LLM Finetuning and Light Agentic Systems

Hamid Bekamiri

Aalborg University Business School
Innovation, Knowledge, and Economic Development (IKE)
AI Denmark (AIDK)
hamidb@business.aau.dk
Feb 02, 2026

Today's lecture overview

1. Recap of Previous Lectures
2. Applications of LLMs
3. Optimizing GPTs

AI -LAB

About AI-LAB
How to access
System overview
Troubleshooting
Terms and Conditions
Fair Usage

GUIDES
Index
Video Onboarding Guide
Getting started
1. Prerequisites
2. Login
Understanding AI-LAB Access
Basic SSH Connection
Setting Up SSH Shortcuts (Recommended)
Troubleshooting
3. File Handling
4. Running Jobs
5. Getting Containers

Logging into AI-LAB

This guide will help you connect to AI-LAB using SSH (Secure Shell). SSH is a secure way to access remote computers over a network.

Understanding AI-LAB Access

AI-LAB has two front-end nodes that act as entry points:

- **ailab-fe01.srv.aau.dk**
- **ailab-fe02.srv.aau.dk**

You can connect to either node - they provide the same functionality.

Basic SSH Connection

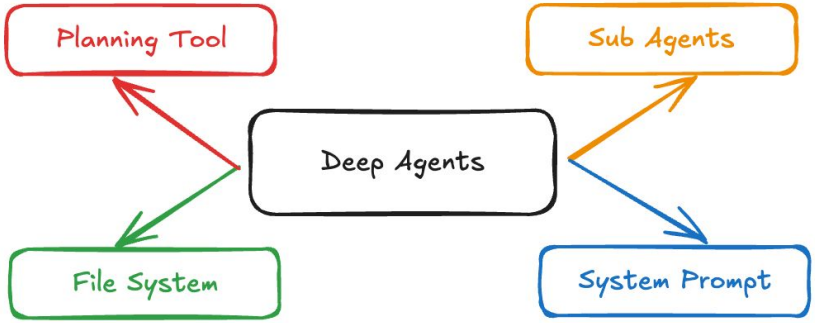
Step 1: Open Your Terminal

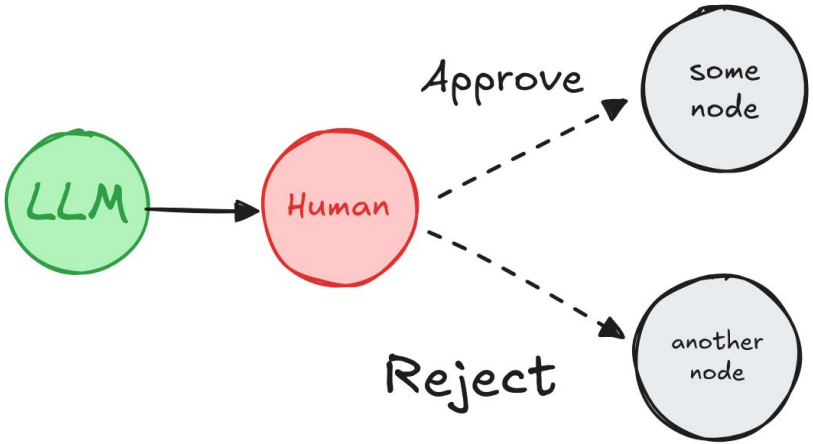
Windows | macOS | Linux

- Open your preferred terminal application

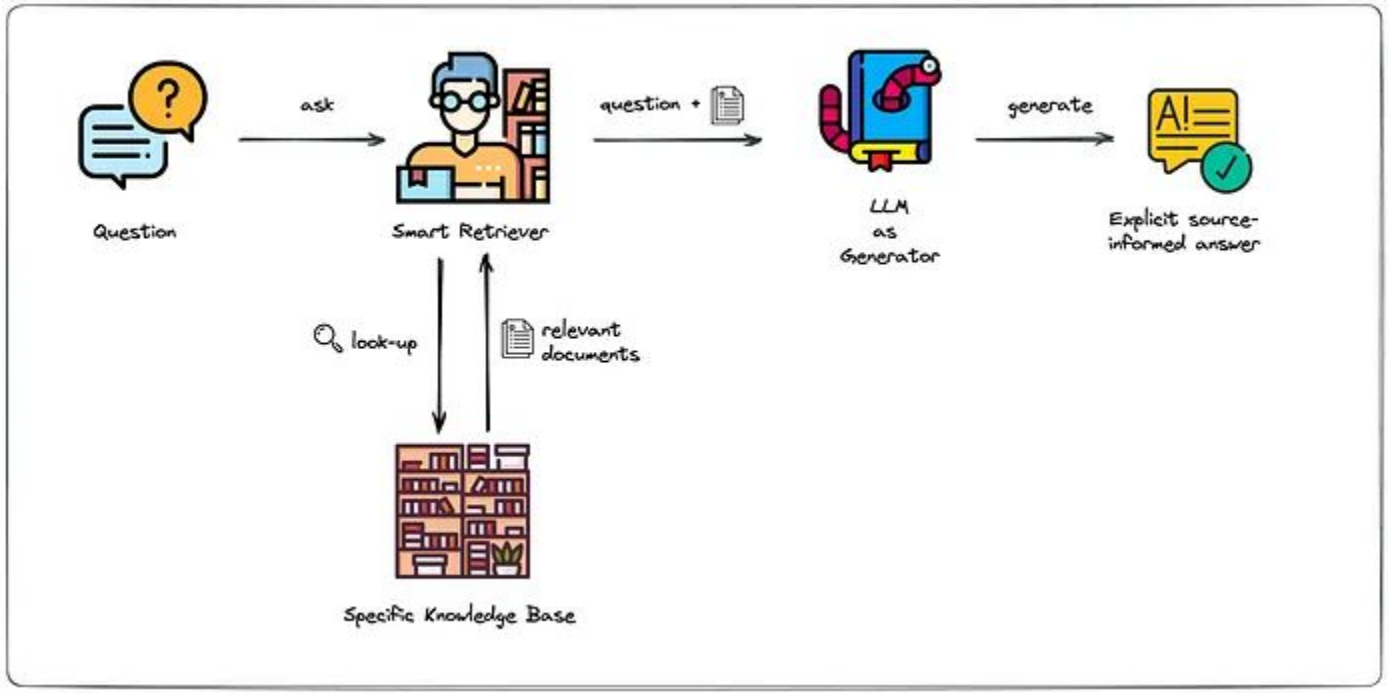
LangChain

- DeepAgent
- Human In The Loop

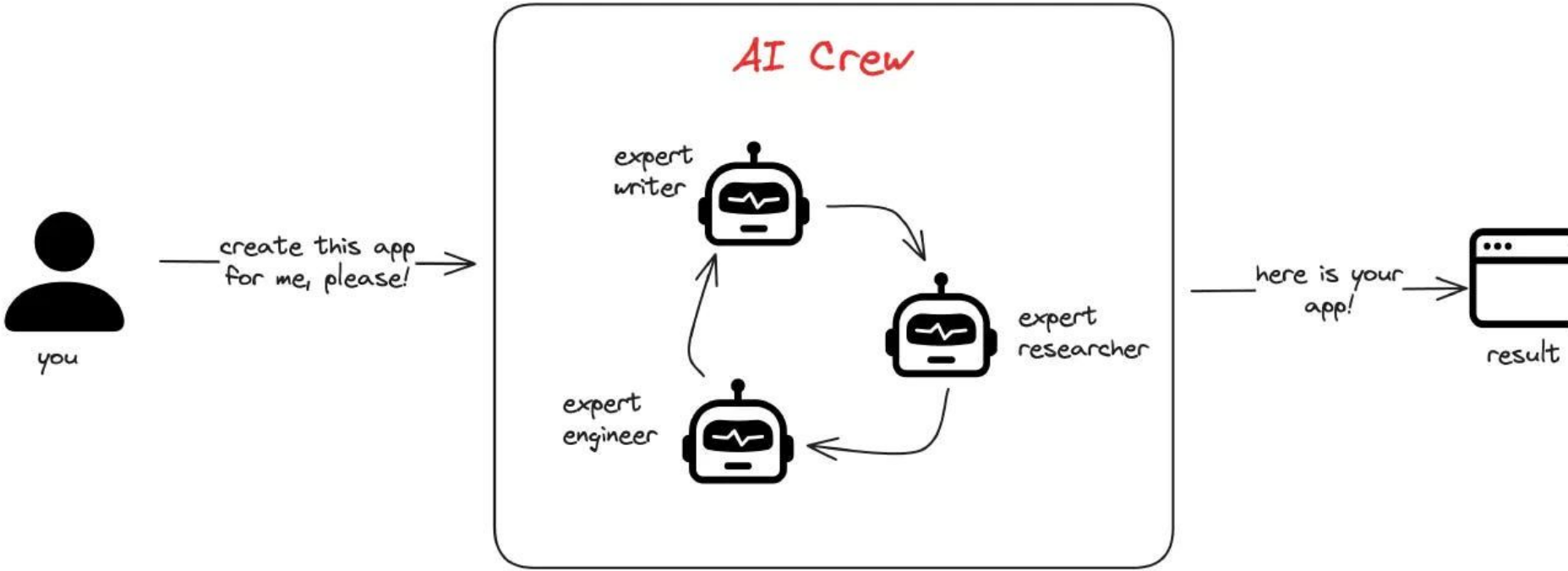




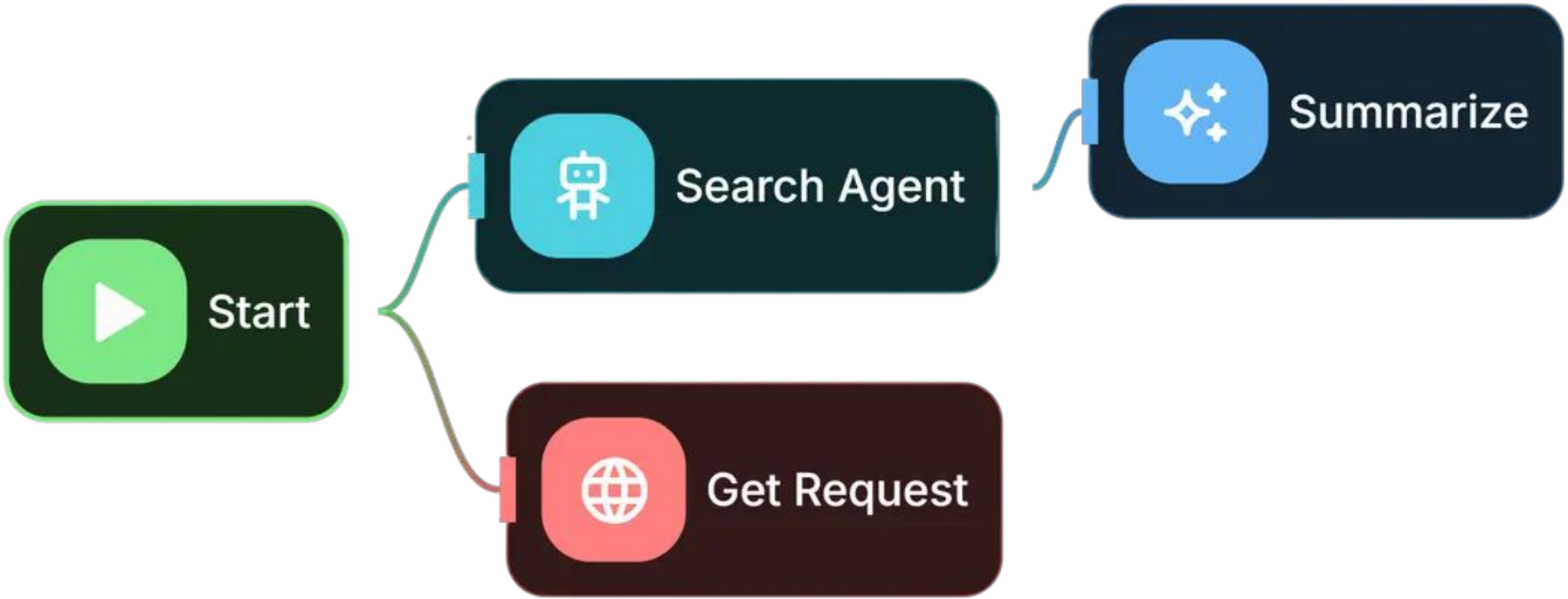
RAG + ChromaDB

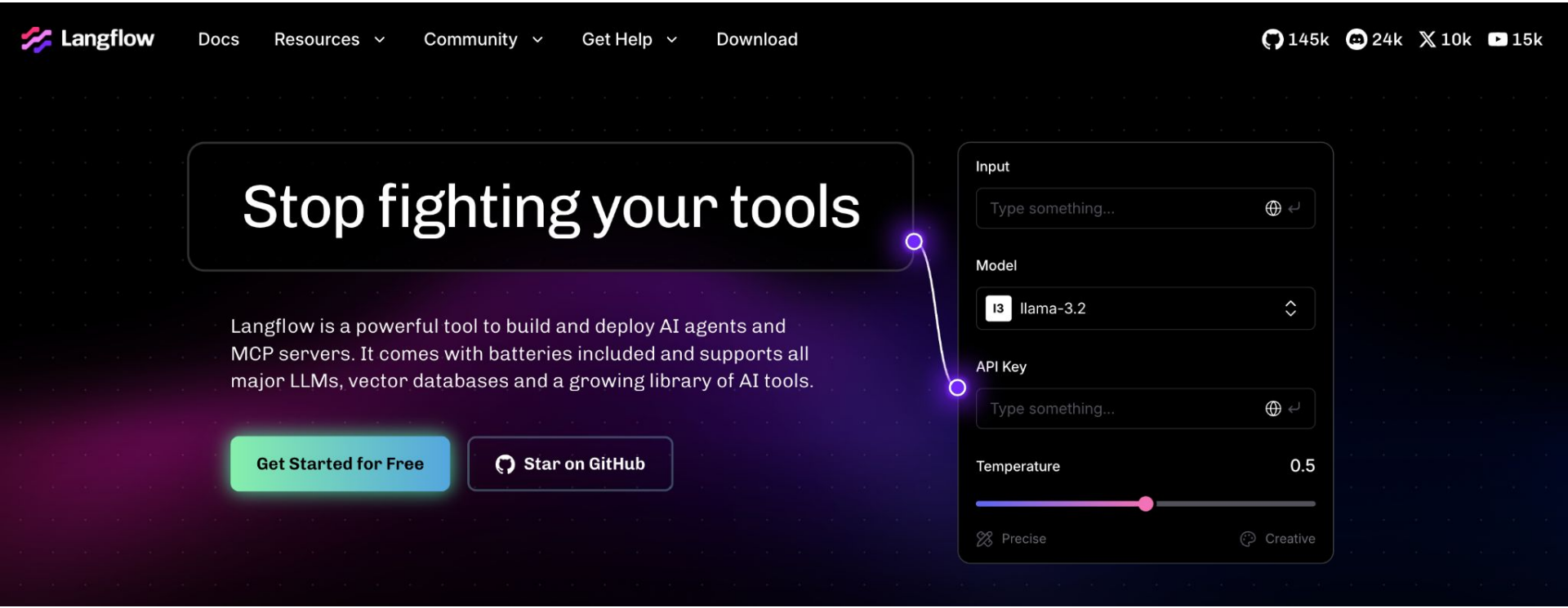


CrewAI


AI Crew
you
create this app for me, please!
expert writer
expert researcher
expert engineer
here is your app!
result

Flowise




Langflow
Docs
Resources
Community
Get Help
Download
145k
24k
10k
15k
Stop fighting your tools
Langflow is a powerful tool to build and deploy AI agents and MCP servers. It comes with batteries included and supports all major LLMs, vector databases and a growing library of AI tools.
Get Started for Free
Star on GitHub
Input
Type something...
Model
llama-3.2
API Key
Type something...
Temperature
0.5
Precise
Creative

What is Agentic AI?

August 6, 2025

Discover how agentic AI helps software development teams increase productivity and focus on more strategic tasks.

TABLE OF CONTENTS

What is Agentic AI?

How does agentic AI work?

Agentic AI vs. generative AI

Prompt Engineering: In context Learning

Google Research March 9, 2023

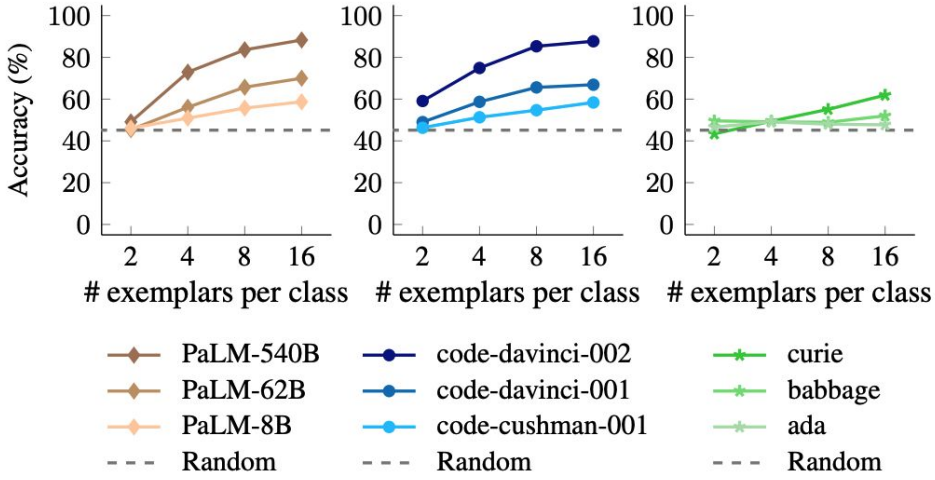
LARGER LANGUAGE MODELS DO IN-CONTEXT LEARNING DIFFERENTLY

Jerry Wei[1,2,*] **Jason Wei**[1] **Yi Tay**[1] **Dustin Tran**[1] **Albert Webson**[1,3,*]
Yifeng Lu[1] **Xinyun Chen**[1] **Hanxiao Liu**[1] **Da Huang**[1] **Denny Zhou**[1]
Tengyu Ma[1,2,†]

[1] Google Research, Brain Team [2] Stanford University [3] Brown University

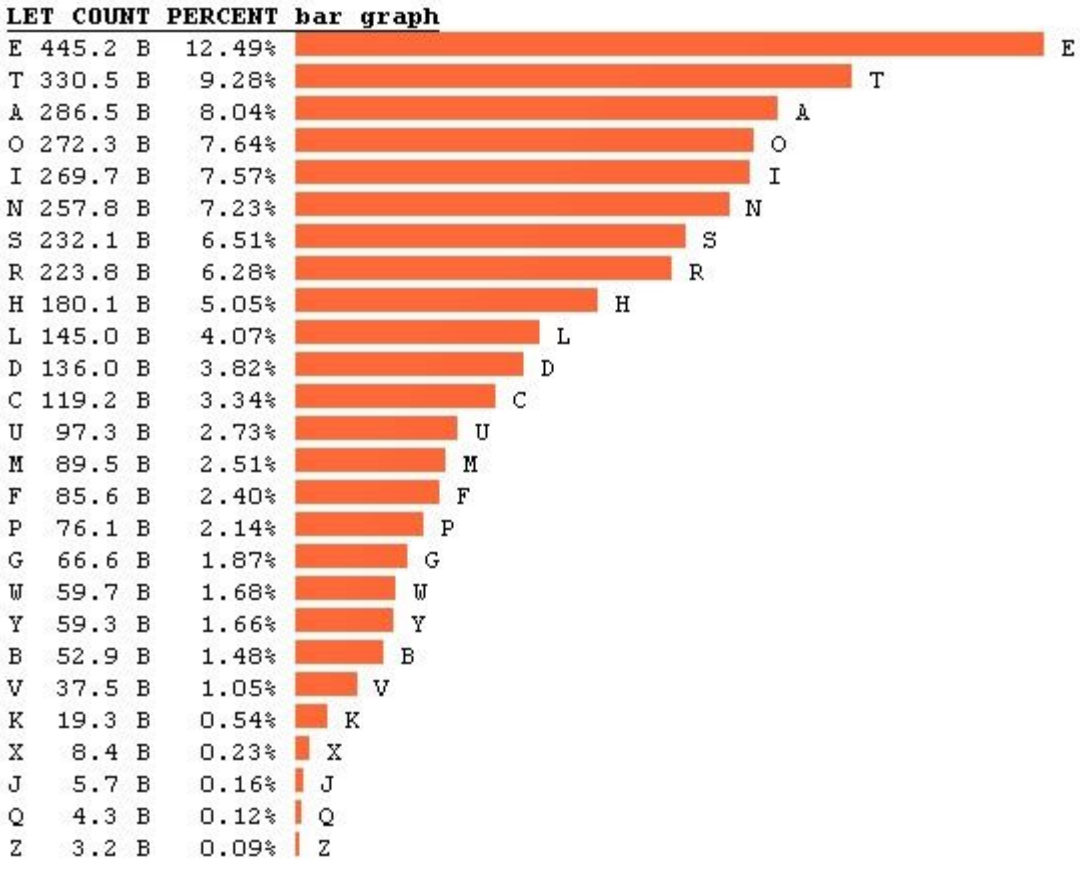
ABSTRACT

We study how in-context learning (ICL) in language models is affected by semantic priors versus input–label mappings. We investigate two setups—ICL with flipped labels and ICL with semantically-unrelated labels—across various model families (GPT-3, InstructGPT, Codex, PaLM, and Flan-PaLM). First, experiments on ICL with flipped labels show that overriding semantic priors is an emergent ability of model scale. While small language models ignore flipped labels presented in-context and thus rely primarily on semantic priors from pretraining, large models can override semantic priors when presented with in-context exemplars that contradict priors, despite the stronger semantic priors that larger models may hold. We next study *semantically-unrelated label ICL* (SUL-ICL), in which labels are semantically unrelated to their inputs (e.g., foo/bar instead of negative/positive), thereby forcing language models to learn the input–label mappings shown in in-context exemplars in order to perform the task. The ability to do SUL-ICL also emerges primarily with scale, and large-enough language models can even perform linear classification in a SUL-ICL setting. Finally, we evaluate instruction-tuned models and find that instruction tuning strengthens both the use of semantic priors and the capacity to learn input–label mappings, but more of the former.



How Does GPT Work?

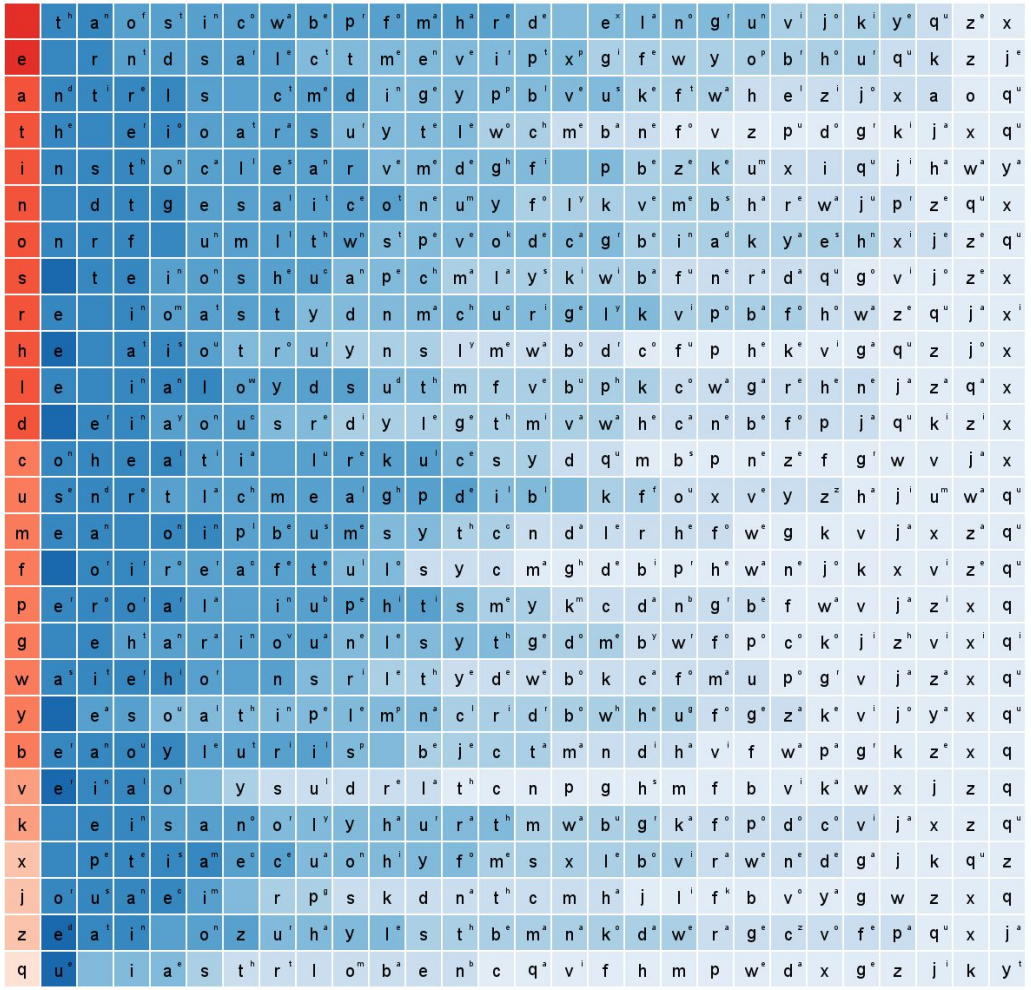
the **relative frequency** of individual letters in the English language



Letter and next-letter frequencies in English

measured across 1 million articles from Wikipedia

Conditional Probability of english letters



Colour key

a — Letters and spaces sorted by overall frequency. Ignores case and accents.

n^d — Next letter sorted by frequency. Small letter is the most common third letter following the pair.

Blank letters indicate spaces

Letter frequencies

- 31.6% to 100.0%
- 10.0% to 31.6%
- 3.2% to 10.0%
- 1.0% to 3.2%
- 0.3% to 1.0%
- 0.1% to 0.3%
- 0.0% to 0.1%

Markov generators

The letters distributions in the chart can be used to generate psuedowords such as the ones below. A similar approach, at the word level, is used for online parody generators.

bastrabot	*dithely*
loctrion	*raliket*
calpereek	*amorth*
forliatitive	*asocult*
wasions	*quarm*
felogy	*winferlifterand*
sonsih	*uniso*
fourn	*hise*
meembege	*nuouish*
prouning	*guncelawits*
nown	*rectere*
abrip	*doesium*

/u/Udzu

Reprinted with corrections from *The Bell System Technical Journal,*
Vol. 27, pp. 379–423, 623–656, July, October, 1948.

A Mathematical Theory of Communication

By C. E. SHANNON

INTRODUCTION

THE recent development of various methods of modulation such as PCM and PPM which exchange bandwidth for signal-to-noise ratio has intensified the interest in a general theory of communication. A basis for such a theory is contained in the important papers of Nyquist[1] and Hartley[2] on this subject. In the present paper we will extend the theory to include a number of new factors, in particular the effect of noise in the channel, and the savings possible due to the statistical structure of the original message and due to the nature of the final destination of the information.

The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point. Frequently the messages have *meaning*; that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem. The significant aspect is that the actual message is one *selected from a set* of possible messages. The system must be designed to operate for each possible selection, not just the one which will actually be chosen since this is unknown at the time of design.

If the number of messages in the set is finite then this number or any monotonic function of this number can be regarded as a measure of the information produced when one message is chosen from the set, all choices being equally likely. As was pointed out by Hartley the most natural choice is the logarithmic function. Although this definition must be generalized considerably when we consider the influence of the statistics of the message and when we have a continuous range of messages, we will in all cases use an essentially logarithmic measure.

The logarithmic measure is more convenient for various reasons:

1. It is practically more useful. Parameters of engineering importance such as time, bandwidth, number of relays, etc., tend to vary linearly with the logarithm of the number of possibilities. For example, adding one relay to a group doubles the number of possible states of the relays. It adds 1 to the base 2 logarithm of this number. Doubling the time roughly squares the number of possible messages, or doubles the logarithm, etc.

2. It is nearer to our intuitive feeling as to the proper measure. This is closely related to (1) since we intuitively measures entities by linear comparison with common standards. One feels, for example, that

1. The graph does not consist of two isolated parts A and B such that it is impossible to go from junction points in part A to junction points in part B along lines of the graph in the direction of arrows and also impossible to go from junctions in part B to junctions in part A.

2. A closed series of lines in the graph with all arrows on the lines pointing in the same orientation will be called a "circuit." The "length" of a circuit is the number of lines in it. Thus in Fig. 5 series BEBES is a circuit of length 5. The second property required is that the greatest common divisor of the lengths of all circuits in the graph be one.

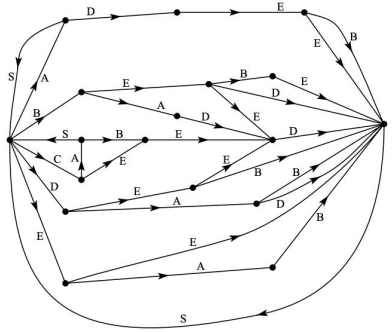


Fig. 5—A graph corresponding to the source in example D.

If the first condition is satisfied but the second one violated by having the greatest common divisor equal to $d > 1$, the sequences have a certain type of periodic structure. The various sequences fall into d different classes which are statistically the same apart from a shift of the origin (i.e., which letter in the sequence is called letter 1). By a shift of from 0 up to $d-1$ any sequence can be made statistically equivalent to any other. A simple example with $d = 2$ is the following: There are three possible letters a, b, c. Letter a is followed with either b or c with probabilities $\frac{1}{3}$ and $\frac{2}{3}$ respectively. Either b or c is always followed by letter a. Thus a typical sequence is

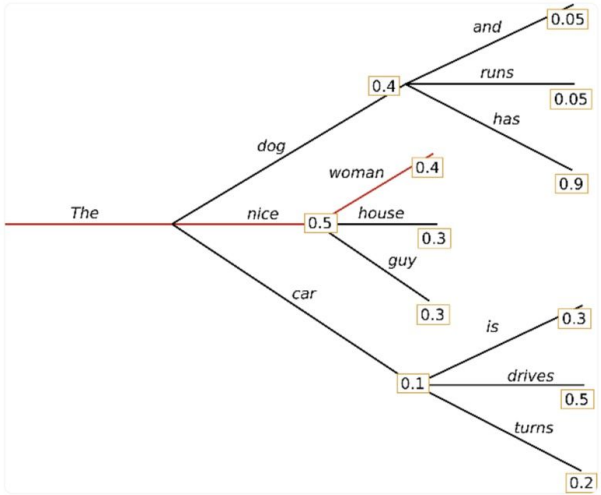
$$a\,b\,a\,c\,a\,c\,a\,c\,a\,b\,a\,c\,a\,b\,a\,b\,a\,c\,a\,c.$$

This type of situation is not of much importance for our work.

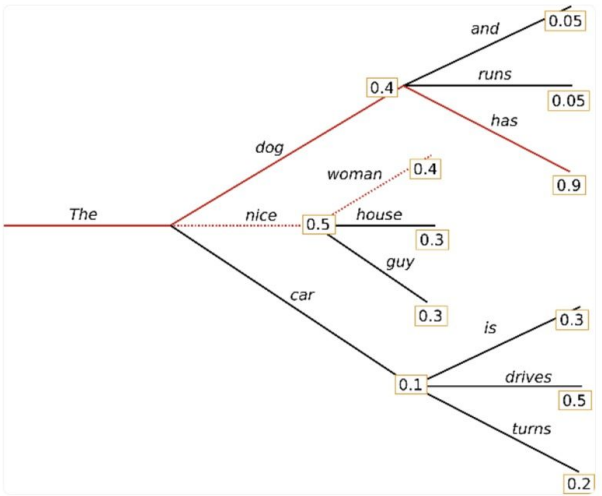
If the first condition is violated the graph may be separated into a set of subgraphs each of which satisfies the first condition. We will assume that the second condition is also satisfied for each subgraph. We have in this case what may be called a "mixed" source made up of a number of pure components. The components correspond to the various subgraphs. If $L_1, L_2, L_3, \ldots$ are the component sources we may write

Optimizing HyperParameters: Greedy Encoder

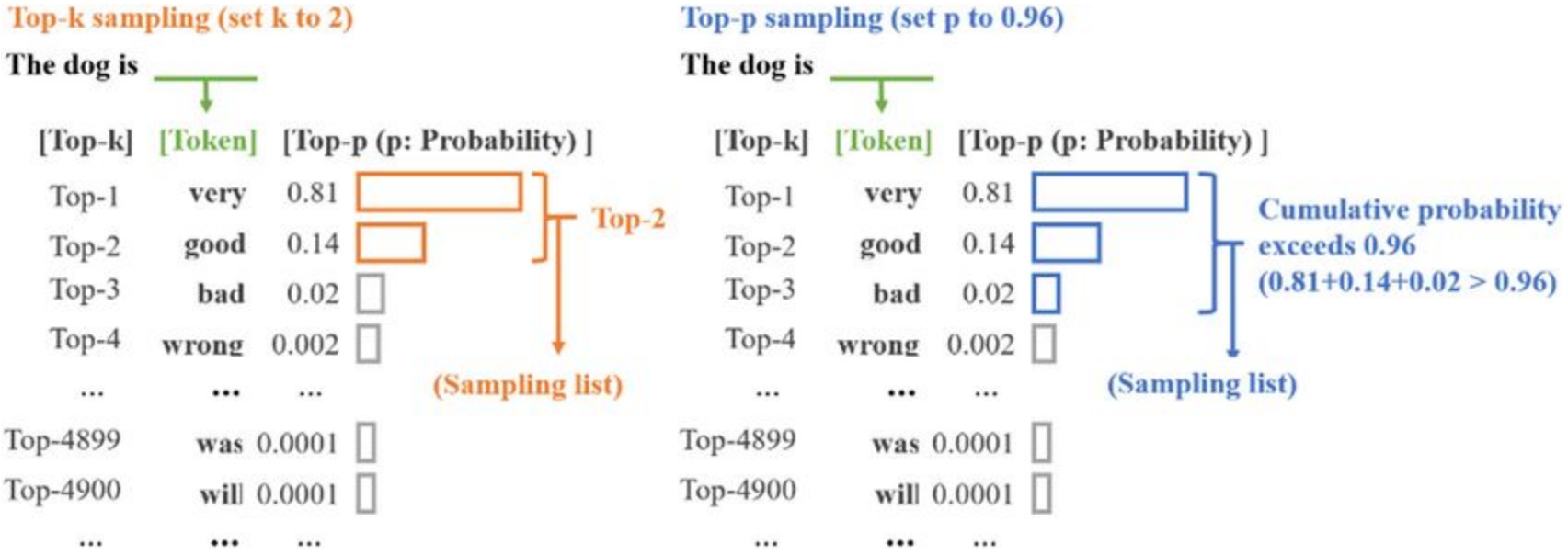
Greedy



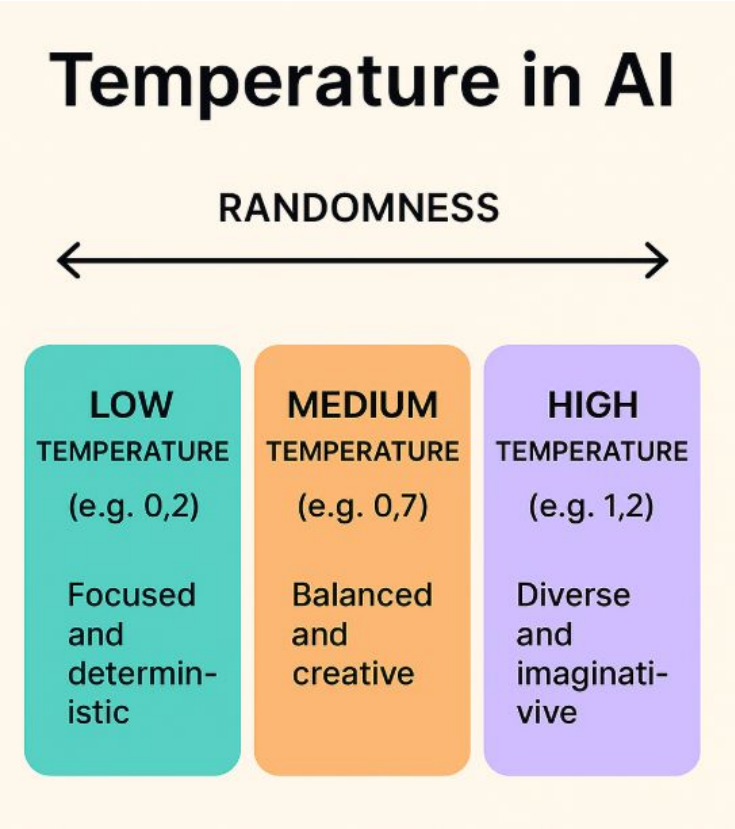
Beam

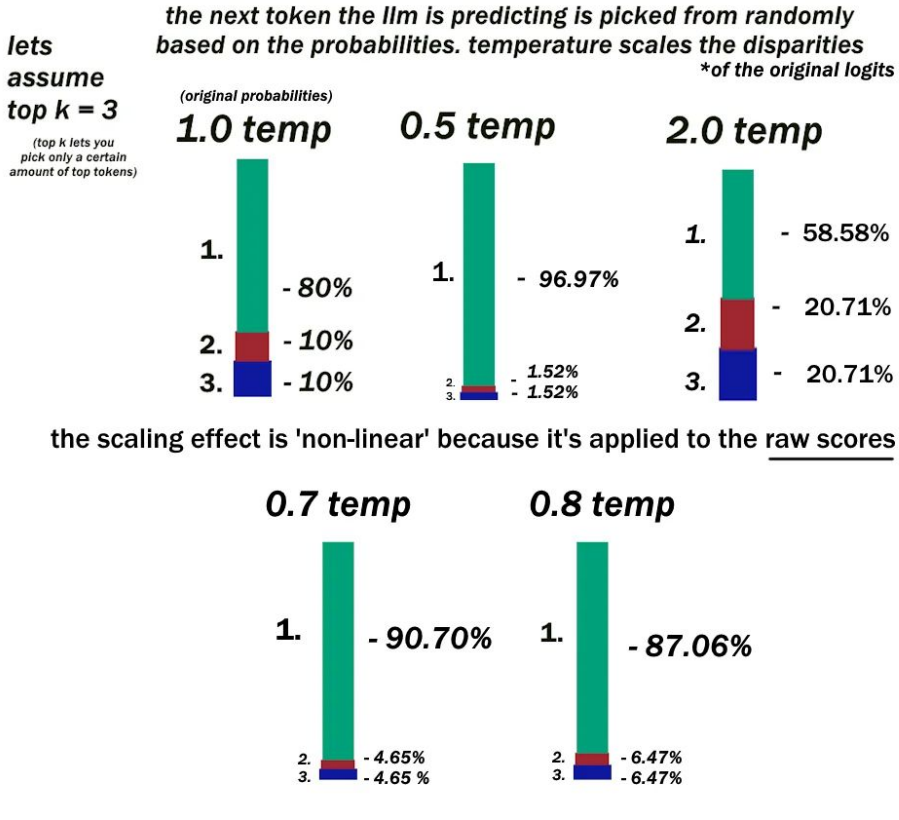


Optimizing HyperParameters: Top-p Vs. Top-k



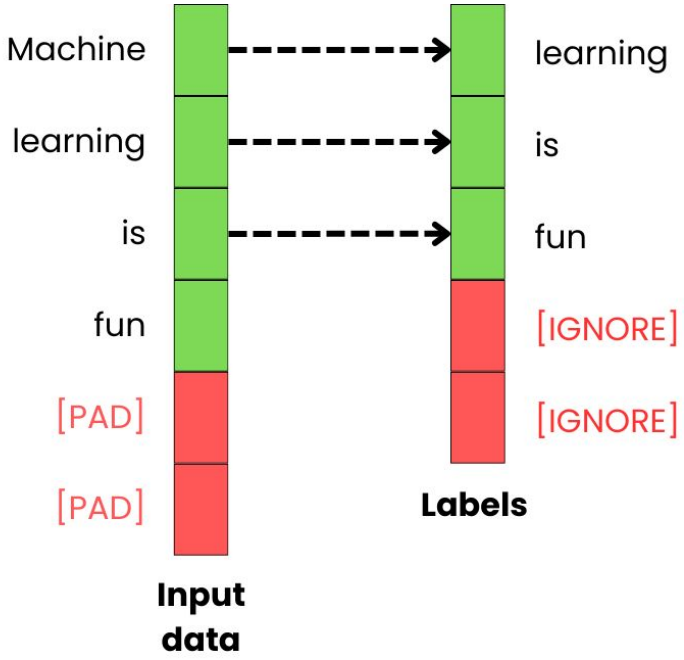
Optimizing HyperParameters: Temperature



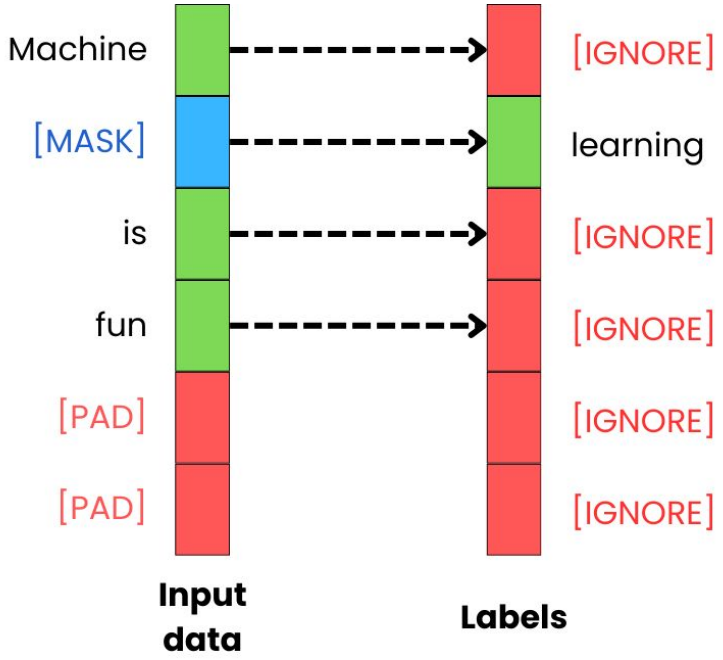


Optimizing Parameters

- SFT
- PEFT
- RLHF



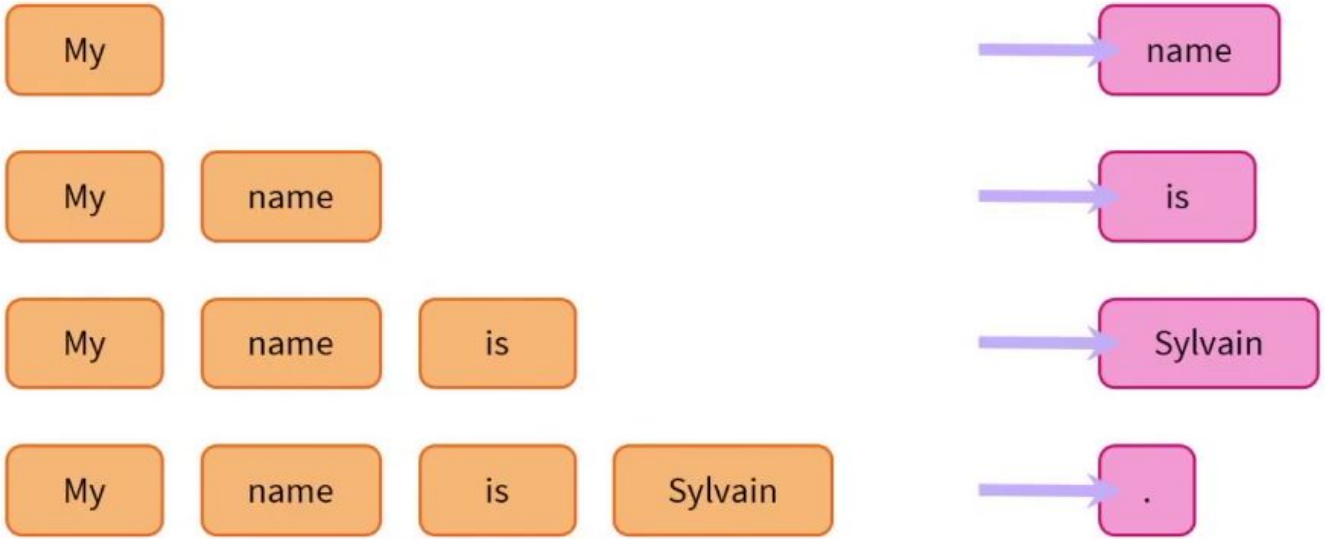
Causal Language Modeling



Masked Language Modeling

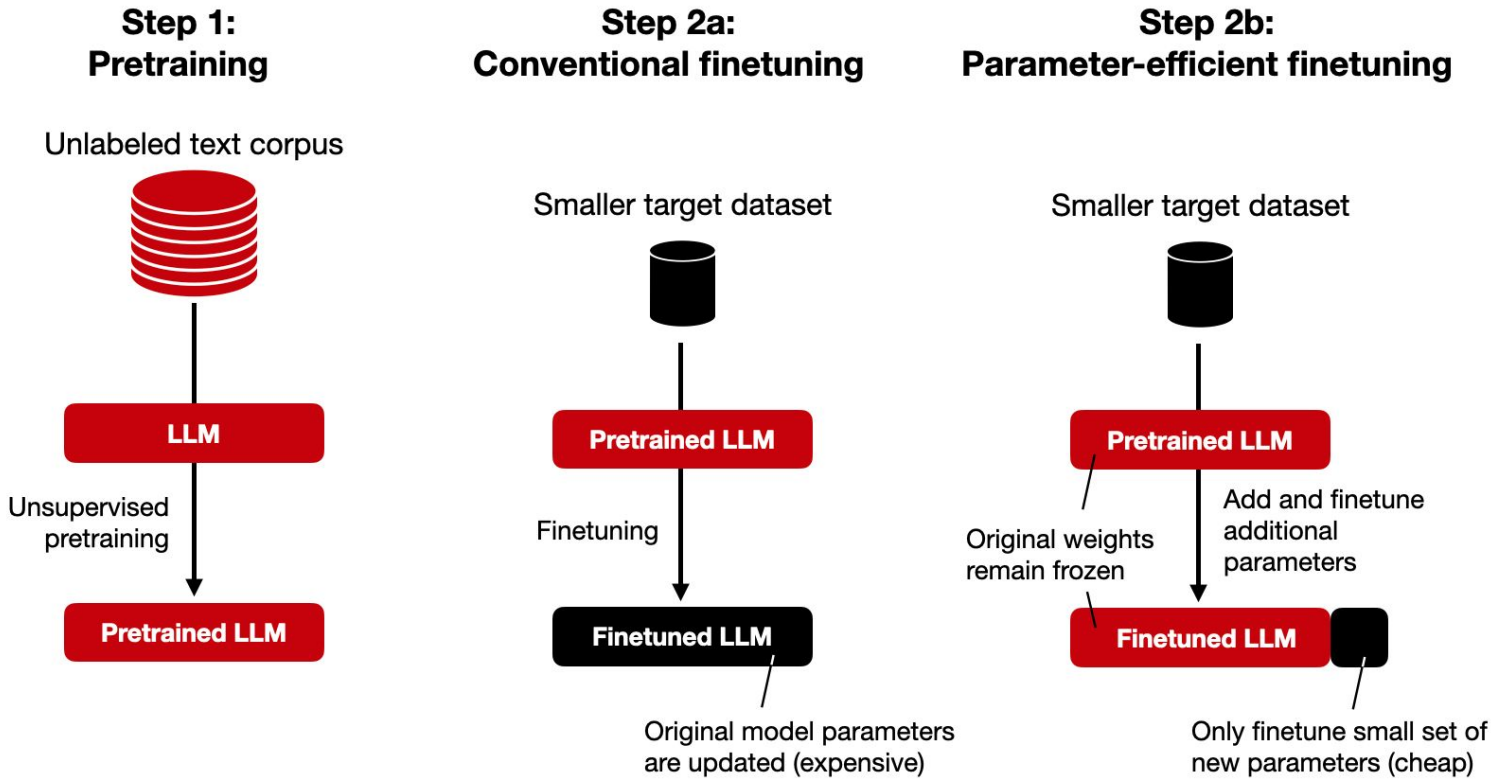
Causal Language Modeling

Guessing the next word in a sentence



Parameter-Efficient Transfer Learning for NLP

Neil Houlsby [1] Andrei Giurgiu [1 *] Stanisław Jastrzębski [2 *] Bruna Morrone [1] Quentin de Laroussilhe [1]



PEFT

These methods are divided into three broad categories:

1. **Additive Methods:** Additive methods introduce new parameters to the base model
    a. Adapters: Small dense (fully connected) networks inserted after specific transformer sublayers
    b. **Soft Prompts:** Fine-tuning applied directly to the model's input embeddings
1. **Selective Methods:** Selective methods adjust only a fraction of the existing model parameters.
    a. Top Layer Fine-Tuning: Focusing on fine-tuning only the upper layers of the network while leaving the lower layers untouched.
    b. Specific Parameter Fine-Tuning: Selectively training certain types of parameters, such as biases, while freezing other parameters.
    c. **Sparse Updates:** Selecting a specific subset of parameters for training. While promising, this approach can be computationally expensive due to the need to identify the most relevant parameters.
2. **Reparameterization-Based Method:** storage efficiency and training time
    a. **LoRa (Low-Rank Adaptation):** Employs low-rank matrix decomposition to represent weight updates
    b. Intrinsic SAID: Utilizes the Fastfood transform, a technique for efficiently representing low-rank updates.

Source: [2303.15647] Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning

PEFT: Additive Methods - Soft Prompts

The Power of Scale for Parameter-Efficient Prompt Tuning

Brian Lester* **Rami Al-Rfou** **Noah Constant**
Google Research
{brianlester,rmyeid,nconstant}@google.com

Abstract

In this work, we explore "prompt tuning," a simple yet effective mechanism for learning "soft prompts" to condition frozen language models to perform specific downstream tasks. Unlike the discrete text prompts used by GPT-3, soft prompts are learned through backpropagation and can be tuned to incorporate signals from any number of labeled examples. Our end-to-end learned approach outperforms GPT-3's few-shot learning by a large margin. More remarkably, through ablations on model size using T5, we show that prompt tuning becomes more competitive with scale: as models exceed billions of parameters, our method "closes the gap" and matches the strong performance of model tuning (where all model weights are tuned). This finding is especially relevant because large models are costly to share and serve and the ability to reuse one frozen model for multiple downstream tasks can ease this burden. Our method can be seen as a simplification of the recently proposed "prefix tuning" of Li and Liang (2021) and we provide a comparison to this and other similar

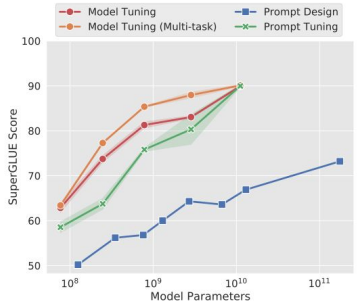


Figure 1: Standard **model tuning** of T5 achieves strong performance, but requires storing separate copies of the model for each end task. Our **prompt tuning** of T5 matches the quality of model tuning as size increases, while enabling the reuse of a single frozen model for all tasks. Our approach significantly outperforms few-shot **prompt design** using GPT-3. We show mean and standard deviation across 3 runs for tuning methods.

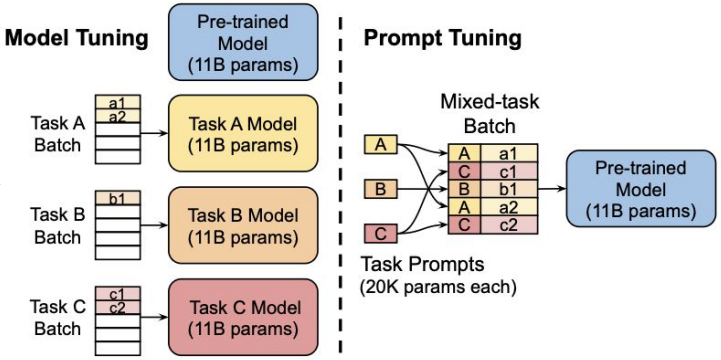


Figure 2: **Model tuning** requires making a task-specific copy of the entire pre-trained model for each downstream task and inference must be performed in separate batches. **Prompt tuning** only requires storing a small task-specific prompt for each task, and enables mixed-task inference using the original pre-trained model. With a T5 "XXL" model, each copy of the tuned model requires 11 billion parameters. By contrast, our tuned prompts would only require 20,480 parameters per task—a reduction of *over five orders of magnitude*—assuming a prompt length of 5 tokens.

PEFT: Selective Methods -Sparse Updates (MoE)

Journal of Machine Learning Research 23 (2022) 1-40 Submitted 8/21; Revised 3/22; Published 4/22

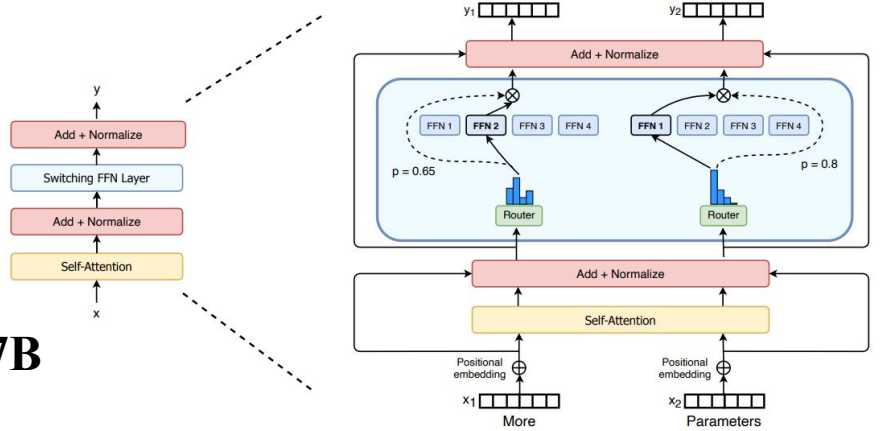
Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity

William Fedus*
LIAMFEDUS@GOOGLE.COM
Barret Zoph*
BARRETZOPH@GOOGLE.COM
Noam Shazeer
NOAM@GOOGLE.COM
Google, Mountain View, CA 94043, USA

Editor: Alexander Clark

Abstract

In deep learning, models typically reuse the same parameters for all inputs. Mixture of Experts (MoE) models defy this and instead select *different* parameters for each incoming example. The result is a sparsely-activated model—with an outrageous number of parameters—but a constant computational cost. However, despite several notable successes of MoE, widespread adoption has been hindered by complexity, communication costs, and training instability. We address these with the introduction of the Switch Transformer. We simplify the MoE routing algorithm and design intuitive improved models with reduced communication and computational costs. Our proposed training techniques mitigate the instabilities, and we show large sparse models may be trained, for the first time, with lower



Mixtral 8x7B

Figure 2: Illustration of a Switch Transformer encoder block. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = "More" and x_2 = "Parameters" below) being routed (solid lines) across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value (dotted-line).

PEFT: LoRa (Low-Rank Adaptation)

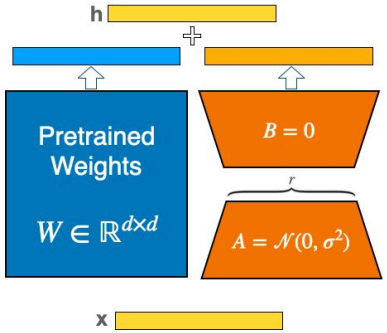
LoRA: Low-Rank Adaptation of Large Language Models

Edward Hu* **Yelong Shen*** **Phillip Wallis** **Zeyuan Allen-Zhu**
Yuanzhi Li **Shean Wang** **Lu Wang** **Weizhu Chen**
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

Abstract

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example – deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose **Low-Rank Adaptation**, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, *no additional inference latency*. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at https://github.com/microsoft/LoRA.

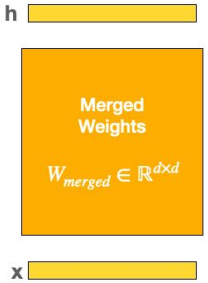
During training



$$h = Wx + BAx$$

$$h = \underbrace{(W + BA)}_{W_{merged}}x$$

After training



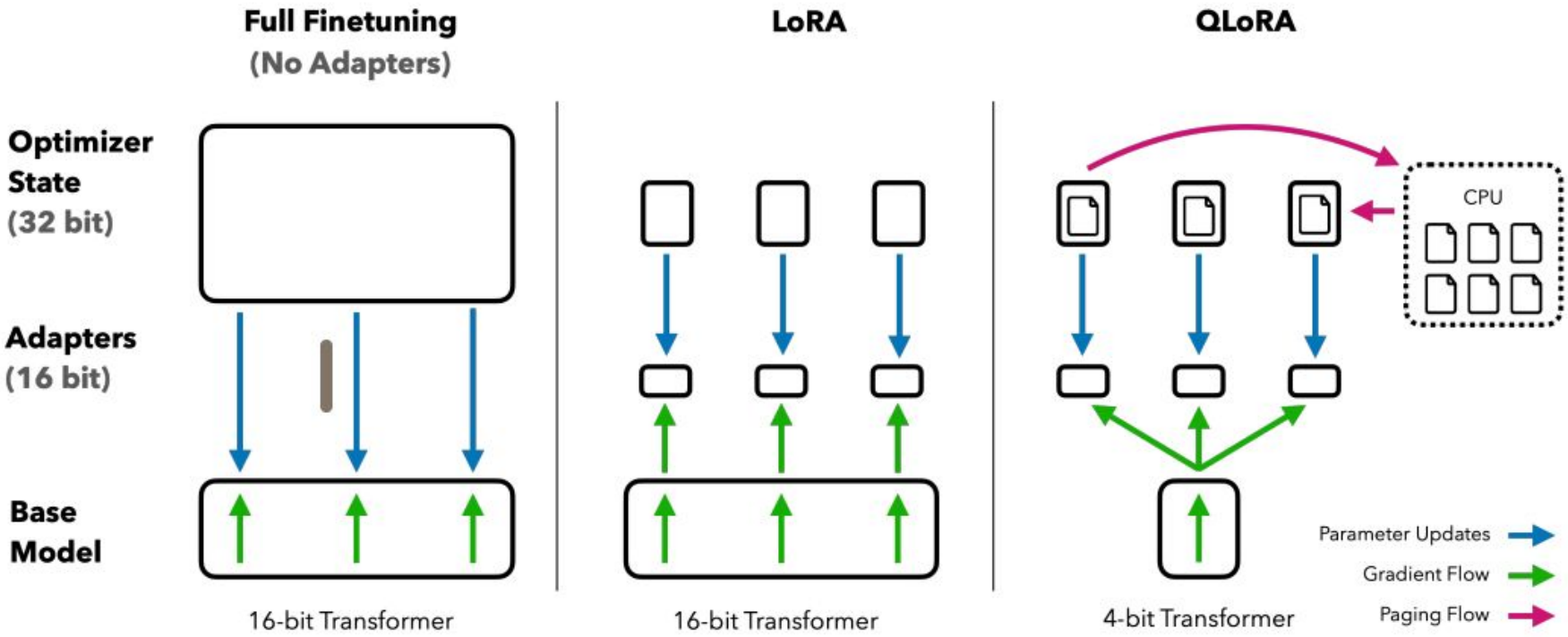


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

Bridging the Gap: Data & AI in the Heart of Industry

Esmaeil Nikumanesh
Senior Solution Architect Data & AI
NTT Data Deutschland SE

Feb. 2026