_i of it being expanded next and the action a_i to apply to it (see Figure 1.a1). The ET utilizes a standard transformer architecture and does not share weights with the OT. We propose to use a transformer, instead of a different neural network, so that each node can attend to the other nodes when "deciding" *if* and *how* it should be expanded. Therefore, the ET leverages the current global state of the search process when computing the actions and expansion probabilities. This is analogous to non-markovian/history-dependent policies in Reinforcement Learning, which look at past states and actions when deciding the action to apply next. Just as the OT, the ET also receives the input tokens in addition to the search nodes. These tokens serve to provide additional context to the ET, encoding crucial information about the task, such as its initial state and goal. Were they not provided, this information would need to be encoded in the search nodes, thus potentially hindering the learning process.

Node transition. Next, the **Transition Transformer (TT)** receives the input tokens along with the sequence of search nodes and the actions predicted by the ET. For each node-action pair (n_i, a_i), it predicts the next node n_i' resulting from the application of a_i at n_i (see Figure 1.a2). Therefore, the goal of the TT is to learn the action model (also known as world model or transition function). Analogously to the OT and ET, the TT utilizes a standard transformer ar-

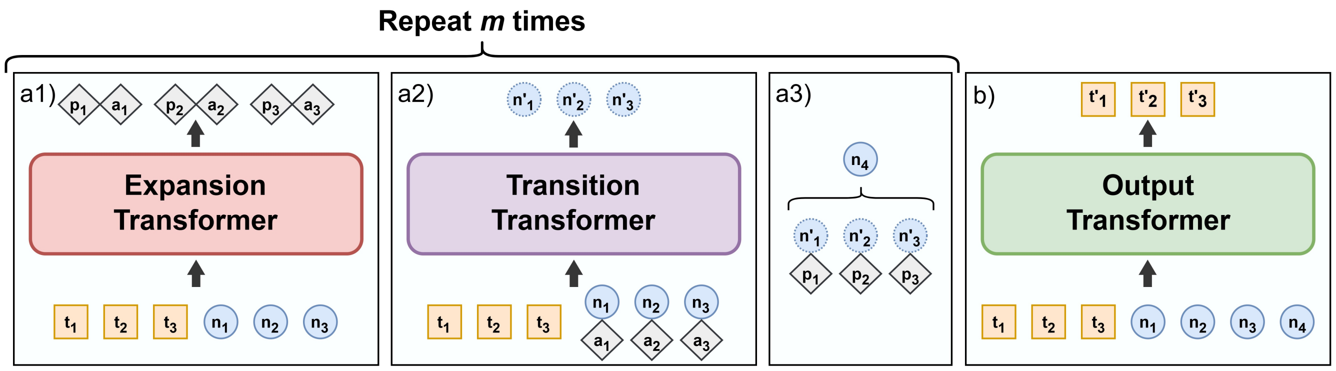

Figure 1: **Search Transformer architecture.** Search transformers carry out two different phases when solving a task: **a) search** and **b) output prediction**. The search phase is composed of three steps (a1 to a3), whose application results in a new search node n_i each time. By iteratively applying steps a1-a3 m times (the figure assumes $m = 4$), we obtain the sequence of search nodes $n_1, ..., n_m$, which encode useful information about the task to solve. We now explain these steps in detail. **a1)** At each search iteration, the **Expansion Transformer (ET)** receives the input tokens and search nodes expanded so far. For each node n_i, it predicts the probability p_i of the node being expanded next and the action a_i to apply to it. **a2)** Next, the **Transition Transformer (TT)** receives the input tokens alongside the sequence of nodes $n_1, ..., n_m$ and predicted actions $a_1, ..., a_m$. For each node n_i, it predicts the next node n'_i resulting from the application of action a_i. **a3)** Finally, the next node n_{m+1} of the search is obtained from the sequence of next node candidates $n'_1, ..., n'_m$. At test time, we sample a candidate node according to probabilities $p_1, ..., p_m$ whereas, at training time, n_{m+1} is obtained as a weighted average or with the Gumbel-softmax trick (Jang, Gu, and Poole 2017). Once search has concluded, the output prediction phase (b) takes place. The **Output Transformer (OT)** receives the input tokens $t_1, ..., t_n$ and nodes $n_1, ..., n_m$ expanded during the search, and predicts the output tokens $t'_1, ..., t'_n$ autoregressively.

chitecture and does not share weights with them. Moreover, it also receives the input tokens as additional context. In order to allow each (n_i, a_i) pair to attend to the input tokens when predicting n'_i, we decided to encode the action model as a transformer instead of as an alternative model (e.g., a multilayer perceptron). Nonetheless, we forbid each (n_i, a_i) pair from attending to other (n_j, a_j) pairs. This encodes the strong inductive bias that the next node n'_i only depends on the current node n_i and action a_i applied to it, being independent from other nodes. In other words, we assume the action model is markovian, i.e., it does not depend on the past history of actions and states (nodes). Finally, most transformers make use of position encodings to represent the position/index of each token in the input sequence. In the case of a search process, we do not care about the index of each node but, rather, about the topology of the search tree, i.e., for each node we may want to know its parent and the action that was applied to obtain it. Instead of manually encoding this information into each node embedding, we let the TT learn it automatically, in analogous fashion to learnable position encodings in transformers.

Next node generation. In the previous step, the TT obtained for each node n_i its corresponding next node n'_i, according to the action a_i selected by the ET. However, we should only expand a single node at each iteration of the search process to prevent exponential growth in the number of nodes. For this reason, we need to aggregate all the next node candidates $n'_1, ..., n'_m$ into a single next node n_{m+1}, which then will be appended to the sequence of expanded nodes $n_1, ..., n_m$ (see Figure 1.a3). At test time, this is straightforward to do: we can simply sample a node from $n'_1, ..., n'_m$ according to the expansion probabilities $p_1, ..., p_m$ predicted by the ET. At training time, however, a different approach is needed, as this sampling process is not differentiable. The simplest approach would be to obtain n_{m+1} as the weighted average of $n'_1, ..., n'_m$, where the aggregation weights would be given by the probabilities $p_1, ..., p_m$. Nonetheless, we believe a better approach would be to utilize the Gumbel-sofmax trick (Jang, Gu, and Poole 2017) to approximate sampling a node from $n'_1, ..., n'_m$ in such a way gradients can flow through.

Halting the Search

As explained above, Search Transformers carry out a search process in order to obtain a sequence of nodes $n_1, ..., n_m$ that are provided to the OT as additional information for solving the task. An integral part of this process is deciding when to halt the search. Intuitively, the more nodes are expanded, the better the accuracy/quality of the solution predicted by the OT should be, as more useful information about the task can be discovered during the search. For instance, when solving a mathematical equation, Search Transformers could explore all possible sequences of mathematical transformations until one that solves the equation is found, in a process analogous to brute-force search. However, expanding more nodes means incurring in higher computational costs. For this reason, it is of utmost importance to expand *just the right number* of nodes required to solve the task, thus achieving a balance between accuracy and computational efficiency. The optimal number of nodes to expand should vary from task to task, as harder tasks generally require more nodes (i.e., more computation) in order to be solved successfully. Therefore, Search Transformers should learn to adapt their computational budget (i.e., number of expanded nodes) to the task provided as input, in what is known as *adaptive test-time computation*.

In order to achieve this, we propose an approach inspired by the method in (Graves 2016). We define the *utility* u_i of a node n_i as a measure of how useful it is for the OT to solve the task. Utility can be seen as a continuous approximation of the concept of a goal: whereas a node is either a goal or not, its utility is a real value between 0 and 1, where 0 means the node contains no useful information at all and 1 corresponds to maximum utility. Although goal nodes should be assigned high utility, this measure is not limited to them. For example, nodes contained in the path from the initial node to the goal may also exhibit high utility (e.g., in case we need to explain how the goal was reached), or nodes corresponding to dead-ends, which could be useful for providing contrastive explanations (Stepin et al. 2021). Based on this notion of utility, we define the *utility threshold* U as the total utility that must be attained in order to solve the input task successfully. During search, we expand nodes $n_1, ..., n_m$ until the sum of their utilities $\sum_{i=1}^{m} u_i$ reaches U, at which point we halt the search and provide the sequence of nodes to the OT. Therefore, this parameter U controls the computational budget of the search: the higher its value is, the more nodes are expanded on average.

Now, an important question arises: how can we estimate these utility values? Our proposal is to approximate the utility u_i of a node n_i as its (normalized) attention score by the OT. This attention score measures how much the OT *attends to* (i.e., relies on) n_i when predicting the output tokens so, the higher this value is, the more useful n_i should be for solving the task (since the OT would not attend to n_i otherwise). An important limitation of this utility definition, however, is that it can only be computed *after* the search has concluded and all nodes $n_1, ..., n_m$ have been expanded. In contrast, we would need to have access to the node utility values $u_1, ..., u_m$ *during* the search, in order to decide when to halt it. We propose to overcome this limitation by training the TT to also predict the utility u'_i of next node candidates n'_i as they are generated, so this information is available during search.

At training time, we propose to set the utility threshold U to a high value so a large number of nodes are expanded (this would be analogous to performing *exploration* in Reinforcement Learning). For instance, if we set $U = 2$, then the Search Transformer would expand twice the number of nodes it deems necessary for solving the task. Given the sequence $n_1, ..., n_m$ of expanded nodes, we can then retrospectively determine the subsequence $n_1, ..., n_k$ ($k \leq m$) of nodes that should have been expanded, in order to attain a better balance between accuracy and efficiency. Let us provide a concrete example. Assume the model expands five nodes n_1, n_2, n_3, n_4, n_5, whose ground-truth utility values (i.e., OT attention scores, different from *predicted* utilities) are $0.4, 0.05, 0.3, 0.15, 0.1$ (note how these values add up to 1, since they are normalized). It can be observed that nodes n_1 and n_3 concentrate most of the utility, with values of 0.4 and 0.3, respectively. Therefore, it makes sense to halt the search after expanding node n_3, since nodes n_4 and n_5 have low utility values and, thus, provide little useful information to the OT. There exist many different alternatives for deciding at which node to stop the search. We propose a simple approach where we halt the search at the first node n_k whose ground-truth *cumulative utility* $\sum_{i=1}^{k} u_i$, analogous to the cumulative distribution function *cdf* in statistics, surpasses some predefined value $\tau \in (0, 1)$. In our previous example, if we set $\tau = 0.5$, then $n_k = n_3$.

Once we have obtained the subsequence $n_1, ..., n_k$ of nodes that should have been expanded, we re-normalize their ground-truth utilities so that they add up to 1 again, obtaining a new sequence of utility values $u_1^*, ..., u_k^*$. Then, the TT is trained to predict u_i^* as the utility of each node n_i, e.g., by minimizing the Mean Squared Error (MSE) loss. This is equivalent to training the TT to halt the search after node n_k, assuming $U = 1$. At test-time, U acts as a hyperparameter for dynamically controlling the (expected) number of expanded nodes, thus balancing accuracy and efficiency.

Conclusion

In this preliminary work, we have proposed Search Transformers, an extension of the popular transformer architecture for *learning to reason*. Search Transformers learn to carry out a search process in order to discover useful information about the task, which is then provided as additional context when predicting the output tokens. Search operators (e.g., node selection and successor generation) are formulated in terms of attention-based operations, thus allowing end-to-end training via back-propagation. Furthermore, we devised a differentiable method for learning when to halt the search, which allows Search Transformers to perform *adaptive test-time computation* (Snell et al. 2024).

In future work, we plan to implement our method and compare it with alternative approaches, e.g., standard transformers, Universal Transformers and other learn-to-reason models, evaluating their performance on several reasoning benchmarks such as Sequential Decision Making, math and logical deduction. We will also explore how the ideas introduced in this work can be extended to incorporate additional prior knowledge and stronger, symbolic inductive biases into our model. One possibility is to constrain Search Transformers to follow a particular search strategy. For instance, by always expanding the last node, they would learn to carry out a depth-first search. A different, but compatible, approach is to impose some kind of structure on latent representations. For example, we could constrain node embeddings to utilize discrete variables (Van Den Oord, Vinyals et al. 2017), in a manner reminiscent of symbolic representations like PDDL.

To conclude, we hope that the ideas presented in this work will pave the way towards the development of better learn-to-reason approaches. We envision such methods becoming the backbone of LLMs and other foundational models in the future, allowing them to adquire both System-1 and System-2 capabilities during their extensive pretraining process, instead of having to resort to post-hoc reasoning techniques such as Chain-of-Thought or Self-refinement.

Acknowledgements

This work has been partially funded by the Grant PID2022-142976OB-I00, funded by MICIU/AEI/ 10.13039/501100011033 and by "ERDF/EU", as well as the Andalusian Regional predoctoral grant no. 21-111-PREDOC-0039 and by "ESF Investing in your future".

References

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D. M.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. In *Advances in neural information processing systems*.

Dehghani, M.; Gouws, S.; Vinyals, O.; Uszkoreit, J.; and Kaiser, L. 2019. Universal Transformers. In *7th International Conference on Learning Representations*.

Dong, H.; Mao, J.; Lin, T.; Wang, C.; Li, L.; and Zhou, D. 2019. Neural Logic Machines. In *7th International Conference on Learning Representations*.

Du, X.; Liu, M.; Wang, K.; Wang, H.; Liu, J.; Chen, Y.; Feng, J.; Sha, C.; Peng, X.; and Lou, Y. 2024. Evaluating large language models in class-level code generation. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 1–13.

Farquhar, G.; Rocktäschel, T.; Igl, M.; and Whiteson, S. 2018. TreeQN and ATreeC: Differentiable Tree-Structured Models for Deep Reinforcement Learning. In *6th International Conference on Learning Representations*.

Gandhi, K.; Lee, D.; Grand, G.; Liu, M.; Cheng, W.; Sharma, A.; and Goodman, N. D. 2024. Stream of Search (SoS): Learning to Search in Language. *arXiv*.

Graves, A. 2016. Adaptive computation time for recurrent neural networks. *arXiv*.

Graves, A.; Wayne, G.; and Danihelka, I. 2014. Neural Turing Machines. *arXiv*.

Guez, A.; Mirza, M.; Gregor, K.; Kabra, R.; Racanière, S.; Weber, T.; Raposo, D.; Santoro, A.; Orseau, L.; Eccles, T.; et al. 2019. An investigation of model-free planning. In *International Conference on Machine Learning*, 2464–2473.

Hao, S.; Gu, Y.; Ma, H.; Hong, J. J.; Wang, Z.; Wang, D. Z.; and Hu, Z. 2023. Reasoning with Language Model is Planning with World Model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 8154–8173.

Hao, S.; Sukhbaatar, S.; Su, D.; Li, X.; Hu, Z.; Weston, J.; and Tian, Y. 2024. Training Large Language Models to Reason in a Continuous Latent Space. *arXiv*.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; Muise, C.; Brachman, R.; Rossi, F.; and Stone, P. 2019. *An introduction to the planning domain definition language*, volume 13. Springer.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical Reparameterization with Gumbel-Softmax. In *5th International Conference on Learning Representations*.

Kahneman, D. 2011. *Thinking, fast and slow*. Farrar, Straus and Giroux.

Kasneci, E.; Seßler, K.; Küchemann, S.; Bannert, M.; Dementieva, D.; Fischer, F.; Gasser, U.; Groh, G.; Günnemann, S.; Hüllermeier, E.; et al. 2023. ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and individual differences*, 103: 102274.

Lehnert, L.; Sukhbaatar, S.; Su, D.; Zheng, Q.; Mcvay, P.; Rabbat, M.; and Tian, Y. 2024. Beyond A*: Better planning with transformers via search dynamics bootstrapping. *arXiv*.

Li, Y.; Wang, S.; Ding, H.; and Chen, H. 2023. Large language models in finance: A survey. In *Proceedings of the fourth ACM international conference on AI in finance*, 374–382.

Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Min, B.; Ross, H.; Sulem, E.; Veyseh, A. P. B.; Nguyen, T. H.; Sainz, O.; Agirre, E.; Heintz, I.; and Roth, D. 2024. Recent Advances in Natural Language Processing via Large Pre-trained Language Models: A Survey. *ACM Computing Surveys*, 56(2): 30:1–30:40.

Mirzadeh, I.; Alizadeh, K.; Shahrokhi, H.; Tuzel, O.; Bengio, S.; and Farajtabar, M. 2024. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv*.

Núñez-Molina, C.; Mesejo, P.; and Fernández-Olivares, J. 2024. A review of symbolic, subsymbolic and hybrid methods for sequential decision making. *ACM Computing Surveys*, 56(11): 1–36.

OpenAI. 2024. Learning to reason with LLMs.

Paul, D.; Ismayilzada, M.; Peyrard, M.; Borges, B.; Bosselut, A.; West, R.; and Faltings, B. 2023. Refiner: Reasoning feedback on intermediate representations. *arXiv*.

Plaat, A.; Wong, A.; Verberne, S.; Broekens, J.; van Stein, N.; and Back, T. 2024. Reasoning with large language models, a survey. *arXiv*.

Saha, S.; Prasad, A.; Chen, J. C.-Y.; Hase, P.; Stengel-Eskin, E.; and Bansal, M. 2024. System-1. x: Learning to balance fast and slow planning with language models. *arXiv*.

Shinn, N.; Cassano, F.; Gopinath, A.; Narasimhan, K.; and Yao, S. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Snell, C.; Lee, J.; Xu, K.; and Kumar, A. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv*.

Stepin, I.; Alonso, J. M.; Catala, A.; and Pereira-Fariña, M. 2021. A survey of contrastive and counterfactual explanation generation methods for explainable artificial intelligence. *IEEE Access*, 9: 11974–12001.

Su, D.; Sukhbaatar, S.; Rabbat, M.; Tian, Y.; and Zheng, Q. 2024. Dualformer: Controllable fast and slow thinking by learning with randomized reasoning traces. *arXiv*.

Tamar, A.; Wu, Y.; Thomas, G.; Levine, S.; and Abbeel, P. 2016. Value iteration networks. *Advances in neural information processing systems*, 29.

Thirunavukarasu, A. J.; Ting, D. S. J.; Elangovan, K.; Gutierrez, L.; Tan, T. F.; and Ting, D. S. W. 2023. Large language models in medicine. *Nature medicine*, 29(8): 1930–1940.

Toyer, S.; Trevizan, F.; Thiébaux, S.; and Xie, L. 2018. Action schema networks: Generalised policies with deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Valmeekam, K.; Stechly, K.; and Kambhampati, S. 2024. LLMs Still Can't Plan; Can LRMs? A Preliminary Evaluation of OpenAI's o1 on PlanBench. *arXiv*.

Van Den Oord, A.; Vinyals, O.; et al. 2017. Neural discrete representation learning. *Advances in neural information processing systems*, 30.

Veličković, P.; and Blundell, C. 2021. Neural algorithmic reasoning. *Patterns*, 2(7).

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837.

Zelikman, E.; Wu, Y.; Mu, J.; and Goodman, N. 2022. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35: 15476–15488.