

Training a Massively Multimodal Transformer on YouTube Data:
Pre-training and Parameter Efficient Fine-Tuning on HPC Infrastructure

BY

KASTAN DAY

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2023

Urbana, Illinois

Adviser:

Research Associate Professor Dr. Volodymyr Kindratenko

ABSTRACT

In machine learning, the widespread adoption of pre-trained large language models (LLMs) across many domains has disrupted conventional wisdom on the right model to use for the job. This work investigates the advantages of pre-training large language models from scratch over fine-tuning existing ones in specific scenarios, such as exploring the importance of custom tokenizers for domain-specific applications, addressing information leakage concerns, and examining the use of LLMs in non-traditional applications such as time-series forecasting. When pre-training is unnecessary, this work argues that select parameter efficient fine-tuning (PEFT) methods are strictly superior to traditional-fine tuning for data and computational efficiency and should be preferred in nearly all cases. Furthermore, after PEFT, it is ideal to further sculpt the outputs of one's LLM with Reinforcement Learning with Human Feedback (RLHF). This work argues that reinforcement learning (RL), rather than any form of supervised fine tuning (SFT), is preferable to achieve truthfulness without hallucination. Practitioners should seek to leverage the benefits of RLHF via RL with AI feedback (RLAIF), an effective, fast, and economical alternative to human feedback that retains the benefits of reward modeling for factuality. Additionally, the paper discusses the three most successful learning objectives in multimodal transformers and the challenges they face in aligning distinct embedding spaces. I present my own model Video Pre-trained Transformer: A Multimodal Mixture of Pre-Trained Experts for video question answering tasks as benchmarked against VQAv2. The importance of modern ML-first databases and filesystems is explored in the context of multimodal, multi-model AI systems for fast and flexible data throughput on HPC and multi-cloud systems. Together, the paper represents the state of open-source LLMs and opportunities for researchers and practitioners to combine the attractive properties of PEFT, RLAIF, and multi-modal transformers that paves the way for the next few years of growth in AI capabilities.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION.....	1
CHAPTER 2: PRE-TRAINING VS FINE-TUNING	3
CHAPTER 3: PEFT: PARAMETER EFFICIENT FINE-TUNING.....	7
CHAPTER 4: DEBUNKING THE TRUE PURPOSE OF RLHF	14
CHAPTER 5: MULTIMODAL TRANSFORMERS.....	21
5.1 Cross Attention.....	21
5.2 Small MLP Mapping Projection.....	22
5.3 Joint Embedding and Contrastive Learning	23
CHAPTER 6: VIDEO PRE-TRAINED TRANSFORMER: A MULTIMODAL MIXTURE OF PRE-TRAINED EXPERTS.....	26
6.1 Contributions	28
6.2 Results	31
6.3 Compute Infrastructure.....	32
6.4 Data Infrastructure.....	33
6.5 Conclusion	36
REFERENCES	38

CHAPTER 1: INTRODUCTION

When is it necessary to fully pre-train a model versus simply fine-tuning an existing open-source model with your own data? I find that fine-tuning is always more effective than fully pre-training a model except in two cases (1) when one’s target domain has zero overlap with any existing open-source model such as for specialized chemistry or physics problems where the data format is unusual, such as chemical sequences, and no existing model can understand it. Or (2) when one proposes a novel architecture, full pre-training is always required. Yet, today, full pre-training requires less developer time and expertise, while being more efficient than ever thanks to innovation in parallel and distributed model training offered by native PyTorch Fully Sharded Data Parallel (FSDP) and DeepSpeed Zero Redundancy Optimizer (ZeRO). Chapter 2 discusses further reasons to pre-train vs finetune a model with standout examples from industry.

For ML practitioners, fine-tuning large language models (LLMs) has never been better thanks to a proliferation of parameter efficient fine-tuning (PEFT) methods, most notably Low Rank Adaptation (LoRA). These methods freeze 100% of the original parameters of the model and strategically add small amounts of new parameters, in the range of 0.1% to 1% of the original model size, to be learned during fine-tuning. The most successful methods outperform traditional fine-tuning on downstream evaluations while being several orders of magnitude faster to train with smaller memory devices. Today it’s possible to fine-tune 6B parameter models on a single consumer 24 GB GPU, or much larger models using ZeRO parameter offload to the system’s DRAM memory or even SSD. With these innovations, fine-tuning is nearly universally accessible paving the way for customized machine learning (ML) for every purpose.

Finally, the transformer architecture continues to evolve and scale to new domains. Starting with text then images and now multi-modal visual-language+ transformers take top spot on most ML benchmarks, with the notable exception of tabular data where gradient boosted decision trees still shine. I review the state-of-the-art multi-modal transformers, classify three successful categories of self-supervised learning objectives, and finally present my own architecture of a four-modality mixture-of-experts transformer name Video Pretrained Transformer.

This work’s primary contributions are:

1. A review of pre-training vs fine-tuning, and when to deploy each tool, with a deep analysis of the proliferation of PEFT methods that make fine-tuning more powerful than ever.

2. An analysis of Reinforcement Learning with Human Feedback (RLHF) as a fine-tuning method an argument that reinforcement learning reward modeling prevents LLM hallucinations more effectively than traditional fine tuning.
3. A review of multimodal transformer learning methods, with an emphasis on architectures making efficient use of existing pre-trained LLMs and minimize the need for re-training.
4. A novel multimodal transformer architecture building on the “Embedding → Trunk → Head” design pattern to leverage the immense capabilities of existing open-source models while minimizing necessary compute during fine-tuning.
5. A novel use of scene graphs, in addition to language, vision and audio, as structured information into for multimodal transformers.
6. An uncommon training corpus of the recently introduced YT-Temporal-1B dataset [1], augmented with 25k additional hand selected interesting YouTube videos.

CHAPTER 2: PRE-TRAINING VS FINE-TUNING

In which scenarios are pre-training a LLM from scratch more advantageous than fine-tuning an existing one? While fine-tuning a pre-trained model has been the standard approach in various natural language processing tasks, there are specific cases where training a new model is more appropriate.

Firstly, pre-training a new LLM is necessary when a dramatically different tokenizer is required. Tokenization plays a crucial, often under-appreciated [2], role in LLMs, as it determines the granularity of the input representation and may make it substantially easier or harder for a model to represent concepts like number sequences, chemical formulations, technical jargon and all forms of other domain-specific language. For instance, if the target application involves a language with unique morphological features, a specialized domain with a distinct vocabulary, or specific use cases, a custom tokenizer might be needed to capture these characteristics effectively. Domain-specific applications, such as medical, chemistry, or financial analysis (e.g., Bloomberg dataset [3]), often require tailored tokenization to handle specialized terminologies and notations. Similarly, time-series analysis with Transformers in areas like sensor or manufacturing monitoring (e.g., IBM's approach to timeseries transformers [4]) may necessitate a tokenizer that can efficiently process sequential data. In such cases, pre-training a new model with the desired tokenizer is more beneficial than fine-tuning an existing one, as the latter may not be able to adapt to the new tokenization scheme efficiently. All modern transformers use Byte Pair Encoding (BPE) tokenizers, and there is a steady trend to increase the vocab size. Most popular open-source transformers today use ~50k tokens, often directly copied from the GPT-2 tokenizer. However, Google's most capable model for the last several years has been the Pathways Language Model (PaLM) which uses a very large vocabulary size at 256k tokens [5]. Especially as models get wider and deeper, the number of trainable params allocated to token embeddings becomes proportionally very low such that it makes sense to scale vocab-size with model size to obtain the maximum benefit from increased scale.

Another scenario where pre-training from scratch is preferred is when information leakage is a significant concern. Pre-trained LLMs have been shown to inadvertently memorize and reveal sensitive information from their training data [6], [7]. This leakage can pose privacy risks, especially when dealing with confidential or proprietary data. In these situations, pre-training a new LLM on a carefully curated dataset can help mitigate the risk of information leakage and



Figure 2.1 Visualizing some of the 604 tasks and actions that Gato was trained on. Figure from [6].

ensure data privacy. Importantly, the cost to pre-train and fine-tune a model from scratch is decreasing. MosaicML quotes the cost of training a 2.7B model on 300B tokens, using a placeholder pricing of \$2/A100/hr, was \sim \$38,000 [8]. This brings fully domain-specific models within reach for a much wider range of use cases.

Gato: A Generalist Agent. A key contribution of Gato [9] is essentially “tokenizers are all you need.” The Gato tokenizer is unique in its ability to handle diverse modalities such as text, images, proprioception, joint torques, button presses, and other discrete and continuous observations and actions, as shown in Figure 2.1. To enable processing this multi-modal data, the authors of the paper designed a tokenization scheme that serializes all data into a flat sequence of tokens. In this representation, Gato can be trained and sampled from akin to a standard Encoder-Decoder, or Seq2Seq, language model. During deployment, sampled tokens are assembled into dialogue responses, captions, button presses, or other actions based on the context.

The tokenizer is highly engineered, in a manner akin to feature engineering in traditional tabular decision-tree style ML, with common-sense decisions made to tokenize each modality. For example, text is encoded via SentencePiece with 32,000 subwords into the integer range $[0, 32000)$. Images are first transformed into sequences of non-overlapping 16×16 patches in raster order. Each pixel in the image patches is then normalized between $[-1, 1]$ and divided by the square-root of the patch size. Discrete values are flattened into sequences of integers in row-major order. Continuous values are first flattened into sequences of floating-point values in row-major order. The values are mu-law encoded to the range $[-1, 1]$ if not already there, then

discretized to 1024 uniform bins. Since all modalities are projected into the same vocabulary space, this uniquely enables Gato to handle many inputs and tasks with a single set of weights.

Vid2Seq: injecting special tokens. The Vid2Seq model [10] uses the standard T5 tokenizer but appends new “timestamp tokens,” as shown in figure 2.2, to the end of the tokenizer integer range that allow the model to predict both *what* events happen in videos and *when* they occur. This allows the model to seamlessly understand and generate sequences of tokens containing both textual semantic information and temporal localization information grounding each text sentence in the video. The time tokens represent relative timestamps in a video, as the video of arbitrary duration is quantized into 100 equally spaced timestamps.

Domain-specific LLMs: BioMedLM [11] & BloombergGPT [3]. The BloombergGPT tokenizer is unique because it uses a Unigram model instead of a greedy merge-based sub-word tokenizer, such as BPE or WordPiece. The authors trained their tokenizer on The Pile dataset [12] and used a split and merge approach to handle the large size of the dataset. The final tokenizer has 7 million tokens and a vocabulary size of 131,072. The authors selected this vocabulary size based on experiments with vocabulary sizes ranging from 25,000 to 550,000. Their heuristic was to choose the vocabulary size that led to the smallest encoded representation of the C4 dataset. The authors carefully designed and trained their tokenizer to achieve high performance on financial tasks while maintaining competitive performance on general-purpose LLM benchmarks.

The BioMedLM, also called PubMedGPT 2.7B, uses a custom tokenizer trained on PubMed abstracts. The authors of the model found it important to use a tokenizer trained on in-domain

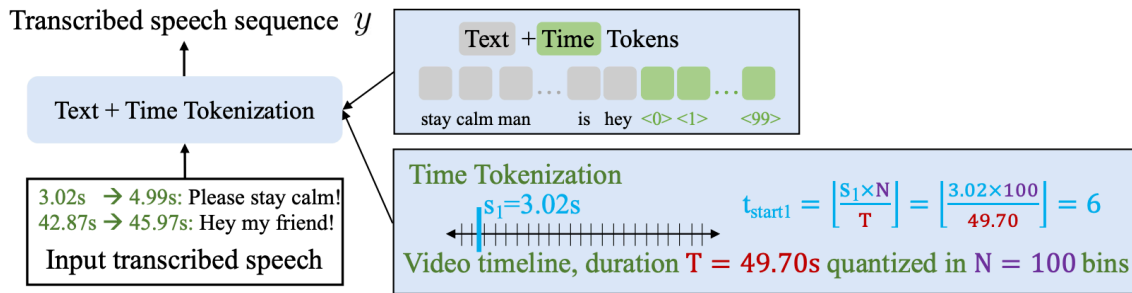


Figure 2.2 Timestamp tokens in Vid2Seq. This creative approach injects 100 timestamp tokens into the vocabulary, where each token represents a relative position in the video, essentially like a ‘percent complete’ marker, which enables arbitrary length videos to be represented by 100 tokens. The limitation is longer videos are truncated. Figure from [7].

text to maximize performance on downstream tasks. A key benefit of this approach is that common biomedical terms are represented as entire tokens. For instance, terms such as "chromatography", "cytotoxicity", "Immunohistochemistry", "photosynthesis", and "probiotic" are tokenized into single tokens by the biomedical tokenizer and multiple tokens by the standard GPT-2 tokenizer. This allows the model to encode information about these concepts in their individual token representations rather than spread out across subword tokens like "oh" shared with many other terms.

Non-traditional Transformer Applications: Time-series forecasting, also called Long Sequence Time-series Forecasting or the LSFT problem. "A Time Series is worth 64 Words" proposes an efficient design for Transformer-based models in multivariate time series forecasting and self-supervised representation learning [4]. The approach involves segmenting time series into subseries-level patches and using channel-independence to reduce computation and memory usage. The proposed model outperforms other baselines in supervised learning and shows promising capability in self-supervised representation learning and transfer learning. The key contributions of this work are the introduction of patching and channel-independent structure, which enable the model to capture local semantic information, benefit from longer look-back windows, and efficiently work with timeseries data in a transformer architecture.

In a similar work, Multivariate Time Series Representation Learning [13], the framework includes an unsupervised pre-training scheme, which offers substantial performance benefits over fully supervised learning on downstream tasks. The authors evaluate their framework on several public multivariate time series datasets from various domains and demonstrate that it performs significantly better than the best currently available methods for regression and classification. These findings represent an important landmark, presenting the first unsupervised method shown to push the limits of state-of-the-art performance for multivariate time series regression and classification.

A key limitation of using transformers for time-series forecasting is the limited context window size, and the quadratic computation time of vanilla self-attention. Informer [14] addresses these issues by (1) introducing sparsity in attention, dubbed ProbSparse attention, reducing the cost of attention from $O(S^2H)$ to $O(S \log S)$ where S is sequence length and H is hidden dimension. Additionally, the authors reduce the memory footprint of the layers of transformer blocks from $O(S^2N)$ to $O(N S \log S)$ where N is the number of transformer blocks

by simply cutting in half the size of the embedding passed between transformer blocks using a distillation operation. A plethora of optimizations are emerging to increase the computational efficiency of Transformers, in the next section I examine which methods are best in practice.

CHAPTER 3: PARAMETER EFFICIENT FINE-TUNING

The rapid development of pre-trained language models (PLMs) has revolutionized the field of natural language processing, enabling state-of-the-art performance on a wide range of downstream tasks. However, fine-tuning these large-scale PLMs can be computationally expensive and resource-intensive, which limits their applicability in many real-world scenarios. To address this challenge, Parameter-Efficient Fine-Tuning (PEFT) methods have emerged as a promising alternative, allowing for efficient adaptation of PLMs to various downstream applications without the need to fine-tune all the model's parameters [15]. The mechanism of action of PEFT methods have touched every module of the transformer architecture as taxonomized in figure 3.1. Nearly always practitioners use both math-preserving systems optimizations, most notably DeepSpeed ZeRO optimizer sharding and CPU offload [16]–[18], alongside math-altering fine-tuning methods, most notably LoRA and the updated Adaptive Budget Allocation LoRA (AdaLoRA) [19], [20].

LoRA, as presented in figure 3.2, presents a groundbreaking method that enables the training of 20-Billion parameter LLMs on a single 24GB consumer GPU, and a 50-60B parameter model on a single 80GB A100 [21], [22]. It's even possible to fine-tune OPT-6.7B (13GB in float16) in

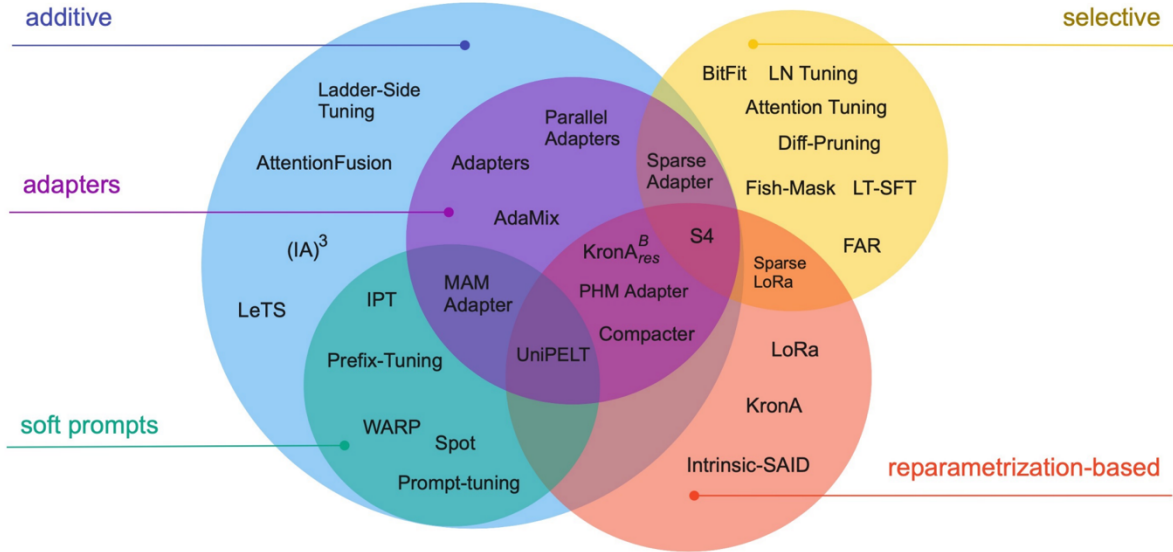


Figure 3.1 A taxonomy of PEFT methods examined in [18]. The authors identify three main classes of methods: Addition-based, Selection-based, and Reparameterization-based. Additive methods are further classified into adapter-like methods and soft prompts.

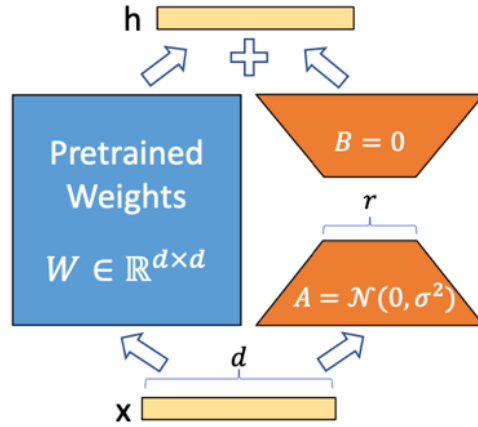


Figure 3.2 LoRA architecture visualizing a single weight matrix, like attention K, Q, V or FFN. There is one adapter per weight matrix in the Transformer, except embedding and projection layers. On the left are the pre-trained weights, which are 100% frozen. On the right are the low rank adapters. Figure from [16].

a free Google Colab notebook. This innovative approach demonstrates both superior task performance and superior training speed compared to all other fine-tuning techniques considered here. LoRA works by freezing the pre-trained model, and simply adding an auto-encoder style ‘bottleneck’ to each transformer block. During finetuning, only the LoRA parameters are updated and added to the output of the pre-trained weights. The authors attribute this success to the theoretical underpinnings of Singular Value Decomposition (SVD) and low-rank adaptation, although the extent to which these concepts are rigorously supported remains a topic of debate within the machine learning community. A key aspect of LoRA is the utilization of SVD to determine which ranks of the matrix should be selected for modification and which should remain unchanged. This selective process contributes to the overall efficiency and effectiveness of the method, making it a promising avenue for further exploration in large-scale model training.

LoRA offers several benefits over other Adapter design patterns, including improved interpretability of update weights, emphasis on important patterns, and the ability to merge weights into the original pre-training weights after fine-tuning without introducing additional latency or computational overhead during inference. Motivated by theory, LoRA specifically employs “rank decomposition matrices” as the learnable parameters because in ML theory “it is well known that for many deep learning tasks, especially those with a heavily over-parametrized

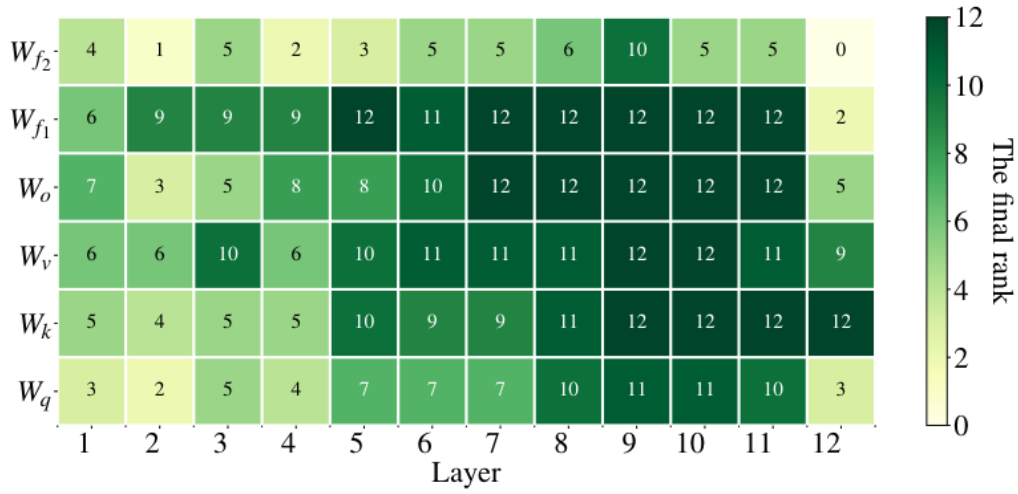


Figure 3.4 AdaLoRA’s importance metric allocates the bulk of trainable parameters towards the later layers of the model, except for the final layer. Notably, all parameters except the final feed-forward layer were allocated at least some amount of fine-tuning capacity. The x-axis is the layer index, and the y-axis represents different types of adapted weight matrices. Figure from [19].

neural network, the learned neural network will enjoy low-rank properties after training.”

Therefore, a rank decomposition matrix offers the minimum number of parameters with the maximal ability to affect the latent representation because it can best manipulate the low rank pre-trained weights. The widely used implementation of LoRA adapts too all major weight

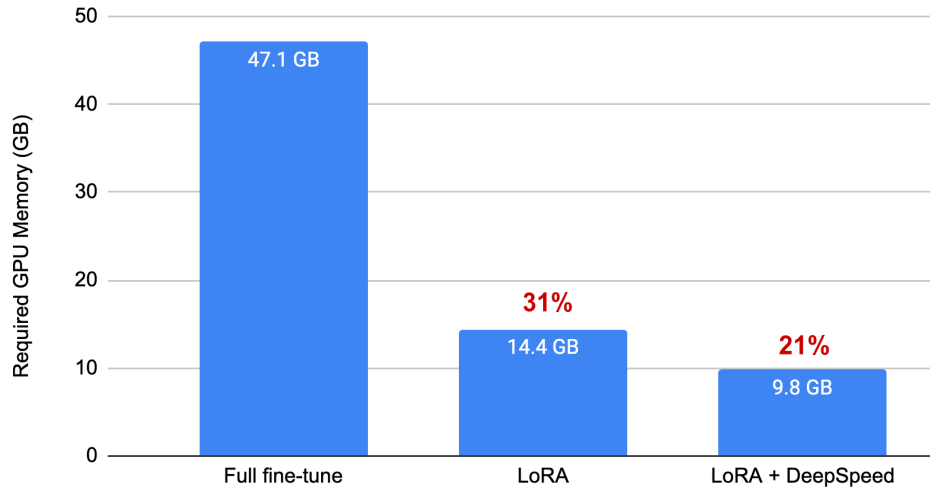


Figure 3.3 Dramatic 70 to 80% GPU memory savings using LoRA and DeepSpeed stage 3 CPU offload make large model fine-tuning available to researchers of any scale. Testing T0-XL 6B model memory savings using data collected from [13].

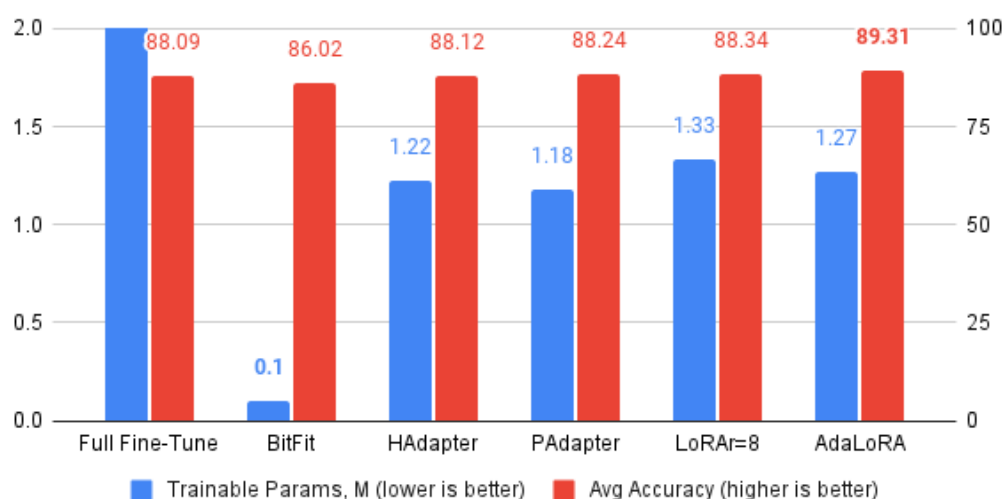


Figure 3.5 Comparing PEFT methods in computational efficiency vs accuracy. LoRA performs best and is tied for least number of trainable params at 775M trainable params, which is 1/10th of a percent or 1,000x less than full fine-tuning. Data from experiments in AdaLoRA [18].

matrices of a transformer block: query, key and value in self attention and all MLP blocks, yet it does not affect the embedding layers. By inspecting the surprisingly strong correlation of the weights learned in LoRA versus the pre-trained weights, as shown in figure 3.4, the authors claim that low-rank adaptation “potentially *amplifies the important features* for specific downstream tasks that were learned but not emphasized in the general pre-training model” [19].

When used to fine-tune a GPT-3-style-model, LoRA achieved superior accuracy on all examined downstream tasks, WikiSQL and MNLI-m, with 10,000 times fewer parameters that required only 1/3rd of the GPU memory, as shown in figure 3.3. LoRA does not alter the original, or base, model and requires all parameters during the forward pass, which is why the GPU memory savings are not greater despite having so few trainable parameters. Another key benefit of LoRA is that it is completely invisible, performance wise, after it is complete. The learned weights are merged into the original pre-training weights after fine-tuning, thus not introducing any latency or computational overhead during inference, making it certain to work with existing systems and SLAs.

Hugging Face PEFT’s [15] support for LoRA makes it by far the most practical method of fine-tuning LLMs with custom data and as little as a single consumer GPU. Moreover, integration with Accelerate makes efficient use of additional heterogenous GPUs, again making

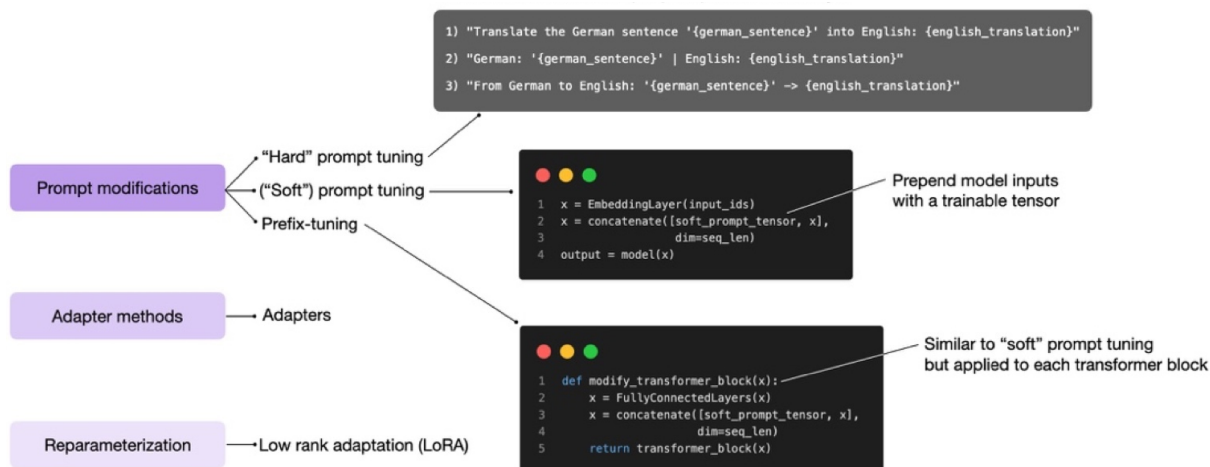


Figure 3.6 A visual comparison between Prompt Tuning vs Prefix Tuning vs LoRA [17].

it an ideal choice for researchers without racks of identical servers. From my involvement in communities of ML practitioners, it appears that LoRA is commonly used and with great fanfare, like in many RLHF libraries like TRL [23] and TRLX [24]. Moreover, I have not seen any notable ML projects using any other PEFT method. A comparison of PEFT methods' efficiency and accuracy on downstream tasks is made in figure 3.5.

Prompt Tuning Methods are a class of parameter-efficient techniques that enable the adaptation of PLMs to specific tasks by optimizing the input prompts rather than fine-tuning the entire model. These methods have gained attention due to their ability to achieve competitive performance with reduced computational and storage costs.

"Hard" prompt tuning refers to traditional prompt engineering, where humans try to find the best permutation of natural language to constrain the model's outputs. Hard prompts are made of discrete non-differentiable vocab tokens. "Soft" prompt tuning refers to skipping the tokenization step and working directly with a fixed number of word embeddings, which are differentiable. In soft prompting, the model weights are kept frozen and gradient descent is used to update the embedding vectors in the first n embeddings, which is the soft prompt. Finally, P-tuning is the same as prompt tuning, but where the soft-prefix is applied to the input of every transformer block, not just the initial input sequence. A visualization of the differences is provided in figure 3.6. Here, I provide a brief description of four notable prompt tuning methods:

Prefix Tuning [25] is the original work introducing "soft prompts" wherein the authors introduce a method that optimizes continuous tokens, or prefix, which are prepended to the input

sequence. This prefix is task-specific and learned during training, allowing the model to generate task-appropriate responses without modifying the underlying PLM's parameters. Importantly, the context size is not altered, simply the first few tokens of the input are hijacked to be used as learnable soft-prompts and the input is truncated if necessary to fit in the remaining context window. **Prompt Tuning** [26] further demonstrated that as the size of the PLM increases, prompt tuning becomes more parameter-efficient and can achieve competitive performance with a smaller number of trainable parameters compared to full fine-tuning, which achieving nearly the same if not identical performance.

P-Tuning [27] extends prefix tuning by adding a learnable prefix to each transformer block, in addition to the initial input. This allows the network to learn the right “prompt” to fine-tune the communication between each transformer block. It’s similar to LoRA but without the bottleneck layers. **P-Tuning v2**’s [28] follow up work mirrors AdaLora addresses the question: how can I allocate the parameter budget adaptively according to importance of modules to improve the performance of parameter-efficient fine-tuning? P-Tuning proposes learning to completely skip certain transformer blocks which reduces the cost of both the forward and backward pass. It also further demonstrates that prompt tuning can achieve competitive performance across a wide range of tasks and model scales, making it a versatile and efficient alternative to full fine-tuning.

LLaMA-Adapter with Zero-Init Attention [29] provides the key insight that initializing adapter layers to zero “prevents disturbing the loss at the beginning of fine-tuning” and

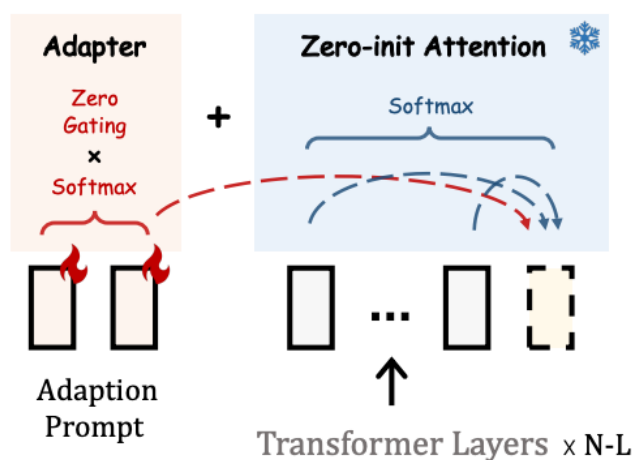


Figure 3.7 Details of LLaMA-Adapter. The red arrow represents cross attention paid to the Adapter embeddings, which are added to the input of each transformer layer. Figure adapted from [28].

empirically performs much better than random-init of adapter layers, as detailed in figure 3.7. The intuition is that zero-init starts the model training from right where it left off, and, as part of fine tuning, progressively and additively change adapt the weights without any disturbance or discontinuity in the learning process.

Common between all implementations, PEFT methods are designed to be more data and compute efficient than traditional fine-tuning methods by leveraging the fundamental observation that it's cheaper to freeze the pre-trained parameters and strategically add new learnable parameters than to fine-tune all the existing pre-trained parameters. The amount of data required for PEFT to work effectively depends on various factors, such as the complexity of the task, the quality of the pre-trained language model, and the desired performance level. Data size guidelines and scaling laws are presently turbulent as researchers reconcile tension between the emergence of new capabilities as model size scales with how well small models perform when given unprecedented amounts of training data. The next section explores the benefits of high-quality fine-tuning data in the form of direct human feedback.

CHAPTER 4: DEBUNKING THE TRUE PURPOSE OF RLHF

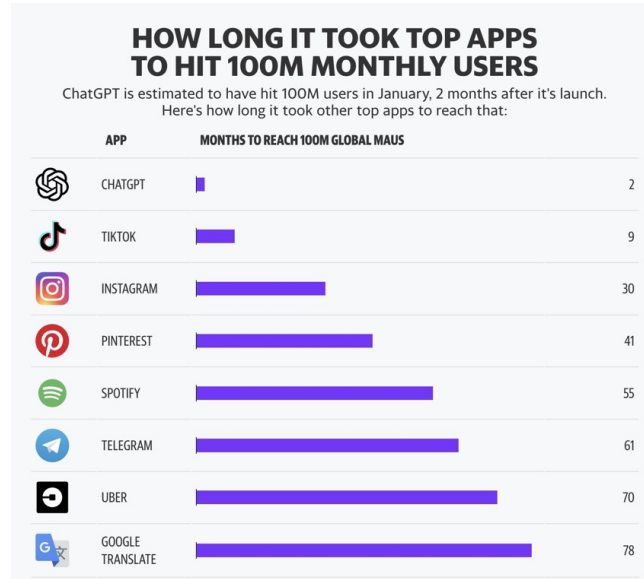


Figure 4.1 ChatGPT’s Time from product launch to reach 100M MAU, global leaderboard [22].

RLHF has been lauded as the primary innovation that transformed GPT-3 from a groundbreaking technical achievement into the #1 fastest growing product in human history: it had 1M users in 5 days, and 100M users in 2 months [30], best illustrated in figure 4.1. However, despite sharing broad outlines, important details of the process have not been shared by OpenAI, and the ML community needs clarity on best practices for replicating this process with equal success. Here I discuss the fundamental intuitive difference between pre-training and fine-tuning, and Section 5 discusses the current State of The Art (SOTA) self-supervised learning objectives used in pre-training.

The Purpose of Pre-training. The fundamental property of pre-training is *compression* via a *self-supervised learning objective* suitable for unlabeled data, i.e., the AI model learns to reproduce terabytes of training data using only gigabytes of model parameters (1,000x compression). To compress the training data most effectively, memorization is insufficient, and the AI learns increasingly useful patterns. Eventually, the best solution is to model the underlying process that generated the data in its training corpus, i.e., learn a model of human behavior, as projected as text on the internet, to predict what humans will say next. Ilya Sutskever, Chief Scientist OpenAI, says it this way “eventually you need to understand the true underlying process that produced the data to predict the data well, to compress it well, you need

to understand more and more about the world that produced the data” [31]. Sutskever further claims that despite images being a more natural representation of some concepts, it is, in principle, possible to learn any concept from text alone. He justifies this by examining the embedding vectors of color names, like red and pink, in GPT-3 and claims that similar colors have similar embeddings, and the model excels at the compositional reasoning necessary to make any color by mixing arbitrary other colors. Certainly, he concedes, some concepts are more densely and richly represented in images, but he maintains that text alone is sufficient for practically any reasoning task.

The purpose of pre-training, then, is to learn a model of the underlying process that produced the training data. It’s focused on the inputs; it’s designed to capture generalizable knowledge. Via compression, the AI learns a predictive model of the world.

Debunking RLHF. On the other hand, RLHF is concerned with the outputs. This dichotomy is depicted in figure 4.2. The “Human Feedback” in RLHF is crucial to learn objectives that are difficult to define explicitly, like helpfulness and harmlessness. It’s to *sculpt the outputs* into the right style, structure, and detail; to teach it that “hallucination is never okay” [31].

For example, consider how a model would respond to an out of domain prompt before and after RLHF. If a model has never seen a concept during pre-training, then it won’t know about it, like a student who never took a course in school. Without RLHF, the model might produce random noisy outputs, or latch onto the wrong concept, when asked about unfamiliar topics. But after RLHF, the model would respond like a professional colleague, saying something like ‘As an AI Language Model, my training data only contains information from 2021 and before, so I don’t know how to help with your question.’

Reward Models will be Foundation Models. My assertion is that much like GPT-3/4 is a foundation model suitable for many downstream tasks, Reward Model Foundation Models (RM FM) will be wildly useful as well. OpenAI’s stated policy [32] is to create a base RM tuned with wide, all-inclusive content guidelines, then will create a suite of fine-tuned versions tuned for different sensibilities on content moderation, willingness to answer potentially harmful



Figure 4.2 Pre-training builds a world model and RLHF constrains the output.

questions, verbosity and many more dimensions of preferences. These reward models are akin to personalities that that chatbot can adopt. Moreover, there exists another class of RMs that tune LLMs to perform duties other than “chat” such as use tools like web browsers, APIs, and databases. These models are tuned to be workers, often controlled by other AIs, instead of interfacing directly with humans.

To organize the tool-using LLMs, other models will be tuned to be managers. Taken to the extreme, it’s conceivable that every knowledge-worker job today will have an AI equivalent, tuned for just that role. This enables multi-AI teams to work together on long-horizon projects, leaning on the org chart of existing, say, SAAS companies to define the information flow between AI workers. With GPT-4, these designs seem possible for the first time thanks to the incredible instruction-following capability of that model, but it’s still early and these multi-AI collaboration systems may require domain-specific RMs to find success in practical, and therefore difficult and long-horizon tasks.

Calibration: LLMs Know what they Know. Fortunately, LLMs, mostly, know what they know [33]. This is particularly true for GPT-4, which can estimate its own probability of being correct with 82% accuracy, across the entire curve from 0% to 100% confidence. However, this is only true for the pre-trained “base” model and the RLHF process, by OpenAI, destroyed this calibration. This makes sense because humans are usually on the extreme ends of “I know” or “I don’t know” with very little consideration for confidence levels between 10% and 90%, especially as a paid data labeler who is not incentivized to write, for example, answers with 50% confidence. This is perfectly illustrated in Figure 4.3, where after RLHF the model is either 0%, or 90% confident in its correctness on each extreme, and otherwise only approximately 35% confident thus ruining the model’s ability to know what it knows [34].

Moreover, RLHF for chat purposes substantially hurts the reasoning and test-taking performance of GPT-4. By forcing the model to avoid sensitive topics, it’s learning that it’s “okay to lie” in some cases and is encouraged to withhold information. The OpenAI authors claim that RLHF degrades exam performance in the absence of “active effort,” which is substantiated by Microsoft researchers who had access to an early version of the GPT-4 “base model pre RLHF” and claim from their subjective experience it was noticeably more capable before final release, and detail their qualitative experiments where GPT-4 demonstrated “sparks of AGI” [35].

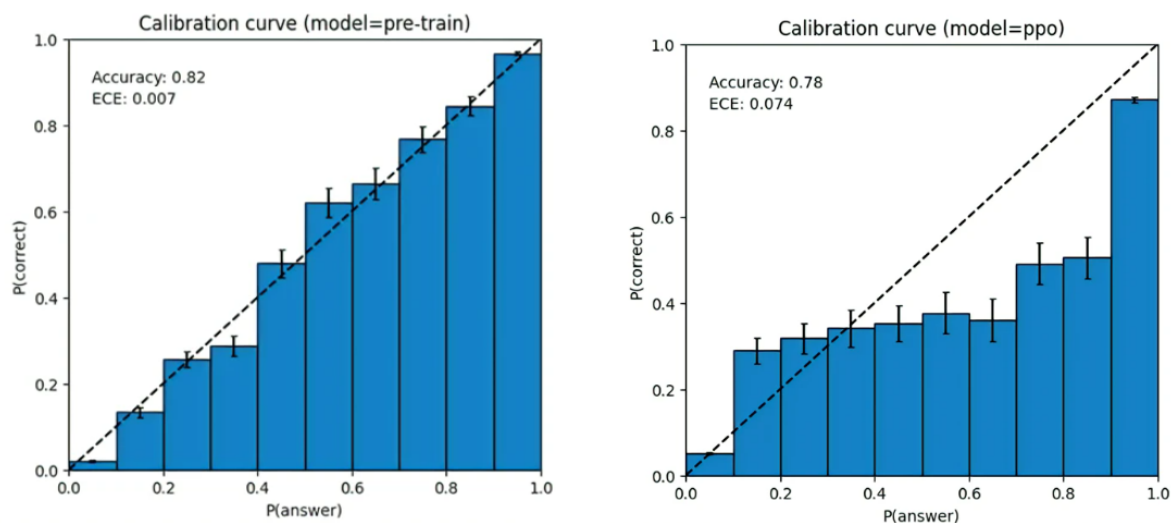


Figure 4.3 RLHF destroys GPT-4s ability to know what it knows. “Left: Calibration plot of the pre-trained GPT-4 model on an MMLU subset. The model’s confidence in its prediction closely matches the probability of being correct. The dotted diagonal line represents perfect calibration. Right: Calibration plot of post-trained PPO GPT-4 model on the same MMLU subset. Our current process hurts the calibration quite a bit.” Figure from [32].

Why RL, not Supervised Fine Tuning, is Required for Truthfulness without Hallucination. Why should we use Reinforcement Learning (RL) as a reward signal instead of FLAN-style [36] instruction tuning? It seems instruction tuning should be sufficient for the model to learn to follow commands and even generalize to compose commands and handle new ones never seen in training. So, why use a reward model and Proximal Policy Optimization (PPO) to provide a very similar signal? First let’s examine a few general strengths of RL, then the reason why Supervised Fine-Tuning (SFT) fundamentally encourages hallucination. **(1) Give the model negative feedback.** RL enables both positive and negative reward values, allowing practitioners to penalize bad outputs, unlocking a powerful learning signal missing in SFT. **(2) Let the model test its own hypothesis.** RL enables practitioners to sample multiple possible answers from a model and thereby allow the model to “guess and check” multiple “hypotheses” about how to answer the question. The model then gets feedback on the reward values for each answer. This feedback loop is much more powerful, akin to the difference between passive lectures and difficult problem sets for human students. In some sense, the model can “test its assumptions” about the world and fill in blind spots that it never encountered during pre-training or SFT.

Let's examine three key uses of language models:

1. **Grounded** response that only answers about provided context: "Summarize this text: {context}" or Factual Document QA from "{context}, question" pairs.
2. **Internal Knowledge** response: "What are common causes of the flu?"
3. **Creative** response: "Write a story about..."

Empirical evidence shows that situation (1), text-grounded tasks in the supervised learning paradigm, are very effective; these tasks have driven the breakout success story of modern deep learning. Situation (3) is less concerned with truthfulness, and more concerned with compositional reasoning and style transfer which have also seen great empirical success in many domains of deep learning. However, situation (2), factual QA from internal knowledge, has been widely criticized as "hallucination" which, broadly, are coherent but incorrect answers.

Think of an LLM as having knowledge about the world. It will perform horribly if you ask it about something it doesn't know and can be prone to hallucinating incorrect facts. Say we take this model and fine-tune *just the last few layers* with a new dataset of, say, movie title and summaries. Since knowledge of those movies does not exist in the "world model" stored in the bulk of the model parameters, the fine-tuning is teaching the model that 'despite not knowing about this movie, you should still talk about it authoritatively as if you knew about it.'

That, in John Schulman's eyes, is a major reason LLMs lie: "they're taught that they should respond to things they do not know" [37]. John Schulman, one of the leaders of RL and RLHF at OpenAI, spoke at Berkeley to review RLHF in GPT-4 and his favorite open questions. He made the claim that supervised fine-tuning a LLM with new facts *fundamentally teaches it to lie and hallucinate*. We're teaching it to ignore the world model it built during pre-training and just make things up. Schulman's surprising takeaway is that "unless you have a way of looking at what's in the model, you can't train a model to be truthful with behavior cloning (note: "behavior cloning" is RL parlance to mean SFT)" [37].

In contrast to SFT, RL penalizes lying and hallucination because in the long run it will get back scores for made up answers (which is likely to be wrong) and will be rewarded for answering correctly thus learning to use its internal knowledge or abstain. RL is different because we sample from the model itself and compare the generation to a ground truth answer. This fundamentally bypasses the problem that "we can't inspect what the model knows or doesn't know" because it was forced to generate the completion, and the RM scores the result.

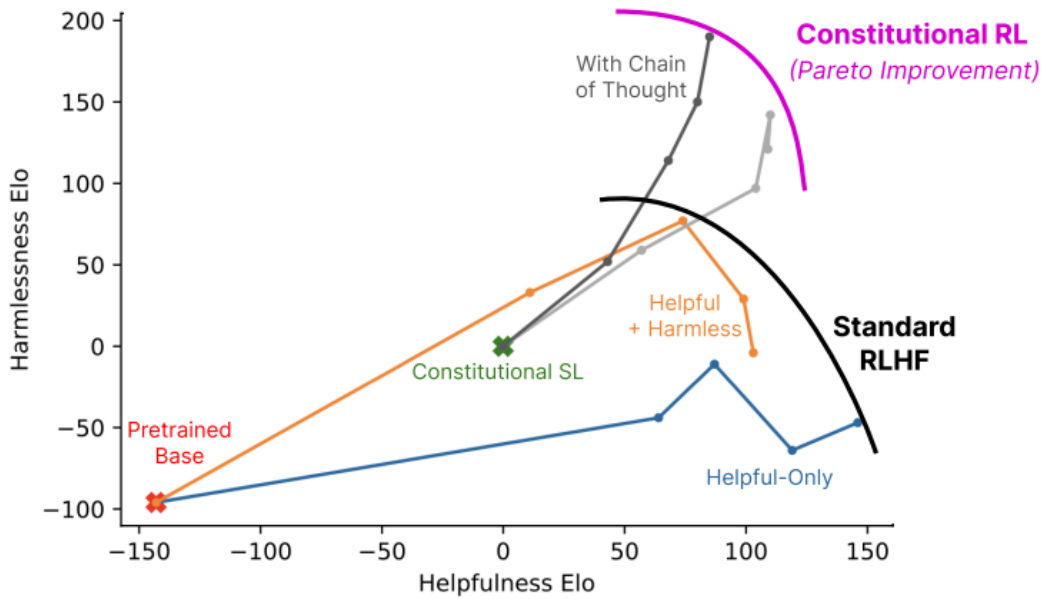


Figure 4.4 Demonstrates that RLAIIF, here called Constitutional RL, outperforms human crowd workers at improving the helpfulness and harmlessness of a 52B param RL model. Figure from [35].

RLAIIF: RL With AI Feedback. To avoid the dramatic expense and difficulty of collecting truly excellent human-written responses, we can use an auxiliary LLM to give feedback to the one being trained. This method is based on the observation that supervised training is highly effective for tasks such as determining semantic equivalence between two texts or identifying facts present in one text but absent in another. RLAIIF also involves decomposing tasks into generating question-answer pairs from a given text and subsequently determining if answers to those questions can be found in another text. By utilizing AI feedback from these tasks, RLAIIF aims to enhance the generalization capabilities of RL models, making them more adept at handling a wide range of text-grounded tasks, as shown in figure 4.4. This approach has the potential to contribute significantly to the ongoing advancements in reinforcement learning research by tightening the feedback loop for researchers to innovate without humans in the loop.

RLAIIF also enables multi-loss self-supervised training. Moving one step beyond ‘next token prediction, one can easily imagine adding a loss penalty for factualness, fluency, and the following a set of “constitutional” guidelines, like helpfulness and harmlessness [38], as shown in figure 4.4. Used in a Chain of Thought prompt sequence [39] the authors show that their LLMs are capable of iterative refinement against a series of separate evaluations against an arbitrary number of “constitutional value statements” posed by evaluators. In this study, RLAIIF

outperformed the human alternative, RLHF, and poses an extremely promising future direction of training LLMs with a consortium of expert teachers that critique the outputs of the base LLM. Finally, RLAIIF provides a direct research path to the development of an abundance of domain-specific Reward Models that can tune any given LLM towards “harmless and helpful” or “only output valid JSON” or “only answer when certain” or “only answer in poetry.” However, to reach the full potential of these LLMs, users want direct manipulation of more than just plaintext. But how can other modalities best augment transformers?

CHAPTER 5: MULTIMODAL TRANSFORMERS

Multimodal transformers aim to reason over multiple modalities of information which requires that all modalities are projected into the same latent embedding space of the model, meaning that a photo of a cat and a description of that photo should both map to the same vector in the model’s latent space; they should have a similar cosine similarity. However, the best way to align distinct embedding spaces, like say from separate text and image encoders, is an open question and will be the focus of this section. The three broad categories studied are (1) cross attention, (2) a small MLP projection layer, also called an “adaptor” or “mapping” layer, and (3) (contrastive) joint embedding methods. In this context, embeddings are typically the last hidden states from a transformer encoder, or the hidden states from the bottleneck layer of an autoencoder.

5.1 CROSS ATTENTION

Cross attention is a flexible and efficient method of combining arbitrary embeddings into a single transformer. One design pattern is to use a single autoregressive, “causal” transformer decoder to generate text, and multiple transformer encoders, often one encoder per modality, that generate embeddings, as in [10], [40]–[42]. This design pattern is best exemplified by Prismer and Vid2Seq, shown in figure 5.1. Cross attention, which is typically used to integration information between any encoder-decoder transformer, is extended in the multi-modal setting to

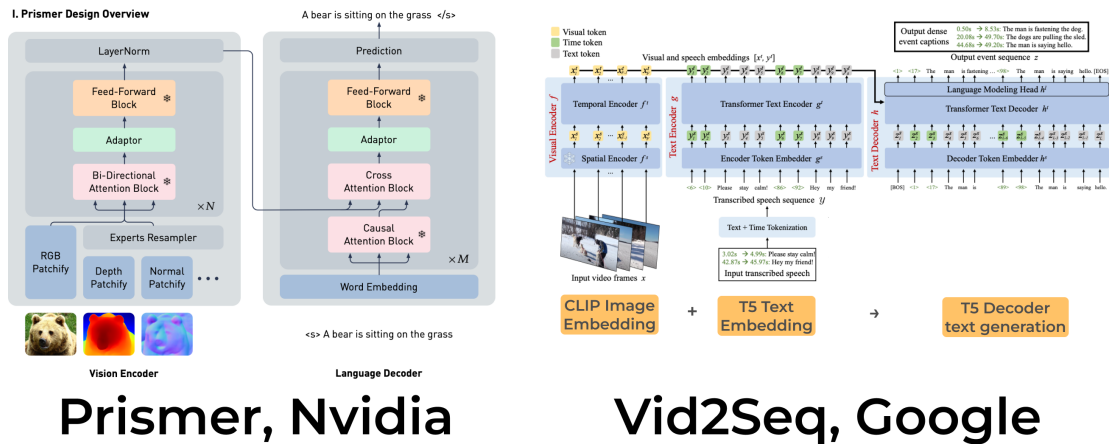





Figure 5.1 Cross Attention between Text & Image embeddings. Prisma and Vid2Seq are representative multimodal transformers using cross attention to incorporate information from multiple encoders into a single generative decoder. Figure adapted from [7], [39].

integrate information from each and every encoder into a single decoder. The decoder learns Queries, and the encoders learn the Keys and Values of the attention mechanism. The outputs from the encoders are typically flattened and concatenated into a single sequence and then passed through another standard transformer encoder. This pattern is used in Flamingo [41] and Prismer [40], where the authors name this process of collecting and compressing multiple encoders an “Expert Resampler.” The Expert Resampler serves two purposes (1) to further compress the multimodal features, hopefully reducing redundant information and creating a compact non-linear synthesis of the modalities and (2) to project possible variable length embeddings from each upstream modality into a single consistent-length embedding that can be queried, via cross attention Queries, by the downstream text decoder.

Image-conditioned QA. The word “conditioned” implies we don’t have direct control over how the visual information is used, since the attention weights learn to query information from the visual representation when it reduces loss. In some sense, the photo is always “attached globally” like an email attachment, instead of being an “in line” photo in a multi-modal sentence, e.g. “Does this  look like a  or 

5.2 SMALL MLP MAPPING PROJECTION

This fabulously simple method uses as little as a single layer to project the embedding from one model into the embedding space of another model. This project layer often maps the output embedding from, say, ResNet or ViT into the embedding dimension of a transformer. This mapping is often from a 768 or 1024 image embedding to 1024 or 2048 token embedding dimension in the transformer. This approach enables “flexible multimodal sentences” where images, or even robot states, can be flexibly placed in any part of the input sequence, as shown in figure 5.2. One simply “projects” the image embedding into the language-model’s embedding space, which makes a photo of a cat and a description of one exactly identical in the model’s latent space. MLP projection layers in the context of the state-of-the-art CLIP and Tip-Adapter models employ a learned affine transformation for mapping embeddings across different dimensions, and these MLP layers are always shallow, often just a single layer, or at most 3 small layers. The authors of Prismer and Flamingo suggest that shallow adapters are best to avoid ‘ruining’ the embedding being produced by your upstream model. In my proposed

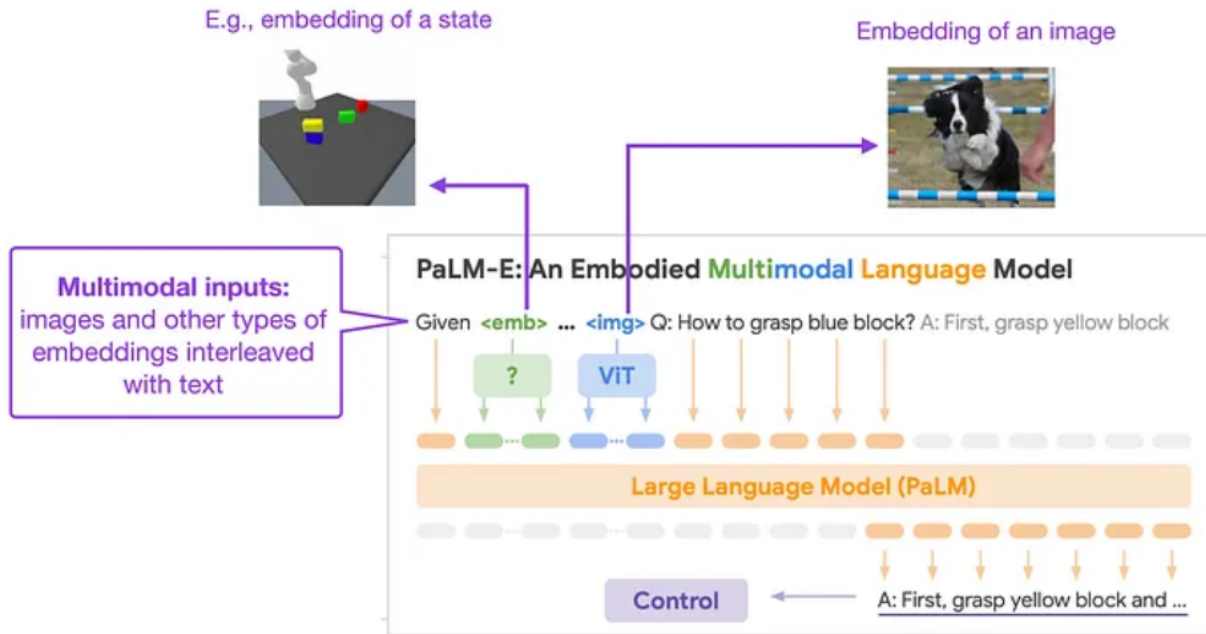


Figure 5.2 Multimodal sentences in Palm-E allow the flexible placement of images and robot movements anywhere in the sentence, making it easy to use images as context, i.e. “given this image of a kitchen, where should we move to access the refrigerator” or the object of a question like I want to move to the refrigerator, is <this picture> right? Figure modified from [37].

architecture, I doggedly avoided the use of adapter layers in order to protect the quality of the embeddings produced by the expert SOTA model in each modality. However, this was inconvenient because it limited my selection of experts to those that had matching hidden dimension (1024) used in the backbone T5 transformer. This illustrates why adapters are wonderfully useful because they’re quick to train, can facilitate any two or more models to be used together, are conceptually simple, and enable maximum flexibility in multimodal sentences.

5.3 JOINT EMBEDDING AND CONTRASTIVE LEARNING

Joint Embeddings take in a single input, for example an image, and feeds it to two or more models. The first model receives the unmodified image, and the second model receives distorted version. Finally, the loss function incentivizes the two models to output the same embedding despite seeing slightly different images. This forces the model to become invariant to the distortions, often called data augmentation, and in the processes learns a more robust representation of each concept since many different “distorted views” of the object all return the

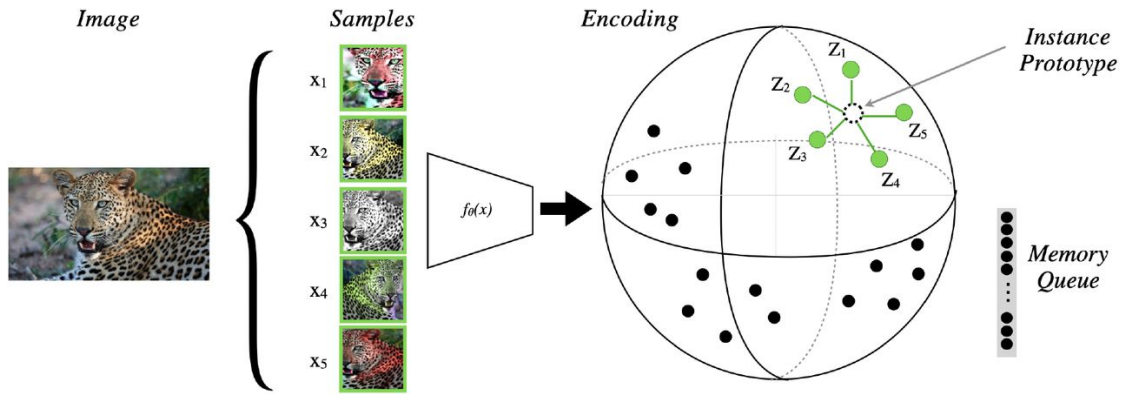


Figure 5.3 Contrastive learning is typically L2 normalized, meaning the norm (and sum) of the vector is 1, so embedding vectors are projected onto a unit sphere in high dimensional space, enabling cosine similarity to be an effective measure of conceptual relevance. Figure from [40].

same classification. This is visualized in figure 5.3 where we see five sample images that all look quite similar, they're all classes of big wild cats, and we see that their embeddings all reside in a tight cluster in the embedding space.

The downside of this method is that it requires an abundance of domain-specific data augmentations to learn robust representations of concepts. Prior works like DETR [43] have been criticized [44] for employing complex hand-crafted data augmentations that start to feel like the authors are cheating on benchmarks by hand-crafting features, much like feature engineering in tabular data learning regimes. Another concern is preventing trivial solutions, such when both

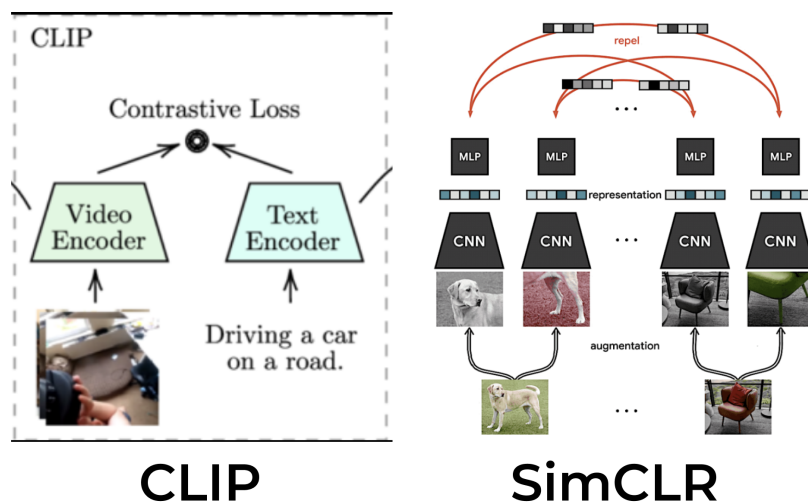


Figure 5.4 Embedding alignment via contrastive learning illustrated via representative architectures CLIP and SimCLR. Figure adapted from [44]–[46].

models always output all 0s (or any constant output) – this is a perfectly aligned representation but is not helpful [45]. Therefore, the energy function that calculates the similarity between the two representations must also have a penalty to prevent constant output solutions.

Some of the most successful methods are multimodal, where contrastive learning is used to “align the embedding space” of two separate pre-trained models. This application is exemplified by CLIP and SimCLR shown in figure 5.4, and more recently in Merlot Reserve [1]. As a direct alternative to using a small MLP projection layer this method allows large pre-trained models to be used together, but without the advantage of constructing multimodal sentences, instead they simply enable “multimodal search” since each modality is projected into the same space, we can easily use k-nearest neighbor search or cosine similarity to find corresponding data samples in other modalities. Yet, these self-supervised methods are most powerful when combined into a multi-modal transformer as we’ll see in the next chapter.

CHAPTER 6: VIDEO PRE-TRAINED TRANSFORMER: MULTIMODAL MIXTURE OF PRE-TRAINED EXPERTS

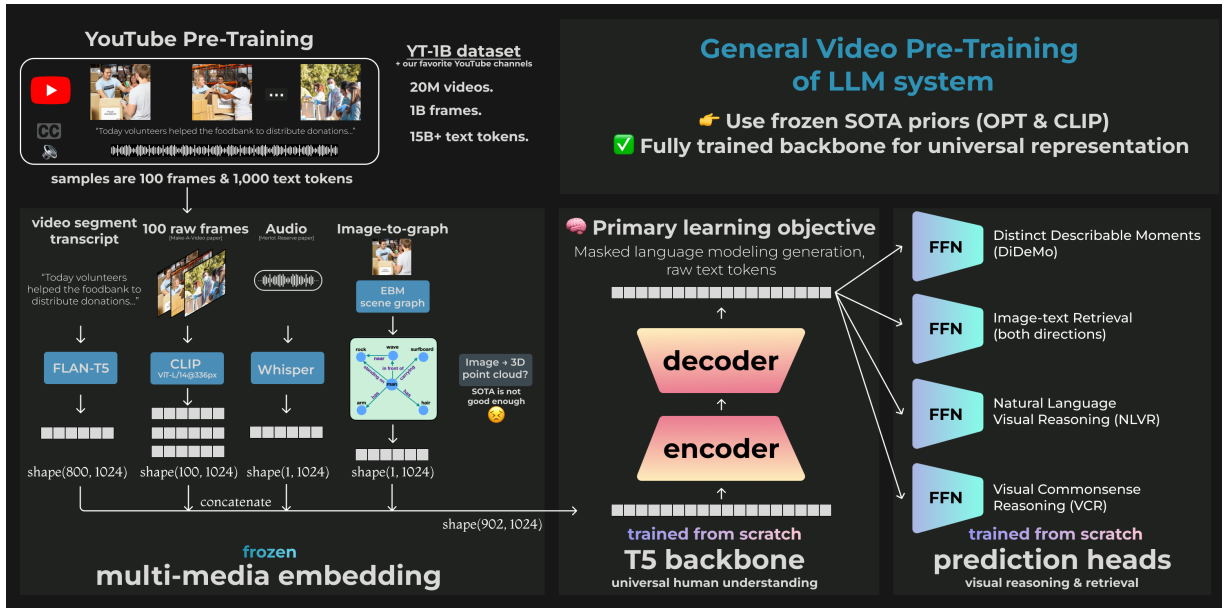


Figure 6.1 VPT learns multimodal representations of videos from four sources: image-sequences, raw audio, Whisper-generated text captions, and OpenPSG scene graphs). The backbone is trained to predict the words spoken in a video given encodings of the frames, audio, and scene graph, as well as an encoding of the directly preceding text. Figure from my own paper [41].

As the culmination of my graduate studies, I propose my own model architecture Video Pre-trained Transformer (VPT): A Multimodal Mixture of Pre-trained Experts [46], and the code can be found on GitHub [47]. My design as detailed in Figure 6.1 follows the “Embedding → Trunk → Head” pattern I first noticed succeeding in DINO [48] and Alphafold2 [49]–[51]. Now in mid-2023, the same pattern is successful in PaLM-E [5] and Vid2Seq [10] from Google, Prismer [40] from Nvidia, and the Autopilot system at Tesla as shown at AI day 2021 [52] and 2022 [53]. My design attempts to make the best possible use of existing pre-trained models by leveraging their information rich embeddings, and using a transformer encoder-decoder, namely Flan-T5, to combine multiple pre-trained models together to understand videos. GPT or T5, using text only, can predict words in video transcripts quite well, so in this work the intuition is to add complementary information, specifically video frames, audio and, as a novel contribution, structured information in the form of semantic segmentation scene graphs. I choose these models

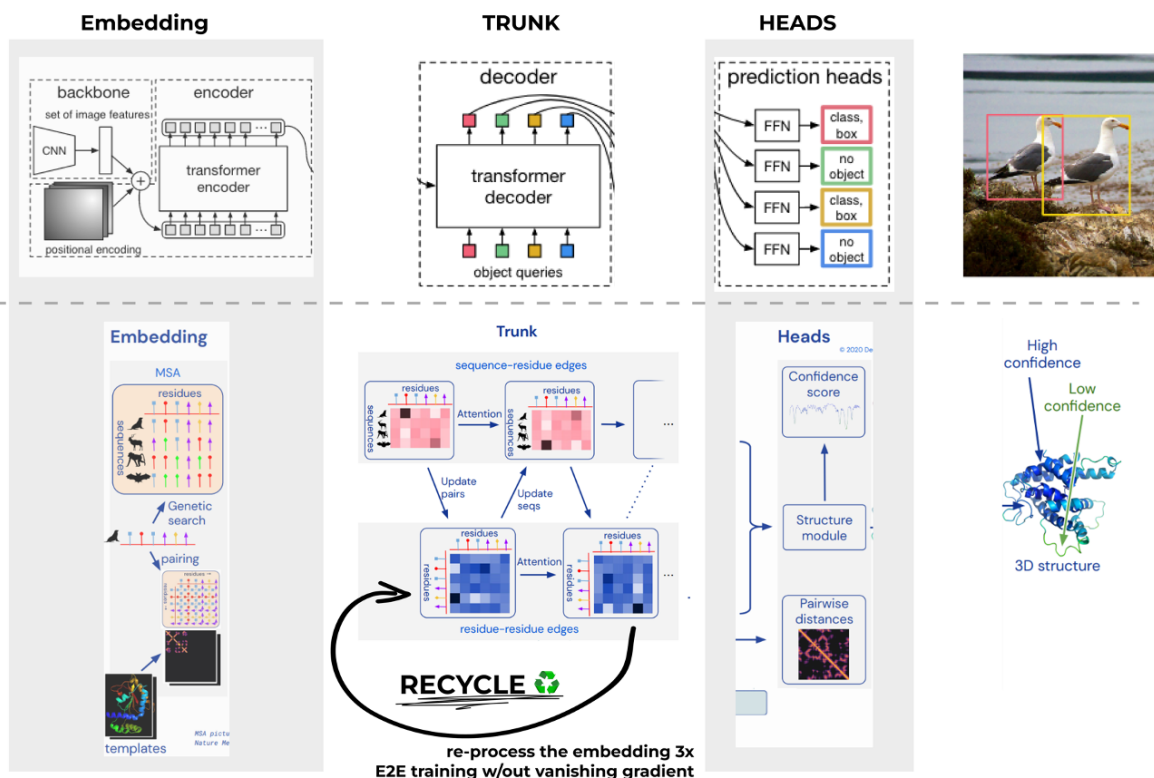


Figure 6.2 An annotated visualization of the “Embedding → Trunk → Head” design paradigm. (Top) DETR from FAIR [49] and (bottom) AlphaFold 2 from Deepmind [45] are two groundbreaking examples of this pattern in large scale transformers. Figure adapted from my own paper [41].

because each goes beyond ‘SOTA performance’ and exhibit truly world class performance and ‘generalized’ knowledge that will extend to the extremely diverse world of YouTube videos.

My architecture is a multi-modal video to text LLM that combines both encoder-decoder and autoregressive transformers and trained on visually interesting YouTube data that performs visual question answering tasks.

VPT’s primary contributions are:

1. A novel multimodal transformer architecture building on the “Embedding → Trunk → Head” design pattern, visualized in figure 6.2, to leverage the immense capabilities of existing open-source models while minimizing necessary compute during fine-tuning.
2. A novel use of Scene Graphs, in addition to language + vision + audio, as structured information into for multimodal transformers.
3. An uncommon training corpus of the recently introduced YT-Temporal-1B dataset, augmented with 25k additional YouTube videos I hand selected as high quality.

4. A suite of experiments exploring the pre-train vs finetune tradeoff.

6.1 CONTRIBUTIONS

VPT is a learned mixture of pre-trained expert models, and here I discuss each expert and future ones that were impossible for this work but are now starting to become a reality. **Audio** captures supplementary information like emotion in language, the sound effects of the world like “applause” or “car sounds” or “wild animal sounds” or the semantic content of music to aid the model in the standard autoregressive next word prediction objective. I concluded Google AudioSet [54] is the best dataset and benchmark for selecting an audio model that focused on maximally diverse range of “sound effects” instead of human speech. I used the corresponding PapersWithCode leaderboard to select the best performing and most generalizable audio classification and tagging model [55] and decided on Pre-trained Audio Neural Networks (PANNs) [56], [57].

Video content is clearly helpful to next word prediction by showing the content of the world, in particular capturing thematic information that may not be explicitly spoken by the hosts of the video since it's already shown on screen. The obvious choice at the time was CLIP for its unprecedented generalization performance. I elected for the largest and best performing version `openai/clip-vit-large-patch14-336` on Hugging Face, which conveniently matched the resolution of the 360p (360x640) video downloads. More recently, OpenCLIP models [58] trained on the new LION-5B dataset [59] modestly outperform the original CLIP models especially in multilingual and non-English tasks, and would be my recommendation today.

Finally, **semantic segmentation with scene graph generation** models show the relation between who and what is in the frame, including if people are holding objects or sitting/standing near other people and the content or setting of the background scene. Soon it will be possible to reconstruct the content of a standard YouTube video in 3D space using the video alone, enabling even greater semantic understanding of the content, but this capability was out of reach during the majority of this project. However, the recent work and vibrant community behind Neural Radiance Fields (NeRFs) [60], [61], which are functionally a form of photo-realistic 3D models made with ML, are close to bringing this capability within reach [62]–[64]. To my knowledge, VPT is the first work to use scene-graph information in multimodal transformers and would have been the first to include 3D information as well if those models would have been ready in time.

Smarter video sampling. All other prior work studied uses uniform sampling, typically at keeping one frame per second from the video [10], or by splitting the any length video into a fixed number of images [65]–[68]. The intuition is to apply more compute to more intelligently select the interesting frames worth sampling. The method first detects “cuts” in the video edit when the video cuts to new camera angles and B-roll where new information is likely present. At this cut it samples two frames: one frame shortly after the cut and one frame at the center timestamp between cuts. In this way, I inject prior intuition that cuts themselves, which are created by a skilled editor, carry significant information. This strategy of sampling right after a cut, and a 2nd “backup” point in the middle, biases the model towards YouTube content to disproportionately sample only salient moments from videos. If there a few cuts in a video, this is a red flag, and the footage was often was not a suitable video and was discarded. In the remaining cases with few cuts, the sampling defaults to a baseline at one FPS. Future work should further investigate how to extract the most salient moments from videos, such as by throwing out frames with no new semantic value or by introducing placeholder tokens to the vocabulary that represent “no change in video” to stand-in for uneventful video frames. Furthermore, one could exploit the additional metadata available on YouTube by using timestamps mentioned in the comments as interesting moments, by heavily sampling the most popular segments of videos, by leveraging ‘chapter’ information, view count, or the semantic content of YouTube recommendations, just to name a few.

Pre-trained vs Random-Init Backbone. A key open research question is whether the T5 “backbone” network be initialized with the pre-trained T5 weights or randomly initialized weights? Starting from the pre-trained weights, from intuition, may be helpful because that model is one extremely successful variant of the thousands or millions of T5 training runs that have occurred; akin to the lottery ticket hypothesis [69] the one network that survives all the hyperparameter tuning may have some unusually productive circuitry for learning, or even learning to learn. On the other hand, starting from pre-trained weights that are tuned for a totally different task may be harmful because it could limit how much the model is able to adjust to a new domain; perhaps it contains 0-weight connections that prevent circuits from ever becoming active again, thus limiting the pre-trained models’ potential to learn new, foreign, concepts. Yet, finally, the same dilemma exists for random-init weights: it’s a ‘green field’ to learn new things, but we could get unlucky with bad random initializations or waste training data and compute re-

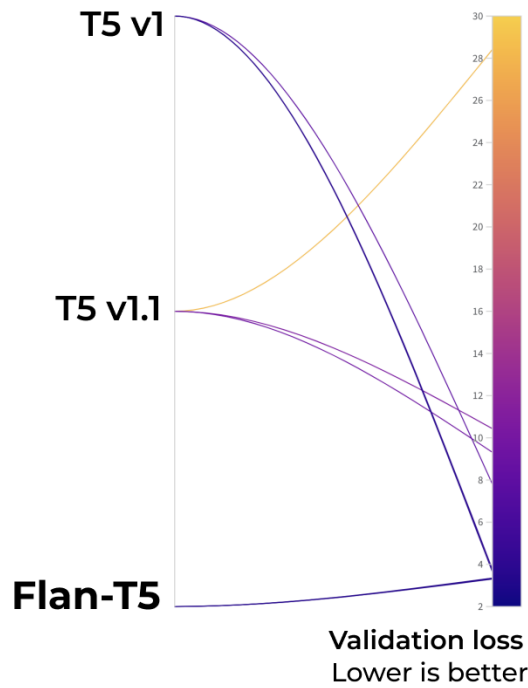


Figure 6.3 A hyperparameter sweep investigating if pre-trained or random-init models are preferable for only minimally related downstream tasks. I compare `google/flan-t5-large` vs `google/t5-v1_1-large` vs `t5-large`. In all cases, the Flan version of the model performed the best, leading to the lowest validation cross entropy loss. Unexpectedly, T5 v1.1 performed consistently worse than v1, i.e., `t5-large`, demonstrating unpredictable performance.

learning simple patterns or meta-learning capabilities that may already exist in the pre-trained weights.

I systematically investigate this tradeoff using a hyperparameter sweep of different model initializations, as shown in Figure 6.3. I confirmed prior work [70] that evaluated the fine-tuning performance of a wide range of open source LLMs and concluded that FLAN-T5 performed the best. There’s something special about that model that lends itself to exceptional fine-tuning performance, perhaps due to the fact that it was trained with class-leading 473 open-source instruction-following NLP datasets. The dramatic difference in my measured performance between Flan-T5 and other versions of T5 further suggests that certain models may simply ‘win the lottery’ by learning exceptional ‘learning to learn’ meta-structures that facilitate generalization.

6.2 RESULTS

I report experimental results from Visual Question Answering (VQAv2) benchmark in Table 6.4 and examine qualitative results in Figure 6.5. Further ablations report the significance of YouTube pre-training and the contribution of Scene Graphs as a pre-trained expert. For VQA, the model is pre-trained on YouTube data and fine-tune using the training and validations sets from VQA. Finally, evaluations are made using the standard, completely held out, VQA test set. First, I compare the model to a naïve baseline of always guessing the most common classes, which are by far ‘Yes’ and ‘No.’ I find that VPT outperforms this baseline by a 7% increase in accuracy (0.2389 baseline vs 0.3098 ours). Furthermore, I find that YouTube pre-training positively contributes to downstream performance on the benchmark, increasing the model’s score by 7%. Finally, I note that including scene graph information only hurt the performance of the model and speculate that this is likely due to a bug during preprocessing that mismatched the scene graph results with the rest of the training data, therefore making the scene graph information maliciously incorrect because it described unrelated video footage. Nevertheless, I’m encouraged by the significantly better than random performance even with the existence of bugs in the dataset. Follow-up evaluations will target Television Question Answering (TVQA) [71], [72], which is a remarkable dataset of American sitcom TV shows like The Big Bang Theory and Friends with roughly one-minute sections of videos paired with detailed questions asking about what characters were doing/holding, asking why they were upset, and occasionally asking about longer-term motivations of characters in the show. Additionally, Distinct Describable Moments (DiDeMo) [73] offers significant potential for precise retrieval tasks, for instance, what is the exact timestamp that the girl jumps onto the horse? DiDeMo also includes difficult counting questions like “how many oranges are on the table?” where some of the

	Accuracy	Iterations
Base T5 + VQA Finetuning	0.2390	20k
Base T5 + YouTube Pre-training + VQA Finetuning	0.2555	20k
Base T5 + YouTube Pre-training (only Yes/No)	0.2389	20k
Base T5 + YouTube Pre-training + VQA Finetuning (w/o scene graph)	0.3098	20k

Table 6.4 Ablation showing the impact of scene graph information and/or YouTube pre-training on VQA benchmark. Figure from [41].

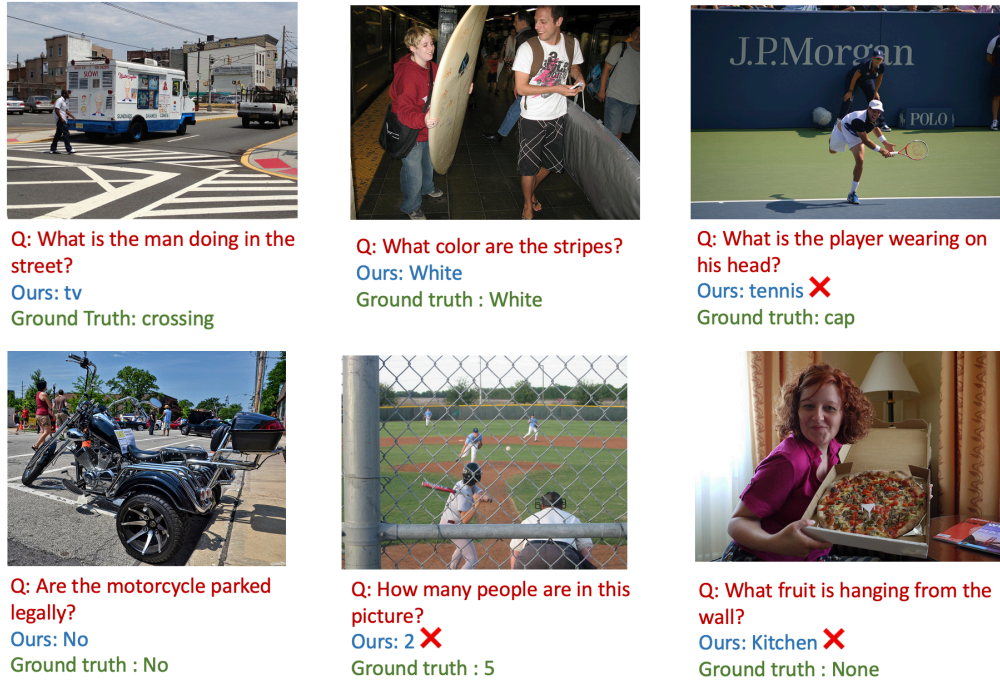


Figure 6.5 A qualitative evaluation of VPT against the VQA dataset shows the model outputs, even when wrong, are almost always semantically relevant but less sophisticated than the ground truth. For instance, the model recognized the two foreground baseball players but not the three additional ones that are barely visible in the background. Similarly, it fails the trick question of “Q: what fruit is hanging from the wall? A: None” probably because it associates the combination of the word “wall” and the picture of the pizza with kitchen.

information is occluded, requiring long-time-horizon context and memory. Both evaluations will be included in follow up work to VPT.

6.3 COMPUTE INFRASTRUCTURE

Performing inference with five different large language models over millions of videos, require extensive compute and data engineering. It requires four GPU years to run *Whisper-Medium*, the compute/performance sweet spot, on one million videos on a HPC node (4xA40 GPU, 64-core EPYC 7763 CPU and 256GB DDR4 RAM) which is equivalent to \$1.3 million on a functionally identical AWS p3 node. Attention was paid to achieving an average of 94.5% GP realization which is far above the industry average for LLM training workloads. CLIP, specifically *openai/clip-vit-large-patch14-336* on Hugging Face, although 15% faster

than Whisper-Medium, required a similar distributed inference setup. OpenPSG scene-graph generation achieved exceedingly high performance due to running on top of a highly optimized Detectron2 [74] implementation. Yet, I am unable to report on total compute requirements since many iterations of experimentation resulted in massive compute spend prior to final training runs. During finalized preprocessing I used up to 200 node parallelism on the Delta supercomputer, which worked well and put extreme demands on the data infrastructure.

6.4 DATA INFRASTRUCTURE

For pretraining data YT-Temporal-1B dataset introduced by Merlot Reserve [1] was recreated by redownloading the collection of 20M YouTube videos. The Merlot authors collected diverse videos via breadth first search through YouTube’s recommended videos with a wide variety of starting videos, then used MTurk crowd workers to train a classifier on specific metadata about a random selection of 2,000 videos with labels like news, video games, slide shows, and more. Then this data was used to train a classifier to filter out undesirable content, such as an over-supply of news media which they found to be problematic in their original Merlot paper. Finally, the dataset contains 20M videos that have a high density of words per minute and visually interesting and diverse content. Manual inspection of thousands of included videos confirms it’s the most diverse collection of videos that are both visually and verbally engaging I’ve seen in a machine learning dataset.

I further augment YT-Temporal-1B with handpicked YouTube videos wherein the spoken words have high relevance to the visuals presented in the video, as is often the case with voiceover, MythBusters-style content, and high-quality YouTube “explainer” videos. This process accounts for 23,582 additional videos in the dataset, and the original paper presents the full list of channels used. However, due to the immense size of 20M videos, I was only able to collect 2.8M videos, which took 6 weeks on a symmetrical one-gigabit internet connection. This corpus amounts to 330 thousand hours (38 years) of video content, requiring 76 TB of storage in 360p, as required for the 336x336 resolution CLIP-ViT encoder.

Database Requirements in Machine Learning. When pre-processing the video data python-native data formats fall woefully short in parallel and distributed workloads with varied datatypes and data shapes. For text data, Apache Parquet’s columnar format is world-class, and feels like it totally addresses that need, but the options are less modern and less useful for ragged,

meaning variable-size and variable-shape, collections of matrices. I tested all the following tools, here ordered from simplest to most full featured: JSON → np & pd → Parquet, PyArrow → Dask, Zarr, Xarray → NetCDF4, HDF5.

Between these tools Zarr and HDF5 certainly could solve the problem with high performance, but have difficult to use APIs, and do not support ragged arrays, meaning one would still have to create separate collections for each of his or her many modalities during preprocessing. Moreover, each data structure required significant manual tuning to match the chunk size of the array on disk to the filesystems' native chunk size. All these manual tuning steps introduce sources of error and performance limitations, which pushed me to keep searching for a better tool for my problem. Enter ML-first database solutions.

The essential criteria for selecting a suitable database for machine learning data preprocessing and training applications include:

- Single-node thread-safe and distributed read and write capabilities, allowing for mutable databases with in-place updates.
- Columnar $O(1)$ indexing to enable efficient sample retrieval and, ideally, vector-based similarity search.
- On-the-fly compression, incorporating both sample-based compression (e.g., JPEG) and chunk-based compression (e.g., LZ4 and ZSTD).
- Support for ragged arrays, as vectors may have significantly different shapes, rendering padding to a uniform specification impractical.
- Atomic (ACID) guarantees to ensure consistency in the database.
- “Nice to haves” include (git-like) data checkpointing, and redundancy, both of which may be provided by the filesystem.

Based on these requirements, I ultimately choose DeepLake, the Data Lake for Deep Learning [75], as the most capable, stable and user friendly data lake solution for deep learning applications. Deep Lake offers optimal compression and infinite flexibility due to its unique data layout on disk with both columns and chunking, which is further explained in Figure 6.6.

The most capable competitor to Deep Lake is the FiftyOne database whose founders have published over 250 papers during 30 years of academic experience [76]. Their platform is focuses on heavily on images and 3D point clouds for industry CV and ML workflows. Although appearing very similar to Deep Lake I found that it lacked the parallel and distributed processing

capabilities I required and lacked some of the streaming features, like streaming from S3 directly to GPU nodes, that make big data workflows much simpler in multi-cloud environments. Data streaming offers incredible flexibility and cost savings because I can keep my data wherever it's cheapest and spin up GPU nodes wherever they're cheapest at any given moment and quickly and reproducibly stream training data to the GPUs. Another competitor, LanceDB [77], is a young startup in this space, but focuses more on the ML inference rather than data preprocessing and ML training, with strong support for cost-efficient serverless vector retrieval and similarity search. Finally, DuckDB [78] is an established product in vector databases, supporting SQL's Online Analytical Processing (OLAP) of multi-dimensional arrays of data, which again works perfectly for ML inference, but like LanceDB, is not appropriate for concurrent writes or distributed data preprocessing. Finally, Ray Data [79] provides excellent parallel and distributed preprocessing features but does not provide its own persistent data structures, instead focusing on high performance shared in-memory objects, meaning a companion database is still required. I used Ray to implement all parallelism used in this work, paired with Deep Lake as the persistent datastore and high-performance streaming PyTorch Dataloader.

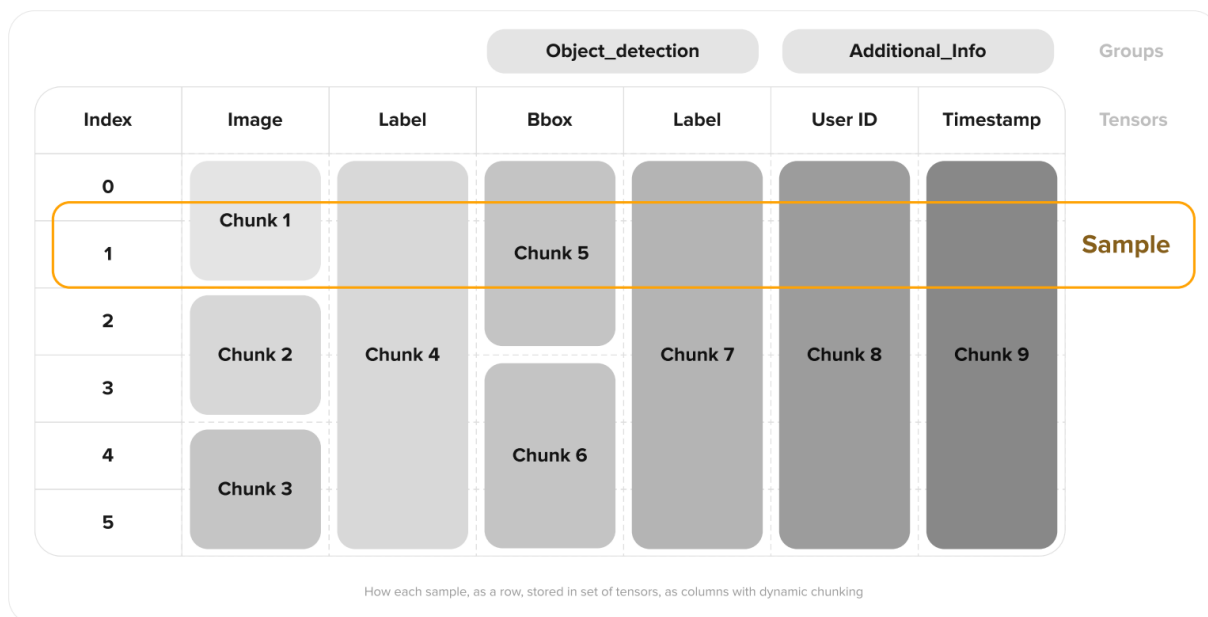


Figure 6.6 DeepLake data layout on disk. The use of variable chunk sizes for chunk-based compression fundamentally enables ragged arrays, or storing matrices of different shapes, efficiently on disk in a single database. Furthermore, it enables each modality, such as images and audio, to use domain-specific compression algorithms like JPEG, MP3 and LZ4, an essential feature for big data ML. Figure from [49].

Furthermore, I tested the most capable file system to pair with Deep Lake. While using Deep Lake on the NCSA Delta GPU supercomputer [80], the preprocessing was repeatedly bottlenecked by the central filesystem performance, specifically slow metadata retrieval that bottlenecked training for hours before the GPUs were saturated. Even worse, performance was unpredictable and experienced dramatic swings due to the multitenant nature of large file servers. Therefore, it's crucial to use efficient data access patterns, or database design decisions, that complement the filesystem's preferred layout on disk.

To this end, I interviewed JD Maloney, Sr. HPC Storage Engineer and designer of Delta's storage infrastructure, about the tradeoffs of different filesystems and how I may avoid my current bottlenecks. JD explained the three generations of filesystems, starting with traditional NFS, then the first offerings of the supercomputing world: Lustre and GPFS, which were natively parallel, scalable offerings, but optimized for HDDs and large file throughput. Today, third generation filesystems are optimized for, or in some cases exclusively support, flash/SSD storage media. These offerings are few but powerful: Vast and Weka. JD explained how supercomputer designers estimate filesystem needs and provision metadata servers to support the anticipated workloads. On Delta, the primary "hot" storage system uses 14 NVMe SSDs just for metadata lookup, achieving 80k to 90k file creates per second, which is 2x the metadata performance of Blue Waters. However, my workload was naïvely reading and writing millions of small files, leading to large bottlenecks in metadata operations. Despite recording up to 12 Gbps of network throughput, much more was expected the 100 Gbps networking. With this information about metadata server bottlenecks, I transformed my dataset from LOSF, "Lots of Small Files," to one Deep Lake database thereby eliminating the metadata bottleneck and allowing me to reach 75+ Gbps network throughput with fully saturated GPUs.

In follow-up work, I analyzed the optimal database and filesystem combination for multimodal ML workloads, focusing on DeepLake DB and Weka FS on AWS/Azure and a preprint should be posted on Arxiv by July 2023, and the benchmarking code is available on GitHub [81].

6.5 CONCLUSION

Compute is less critical than ever due to the abundance of high-quality open-source models that may outright solve your problem, or which you can PEFT to fit your problem using only

consumer-class GPUs. For the rare case where full-pre-training is necessary for entirely new domains or model architectures, industry adoption of AI has created the right incentives for industry to solve the problem of large model training with distributed GPUs via platforms like AWS SageMaker [82] and MosaicML [83].

On the other hand, data is more important than ever. The winds of the scaling laws have shifted back from prioritizing unimaginably large models to emphasizing unimaginably large and high-quality data. Open-source models are still a long way away from utilizing all the publicly available data on the internet, let alone all the private data. Pre-training on a curated collection of the best public datasets and fine-tuning on proprietary data will be an effective strategy. AI projects should pay close attention to data quality infrastructure.

Finally, using simulated environments for data generation holds promise, but is still grappling with the simulation-to-reality gap. If successful, it would transform data scarcity issues into a function of computational resources required to operate the simulation. Current research focuses on applications in robotic tasks, but with creativity simulations can work for many scientific applications as well. Nvidia Omniverse and Isaac Sim emerge as the two most promising avenues for machine learning, particularly reinforcement learning. Meanwhile, Unreal Engine offers easy accessibility to the open-source community. The success of simulation would turn data problems into compute problems, promising even greater scalability. Until then, data and compute are all we need.

REFERENCES

- [1] “MERLOT Reserve: Neural Script Knowledge through Vision and Language and Sound - <https://rowanzellers.com/merlotreserve/>.” <https://rowanzellers.com/merlotreserve/> (accessed Apr. 16, 2023).
- [2] S. J. Mielke *et al.*, “Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP.” arXiv, Dec. 20, 2021. doi: 10.48550/arXiv.2112.10508.
- [3] S. Wu *et al.*, “BloombergGPT: A Large Language Model for Finance.” arXiv, Mar. 30, 2023. doi: 10.48550/arXiv.2303.17564.
- [4] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, “A Time Series is Worth 64 Words: Long-term Forecasting with Transformers.” arXiv, Mar. 05, 2023. doi: 10.48550/arXiv.2211.14730.
- [5] A. Chowdhery *et al.*, “PaLM: Scaling Language Modeling with Pathways.” arXiv, Oct. 05, 2022. doi: 10.48550/arXiv.2204.02311.
- [6] N. Carlini *et al.*, “Extracting Training Data from Large Language Models.” arXiv, Jun. 15, 2021. doi: 10.48550/arXiv.2012.07805.
- [7] T. B. Brown *et al.*, “Language Models are Few-Shot Learners.” arXiv, Jul. 22, 2020. doi: 10.48550/arXiv.2005.14165.
- [8] “BioMedLM: a Domain-Specific Large Language Model for Biomedical Text.” <https://www.mosaicml.com/blog/introducing-pubmed-gpt> (accessed Apr. 24, 2023).
- [9] S. Reed *et al.*, “A Generalist Agent.” arXiv, Nov. 11, 2022. doi: 10.48550/arXiv.2205.06175.
- [10] A. Yang *et al.*, “Vid2Seq: Large-Scale Pretraining of a Visual Language Model for Dense Video Captioning.” arXiv, Mar. 21, 2023. doi: 10.48550/arXiv.2302.14115.
- [11] “Stanford CRFM - <https://crfm.stanford.edu/2022/12/15/pubmedgpt.html>.” <https://crfm.stanford.edu/2022/12/15/pubmedgpt.html> (accessed Apr. 24, 2023).
- [12] L. Gao *et al.*, “The Pile: An 800GB Dataset of Diverse Text for Language Modeling.” arXiv, Dec. 31, 2020. Accessed: Jun. 07, 2022. [Online]. Available: <http://arxiv.org/abs/2101.00027>
- [13] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff, “A Transformer-based Framework for Multivariate Time Series Representation Learning,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, in KDD ’21.

- New York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 2114–2124.
doi: 10.1145/3447548.3467401.
- [14] H. Zhou *et al.*, “Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting.” arXiv, Mar. 28, 2021. doi: 10.48550/arXiv.2012.07436.
 - [15] “🤗 PEFT.” Hugging Face, Apr. 15, 2023. Accessed: Apr. 15, 2023. [Online]. Available: <https://github.com/huggingface/peft>
 - [16] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, “ZeRO: Memory Optimizations Toward Training Trillion Parameter Models.” arXiv, May 13, 2020. doi: 10.48550/arXiv.1910.02054.
 - [17] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, “ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning.” arXiv, Apr. 15, 2021. doi: 10.48550/arXiv.2104.07857.
 - [18] R. Y. Aminabadi *et al.*, “DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale.” arXiv, Jun. 30, 2022. doi: 10.48550/arXiv.2207.00032.
 - [19] E. J. Hu *et al.*, “LoRA: Low-Rank Adaptation of Large Language Models.” arXiv, Oct. 16, 2021. doi: 10.48550/arXiv.2106.09685.
 - [20] Q. Zhang *et al.*, “Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning.” arXiv, Mar. 18, 2023. doi: 10.48550/arXiv.2303.10512.
 - [21] “StackLLaMA: A hands-on guide to train LLaMA with RLHF.” <https://huggingface.co/blog/stackllama> (accessed Apr. 25, 2023).
 - [22] “Fine-tuning 20B LLMs with RLHF on a 24GB consumer GPU.” <https://huggingface.co/blog/trl-peft> (accessed Apr. 25, 2023).
 - [23] “lvwerra/trl: Train transformer language models with reinforcement learning.” <https://github.com/lvwerra/trl> (accessed Apr. 16, 2023).
 - [24] “CarperAI/trlx: A repo for distributed training of language models with Reinforcement Learning via Human Feedback (RLHF).” <https://github.com/CarperAI/trlx> (accessed Apr. 16, 2023).
 - [25] X. L. Li and P. Liang, “Prefix-Tuning: Optimizing Continuous Prompts for Generation,” in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1:*

- Long Papers*), Online: Association for Computational Linguistics, Aug. 2021, pp. 4582–4597. doi: 10.18653/v1/2021.acl-long.353.
- [26] B. Lester, R. Al-Rfou, and N. Constant, “The Power of Scale for Parameter-Efficient Prompt Tuning.” arXiv, Sep. 02, 2021. doi: 10.48550/arXiv.2104.08691.
 - [27] X. Liu *et al.*, “GPT Understands, Too.” arXiv, Mar. 18, 2021. doi: 10.48550/arXiv.2103.10385.
 - [28] X. Liu *et al.*, “P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks.” arXiv, Mar. 20, 2022. doi: 10.48550/arXiv.2110.07602.
 - [29] R. Zhang *et al.*, “LLaMA-Adapter: Efficient Fine-tuning of Language Models with Zero-init Attention.” arXiv, Mar. 28, 2023. doi: 10.48550/arXiv.2303.16199.
 - [30] “ChatGPT’s on track to surpass 100 million monthly users faster than TikTok or Instagram: UBS,” *Yahoo Finance*, Feb. 02, 2023. <https://finance.yahoo.com/news/chatgpt-on-track-to-surpass-100-million-users-faster-than-tiktok-or-instagram-ubs-214423357.html> (accessed Apr. 22, 2023).
 - [31] “Ilya Chief Scientist OpenAI - Eye on AI Podcast Episode #116,” *Eye On AI*, Mar. 15, 2023. <https://www.eye-on.ai/podcast-archive> (accessed Apr. 22, 2023).
 - [32] *Sam Altman: OpenAI CEO on GPT-4, ChatGPT, and the Future of AI | Lex Fridman Podcast #367*, (Mar. 25, 2023). Accessed: Apr. 24, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=L_Guz73e6fw
 - [33] S. Kadavath *et al.*, “Language Models (Mostly) Know What They Know.” arXiv, Nov. 21, 2022. doi: 10.48550/arXiv.2207.05221.
 - [34] OpenAI, “GPT-4 Technical Report.” arXiv, Mar. 27, 2023. doi: 10.48550/arXiv.2303.08774.
 - [35] S. Bubeck *et al.*, “Sparks of Artificial General Intelligence: Early experiments with GPT-4.” arXiv, Apr. 13, 2023. doi: 10.48550/arXiv.2303.12712.
 - [36] H. W. Chung *et al.*, “Scaling Instruction-Finetuned Language Models.” arXiv, Dec. 06, 2022. doi: 10.48550/arXiv.2210.11416.
 - [37] *John Schulman - Reinforcement Learning from Human Feedback: Progress and Challenges*, (Apr. 19, 2023). Accessed: Apr. 22, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=hhiLw5Q_UFg

- [38] Y. Bai *et al.*, “Constitutional AI: Harmlessness from AI Feedback.” arXiv, Dec. 15, 2022. doi: 10.48550/arXiv.2212.08073.
- [39] J. Wei *et al.*, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,” *arXiv.org*, Jan. 28, 2022. <https://arxiv.org/abs/2201.11903v6> (accessed Apr. 25, 2023).
- [40] S. Liu, L. Fan, E. Johns, Z. Yu, C. Xiao, and A. Anandkumar, “Prismer: A Vision-Language Model with An Ensemble of Experts.” arXiv, Mar. 11, 2023. doi: 10.48550/arXiv.2303.02506.
- [41] J.-B. Alayrac *et al.*, “Flamingo: a Visual Language Model for Few-Shot Learning.” arXiv, Nov. 15, 2022. doi: 10.48550/arXiv.2204.14198.
- [42] J. Li, D. Li, S. Savarese, and S. Hoi, “BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models.” arXiv, Jan. 29, 2023. doi: 10.48550/arXiv.2301.12597.
- [43] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, “End-to-End Object Detection with Transformers,” *ArXiv200512872 Cs*, May 2020, Accessed: Nov. 27, 2021. [Online]. Available: <http://arxiv.org/abs/2005.12872>
- [44] *DETR: End-to-End Object Detection with Transformers (Paper Explained)*, (May 28, 2020). Accessed: Apr. 28, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=T35ba_VXkMY
- [45] “Joint Embedding Methods - Contrastive · Deep Learning - <https://atcold.github.io/NYU-DLSP21/en/week15/15-1/>” <https://atcold.github.io/NYU-DLSP21/en/week15/15-1/> (accessed Apr. 22, 2023).
- [46] K. Day, D. Christl, R. Salvi, and P. Sriram, “Video Pre-trained Transformer: A Multimodal Mixture of Pre-trained Experts.” arXiv, Mar. 24, 2023. doi: 10.48550/arXiv.2304.10505.
- [47] K. Day, “Video Pre-trained Transformer: A Multimodal Mixture of Pre-trained experts - Github.” Apr. 27, 2023. Accessed: Apr. 27, 2023. [Online]. Available: <https://github.com/KastanDay/video-pretrained-transformer>
- [48] M. Caron *et al.*, “Emerging Properties in Self-Supervised Vision Transformers,” *arXiv.org*, Apr. 29, 2021. <https://arxiv.org/abs/2104.14294v2> (accessed Apr. 25, 2023).

- [49] G. Ahdritz *et al.*, “OpenFold: Retraining AlphaFold2 yields new insights into its learning mechanisms and capacity for generalization,” *Bioinformatics*, preprint, Nov. 2022. doi: 10.1101/2022.11.20.517210.
- [50] J. Jumper *et al.*, “Highly accurate protein structure prediction with AlphaFold,” *Nature*, vol. 596, no. 7873, Art. no. 7873, Aug. 2021, doi: 10.1038/s41586-021-03819-2.
- [51] J. Skolnick, M. Gao, H. Zhou, and S. Singh, “AlphaFold 2: Why It Works and Its Implications for Understanding the Relationships of Protein Sequence, Structure, and Function,” *J. Chem. Inf. Model.*, vol. 61, no. 10, pp. 4827–4831, Oct. 2021, doi: 10.1021/acs.jcim.1c01114.
- [52] *Tesla AI Day 2021*, (Aug. 19, 2021). Accessed: Apr. 27, 2023. [Online Video]. Available: <https://www.youtube.com/watch?v=j0z4FweCy4M>
- [53] *Tesla AI Day 2022*, (Sep. 30, 2022). Accessed: Apr. 27, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=ODSJsviD_SU
- [54] J. F. Gemmeke *et al.*, “Audio Set: An ontology and human-labeled dataset for audio events,” in *Proc. IEEE ICASSP 2017*, New Orleans, LA, 2017.
- [55] “Papers with Code - AudioSet Dataset.” <https://paperswithcode.com/dataset/audioset> (accessed Apr. 27, 2023).
- [56] Q. Kong, Y. Cao, T. Iqbal, Y. Wang, W. Wang, and M. D. Plumbley, “PANNs: Large-Scale Pretrained Audio Neural Networks for Audio Pattern Recognition.” *arXiv*, Aug. 23, 2020. doi: 10.48550/arXiv.1912.10211.
- [57] qiuqiangkong, “PANNs: Large-Scale Pretrained Audio Neural Networks for Audio Pattern Recognition.” Apr. 26, 2023. Accessed: Apr. 27, 2023. [Online]. Available: https://github.com/qiuqiangkong/audioset_tagging_cnn
- [58] G. Ilharco *et al.*, “OpenCLIP.” Jul. 2021. doi: 10.5281/zenodo.5143773.
- [59] C. Schuhmann *et al.*, “LAION-5B: An open large-scale dataset for training next generation image-text models.” *arXiv*, Oct. 15, 2022. doi: 10.48550/arXiv.2210.08402.
- [60] “Getting Started with NVIDIA Instant NeRFs,” *NVIDIA Technical Blog*, May 12, 2022. <https://developer.nvidia.com/blog/getting-started-with-nvidia-instant-nerfs/> (accessed Apr. 27, 2023).

- [61] *Why THIS is the Future of Imagery (and Nobody Knows it Yet)*, (Nov. 20, 2022). Accessed: Apr. 27, 2023. [Online Video]. Available: <https://www.youtube.com/watch?v=YX5AoaWrowY>
- [62] K. Gao, Y. Gao, H. He, D. Lu, L. Xu, and J. Li, “NeRF: Neural Radiance Field in 3D Vision, A Comprehensive Review.” arXiv, Dec. 18, 2022. doi: 10.48550/arXiv.2210.00379.
- [63] D. Ramirez, S. Jayasuriya, and A. Spanias, “Towards Live 3D Reconstruction from Wearable Video: An Evaluation of V-SLAM, NeRF, and Videogrammetry Techniques.” arXiv, Nov. 21, 2022. doi: 10.48550/arXiv.2211.11836.
- [64] G. Te, X. Li, X. Li, J. Wang, W. Hu, and Y. Lu, “Neural Capture of Animatable 3D Human from Monocular Video.” arXiv, Aug. 18, 2022. doi: 10.48550/arXiv.2208.08728.
- [65] A. Jaegle, F. Gimeno, A. Brock, A. Zisserman, O. Vinyals, and J. Carreira, “Perceiver: General Perception with Iterative Attention.” arXiv, Jun. 22, 2021. doi: 10.48550/arXiv.2103.03206.
- [66] A. Jaegle *et al.*, “Perceiver IO: A General Architecture for Structured Inputs & Outputs.” arXiv, Mar. 15, 2022. doi: 10.48550/arXiv.2107.14795.
- [67] U. Singer *et al.*, “Make-A-Video: Text-to-Video Generation without Text-Video Data.” arXiv, Sep. 29, 2022. doi: 10.48550/arXiv.2209.14792.
- [68] Y. Ma, G. Xu, X. Sun, M. Yan, J. Zhang, and R. Ji, “X-CLIP: End-to-End Multi-grained Contrastive Learning for Video-Text Retrieval.” arXiv, Sep. 22, 2022. doi: 10.48550/arXiv.2207.07285.
- [69] J. Frankle and M. Carbin, “The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks.” arXiv, Mar. 04, 2019. doi: 10.48550/arXiv.1803.03635.
- [70] S. Longpre *et al.*, “The Flan Collection: Designing Data and Methods for Effective Instruction Tuning.” arXiv, Feb. 14, 2023. doi: 10.48550/arXiv.2301.13688.
- [71] J. Lei, L. Yu, M. Bansal, and T. L. Berg, “TVQA: Localized, Compositional Video Question Answering.” arXiv, May 07, 2019. doi: 10.48550/arXiv.1809.01696.
- [72] J. Lei, L. Yu, T. L. Berg, and M. Bansal, “TVQA+: Spatio-Temporal Grounding for Video Question Answering.” arXiv, May 11, 2020. doi: 10.48550/arXiv.1904.11574.
- [73] L. A. Hendricks, O. Wang, E. Shechtman, J. Sivic, T. Darrell, and B. Russell, “Localizing Moments in Video with Natural Language.” arXiv, Aug. 04, 2017. doi: 10.48550/arXiv.1708.01641.

- [74] “facebookresearch/detectron2.” Meta Research, Apr. 26, 2023. Accessed: Apr. 25, 2023. [Online]. Available: <https://github.com/facebookresearch/detectron2>
- [75] “Deep Lake: Data Lake for Deep Learning.” Activeloop, Feb. 26, 2023. Accessed: Feb. 26, 2023. [Online]. Available: <https://github.com/activeloopai/deeplake>
- [76] B. Moore, “Introducing FiftyOne: A Tool for Rapid Data & Model Experimentation,” *Voxel51*, Sep. 12, 2020. <https://medium.com/voxel51/introducing-fiftyone-a-tool-for-rapid-data-model-experimentation-73c85b8406e1> (accessed May 01, 2023).
- [77] “lancedb/lancedb.” Lance DB, May 01, 2023. Accessed: May 01, 2023. [Online]. Available: <https://github.com/lancedb/lancedb>
- [78] “An in-process SQL OLAP database management system,” *DuckDB*. <https://duckdb.org/> (accessed May 01, 2023).
- [79] “Ray Datasets: Distributed Data Preprocessing — Ray 2.4.0 - <https://docs.ray.io/en/latest/data/dataset.html>.” <https://docs.ray.io/en/latest/data/dataset.html> (accessed May 01, 2023).
- [80] “Delta,” *NCSA*. <https://www.ncsa.illinois.edu/research/project-highlights/delta/> (accessed Apr. 27, 2023).
- [81] K. Day, “fastest-filesystem-and-database-for-multimodal-ML.” Apr. 03, 2023. Accessed: Apr. 27, 2023. [Online]. Available: <https://github.com/KastanDay/fastest-filesystem-and-database-for-multimodal-ML>
- [82] “AWS SageMaker,” *Amazon Web Services, Inc.* <https://aws.amazon.com/sagemaker/> (accessed Apr. 27, 2023).
- [83] “MosaicML | Home.” <https://www.mosaicml.com/> (accessed Apr. 27, 2023).