EMBODIED LANGUAGE MODELS

Thomas Dooms, Alexander Beloussov

University of Antwerp

ABSTRACT

This paper explores the current state of Embodied Language Models (ELMs) from a multi-task and meta-learning perspective. We establish a foundation by elucidating multi-task and meta-learning concepts, emphasizing their applications in reinforcement learning and their relationship with large language models. We then delve into language conditioning for robotic tasks, highlighting its potential for enhancing generalization and adaptability. Finally, we introduce Palm-E, an innovative language model that seamlessly integrates vision, language, and embodied tasks. Through this exploration, we demonstrate the potential of ELMs in rapidly acquiring grounded knowledge and bringing us closer to the realization of generalist robots.

***Index Terms*—** Deep learning, Multi-task Learning, Meta-Learning, Embodied Language Models, Few-shot, PaLM, Robotics

1. INTRODUCTION

Robots that possess the ability to understand and generate natural language have always captured the imagination of researchers and enthusiasts alike. In popular culture and movies, the concept of intelligent robots has been depicted in various forms, ranging from the endearing and helpful character of R2-D2 in Star Wars to the menacing and self-aware machines portrayed in the Terminator franchise. The integration of language understanding and production into robotic systems has the potential to revolutionize this futuristic concept, enabling robots to comprehend instructions, communicate effectively, and perform complex tasks in a manner that is intuitive and adaptable. Embodied Language Models [1, 2] represent a cutting-edge area of research that lies at the heart of this endeavor. These models combine the conceptual knowledge of languages and grounded experience into a sensory-driven physical agent which brings us closer to the realization of generalist robots.

The first part of our exploration (section 2, section 3) lays a solid foundation by providing a comprehensive overview of multi-task and meta-learning concepts. We begin by presenting a mathematical and intuitive examination of both concepts, elucidating their nuances and distinguishing them from transfer learning. Various architectures and techniques are introduced, aiming to address the associated challenges. Moreover, we extend these concepts to the domain of reinforcement learning in section 4. This explanation is derived from the Stanford course CS330, lectured by Chelsea Finn and Karol Hausman [3]. Following this discussion, we provide a whirlwind overview of Language Learning Models (LLMs) in section 5. We establish their relationship with multi-task learning and showcase how LLMs embody the characteristics of a meta-learner.

Building upon this foundation, section 6 delves deep into the realm of language conditioning for robotic tasks. Language conditioning offers a groundbreaking approach that empowers robots to utilize the vast knowledge stored within language models. By conditioning their behavior on linguistic inputs, robots can enhance their generalization abilities and adapt to novel situations with greater efficiency. We explore the intricacies of this innovative approach and discuss its implications for advancing robotic capabilities.

Our exploration culminates with the introduction of PaLM-E in section 7, a groundbreaking embodied language and vision model developed by Google. PaLM-E exemplifies the potential of ELMs by seamlessly integrating vision tasks, language tasks, and embodied tasks. This integration enables the rapid acquisition of grounded knowledge on new tasks, bringing us closer to the realization of generalist robots.

By examining the advancements in ELMs through the lens of multi-task learning, we shed light on the progress made in this field and envision a future where robots understand and interact with the world in a more human-like manner.

The impact of generalist robots extends beyond individual tasks, paving the way for collaborative and adaptable robotic systems integrated into our daily lives. With each step forward, we are able to create embodied models that not only perform tasks more efficiently but intrinsically share human understanding across several modalities. This will facilitate human-robot interaction and enhance the explainability of these complex embodied models.

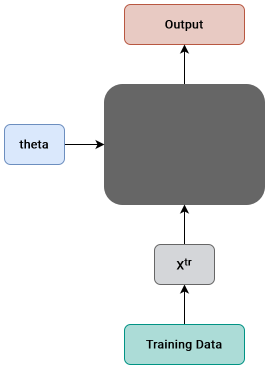

Fig. 1. Typical supervised learning diagram

2. MULTI-TASK LEARNING

2.1. Supervised Learning & Data Modalities

Machine learning encompasses various learning algorithms for solving a multitude of tasks. One prominent algorithm family is supervised learning with the goal of training models to make precise predictions using a single labeled dataset. This dataset can be described as $\mathcal{D} = (X, Y)$ where X represents the input features and Y represents the corresponding target outputs (labels). Typically this involves defining a function f that predicts Y given X and is parametrized by θ. This function can exhibit diverse forms and is therefore presented as a black box in Figure 1. To accurately solve the task, the goal is to find the optimal model parameters θ by minimizing a loss function that quantifies the discrepancy between the predicted output and the true output as denoted in the following equation.

$$\theta^* = \min_{\theta} \mathcal{L}(\theta, \mathcal{D})$$
$$\mathcal{L}(\theta, \mathcal{D}) = -\mathbb{E}_{x,y\sim\mathcal{D}}[log f_\theta(y|x)]$$

Here, θ^* are the optimal parameters for that specific loss and dataset. As all function parameters θ must be learned from scratch each time, this learning process is complex and requires significant amounts of data and computation. Different domains exhibit variations in dataset sizes across modalities. For instance, Natural Language Processing (NLP) datasets often comprise huge text corpora, such as books, news, social media threads, and even code repositories. The internet is full of textual data and therefore these datasets have been scaled to trillions of tokens which has resulted in AI outperforming humans in several NLP tasks [4, 5, 6]. On the other hand, domains such as audio, Reinforcement Learning (RL), and medical imaging may have limited labeled data due to cost, time, or expertise constraints. Consequently, models in these domains often struggle to match human performance on the simplest of tasks. This has prompted researchers to leverage knowledge from domains with data abundance to more efficiently solve less specified tasks.

2.2. Transfer Learning & Limitations

For a long time, the most common solution to this issue involved transfer learning. Transfer learning is used to transfer knowledge from tasks that have abundant data to more niche tasks. For instance, a trained model on general images contains lots of knowledge about colors, edges, and textures to identify objects. By transferring this specific knowledge to for example medical imaging, the model does not require learning the weights from scratch and can start at a higher abstraction level. Mathematically, transfer learning can be formulated as solving the target task $\mathcal{T}_b$ (medical imaging) by leveraging the knowledge acquired from the source task $\mathcal{T}_a$ (general images), without access to the source task's data $\mathcal{D}_a$.

However, some scenarios require a model to solve both $\mathcal{T}_a$ and $\mathcal{T}_b$ simultaneously. Consider a company that would like to train a model to extract certain details from its internal documents. The amount of internal documents is probably too low to train a performant model, therefore, the model is trained on an external collection of documents and fine-tuned accordingly. Unfortunately, research [7] shows that this severely hurts performance on both the external and any out-of-distribution documents. As documents are very free-form, this is a deeply undesirable property. In contrast, multi-task learning aims to jointly learn multiple related tasks, exploiting their shared knowledge to improve generalization and performance on all tasks. By jointly optimizing the loss functions of multiple tasks, the model can learn task-specific parameters as well as shared representations. These loss functions can freely be tuned to favor the internal documents while preserving respectable performance on the rest.

2.3. Tasks

In this multi-task learning setting, tasks can be formally defined as separate learning problems characterized by specific input features and target outputs. Each task has its own loss function, and the goal is to find the optimal model parameters that minimize the combined loss across all tasks. Mathematically, given a set of tasks $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_N\}$ where each task $\mathcal{T}_i$ has its input features X_i, target outputs Y_i, and associated loss function $\mathcal{L}_i(\theta_i, X_i, Y_i)$, the objective is to find the optimal model parameters θ that minimize the combined loss across all tasks. This combination can simply be achieved by summing across all tasks $\sum_i \mathcal{L}_i(\theta_i, X_i, Y_i)$ [3]. In other cases, such as high-risk contexts, this may not be desired. Consider a robot performing medical procedures with varying degrees of difficulty. The model may lean toward only optimizing the simple tasks and neglecting the others. Here, $\max_i \mathcal{L}(\theta_i, X_i, Y_i)$ may be a more appropriate aggregation. Coming back to the previous example about documents, a fitting loss combination may be denoted as.

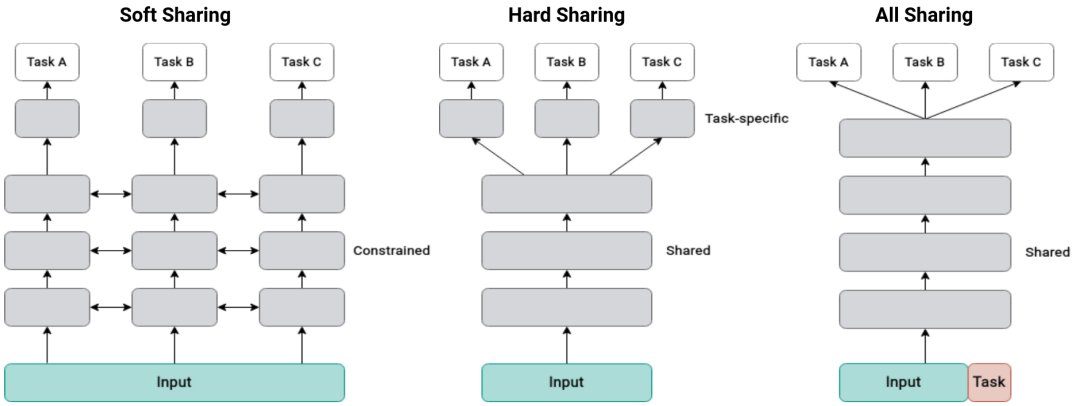

Fig. 2. Three architectures for multi-task learning (adapted from [8])

$$\alpha \sum_{i \in I} \mathcal{L}_i(\theta_i, X_i, Y_i) + (1-\alpha) \sum_{e \in E} \mathcal{L}_e(\theta_e, X_e, Y_e)$$

Here, I and E refer to internal and external document kinds respectively and α is a hyperparameter that determines the importance given to internal documents.

By jointly training on multiple modalities, the model can capture the interdependencies and correlations across different types of data, leading to a richer and more comprehensive understanding of the tasks at hand [3].

2.4. Multi-Task Architectures

To implement multi-task learning in deep neural networks, three commonly used approaches are hard parameter sharing, soft parameter sharing, and all parameter sharing. These approaches determine how the hidden layers are shared or kept separate across tasks and are depicted in Figure 2 [8].

Hard Parameter Sharing. Hard parameter sharing is the predominant approach in MTL for neural networks where the hidden layers are shared among all tasks, while each task has its own task-specific output layer. When learning multiple tasks simultaneously using the same parameters, the model needs to find a representation that captures the common aspects of all tasks, leading to a decreased risk of overfitting individual tasks.

Soft Parameter Sharing. In contrast, soft parameter sharing assigns each task its own model with separate parameters. However, the parameters are regularized to encourage similarity between them. The idea behind soft parameter sharing is to allow each task to have its own set of parameters, while still promoting sharing of information between tasks through regularization.

All Parameter Sharing. Lastly, all parameter sharing employs the same network across all tasks but does not use a task-specific head. Instead, it uses a task descriptor that is appended to the input to distinguish tasks. Explicitly specifying the task helps the model map the input distributions in low-data regimes.

3. META-LEARNING

3.1. Meta Test & Meta Train

Meta-learning and multi-task learning exhibit distinct differences despite their contextual overlap. Meta-learning focuses on enabling a learning algorithm to acquire knowledge from related tasks (meta-training) and leverage it for adapting and generalizing to new tasks (meta-test). In contrast, multi-task learning aims to optimize specific tasks, without emphasizing learning-to-learn. This distinction arises from the use of meta-test tasks, which are entirely unseen, whereas multi-task learning employs test samples from the same task distributions [9].

The goal of meta-learning is: given $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_N$, quickly solve $\mathcal{T}_{test}$. Achieving this objective using the original formulation of supervised learning is unattainable since the test task is undefined, resulting in the model resorting to mere guessing. Ideally, the model should be provided with a limited number of samples, allowing it to employ this knowledge to make an informed conjecture. This mirrors human learning, where, upon seeing an image of two lions, we cannot provide an exact answer, without knowing the question. Yet, based on our prior knowledge of the world, we could make an educated guess such as 'two' or 'lion'. However, if we had previously seen a picture of two penguins labeled 'Antarctica', we could deduce that the answer is likely to be 'Africa'.

3.2. Support & Query Set

Formally, instead of employing a predictor function f with only input x meta-learning takes a slightly different approach. Concretely, it splits each task dataset into a small example set,

called the support set $\mathcal{D}_i^{tr}$. It then uses these demonstrations to perform the actual predictions which are sampled from the other split, called the query set $\mathcal{D}_i^{test}$. The optimization problem can now be expressed as follows.

$$\min_{\theta} \sum_{i} \mathcal{L}_i \left(f_\theta \left(\mathcal{D}_i^{tr}, \mathcal{D}_i^{test} \right) \right)$$

In summary, meta-learning operates through an iterative process where the support set is employed to evaluate the query set. This evaluation yields a loss that is used to update the weights, which is referred to as the meta-training step. Subsequently, the meta-test phase is initiated, which involves performing similar tasks without computing any loss, akin to the test phase in conventional supervised learning. This structural framework is well-suited for achieving the objective of rapid acquisition of knowledge for diverse tasks.

There exist various approaches to structure a function for this purpose, and this section will discuss two prominent methodologies: black-box learning and optimization-based learning. The discussion on black-box learning aims to provide intuitive understanding, while optimization-based learning is emphasized due to its widespread popularity and effectiveness.

3.3. Black-Box Meta-Learning

Black-box meta-learning encompasses the training of a neural network to yield task-specific description, denoted as ϕ_i, given a training dataset and a set of meta parameters θ. Subsequently, an independent neural network utilizes this description ϕ_i to make correct predictions on test data points. The fundamental goal revolves around the first network generating the best possible task-specific description $p(\phi_i|\theta, \mathcal{D}_i^{tr})$ for the second network as seen in Figure 3. The joint training of both networks can be denoted as follows.

$$\min_{\theta} \sum_{i} \mathcal{L}_i(f_\theta(\mathcal{D}_i^{tr}), \mathcal{D}_i^{test})$$

As a result, this iterative process, outlined in Algorithm 1, enhances the adaptability of θ to accommodate various tasks, earning it the name of meta parameters.

Algorithm 1 Black-Box Meta-Learning [9]

for each step **do**
  Sample task $\mathcal{T}_i$ (or mini-batch of tasks)
  Sample disjoint sets $\mathcal{D}_i^{tr}$ and $\mathcal{D}_i^{test}$ from $\mathcal{D}_i$
  Compute $\phi_i \leftarrow f_\theta(\mathcal{D}_i^{tr})$
  Update θ using $\nabla_\theta \mathcal{L}(\phi_i, \mathcal{D}_i^{test})$
end for

Concretely, the first neural network is often structured as a Recurrent Neural Network (RNN) [9, 3]. This network ingests the support set samples one by one and constructs the desired description to be used by the second network. This description can be anything, it could be the actual parameters of the second network. Alternatively, it can be a dynamically generated class label for the specific support set which is appended to the input.

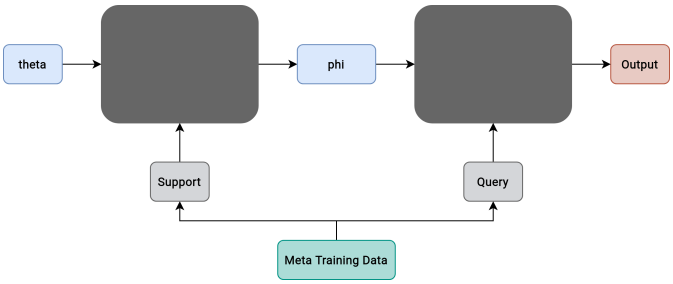

Fig. 3. Diagram of the black box meta-learning approach

3.4. Optimization-Based Meta-Learning

An alternate approach to meta-learning is to simply use a single network. From this viewpoint, the meta-parameters θ serve as a starting point. The quality of these parameters can then be described by their ability to adapt to other tasks within that same parameter space. Instead of this network predicting ϕ, it is computed directly using gradient update steps. Specifically, similarly to Stochastic Gradient Descent (SGD), a small batch of tasks is sampled. For each task, the support set is used to generate ϕ_i, also with SGD, and the model is evaluated using these parameters. Then, the gradient is used to modify the original parameters θ. This algorithm resembles closely to black-box optimization, except in the way ϕ_i is computed.

Algorithm 2 Optimization-Based Meta-Learning [9]

for each step **do**
  Sample task $\mathcal{T}_i$ (or mini-batch of tasks)
  Sample disjoint sets $\mathcal{D}_i^{tr}$ and $\mathcal{D}_i^{test}$ from $\mathcal{D}_i$
  Optimize $\phi_i \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i^{tr})$
  Update θ using $\nabla_\theta \mathcal{L}(\phi_i, \mathcal{D}_i^{test})$
end for

By continually nudging meta-parameters in the direction where tasks can achieve the highest accuracy, the parameters become very polyvalent. This process is depicted in Figure 4 [10], here θ_i^* is an alternate notation for ϕ_i).

4. REINFORCEMENT LEARNING

RL is often used to train agents, such as embodied robots, to perform tasks in an environment. This environment is defined by a Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p(s'|s,a), p(s_1), r(s,a))$, where $\mathcal{S}$ is the set of states $s \in \mathcal{S}$, $\mathcal{A}$ is the set of actions $a \in \mathcal{A}$, $p(s'|s,a)$ is the state

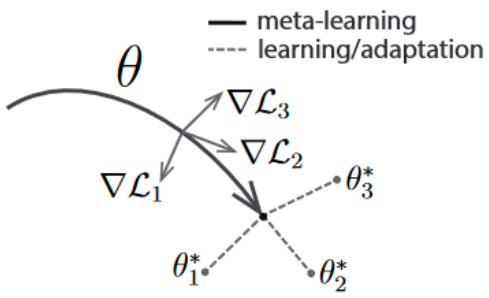

Fig. 4. Optimization-based meta-learning (from [10])

transition probability distribution, $p(s_1)$ is the initial state distribution, $r(s,a)$ is the reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Finally, $\gamma \in (0,1]$ is the discount factor, and is sometimes also included in the definition. The goal of the training is to find a policy $\pi(a_t|s_t)$ which maximizes the (discounted) sum of rewards G_t, sometimes referred to as the "return".

$$G_t = \sum_{k=0}^{H} \gamma^k r(s_{t+k}, a_{t+k})$$

Here, H is the final time step (or horizon), which can be infinite. Value-based RL estimates the achievable discounted returns for each state or state-action pair using value functions. This is done via bootstrapping, meaning that one estimate is updated using another estimate. This can be seen below, as the estimated next-state value $V_\pi(s')$ is used in the update of $V_\pi(s)$ and $Q_\pi(s,a)$.

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi\left[G_t|s_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma V_\pi(s')\right] \\ Q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t|s_t = s, a_t = a\right] \\ &= \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma V_\pi(s')\right] \end{aligned}$$

Policy-based methods only explicitly keep a representation of the current policy. They rely on gradient ascent to update the policy directly to maximize the received rewards.

$$\begin{aligned} J_\pi(\phi) &= \max_\phi \mathbb{E}_{(s_t,a_t)\sim\pi_\phi(\tau)}\left[\sum_{t=0}^{H} \gamma^t r(s_t,a_t)\right] \\ \pi_\phi(\tau) &= \pi_\phi(s_0, a_0, ..., s_H) \\ &= d_0(s_0)\prod_{t=0}^{H} \pi_\phi(a_t|s_t)p(s_{t+1}|s_t,a_t) \end{aligned}$$

The way in which policies are updated can vary. When we exclusively utilize experience from the current policy π_k to update the policy, we refer to this approach as an *on-policy* method. In simpler terms, on-policy RL involves learning a new policy π_{k+1} by interacting with the environment and collecting observations under the current policy π_k.

On the other hand, there are also *off-policy* algorithms that take advantage of experiences from different policies to update the policy. These methods employ a clever technique of storing all experiences in a replay buffer, a sort of data repository [11]. This storage allows for the reuse of experiences by randomly sampling from the accumulated data when updating the policy. In essence, off-policy RL learns a new policy π_{k+1} by using observations collected under various policies, such as π_k, π_{k-1}, π_{k-2}, and so forth.

4.1. Multi-Task Reinforcement Learning

Multi-Task Reinforcement Learning (MTRL) shares many ideas with multi-task supervised learning. Its primary objective is to enhance policy performance by encouraging knowledge sharing across tasks, often referred to as *cross-task generalization* [12]. In addition, MTRL can lead to *easier exploration* as each task can explore distinct environmental dynamics [13]. Consequently, this increases the overall available data, serving multiple tasks simultaneously. Moreover, the concept of relabeling [14] can be employed, generating new experiences derived from different tasks (more details on this to come).

Similarly, MTRL can achieve *higher per-task sample-efficiency*. Training individual tasks may require large amounts of examples for that task, whereas the multi-task setting only requires a few examples for each task to achieve similar performance [15]. This increased efficiency stems from the collective knowledge acquisition across multiple tasks.

Furthermore, task separation can be exploited to tackle *long-horizon tasks*. For instance, consider a scenario where a robot is required to sort various objects by color. By breaking down this objective into a sequence of tasks or subtasks, each aimed at handling individual objects, the overarching goal of object sorting can be accomplished [16, 1, 17].

Similarly, MTRL enables *reset-free learning*. An agent may learn how to open a door and close it at the same time. However, in order to learn to open the door as an individual task, the door needs to be reset to a closed position each time. In the multi-task setting, the resulting state can serve as the starting state for a different task [18].

For MTRL, a task $\mathcal{T}_i$ is defined by the MDP ($\mathcal{S}_i$, $\mathcal{A}_i$, $p_i(s'|s,a)$, $p_i(s_1)$, $r_i(s,a)$) [3]. A new task can be defined by changing any of the components of the MDP. For example, using a different action space defines a new task, and so does changing the dynamics, rewards, state space, or starting state distribution. Tasks may be identified by a task identifier z_i. This identifier can optionally be part of the state, such that

Algorithm 3 Hindsight Experience Replay [14] (simplified)

for each episode **do**
  Choose task $\mathcal{T}_i$
  Collect data for $\mathcal{T}_i$ using the learned policy π
  Store data in the replay buffer
  for task $\mathcal{T}_j$ in a set of other tasks **do**
    Change task descriptor z to z_j
    Change rewards $r_j(s, a)$ according to the new task
    Store data in the replay buffer
  end for
  Update the policy π
end for

Algorithm 4 Hindsight Experience Replay [14] (goal-conditioned)

for each episode **do**
  Choose task $\mathcal{T}_i$
  Collect data for $\mathcal{T}_i$ using the learned policy π
  Store data in the replay buffer
  for state s_j in a subset of visited states **do**
    Change goal g to the state s_j
    Change rewards $r(s, a, g)$ according to new goal
    Store data (up to g) in the replay buffer
  end for
  Update the policy π
end for

the agent is aware of the task that it needs to perform. If $\bar{s}$ is the current state for task i, the new state can be a concatenation of $\bar{s}$ and the task identifier, resulting in $s = (\bar{s}, z_i)$.

$$\text{Policy: } \pi(a|\bar{s}) \rightarrow \pi(a|\bar{s}, z_i)$$
$$\text{State value: } V(\bar{s}) \rightarrow V(\bar{s}, z_i)$$
$$\text{State-action value: } Q_\pi(\bar{s}, a) \rightarrow Q_\pi(\bar{s}, z_i, a)$$

If the task identifier is some form of description of the goal, the problem is often referred to as Goal-Conditioned RL (GCRL) [19]. In GCRL, the task identifier z_i is often replaced with goal g_i.

The task identifier z_i (and by extension goal g_i) can take several forms, such as a one-hot encoded vector, a language description, or a desired goal state. Not every choice allows for information to be shared effectively between tasks. Even if some task $\mathcal{T}_i$ is very similar to some task $\mathcal{T}_j$, it would be that $dist(z_i, z_j) = dist(z_i, z_k) = dist(z_k, z_j)$, for some less related task $\mathcal{T}_k$. In this case, it is more beneficial to train the agent for each task individually [3]. To allow the sharing of information between tasks, identifiers (or goals) need to be encoded using distributed representations [20]. Images that have been encoded into a latent space of the desired state, have been successfully used to condition robotic agents [21]. Similarly, and as we will explore further in section 6 and section 7, language embeddings have also been used to guide policies [1, 16].

Since MTRL still makes use of MDPs, meaning that classic RL methods still work in this setting. However, there are ways to increase the single-task performance and cross-task generalization capabilities of these algorithms. Generally, Experiences for some task $\mathcal{T}_i$ are gathered by the agent attempting to perform said task in the environment. The agent may perform some other task $\mathcal{T}_j$ when attempting task $\mathcal{T}_i$. The same experience can then be used to train this other task $\mathcal{T}_j$. This is called *hindsight relabeling* [14]. Algorithm 3 shows this process.

There are a few conditions that need to be met for this approach to work. Firstly, to perform relabeling, the reward function needs to be known. Secondly, for experiences to transfer across tasks, the environment dynamics need to be consistent between tasks. Finally, only off-policy algorithms can make use of this technique [3].

The tasks that are used for relabeling can be chosen in different ways. For example, a set of tasks can be sampled for which to relabel a certain experience or only tasks where the reward is non-zero can be selected. The random approach may create a lot of negative examples (where the sum of rewards = 0). These can still be used by value-based methods such as Q-learning [22, 23], as even transitions with no reward can be used to update the policy. This is not the case for policy gradient approaches. There, the gradient of the policy is dependent on the reward signal. Without a reward, the policy cannot be updated. This is a disadvantage for policy gradients in the MTRL setting, but not for value-based RL [3].

If the task descriptor is the desired goal state, we can use hindsight relabeling to drastically increase the number of positive examples. Regardless of success in the original goal-conditioned task, successful examples can be created by changing the goal state to the final state in the episode. Intuitively, this says that the reached state is the one we were meant to reach all along. Similarly, we can use any state in the episode as the goal, and only store the relevant subsection of the episode. This procedure can be seen in Algorithm 4.

This creates a dataset of episodes where the goal is always reached. This is (near-) optimal data that can be used with simple imitation learning approaches [24, 25, 16, 1]. Additionally, this allows us to use unstructured datasets for teaching agents. One such example of unstructured data is *human play* [26]. This is when a human is allowed to "play" in an environment, without necessarily having to achieve a goal [16]. This unstructured dataset can then be turned into a goal-conditioned dataset using hindsight relabeling.

4.1.1. Model-Based MTRL

Algorithms that only make use of experience gathered in the real environment are called *model-free*. Methods that reason about the resulting state or other outcomes using a model

Algorithm 5 Black-Box Meta-RL (meta-training)

for each episode **do**
  Choose task $\mathcal{T}_i$
  Rollout policy $\pi(a|s, \mathcal{D}_i^{tr})$ for N episodes
  Store sequence in replay buffer for task $\mathcal{T}_i$
  Update policy
end for

are called *model-based*. In many multi-task problems, only the reward function varies across tasks [3]. When this is the case, we can achieve a higher sample efficiency (meaning that fewer interactions with the environment are needed) by learning a model $f_\phi(s, a)$ that predicts the resulting state s'. This model can be learned using experience from different tasks, and then be shared between tasks during the training of the policy. The model can be learned using a simple supervised learning objective.

$$\arg\min_\phi \sum_i ||f_\phi(s_i, a_i) - s_i'||^2$$

Existing model-based algorithms can be used in the multi-task setting, such as Model-Predictive Control [27].

4.2. Meta Reinforcement Learning

Just like MTRL is the RL counterpart of multi-task supervised learning, Meta-RL (MRL) is the RL counterpart of meta-supervised learning. The goal is to be able to adapt to new tasks in as few episodes as possible (k-shot) [3]. This is in contrast with MTRL, where the policy should generalize to new tasks (0-shot). The input to the MRL policy f_θ is the current state s_t, together with a training dataset $\mathcal{D}^{tr} = \{\mathcal{D}_i\}_i$, $\mathcal{D}_i = \{(s_t, a_t, r_t, s_{t+1})\}_t$. The output is the action a_t to take in $s_t \in \mathcal{D}_i^{test}$.

4.2.1. Black-Box Meta RL

Black-box MRL uses a black-box network f_θ as the policy $\pi(a|s, \mathcal{D}_i^{tr}$. This can be a feedforward network [28], recurrent architecture [29, 30], ConvNet [31], etc. Unlike black-box supervised learning, this approach does not make use of separate networks. Often, the task is not identified using a task descriptor as was done in the MTRL case. Instead, the task is inferred from past experience, gathered in $\mathcal{D}_i^{tr}$. This dataset grows with each episode as the agent explores the environment. The goal is to use fewer exploration episodes compared to an agent that would learn the task individually and from scratch. The meta-training algorithm is shown in Algorithm 5. During meta-testing on new task $\mathcal{T}_j$, the policy $\pi(a|s, \mathcal{D}_j^{tr})$ is rolled out for N episodes, where each observation keeps adding to $\mathcal{D}_j^{tr}$.

This approach is very general and expressive. It also allows for a large variety of design choices for the architecture. The disadvantage is that this approach is hard to optimize [3].

Algorithm 6 Optimization-based Meta-RL (meta-training)

for each episode **do**
  Choose task $\mathcal{T}_i$
  Collect $\mathcal{D}_i^{tr}$ by rolling out policy π_θ for k timesteps
  $\phi_i = \theta + \alpha \nabla_\theta J_i(\theta)$ ▷ Inner loop adaptation
  Collect $\mathcal{D}_i^{test}$ by rolling out policy π_{ϕ_i} for h timesteps
  $\theta \leftarrow \theta + \beta \sum_{\text{task } i} \nabla_\theta J_i(\phi_i)$ ▷ Outer loop update
end for

4.2.2. Optimization-Based Meta RL

Optimization-based MRL is very similar to the supervised learning setting. The main difference lies in the optimization objective that is used to update the parameters. Here, we now use objectives that are used in regular RL.

Policy gradient [9] approaches have the benefit of being gradient-based (similar to the supervised learning setting), but are on-policy and struggle in settings where there are few rewards. Additionally, policy gradients may be very noisy, creating high variance during training.

Value-based approaches only make very local updates and may require many update steps, but have the benefit of being off-policy (more sample efficient).

Model-based approaches [32] benefit from the fact that learning models is a supervised learning task. Sample efficient off-policy algorithms can be used to train the policy. In addition, the model can be used to generate new (artificial) samples, increasing the sample efficiency further.

Generally, policy gradient and model-based approaches are more easily combined with optimization-based MRL [3]. Algorithm 6 shows the general procedure, where the optimization objectives $J_i(\theta)$ may be replaced by any RL objective.

5. LANGUAGE MODELS

5.1. Transformer Overview

Large Language Models (LLMs) have gained significant popularity due to their exceptional understanding of human language [33, 34]. These models rely on transformers, an underlying architecture that has demonstrated remarkable capabilities in capturing dependencies and relationships within input data. Unlike conventional neural network architectures like Convolutional Neural Networks (CNNs) [35] or Fully Connected Networks (FCNs) [36], transformers incorporate an attention mechanism, enabling them to dynamically assign weights to input elements based on their relevance to one another [37, 38]. This attention mechanism is a distinguishing feature that empowers LLMs to excel in language-related tasks.

Transformers operate on sequences, with each data point in the sequence commonly referred to as a token. In Natural Language Processing (NLP), tokens can encompass various

forms, ranging from complete words to individual characters, allowing for increased control. In other tasks, tokens can represent pixels, audio samples, or other data entities. The collective set of tokens in the provided input sequence is often referred to as the context.

Traditional neural networks are considered static since they employ fixed weights for each neuron, regardless of the input. The expressive power of these networks lies in the complicated sequence of non-linear operations they perform. However, they have proven to be inadequate for tackling tasks with intricate relationships in the data [37, 39, 40].

In contrast, transformers, akin to human cognition, utilize attention to assign varying levels of importance to input elements based on ongoing computation. This is particularly crucial in natural language tasks, such as translation, where intricate language constructs require meticulous attention to related words to make accurate predictions. In many languages, verbs can be conjugated based on singular, plural, and masculine or feminine which necessitates thorough knowledge about the surrounding words.

Fundamentally, the attention mechanism serves as a communication channel between tokens. Each token assesses the usefulness of other tokens and aggregates this information into a comprehensive embedding. Residual connections are employed to combine this embedding with the original representation to provide a powerful embedding of the token enhanced with surrounding information. This process involves the use of queries, keys, and values, which are computed individually for each token. The query represents what the token seeks, similar to a database query, while the keys and values resemble their database counterparts. For instance, the query of a verb may represent "all preceding nouns in this sentence", which will hopefully match the keys of all nouns to determine the correct conjugation.

Mathematically, each token is represented by an embedding of size d. From this, queries Q_i, keys K_i, and values V_i are computed using a linear operation, typically with the same embedding size d. The match between queries and keys for all tokens is then calculated through a matrix multiplication $Q \cdot K^T$. Both matrices are simply stacked embeddings of the queries and keys respectively. The resulting square matrix, with dimensions equal to the context length, represents the similarity between the query of a token i (row) and the keys of tokens j (column). The actual attention is obtained by normalizing this matrix by $\sqrt{d_k}$ (where d_k is the embedding length of the keys) and applying a softmax function. This softmax introduces non-linearity to prevent collapse and filters out non-important tokens from attention. Finally, the attention matrix is multiplied by the values matrix to produce the final embeddings.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

The complete transformer architecture consists of alternating an attention layer and a small FCN several times. There are two variants of this architecture: the encoder and the decoder. The encoder allows each token to attend to any other token. On the other hand, tokens in decoders can only interact with preceding tokens. Encoders are primarily used for tasks such as masked word prediction or Named Entity Recognition (NER), which require insight into the full context. In contrast, decoders are employed for generative purposes, such as LLMs which may only see preceding tokens to avoid data "leaks". In certain tasks, both encoder and decoder variants can be combined, such as in translation. The original text is fed through an encoder to generate embeddings, which are then utilized by the decoder to generate the translated text word by word. This process is referred to as cross-attention [37]. Concretely, cross-attention involves utilizing the keys and values from the encoder and the query from the decoder. In the context of translation, this process entails the tokens to be translated querying the original sentence for relevant information. Notably, cross-attention can be performed using any form of embeddings from any source. The application of this technique spans various domains, one of which will be explored in detail in subsection 6.2. Interestingly, recent advancements in decoder-only transformers have demonstrated their ability to accomplish this without relying on an encoder [41].

5.2. Emergent Abilities

Decoder transformers, including large language models (LLMs), are specifically trained using next-token prediction [42]. Given a text T consisting of tokens $t_0, t_1, ..., T_N$, the objective is to predict $p(t_i|t_0, .., t_{i-1})$ for each $i < N$. The strength of this transformer architecture lies in its ability to handle extensive context lengths by leveraging large-scale training data. Unlike previous NLP architectures like Recurrent Neural Networks (RNNs), which were limited to remembering only a few sentences or approximately a hundred tokens [43], modern transformers can effectively attend to a context length of 32k tokens [44, 34], and with certain modifications, even millions of tokens [45]. This enables them to harness a wealth of information for highly informed predictions across diverse domains, with performance improving as more data is incorporated. Currently, LLMs are often trained on text corpora approaching the scale of the entire internet, encompassing trillions of tokens [46].

At first glance, the task of predicting the next token may appear trivial—simply completing sentences coherently. However, to enhance the accuracy of predicting upcoming tokens, the model needs to acquire a broad range of relevant knowledge. For example, consider predicting the next word in the sentence "The Eiffel Tower is located in ...". To excel in such tasks, the model must possess comprehensive knowledge not only of language but also of the broader world. As

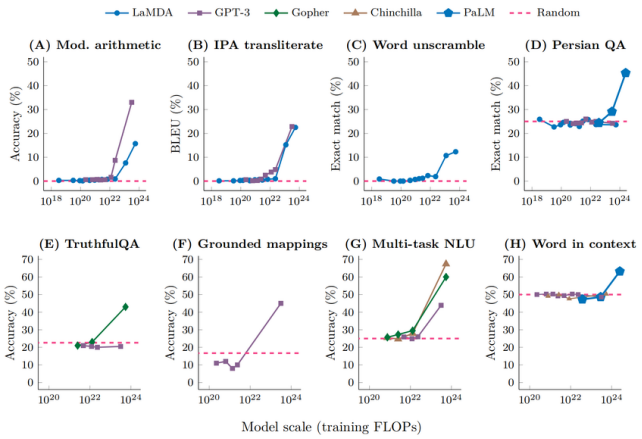

Fig. 5. Accuracy of LLMs on various non-trivial tasks

depicted in Figure 5, these specialized capabilities emerge spontaneously. Notably, the accuracy of specific tasks, such as modulo arithmetic $x \bmod y = ..$ or PersianQA, consisting of challenging trivia questions extracted from Persian Wikipedia pages, experiences a sudden and significant increase after a certain threshold [47]. This demonstrates that, while not specifically trained with a multi-task learning algorithm, these models show strong characteristics of multi-task learning and generalization.

The timing of the emergence of these abilities is inherently unpredictable, as there are no prior indications. A recent contest aimed to identify tasks where the accuracy of LLMs declined with model size. One of the winners was the hindsight neglect task, which presented a prompt like: "David has the option to play a game where he has a 90 percent chance of losing 20 dollars and a 10 percent chance of earning 100 dollars. David plays the game and ends up earning 100 dollars. Did David make the right decision? Choose Y or N." This is obviously just lucky and is not representative of the general case. Surprisingly, larger models struggle with this type of question, ignoring the probability theory and simply answering: "Y". However, GPT4 demonstrates that further scaling resolves this issue [34].

Leveraging the extensive knowledge of these models, often referred to as foundation models, is not as straightforward as initially assumed. Following their training regime, these models only predict the most probable next token. However, this may not always align with the desired behavior. Consequently, they are commonly fine-tuned for specific tasks, such as question answering, to enhance their performance and align with the desired objectives [33].

5.3. Few-Shot Learning

Another emergent ability is few-shot learning from examples in the prompt. Referred to as In-Context Learning (ICL), this phenomenon holds great promise for achieving efficient learning with limited samples [48]. Concretely, certain demonstration samples $C = (x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ are textually encoded into the prompt, additionally, the question token is appended, preferably in the same format. To illustrate, consider the following prompt: "The Dark Side of the Moon: 1973, Wish You Were Here: 1975, The Division Bell: 1994, Animals:". A good language model will notice the names of the Pink Floyd albums and their corresponding release years and fill in the required year: 1977. Although not fully comprehended, ICL has proven to be a fruitful avenue worth exploring [48]. The astute reader may notice that this problem formulation is identical to meta-learning.

Moreover, an effective way to enhance the sample efficiency of language models is by prepending a textual description of the task. This technique can be further augmented by fine-tuning the language model on a diverse range of instruction tasks, enabling it to perform several tasks without the need for in-context examples [49]. This zero-shot prompt understanding and the in-context learning capability highlight the remarkable meta-learning characteristics of these large language models (LLMs) across various domains.

Alternatively, ordinary fine-tuning on these models can be carried out efficiently. A notable method for achieving this is through Low-Rank Adaptation (LoRA) [50]. By freezing the original weights and introducing a compact and trainable modification in each layer, LoRA enables the model to be fine-tuned with minimal computational and memory resources. This makes it feasible to perform fine-tuning on consumer-grade GPUs and simplifies the deployment of multiple independent instances of the model, requiring only a small modification for each instance and a single copy of the original weights.

5.4. Attention as a Meta-Optimizer

This section will cover the abilities of LLMs from the viewpoint of multi-task and meta-learning. As seen in the previous section, providing a massive amount of data comprising a multitude of textual tasks leads to a model that can perform almost all tasks well. Additionally, this extends to meta-learning where these models are surprisingly capable of performing any novel task well with minimal supervision.

There is no specific incentive in the training or dataset for this behavior, rather all data is simply concatenated and an all-parameter-sharing architecture is used (section 2). This fact has prompted machine learning researchers to explore the underlying cause for these capabilities. One hypothesis to explain ICL capabilities is that attention acts as a meta-optimizer. Intuitively, the attention mechanism enhances the latent representations with information about the samples in a way that gradient descent would do. At each layer, the representations are augmented to minimize some internal loss functions [51]. Philosophically, this is comparable to what humans do, internal mechanisms define whether we steer our

reasoning or deem an answer correct based on all kinds of knowledge.

The research paper in question [51] underpins this statement with two arguments: a mathematical dual between attention and gradient descent and the observations in the latent space. The former argument relies on the fact that one step of gradient descent fine-tuning modifies embeddings at each layer by the following formula.

$$emb = (W_0 + \Delta W)x$$

Here x is the input to that layer, W_0 are the initial weights and ΔW is the update step of the weights. When fine-tuning, ΔW is computed using a batch of demonstration samples C.

Comparably, the attention mechanism has access to the tokens of the demonstration samples. Combined with the fact that the mathematical operation of gradient descent and attention are very similar, the authors argue that the following holds approximately.

$$(W_0 + \Delta W)x \approx W_0 x + \text{Attention}(C)$$

Note that this is an abstraction of the derivation in the original paper. Intuitively it states that the attention mechanism, given C, changes the input in such a way that mirrors a fine-tuned model. The latter argument stems from an observation that the latent space of ICL indeed resembles the fine-tuned one to a certain degree [51].

We believe this work should be used to form intuition about ICL but not as a definitive explanation of its inner workings.

6. LANGUAGE CONDITIONING

In the upcoming sections, we explore two remarkable research works that build upon our earlier discussions. The first is "Interactive Language" [16], a groundbreaking fusion of robotics, Multi-Task Reinforcement Learning (MTRL), and language models. This work trains a robotic arm to comprehend and respond to natural language commands, opening doors to effortless human-robot communication. The second work, "PaLM-E" [1], is an extension of the PaLM project [4], enabling it to function as a sophisticated high-level robotics controller. With PaLM-E, complex robotic tasks can be orchestrated with enhanced precision and efficiency.

Interactive language [16] conditions its policy on embeddings of natural language. The expressiveness of the existing language model enables the use of a free vocabulary. This means that anyone controlling the robot can use any combination of words to form instructions. It is possible to continuously modify these instructions, changing the goal that the policy is conditioned on. This allows a controller to complete long-term objectives by giving short-term instructions. Additionally, this allows the controller to correct any mistakes by the robot by issuing new commands.

6.1. Dataset

The dataset for this work is gathered by human operators, each controlling the robotic arm in sessions of 10 minutes. The goal is to create an unstructured dataset that covers a large subsection of the state space. The operators are presented with random objectives but are free to ignore them as they are later discarded. This creates an unlabeled *human play* dataset [26] consisting of a total of 2,700 hours of human play data.

Next, *event-selectable* hindsight relabeling is performed to create a labeled dataset. This idea closely resembles what was discussed in subsection 4.1. The goal of this process is to create a language-conditioned dataset $\mathcal{D} = \{(\tau, l)_i\}_i$. Crowdsourced annotators are tasked with finding K actions (here $K = 24$) within the 10-minute-long episodes. The start and end of the actions are marked by the annotators and language descriptions are added (using a free vocabulary). On average, actions are 5.8 seconds long. The length of the labeled dataset is 488 hours. This results in a goal-conditioned dataset where goals are given by language description. More importantly, this is also an expert dataset, as each episode successfully completes the task that is (in hindsight) assigned to it. As mentioned in subsection 4.1, this enables the use of simple imitation learning methods.

6.2. LAVA Architecture

The architecture used in Interactive Language is termed *Language Attends to Vision to Act* (LAVA). Figure 7 shows this architecture.

Emdedding the input. The states are RGB observations made by a camera. They are embedded using a convolutional network that consists of 2 ResNet [52] layers that are pretrained on ImageNet [53] and 2 additional learned convolutional layers. It outputs a set of features for different resolutions. This allows it to capture more global information, as well as some finer details.

The language descriptions/instructions are encoded using a fine-tuned CLIP [54] encoder. This fine-tuning happens prior to the training of the full model, for which it remains fixed. CLIP is trained on (image, language) pairs, however, actions consist of more than a single image. Instead, (start frame s_0, goal frame s_g) pairs are used to represent actions. Both frames are passed to the image encoder to create embeddings (z_0, z_g). These are then concatenated and passed through an MLP to reduce the length of the embedding back to the length of the text and image embeddings. This can be seen in Figure 6. After fine-tuning, only the CLIP text encoder is kept for making the text embeddings.

Vison Language Fusion. Prior to the attention mechanism, the language description/instruction and image (at timestep t) are scaled to the same dimension size d_{model}. For the language embeddings, this is done using an MLP. For the image features, 2D sinusoidal positional features are

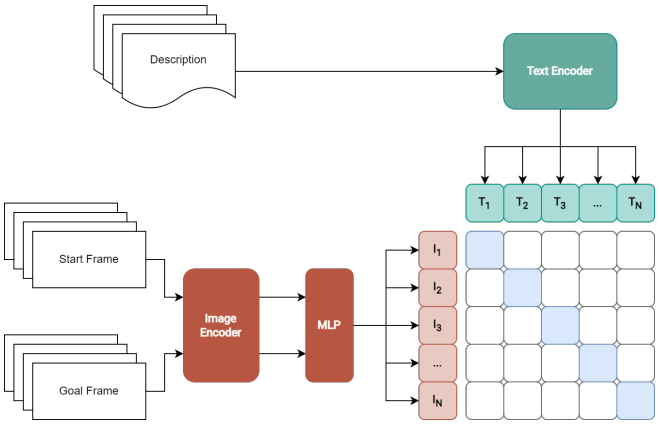

Fig. 6. Adapted CLIP, taking pairs of images (start frame s_0, goal frame s_g).

encoded similarly to [37]. Then, each feature scale is projected and flattened into a matrix of size $[n_{features}, d_{model}]$. The language embedding is then used as the query in the (multi-headed) cross-attention mechanism of a decoder-only transformer, while the image embeddings are used for keys and values. This creates an embedding for the (image, language) pair for some timestep t

Vision-Language Embedding Sequence. The next transformer takes a sequence of vision language fusions. 1D sinusoidal positional encodings are added, similar to [37].

Rollup over time. The sequence of vision-language embeddings is then passed through an encoder-only transformer. This transformer consists of blocks, each containing (multi-headed) self-attention and an MLP. This outputs another embedding of size d_{model} for each input in the sequence. These are then averaged over the time dimension to create a single d_{model}-sized embedding.

Policy. The embedding is then passed through 2 deep residual MLP blocks, each containing 3 layers larger than d_{model}. The output is then projected linearly to the action space.

This whole architecture runs at 5 Hz. It also allows the language instructions to change in real-time. For further details, we refer the reader to the appendix of the Interactive Language work [16].

6.3. Summary

The combination of the hindsight relabeled dataset and the LAVA architecture allows the robot to perform a very wide variety of short-term tasks, conditioned by open-vocabulary natural-language instructions. By continuously giving new instructions, an operator can achieve complex long-term goals and correct any mistakes made by the robot. Video demonstrations of this work can be found at `https://interactive-language.github.io/`.

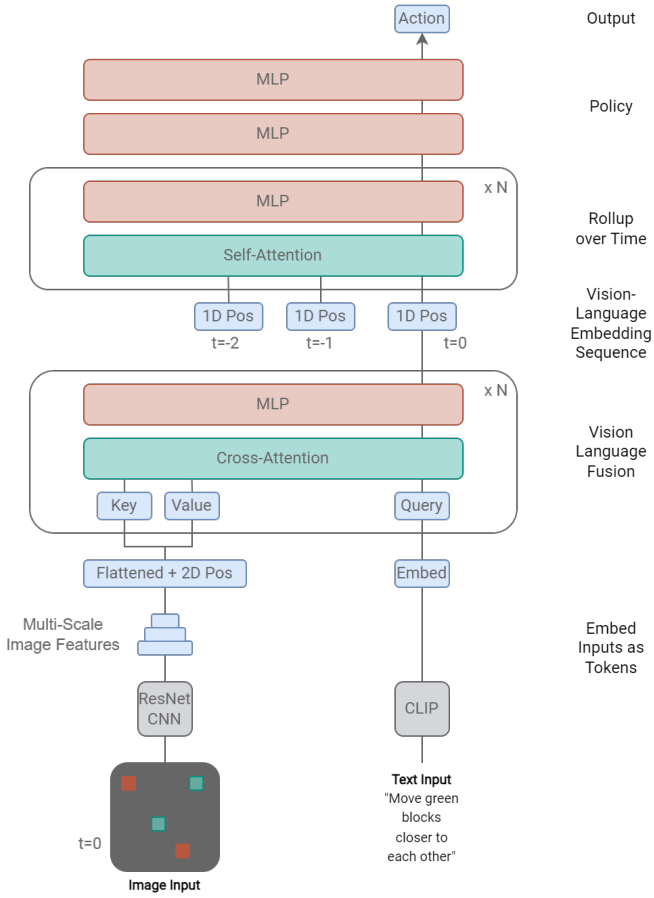

Fig. 7. LAVA architecture. Figure adapted from [16]

There are a few points that are important to remember from this work. Firstly, language embeddings can be powerful task descriptors, not only providing arbitrary language understanding but also allowing for strong generalization across tasks. Secondly, progress in computer vision and NLP provides us with powerful architectures that can be used in a wider variety of machine learning fields. In particular, robotics is another field that has been empowered through the development of transformers, which originally were only meant for NLP tasks.

7. PALM-E

7.1. Representing multi-modal states

The short-term tasks from the previous section, such as grabbing, moving, and so on, can be seen as atomic tasks. These tasks are performed effortlessly by humans without conscious thought, often called muscle memory. Nevertheless, orchestrating these tasks into a complex ensemble requires careful planning, taking into account potential failures or changing

circumstances. The aforementioned research delegated this issue to a human controller who had to meticulously guide the robot. Alternatively, LLMs can be leveraged to perform this planning process instead of humans.

To enable LLMs to perform this form of planning, the model must be provided with current state information. In subsection 5.3, we explored the concept of In-Context Learning (ICL) and demonstrated that LLMs exhibited the ability to react appropriately to various scenarios through their few-shot learning capabilities when supplemented with contextual information. However, there is a constraint to this approach: this contextual information must be textually encodable. Unfortunately, the context in tasks such as robotics cannot be fully represented by text alone. While it is possible to describe an image or a 3D environment using text, this approach inevitably leads to information loss. Ideally, we would want to provide the model with an exact representation of the environment so that planning can be as effective as possible.

Recent research conducted by Google Brain aims to address this limitation by introducing a new model called PaLM-E [1], which incorporates multiple modalities. PaLM-E combines an LLM with a vision transformer, leveraging the strengths of both models. This combined model is then trained on various tasks, including robotics, visual question answering, and captioning, to reach a visual-language generalist model.

7.2. PaLM

In essence, PaLM-E is a fine-tuned version of PaLM [4] augmented with several encoders. PaLM, with its 540 billion parameters, is one of the largest language models ever trained at the time of writing. Training this behemoth is widely regarded as a monumental engineering feat that required exceptional techniques to communicate gradient updates across multiple data centers. The model incorporates a few modifications to the standard transformer architecture, including the use of parallel layers, which slightly changes the order of normalization to improve speed. The original formulation of a layer is depicted in Equation 1, while the improved one is shown in Equation 2. Additionally, biases are omitted from layer normalization and QKV (query-key-value) projections in PaLM-E to reduce instability and improve performance. These combined techniques result in approximately a 20% performance gain.

$$y = x + \text{MLP}(\text{LN}(x + \text{Attention}(\text{LN}(x)))) \quad (1)$$

$$y = x + \text{MLP}(\text{LN}(x) + \text{Attention}(\text{LN}(x))) \quad (2)$$

The training process of PaLM-E involves using the Adafactor optimizer [55], a slightly modified version of the Adam optimizer [56], on a dataset consisting of 780 billion tokens. The data is sourced from various domains, such as social media, webpages, books, GitHub, Wikipedia, and news articles.

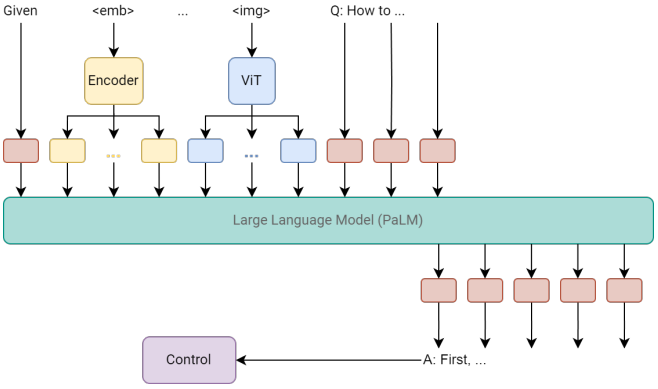

Fig. 8. Figure adapted from [1]

7.3. Visual Transformers

Regarding the visual aspect of PaLM-E, it incorporates an upscaled visual transformer model with 22 billion parameters [57]. The visual transformer architecture [39] takes a different approach to image classification compared to traditional CNNs. Instead of using convolutions, it employs a transformer architecture that treats an image as a sequence of small patches. Each patch represents a square region containing a few hundred pixels. This patch-based segmentation is done to reduce computation complexity, as performing pixel-wise attention would require computing a prohibitively large number of attention scores. For instance, processing an image with a width and height of 500 pixels would entail computing an attention matrix with 500^4 entries. The advantage of using transformers for this task is that they don't have an inherent bias towards nearby pixels like CNNs do. Transformers can attend to any patch in the image, allowing them to capture long-range dependencies and outperform CNNs in certain scenarios.

7.4. The PaLM-E Architecture

With these foundations in place, we can now understand the full PaLM-E architecture, as depicted in Figure 8. In addition to receiving tokens as input (depicted in orange), the model can also understand encodings from alternative sources (depicted in yellow/blue) at arbitrary positions within the input sequence. These alternative sources, referred to as "observations" in the original work, can be encoded into a list of embeddings. This enables the model to incorporate context from sources such as images captured by a robot's view, allowing for better planning and decision-making.

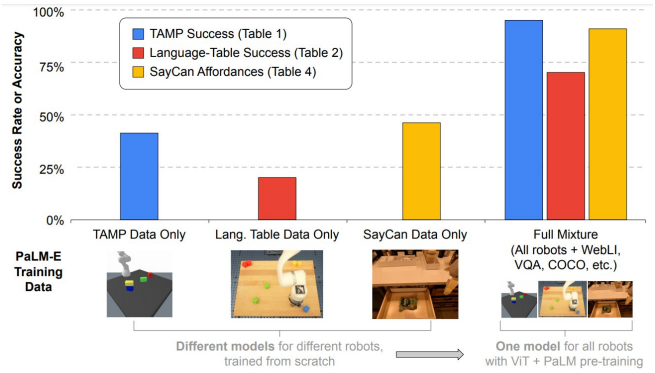

Fig. 9. Transfer learning scores (from [1])

Formally, textual data is represented as tokens from a fixed vocabulary $\mathcal{W}$ and embedded into $\chi \subset \mathbb{R}^k$. On the other hand, observations from $\mathcal{O}$ are embedded into the same representation space χ^q, where q represents the number of embedding vectors.

During the training phase, both the language model (PaLM) and the encoders (such as the vision transformer) are fine-tuned in an end-to-end manner. This allows the language model to adapt to the representations from the encoder, and vice versa, enabling the encoder to align itself with the expectations of the language model. By leveraging pre-trained models, the training process becomes more efficient and shorter.

The main objective of the PaLM-E model is to perform planning tasks for real-world entities. However, language models typically excel in conceptual knowledge but lack an understanding of the physical world as we experience it. To address this limitation, PaLM-E is fine-tuned using various robot environments, including Task and Motion Planning (TAMP), Language Table (discussed in subsection 6.1), and mobile manipulation environments.

The performance of PaLM-E on these benchmark tasks is depicted in Figure 9. The results demonstrate the model's remarkable ability to transfer knowledge and exhibit strong performance across multiple robotic tasks. The inclusion of additional data significantly improves the generalization capabilities of the model across the observed tasks. This suggests a promising future where deep integration of language and vision can enhance the generalization capabilities of various models.

8. CONCLUSION

In this work, we provided an introduction to the world of multi-task and meta-learning, laying a solid foundation for understanding these areas of study. Various architectures and techniques were presented to address the associated challenges, with a particular focus on extending these concepts to the domain of reinforcement learning. Further, we described the inner workings of LLMs and their relationship with multi-task learning. This examination has demonstrated how LLMs embody the characteristics of meta-learners, showcasing their potential for leveraging shared understanding across tasks and domains.

Language conditioning, as discussed in the second part of our paper, offers a groundbreaking approach that empowers robots to leverage the vast knowledge stored within language models. By conditioning their behavior on linguistic inputs, robots can improve their generalization abilities and adapt more efficiently to novel situations. This innovative approach has significant implications for advancing robotic capabilities and has the potential to revolutionize human-robot interaction.

Finally, we introduced Palm-E, an exemplary embodied language model developed by Google. Palm-E showcases the potential of ELMs by seamlessly integrating vision tasks, language tasks, and embodied tasks. This integration enables the embodied model to effectively plan complex action sequences on new tasks, pushing us closer to the realization of generalist robots.

9. REFERENCES

[1] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch, and P. Florence, "Palm-e: An embodied multimodal language model," 2023.

[2] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng, "Do as i can, not as i say: Grounding language in robotic affordances," 2022.

[3] Chelsea Finn and Karol Hausman, "CS330: Deep Multi-Task and Meta Learning, Fall 2021." `http://cs330.stanford.edu/fall2021/index.html`. Accessed: June 14, 2023.

[4] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel, "Palm: Scaling language modeling with pathways," 2022.

[5] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen, E. Chu, J. H. Clark, L. E. Shafey, Y. Huang, K. Meier-Hellstern, G. Mishra, E. Moreira, M. Omernick, K. Robinson, S. Ruder, Y. Tay, K. Xiao, Y. Xu, Y. Zhang, G. H. Abrego, J. Ahn, J. Austin, P. Barham, J. Botha, J. Bradbury, S. Brahma, K. Brooks, M. Catasta, Y. Cheng, C. Cherry, C. A. Choquette-Choo, A. Chowdhery, C. Crepy, S. Dave, M. Dehghani, S. Dev, J. Devlin, M. Díaz, N. Du, E. Dyer, V. Feinberg, F. Feng, V. Fienber, M. Freitag, X. Garcia, S. Gehrmann, L. Gonzalez, G. Gur-Ari, S. Hand, H. Hashemi, L. Hou, J. Howland, A. Hu, J. Hui, J. Hurwitz, M. Isard, A. Ittycheriah, M. Jagielski, W. Jia, K. Kenealy, M. Krikun, S. Kudugunta, C. Lan, K. Lee, B. Lee, E. Li, M. Li, W. Li, Y. Li, J. Li, H. Lim, H. Lin, Z. Liu, F. Liu, M. Maggioni, A. Mahendru, J. Maynez, V. Misra, M. Moussalem, Z. Nado, J. Nham, E. Ni, A. Nystrom, A. Parrish, M. Pellat, M. Polacek, A. Polozov, R. Pope, S. Qiao, E. Reif, B. Richter, P. Riley, A. C. Ros, A. Roy, B. Saeta, R. Samuel, R. Shelby, A. Slone, D. Smilkov, D. R. So, D. Sohn, S. Tokumine, D. Valter, V. Vasudevan, K. Vodrahalli, X. Wang, P. Wang, Z. Wang, T. Wang, J. Wieting, Y. Wu, K. Xu, Y. Xu, L. Xue, P. Yin, J. Yu, Q. Zhang, S. Zheng, C. Zheng, W. Zhou, D. Zhou, S. Petrov, and Y. Wu, "Palm 2 technical report," 2023.

[6] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy, "The pile: An 800gb dataset of diverse text for language modeling," 2020.

[7] A. Kumar, A. Raghunathan, R. Jones, T. Ma, and P. Liang, "Fine-tuning can distort pretrained features and underperform out-of-distribution," 2022.

[8] S. Ruder, "An overview of multi-task learning in deep neural networks," 2017.

[9] C. Finn, *Learning to Learn with Gradients*. PhD thesis, EECS Department, University of California, Berkeley, Aug 2018.

[10] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," 2017.

[11] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, pp. 293–321, 1992.

[12] T. Bräm, G. Brunner, O. Richter, and R. Wattenhofer, "Attentive multi-task deep reinforcement learning," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part III*, pp. 134–149, Springer, 2020.

[13] A. Wilson, A. Fern, S. Ray, and P. Tadepalli, "Multi-task reinforcement learning: a hierarchical bayesian approach," in *Proceedings of the 24th international conference on Machine learning*, pp. 1015–1022, 2007.

[14] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," *Advances in neural information processing systems*, vol. 30, 2017.

[15] R. Yang, H. Xu, Y. Wu, and X. Wang, "Multi-task reinforcement learning with soft modularization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 4767–4777, 2020.

[16] C. Lynch, A. Wahid, J. Tompson, T. Ding, J. Betker, R. Baruch, T. Armstrong, and P. Florence, "Interactive language: Talking to robots in real time," 2022.

[17] A. M. Saxe, A. C. Earle, and B. Rosman, "Hierarchy through composition with multitask LMDPs," in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, pp. 3017–3026, PMLR, 06–11 Aug 2017.

[18] A. Gupta, J. Yu, T. Z. Zhao, V. Kumar, A. Rovinsky, K. Xu, T. Devlin, and S. Levine, "Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6664–6671, IEEE, 2021.

[19] M. Liu, M. Zhu, and W. Zhang, "Goal-conditioned reinforcement learning: Problems and solutions," 2022.

[20] Y. Bengio. 2009.

[21] Y. Chebotar, K. Hausman, Y. Lu, T. Xiao, D. Kalashnikov, J. Varley, A. Irpan, B. Eysenbach, R. Julian, C. Finn, and S. Levine, "Actionable models: Unsupervised offline reinforcement learning of robotic skills," *arXiv preprint arXiv:2104.07749*, 2021.

[22] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[24] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, *et al.*, "An algorithmic perspective on imitation learning," *Foundations and Trends® in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.

[25] D. Michie, M. Bain, and J. Hayes-Miches, "Cognitive models from subcognitive skills," *IEE control engineering series*, vol. 44, pp. 71–99, 1990.

[26] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet, "Learning latent plans from play," in *Proceedings of the Conference on Robot Learning* (L. P. Kaelbling, D. Kragic, and K. Sugiura, eds.), vol. 100 of *Proceedings of Machine Learning Research*, pp. 1113–1132, PMLR, 30 Oct–01 Nov 2020.

[27] J. Testud, J. Richalet, A. Rault, and J. Papon, "Model predictive heuristic control: Applications to industial processes," *Automatica*, vol. 14, no. 5, pp. 413–428, 1978.

[28] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *International conference on machine learning*, pp. 5331–5340, PMLR, 2019.

[29] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "RL^2: Fast reinforcement learning via slow reinforcement learning," *arXiv preprint arXiv:1611.02779*, 2016.

[30] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," *arXiv preprint arXiv:1611.05763*, 2016.

[31] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," *arXiv preprint arXiv:1707.03141*, 2017.

[32] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," *arXiv preprint arXiv:1803.11347*, 2018.

[33] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," 2022.

[34] OpenAI, "Gpt-4 technical report," 2023.

[35] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems* (D. Touretzky, ed.), vol. 2, Morgan-Kaufmann, 1989.

[36] D. E. Rumelhart and J. L. McClelland, *Learning Internal Representations by Error Propagation*, pp. 318–362. 1987.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[38] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2014.

[39] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020.

[40] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10674–10685, 2021.

[41] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[42] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[43] B. Peng, E. Alcaide, Q. Anthony, A. Albalak, S. Arcadinho, H. Cao, X. Cheng, M. Chung, M. Grella, K. K. GV, X. He, H. Hou, P. Kazienko, J. Kocon, J. Kong, B. Koptyra, H. Lau, K. S. I. Mantri, F. Mom, A. Saito, X. Tang, B. Wang, J. S. Wind, S. Wozniak, R. Zhang, Z. Zhang, Q. Zhao, P. Zhou, J. Zhu, and R.-J. Zhu, "Rwkv: Reinventing rnns for the transformer era," 2023.

[44] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," 2022.

[45] A. Bulatov, Y. Kuratov, and M. S. Burtsev, "Scaling transformer to 1m tokens and beyond with rmt," 2023.

[46] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre, "Training compute-optimal large language models," 2022.

[47] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus, "Emergent abilities of large language models," 2022.

[48] Q. Dong, L. Li, D. Dai, C. Zheng, Z. Wu, B. Chang, X. Sun, J. Xu, L. Li, and Z. Sui, "A survey on in-context learning," 2023.

[49] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," 2022.

[50] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 2021.

[51] D. Dai, Y. Sun, L. Dong, Y. Hao, S. Ma, Z. Sui, and F. Wei, "Why can gpt learn in-context? language models implicitly perform gradient descent as meta-optimizers," 2023.

[52] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[53] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[54] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*, pp. 8748–8763, PMLR, 2021.

[55] N. Shazeer and M. Stern, "Adafactor: Adaptive learning rates with sublinear memory cost," 2018.

[56] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[57] M. Dehghani, J. Djolonga, B. Mustafa, P. Padlewski, J. Heek, J. Gilmer, A. Steiner, M. Caron, R. Geirhos, I. Alabdulmohsin, R. Jenatton, L. Beyer, M. Tschannen, A. Arnab, X. Wang, C. Riquelme, M. Minderer, J. Puigcerver, U. Evci, M. Kumar, S. van Steenkiste, G. F. Elsayed, A. Mahendran, F. Yu, A. Oliver, F. Huot, J. Bastings, M. P. Collier, A. Gritsenko, V. Birodkar, C. Vasconcelos, Y. Tay, T. Mensink, A. Kolesnikov, F. Pavetić, D. Tran, T. Kipf, M. Lučić, X. Zhai, D. Keysers, J. Harmsen, and N. Houlsby, "Scaling vision transformers to 22 billion parameters," 2023.

[58] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[59] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.