Meta-Learning with Long-Context Transformers: Algorithm Distillation

Jake Grigsby
jg75962
grigsby@cs.utexas.edu

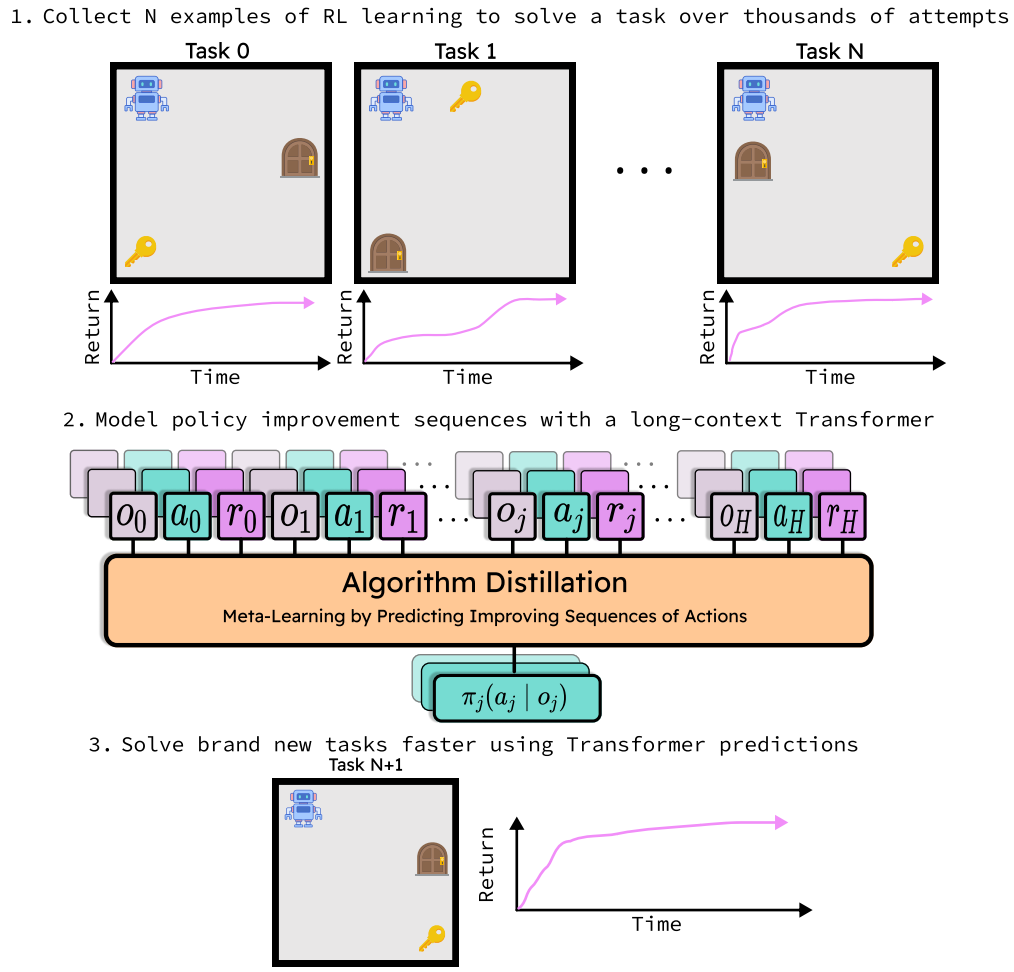

Figure 1: **Algorithm Distillation learns a self-improving decision-making policy that can adapt to new environments without explicit Reinforcement Learning**. Instead, a Transformer is trained to predict sequences of improving actions collected by many independent RL agents. By learning to predict an improving RL agent's next action, our Transformer is not imitating optimal *behavior* but instead imitating the *algorithm* by which that behavior is learned.

1 Introduction

While Transformers [15] originated in Natural Language Processing problems like translation, and are now most popularly used in Large Language Models [4] like GPT-4 [13], sequence modeling can also be applied in domains like robotics and decision-making [3, 5]. Rather than predicting the

continuation of a discrete sequence of language tokens, decision-making data involves predicting actions from a history of continuous observations of an agent's surroundings. However, unlike Reinforcement Learning (RL) models, sequence modeling in a decision-making context cannot learn to improve on its own; we can only learn to imitate the behavior of the data our model was trained on. Ideally we could find *meta-learning* algorithm that "learns to learn" in previously unseen environments [2], but the objective that Transformers are trained to optimize creates a *behavior cloning* (BC) algorithm that fails in novel situations. Algorithm Distillation (AD) [11] is a recent solution to this problem that preserves the sequence modeling objective and Transformer architecture of imitation learning, but cleverly changes the dataset the model is trained on such that the resulting network is actually a meta-learner that improves in new environments automatically. Instead of training on demonstrations of optimal behavior in an environment, AD learns to imitate the sequences of improving performance that are generated by deep RL algorithms. This way, when our agent encounters new environments, the actions we predict imitate the exploration and improvement of the source RL algorithm, and can often mimic its learning process over far fewer timesteps. In other words, our Transformer policy is able to "in-context learn" by using the history of environment interactions in its sequence input to improve its behavior. An overview of the three main steps of the AD algorithm is provided in Figure 1.

This project has two main contributions. First, we create the first (to our knowledge) fully-featured open-source replication of AD. Second, we dive deeper into how AD's change in data format leads to in-context learning as opposed to behavior cloning by ablating this decision in both a meta-learning and multi-task learning setting.

2 Meta-Learning Environment

We choose to replicate the second-most-difficult environment in the original AD experiments:"Dark-Key-To-Door" (visualized in Figure 1). An agent navigates a room that contains a key and door. The agent receives a reward of +1 for navigating to the key, and another reward of +1 for unlocking the door with that key. The optimal total score or "return" is 2, and episodes terminate upon unlocking the door or after 50 timesteps. Although this domain is trivial by the standards of deep RL, this simplicity is offset by AD's incredibly expensive data collection process, which we will discuss in the next section. Importantly, the agent cannot see the location of the key and door, which creates a meta-learning problem where we need to explore the environment to find their position and then exploit this knowledge to learn a high-performance policy. The partial observability of the environment is what separates meta-learning from multi-task learning; if the agent could see the key and door positions, we could learn a single policy that zero-shot generalizes to any new position by training on many key/door combinations [1, 14, 10]. Instead, meta-learning requires optimization over long durations of multiple trials. We are unlikely to find the goal on the first attempt, and RL takes many (thousands) of trials to solve the task. However, the fully-observed multi-task version with key and door information revealed will be a key baseline for us to contrast in-context learning from behavior cloning. We will call this easier setting "*Light*-Key-To-Door" (referring to the agent's ability to "see" the objects in the room).

3 Source RL Data Creation

Imitation learning for decision-making with a sequence model like a Transformer normally follows three simple steps. First, we train a single RL agent to convergence in our environment. Second, we use the agent to demonstrate optimal behavior, saving observation/action pairs (o_t, a_t) as training examples. Finally, we use the expert dataset to train a Transformer to map sequences of observations to action labels. However, this process creates an agent that can only imitate the decisions of the expert in the environment the expert was trained on. Using the environments depicted in Figure 1 as an example, if our expert demonstrator was trained on the key/door positions in task 1, our resulting Transformer imitator would be unable to solve task N+1.

The magic of Algorithm Distillation lies in preserving the well-established Transformer sequence modeling objective, but adjusting the input dataset such that the same training process induces a meta-learner that can adapt to task N+1. Instead of training one RL agent in one environment, AD trains N independent agents in N different environments. Rather than saving the converged "expert" behavior of those N agents, AD saves the entire history of experience that was used by the RL

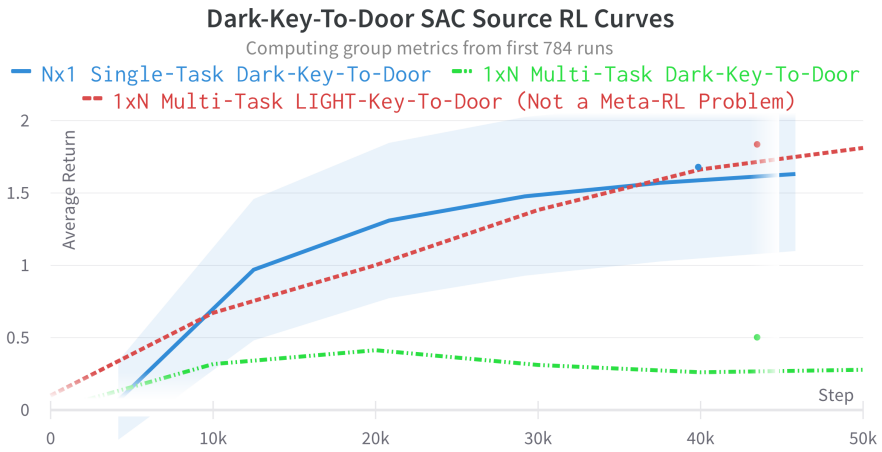

Figure 2: **Learning curves of source RL agents used to collect our sequence datasets**. Blue is AD-Style Nx1. Green is BC-Style 1xN, which fails and highlights the advantage of meta-learning. Red is BC-Style 1xN in the multi-task Light-Key-To-Door environment where meta-learning is not necessary and the policy can directly generalize to every new situation.

algorithm (observation o_t, action a_t, reward r_t and episode termination signal d_t) as it improves over time. The correct action label at each timestep is not determined by expert behavior but by the dynamics of the RL algorithm as it learns from experience. It follows that by learning to predict those actions, a Transformer would actually be learning to "distill" the "source" RL algorithm itself. The Transformer can now succeed in new room N+1 because rather than copying the actions of an expert that has never seen this particular task, it is copying the process by which that expert is generated. The downside of AD is its need to train N-1 more RL agents than regular imitation learning. N is on the order of thousands of tasks to enable generalization to new environments, and training thousands of deep RL agents from scratch can be costly even on a problem as simple as Dark-Key-To-Door.

We reduce the cost of AD dataset collection by roughly 1000x by replacing the sample inefficient A3C [12] used in the original results with a custom highly-tuned variant of SAC [9]. Each individual training run of our SAC agent on a Dark-Key-To-Door task can be completed in $50,000$ timesteps and less than 10 minutes. In order to investigate the way AD's change in dataset format leads to in-context learning, we actually go further than the original AD paper and collect 5 datasets:

- **AD-Style (Nx1): Train N agents on 1 task**. Each training run samples a fixed key and door location that never change. SAC is run for 50k timesteps. Agents improve quickly and the majority of the runs fully solve their task. We train more than 770 runs, which is less than half of AD but should be enough to see reasonable performance according to AD Figure 12 [11].
- **BC-Style (1xN): Train 1 agent on N different tasks**. We run one SAC agent that randomly samples a new key and door location between every environment reset. This means it sees many different tasks during training. We leave it running for 100k timesteps to prevent overfitting of BC on such a simple task. This agent fails, and it's not an issue with SAC: the environment is not solvable in a multi-task format. The key and door are in different (unknown) places each episode. The best the agent can do is wander around randomly and at least not waste time running into walls. We are trying to solve a meta-RL problem, but we are not using a meta-RL agent; AD creates a meta-RL agent out of individual training runs.
- **BC-Style (1xN) Light-Key-To-Door**. Same as above, but we use the non-meta-RL version of the environment. The agent can see the key and door location and directly generalize. The agent learns quickly and converges to a nearly perfect policy. We collect 100k steps to be safe from overfitting. The issue with the BC-style dataset above is not the 1xN training schedule, but a fundamental difference between meta-RL and multi-task-RL.

- **AD-Style (Nx1) Light-Key-To-Door**. Same as Nx1 AD but this time we add key and door information. This has no effect on data collection because each individual agent never sees the key and door location information change, so they might as well not be an input to the network. We take advantage of this to save several hundred GPU hours of compute by only collecting the Light-Key-To-Door version of the AD dataset. Because the key/door inputs do not effect the individual RL algorithms we can discard this information to recover the Dark-Key-To-Door version. We will use this dataset to show how the learning ability of AD changes when its training environments are multi-task rather than meta-learning problems.
- **RL^2 [7, 17] (1xN): Train 1 meta-learning agent on N different tasks**. The key and door are randomized so that the agent sees many tasks, but the locations are held fixed for 10 consecutive episodes. We use a Transformer in RL^2 instead of an RNN. RL^2 is effective and meta-learns quickly on this task. The difference between 1xN SAC (BC-Style) and 1xN RL^2 is that RL^2 is a meta-learning agent already and we do not need to distill a meta-learner from its data.

Each dataset consists of sequences of environment interactions with increasing performance. We can track performance improvement by monitoring the average score of agents throughout training, visualized in Figure 2. The performance RL^2 is shown in the experiments section.

4 Transformer Sequence Modeling

The main advantage of AD is that once we have collected our dataset the training process mirrors the standard causal Transformer sequence prediction routine used in NLP, time series forecasting, computer vision, and many other areas of machine learning. Given a sequence of RL data $(o_{t-c}, a_{t-c}, r_{t-c}, d_{t-c}, \dots, o_{t-1}, a_{t-1}, r_{t-1}, d_{t-1}, o_t)$ we learn to predict the next action a_t. Here c denotes the "context length" and creates a tradeoff between compute and performance. Longer context sequences allow for more accurate modelling of the way the source RL algorithm (in this case, SAC) uses past experience to compute improved actions. However, this comes at the cost of higher runtime and memory requirements for Transformers' self-attention mechanism. We use a pre-norm Transformer architecture [18] with a linear learning rate warmup and learnable position embeddings [16]. FlashAttention [6] is included in the open-source codebase for future scalability but is not necessary at the sequence lengths demonstrated by the official AD results. The official AD uses 4 Transformer layers with a small embedding dimension of 64 and a large feedforward dimension of 2048, which is significantly out-of-proportion to the literature default [15]. These settings are essentially stating that attention across the sequence is less important than feedforward memorization of minor variations [8]. We decide to take this opportunity to test whether this imbalance is an important detail and use a more standard architecture with embedding size 128 and feedforward dimension 768. The architecture, training loop, and most hyperparameters are fixed for all experiments. Our goal is to investigate the results of varying the input dataset and context length.

5 Experiments

We begin by replicating the key result of AD and demonstrating in-context learning in previously unseen environments. The agent begins in an initial state which is passed through our trained Transformer to produce an action. That action is taken in the environment, leading to a reward and a new state. The chosen action, reward, and new state are appended to the end of our input sequence and fed back into the Transformer to get the next action. This process repeats endlessly, trimming the sequence at the maximum context length when necessary and restarting the environment with the same key and door location upon episode termination. Because the agent does not know where the key and door are located, initial performance will be quite low. However, a successful in-context learner would gradually improve over many consecutive trials, reaching a final score of roughly 2 points per episode. We plot this information over the course of $17,500$ timesteps and show that our replication of AD can successfully solve new environments after a few thousand steps (Figure 3) with sufficiently long context lengths. AD can often outperform its source RL algorithm, meaning it converges to the optimal policy in fewer timesteps than the agent that was used to collect its dataset (e.g., compare Figure 3 right to Figure 2 blue). However, our SAC agent is so sample efficient that it is harder to measure this effect here than in the official AD results (which use A3C).

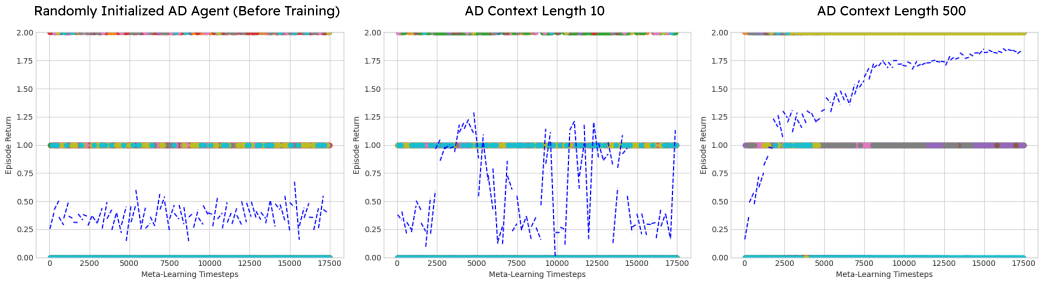

Figure 3: **In-Context Learning Emerges in AD**. At long enough context lengths, AD learns to improve in new environments by observing the history of past timesteps and imitating the improvement of SAC. Compare with official AD [11] Figure 4. Our version of this figure plots a line-of-best-fit across a scatter plot of returns in 20 different environments.

Figure 3 shows that in-context learning emerges somewhere between sequence lengths of 10 and 500 timesteps. The implication is that 10 timesteps (20% of a typical episode) is not enough information to infer how the source RL agent would adapt to the current environment, but 500 timesteps (at least 10 full episodes) is. This is already interesting because SAC is an off-policy algorithm that technically uses *all* past timesteps to develop its next action, and yet the most recent 500 timesteps appears to be a good approximation of its behavior. We explore this further in Figure 4 by varying the context length between 10 and 500. We find that 500 timesteps is actually far more than enough, with in-context learning emerging somewhere between $30 - 50$ timesteps.

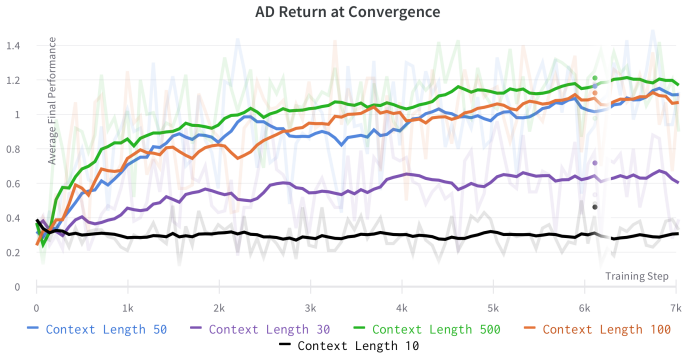

Figure 4: **Comparing AD Context Lengths**. Final performance (at the end of the blue curve in Figure 3) of AD throughout training at various context lengths. In-context learning emerges somewhere between a context length of 30 and 50, with diminishing rate of return. Compare with official AD [11] Figure 7.

.

Next we use our additional dataset types to highlight the way N independent RL improvement sequences create a meta-learner out of what would otherwise be a behavior cloning training loop. When the same Transformer as used in our AD experiments is trained with a standard BC dataset, the policy is unable to generalize to new environments (Figure 5 left). However, if we use the Light-Key-To-Door BC dataset - removing the meta-learning component of the problem - we can simply copy the behavior of the expert and achieve nearly perfect performance from the first attempt (Figure 5 center). Note that this is not due to any in-context learning effect because a context length of just 2 achieves the same results (Figure 5 right).

While we have shown that the generalization to new environments in Dark-Key-To-Door requires N independent examples of RL improvement, we can take this comparison another step further: is the in-context learning effect a result of training independent agents, or training independent agents *in a meta-RL setting*? We train AD with the Light-Key-To-Door dataset variant. Even though we are learning to predict the behavior of different agents, the key and door location information in the

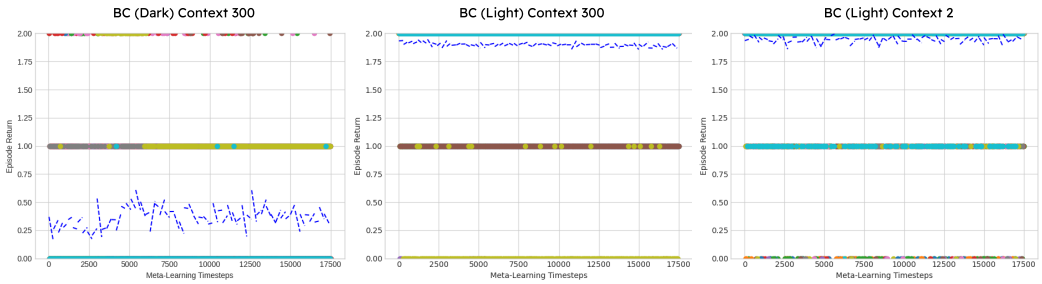

Figure 5: **Behavior Cloning Results**. BC on Dark-Key-To-Door fails because it is not a meta-learning algorithm **(left)**. BC trained on multi-task Light-Key-To-Door zero-shot generalizes easily **(center)**. However, this is not due to any in-context learning as a context length of 2 performs just as well **(right)**.

observation ties the experience together in a way that makes our Transformer learn a BC-style policy that succeeds instantly at any context length (Figure 6 left and center). In other words, the dataset has to be a meta-learning problem to force the sequence model to distill the algorithm rather than simply copying the best demonstrations.

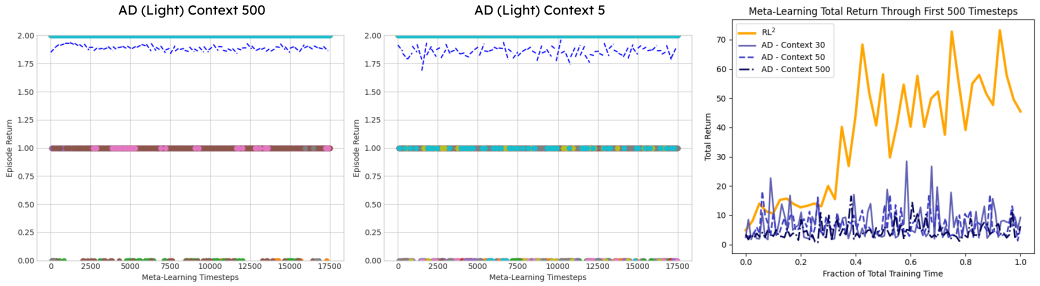

Figure 6: **Meta-Learning, AD, and RL^2**. The meta-learning format is the key ingredient that induces "algorithm distillation" rather than behavior cloning. Even with AD-style independent RL sequences, task information in Light-Key-To-Door removes the in-context learning effect **(left and center)**. Despite improving faster than its source algorithm, AD is still far less sample efficient than RL^2 **(right)**.

Although AD has been shown to be more sample efficient than the individual single-task agents that make up its dataset, it is still not clear whether it is more sample efficient than other meta-learning algorithms. We train a Transformer-based variant of RL^2 [7], which is a meta-learning agent that sees the same information as AD but is trained by pure RL to optimize the return over the first 500 timesteps of a new environment. Like AD, RL^2 is not aware of the key and door location but can intelligently explore a new environment and use a sequence model to remember what it finds in order to adapt its behavior in subsequent attempts. We find that RL^2 learns a far more efficient version of this behavior than AD, almost perfectly solving the task by the 500th timestep while AD does not make meaningful progress in this time and converges thousands of timesteps later. While this does make the long-term usefulness of AD a bit unclear, the method is nonetheless an interesting demonstration of the modeling power of the Transformer architecture and the way that changing the dataset can fundamentally change the behavior that results from a fixed learning algorithm.

The code for this project can be found here: https://github.com/jakegrigsby/algorithm_distillation

References

[1] Marcin Andrychowicz et al. "Hindsight experience replay". In: *Advances in neural information processing systems* 30 (2017).

[2] Jacob Beck et al. “A Survey of Meta-Reinforcement Learning”. In: *arXiv preprint arXiv:2301.08028* (2023).

[3] Anthony Brohan et al. “Rt-1: Robotics transformer for real-world control at scale”. In: *arXiv preprint arXiv:2212.06817* (2022).

[4] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[5] Zichen Jeff Cui et al. “From play to policy: Conditional behavior generation from uncurated robot data”. In: *arXiv preprint arXiv:2210.10047* (2022).

[6] Tri Dao et al. “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness”. In: *arXiv preprint arXiv:2205.14135* (2022).

[7] Yan Duan et al. “RL^2: Fast reinforcement learning via slow reinforcement learning”. In: *arXiv preprint arXiv:1611.02779* (2016).

[8] Mor Geva et al. “Transformer feed-forward layers are key-value memories”. In: *arXiv preprint arXiv:2012.14913* (2020).

[9] Tuomas Haarnoja et al. “Soft actor-critic algorithms and applications”. In: *arXiv preprint arXiv:1812.05905* (2018).

[10] Robert Kirk et al. *A Survey of Generalisation in Deep Reinforcement Learning*. 2022. arXiv: 2111.09794 [cs.LG].

[11] Michael Laskin et al. “In-context Reinforcement Learning with Algorithm Distillation”. In: *arXiv preprint arXiv:2210.14215* (2022).

[12] Volodymyr Mnih et al. “Asynchronous methods for deep reinforcement learning”. In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.

[13] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[14] Vitchyr Pong. “Goal-Directed Exploration and Skill Reuse”. PhD thesis. University of California, Berkeley, 2021.

[15] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[16] Yu-An Wang and Yun-Nung Chen. “What do position embeddings learn? an empirical study of pre-trained language model positional encoding”. In: *arXiv preprint arXiv:2010.04903* (2020).

[17] Jane X Wang et al. “Learning to reinforcement learn”. In: *arXiv preprint arXiv:1611.05763* (2016).

[18] Ruibin Xiong et al. “On layer normalization in the transformer architecture”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 10524–10533.