Intern Weekly Report Summary

MSRA HEX Intern

Jack Zeng

June - September 2023

General goals (not all)

Enhance LLM's ability on edge side, reduce cost on server side

- Edge side: Laptop, smartphone, small business' servers…
- Decrease the difference between smaller LLM and larger LLM

Try to train and implement a "personalized" edge-side LLM

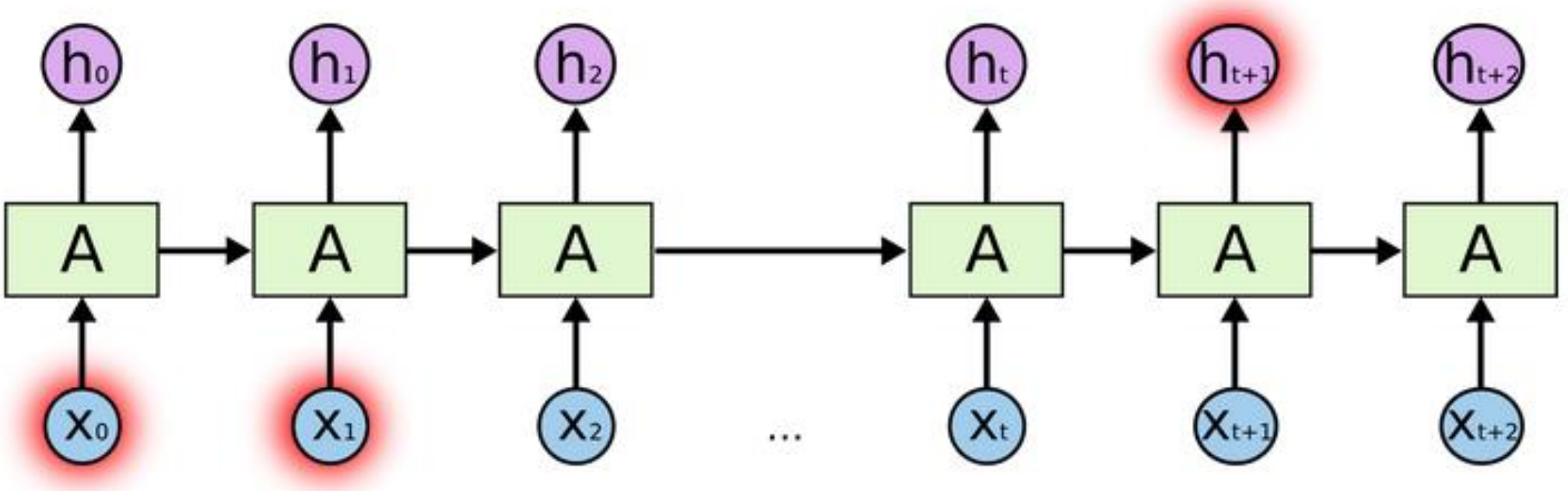
- Could answer private information questions

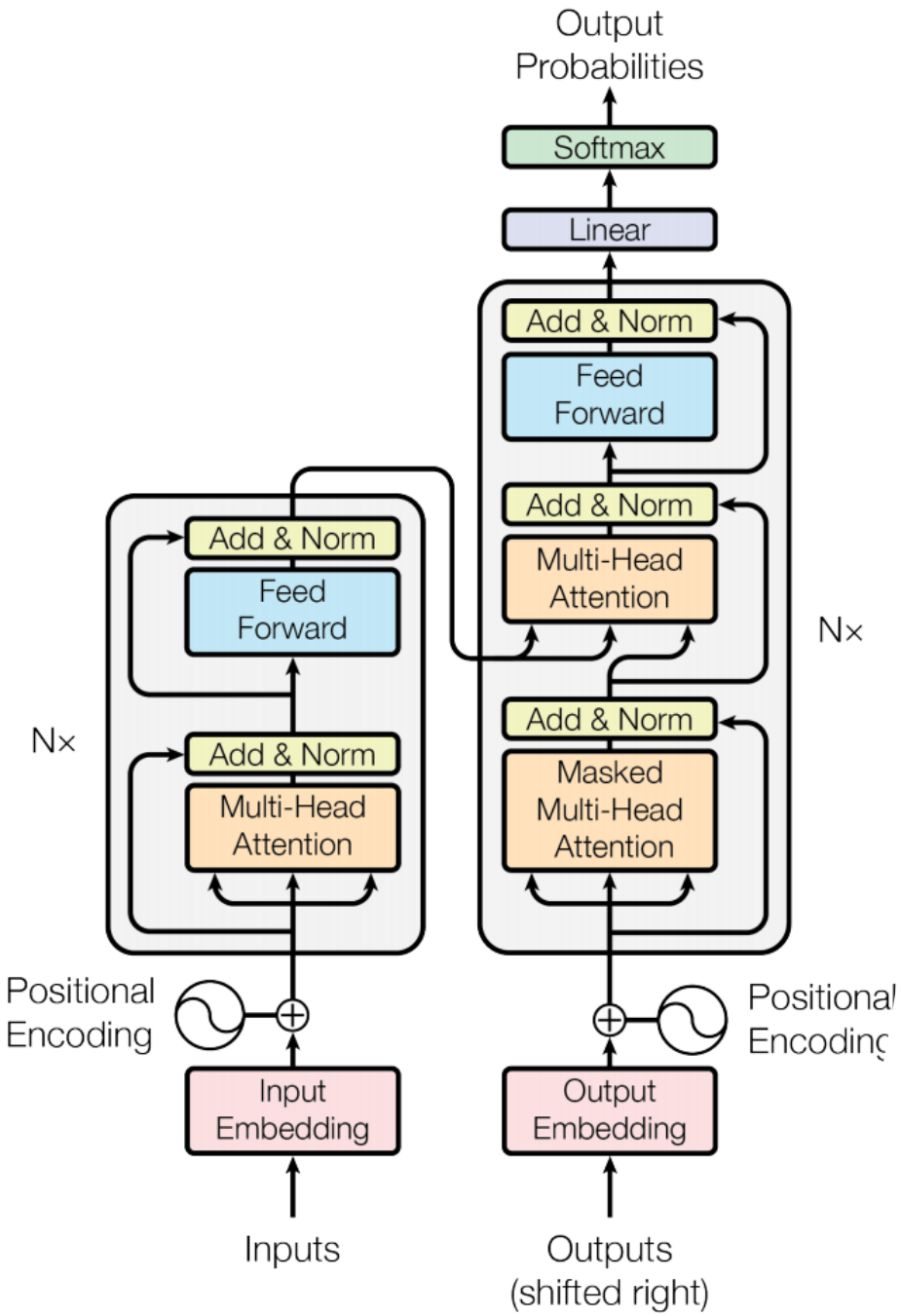
Week 1 & 2

Transformer Intro

Transformer: Why not RNN?

- RNN generates hidden states -> precludes parallelization
- Long-term dependency problem
  - Potential solution: LSTM

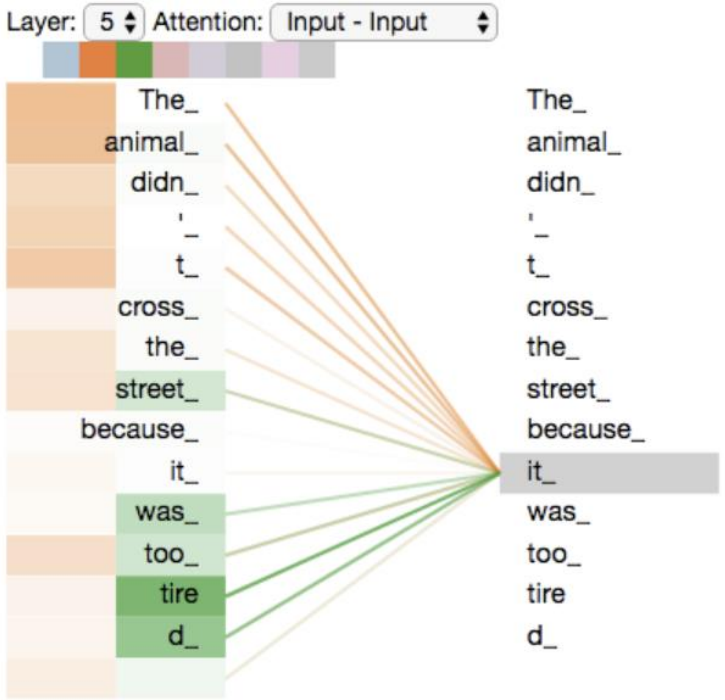




Transformer: Structure

- Left part: encoder
- Right part: decoder
- It consists
  - Adding and Normalization Layer
  - Feed-Forward Layer
  - Multi-Head Attention Layer
  - Positional Encoding

Transformer: Encoder and Decoder in Detail



- Encoder:
  - Self-attention
    - Three matrix (vector) for each word (after embedding): Query, Key and Value
      - **Query** (text in the search bar)
      - **Key** (video title, description)
      - **Value** (best matched videos)
    - Each matrix is the product of embedding matrices and corresponding weights (similarity calculation)
    - $\sqrt{d_k}$: Make gradient values remain stable during training (distribution of softmax function is highly related to d)
    - Multi-head attention: compute multiple attentions from the input information in parallel
    - Purpose of self-attention:
      - Easier to capture long-distance dependency features in sentences

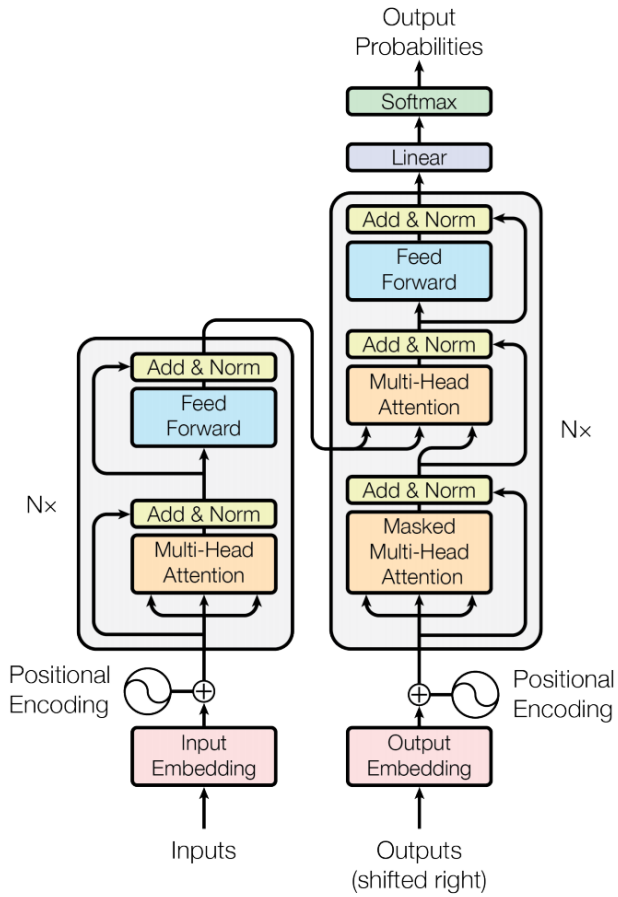
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Transformer: Continue...

- "Encoder-Decoder" Attention (masked attention)
  - Same calculation method as self-attention
  - Q -> last output from decoder
  - K and V -> outputs from encoder
  - Purpose: focus on appropriate place in the input

- Position Encoding: Provide relative location info

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



LoRA (Low-Rank Adaptation)

- Low-rank transformation:
  - Approximate high-dimensional matrix using low dimensional representation
- Purpose: Increase fine tuning efficiency
- Low intrinsic dimension
- Key idea:
  - Decompose weight update matrix into two smaller (A and B on the right)
  - Train matrix A and B only

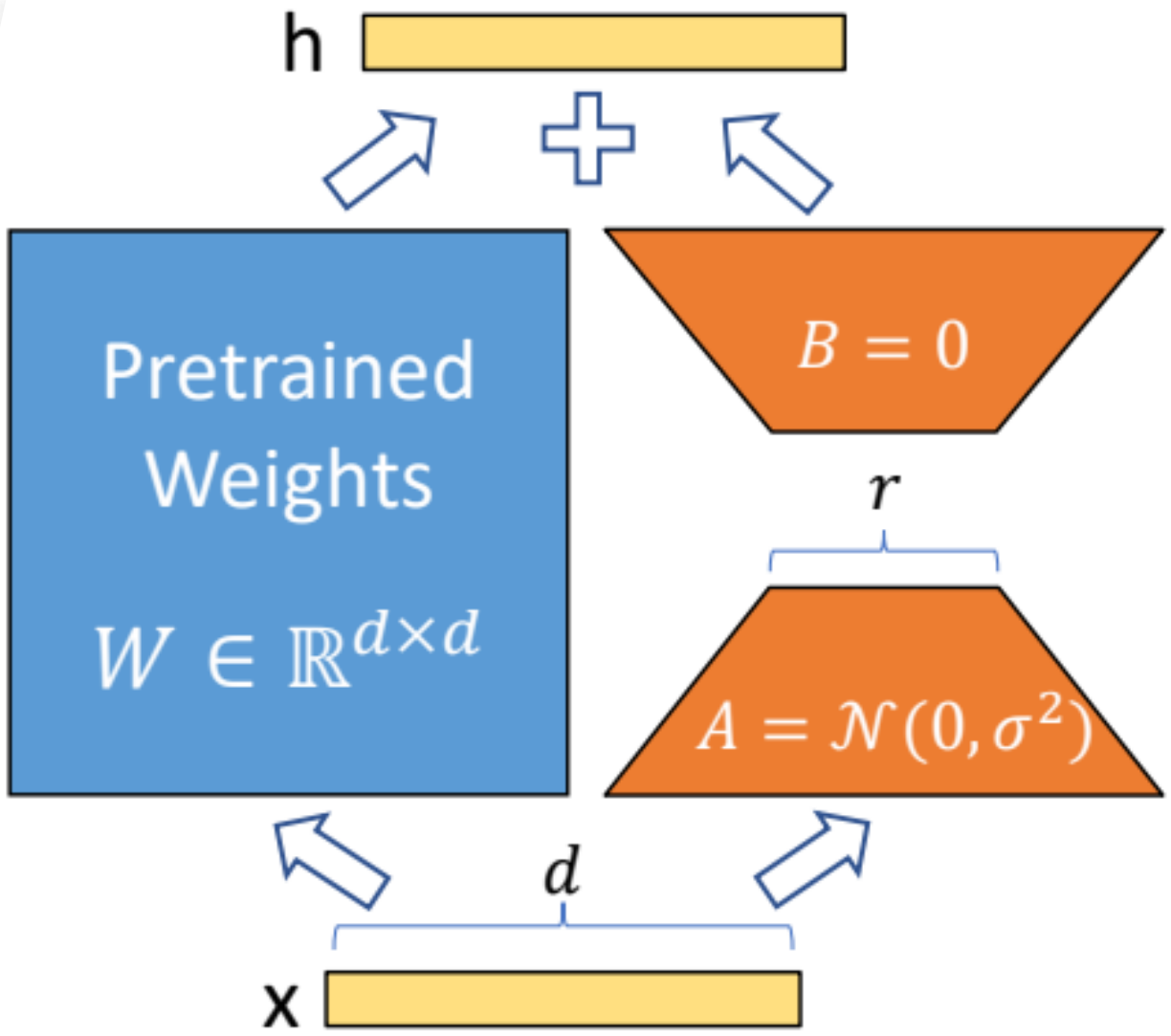


Figure 1: Our reparametrization. We only train A and B.

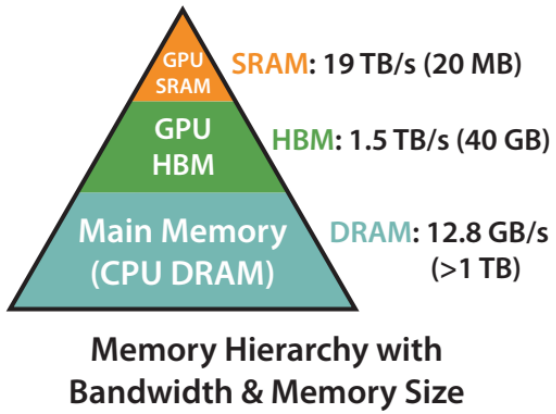
LoRA Applications and Issues (On week 2)

- Using LoRA to fine-tune ChatGLM-6B
- Issues:
  - GPU usage (solved by using Hugging Face's Accelerate)
  - Prompting (not solved yet)

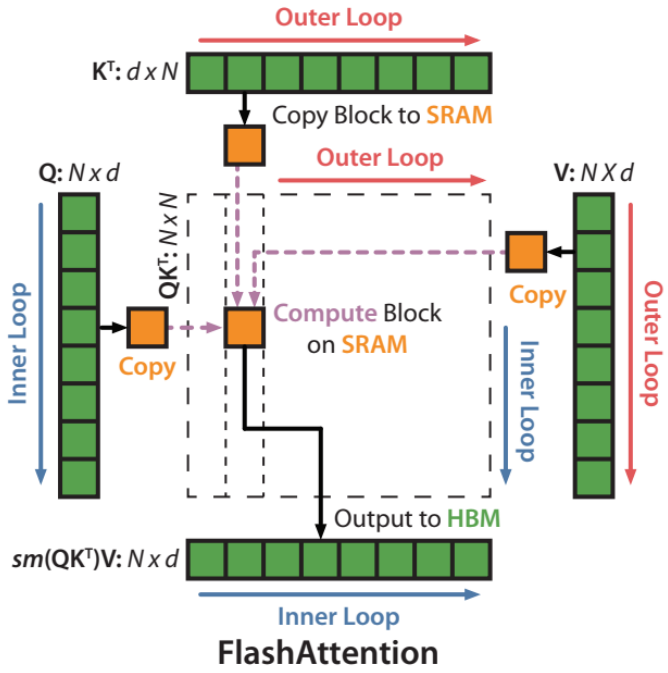
0	..a V100-DGXS-32GB	On	00000000:07:00.0 Off		0
N/A	46C	P0	68W / 300W	20.84GiB / 32.00GiB	0%	Default
1	..a V100-DGXS-32GB	On	00000000:08:00.0 Off		0
N/A	47C	P0	67W / 300W	25.27GiB / 32.00GiB	0%	Default
2	..a V100-DGXS-32GB	On	00000000:0E:00.0 Off		0
N/A	48C	P0	210W / 300W	25.27GiB / 32.00GiB	0%	Default
3	..a V100-DGXS-32GB	On	00000000:0F:00.0 Off		0
N/A	50C	P0	77W / 300W	21.92GiB / 32.00GiB	99%	Default

Flash Attention (Roughly)

- Key idea: dividing operations between faster and slower levels of GPU memory
- Compute-bound and memory-bound
- For transformer, bottleneck -> attention matrix on the HBM and repeating read/write
- Tilting: split the inputs Q, K, V into blocks and combine
- Recompute: store softmax()



Memory Hierarchy with Bandwidth & Memory Size



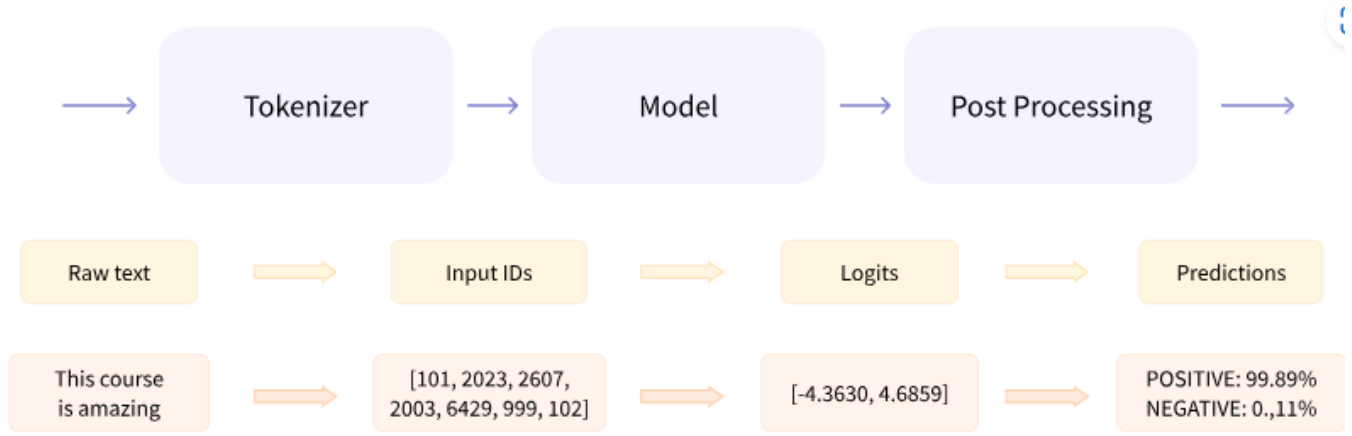
FlashAttention

Week 3

Huggingface Transformer

Huggingface API Structure

- Basic APIs:
  - Transformer
  - Tokenizer
  - Datasets
  - Trainer



Setup and Tokenizer Initialization

- Model: LLaMA 13B
  - Checkpointing Shard: 14GB GPU memory
  - For LLaMA 8B, 8.5GB GPU memory

```python
import torch
# import os
# os.environ["CUDA_VISIBLE_DEVICES"] = "0"
import torch.nn as nn
import bitsandbytes as bits
from transformers import AutoModelForCausalLM, AutoTokenizer, HfArgumentParser, TrainingArguments, LlamaTokenizer, LlamaForCausalLM, LlamaConfig

model = LlamaForCausalLM.from_pretrained(
    "decapoda-research/llama-13b-hf",
    load_in_8bit=True,
    device_map='auto'
)
tokenizer = LlamaTokenizer.from_pretrained("decapoda-research/llama-13b-hf")
if tokenizer.pad_token is None:
    tokenizer.add_special_tokens({'pad_token': '[PAD]'})
    model.resize_token_embeddings(len(tokenizer))
```

Precision and LoRA Setup

```
from peft import LoraConfig, get_peft_model
lora_config = LoraConfig(
    r = 16,
    target_modules = ['q_proj', 'k_proj', 'v_proj', 'o_proj'],
    lora_alpha=32,
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)
model = get_peft_model(model, lora_config)
model.print_trainable_parameters()
```

4]

```
trainable params: 26214400 || all params: 13042088960 || trainable%: 0.20099847563070142
```

LoRA parameter settings

```
for param in model.parameters():
    param.requires_grad = False
    if param.ndim == 1:
        # Cast the small parameters to fp16 for stability
        param.data = param.data.to(torch.float16) # Precision

model.gradient_checkpointing_enable()
model.enable_input_require_grads()

class CastOutputToFloat(nn.Sequential):
  def forward(self, x): return super().forward(x).to(torch.float16)
model.lm_head = CastOutputToFloat(model.lm_head)
```

weight matrix $W_0 \in \mathbb{R}^{d\times k}$, we constrain its update by representing the latter with a low-rank decomposition $W_0 + \Delta W = W_0 + BA$, where $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$, and the rank $r \ll \min(d, k)$. During training, W_0 is frozen and does not receive gradient updates, while A and B contain trainable parameters. Note both W_0 and $\Delta W = BA$ are multiplied with the same input, and their respective output vectors are summed coordinate-wise. For $h = W_0x$, our modified forward pass yields:

$$h = W_0x + \Delta Wx = W_0x + BAx \quad (3)$$

We illustrate our reparametrization in Figure 1. We use a random Gaussian initialization for A and zero for B, so $\Delta W = BA$ is zero at the beginning of training. We then scale ΔWx by $\frac{\alpha}{r}$, where α is a constant in r. When optimizing with Adam, tuning α is roughly the same as tuning the learning rate if we scale the initialization appropriately. As a result, we simply set α to the first r we try and do not tune it. This scaling helps to reduce the need to retune hyperparameters when we vary r (Yang & Hu, 2021).

Dataset Setup (cont.)

```
from datasets import load_dataset
def merge_columns(example):
    example["prediction"] = example["question"] + str(example["answer"]['aliases'])
    return example
data = load_dataset("trivia_qa",'rc.web.nocontext').map(merge_columns)
data['train'][0]['prediction']
```

Example:

```
"Which American-born Sinclair won the Nobel Prize for Literature in 1930?['(Harry) Sinclair Lewis', 'Harry Sinclair Lewis', 'Lewis, (Harry) Sinclair', 'Grace Hegger', 'Sinclair Lewis']"
```

Tokenization

```python
def tokenize(prompt):
    result = tokenizer(
        prompt,
        truncation=True,
        max_length=2048,
        padding="max_length",
        add_special_tokens=True
    )
    return {
        "input_ids": result["input_ids"],
        "attention_mask": result["attention_mask"],
    }
tokenizer.padding = True
data = data.map(lambda samples: tokenize(samples['prediction']), batched=True, remove_columns=columns)
```



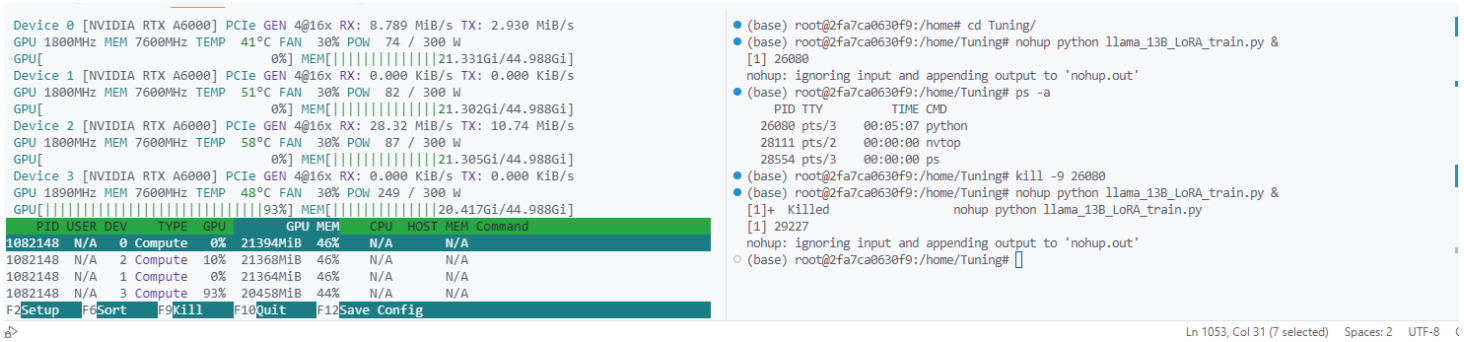
```
Dataset({
    features: ['input_ids', 'attention_mask'],
    num_rows: 76496
})
```

Training

```
from transformers import Trainer, TrainingArguments, DataCollatorForLanguageModeling
import matplotlib.pyplot as plt
trainer = Trainer(
    model = model,
    train_dataset=data['train'],
    eval_dataset=data['test'],
    args=TrainingArguments(
        num_train_epochs=3,
        per_device_train_batch_size=4,
        gradient_accumulation_steps=4,
        warmup_steps=100,
        max_steps=200,
        learning_rate=2e-4,
        fp16=True,
        logging_steps=1,
        optim='adamw_torch',
        output_dir='./results'
    ),
    data_collator=DataCollatorForLanguageModeling(tokenizer, mlm=False)
)
model.config.use_cache = False
trainer.train()
```

Issue (Week 3)

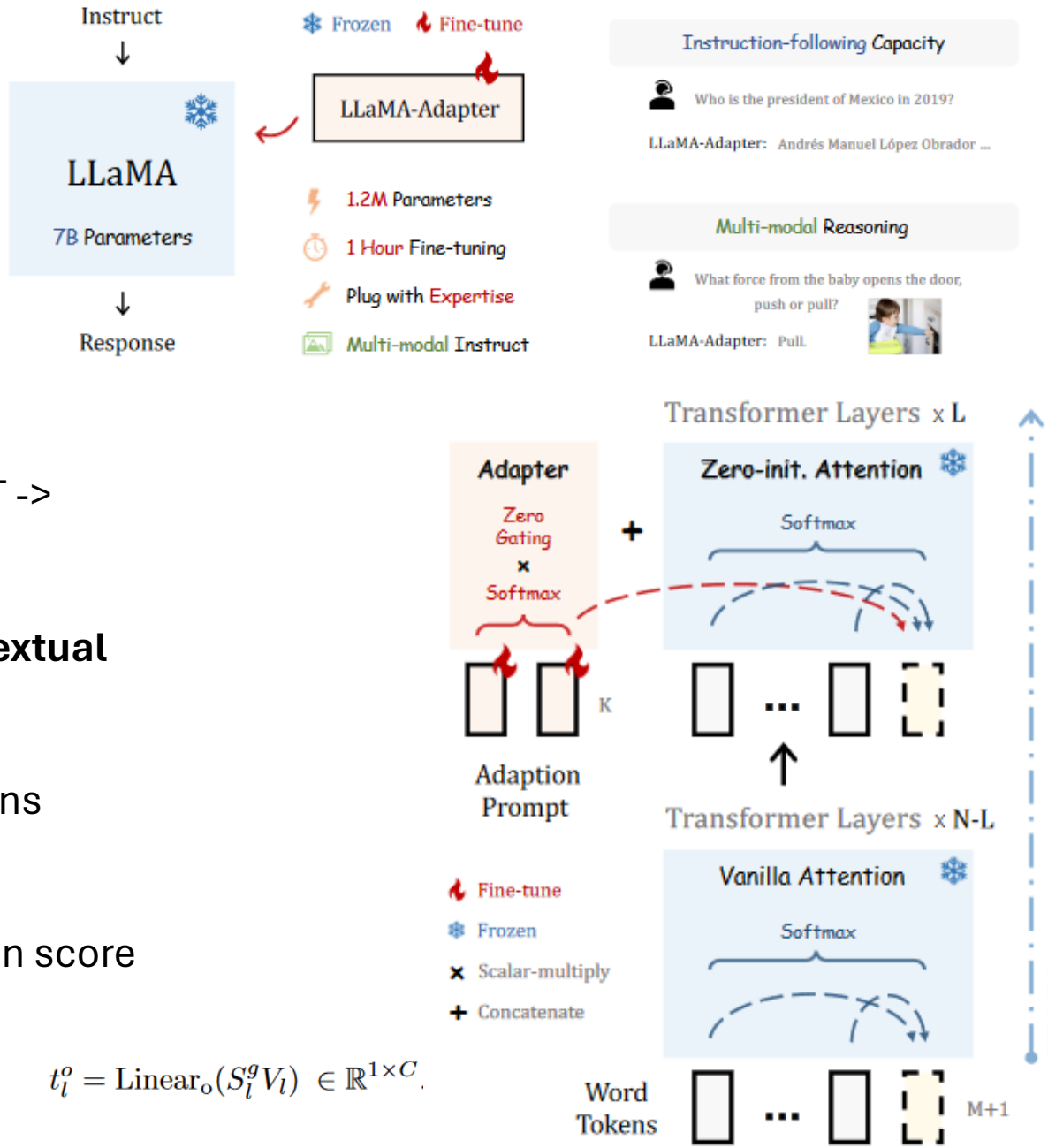
- Potential Solutions:
  - Huggingface Accelerate
  - Pytorch Script: https://huggingface.co/docs/transformers/run_scripts#distributed-training-and-mixed-precision
  - DeepSpeed



Week 4

PEFT Intro

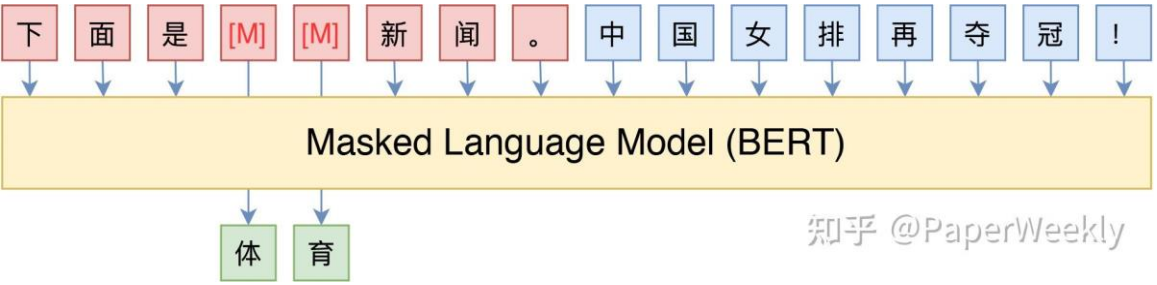
Llama-Adapter

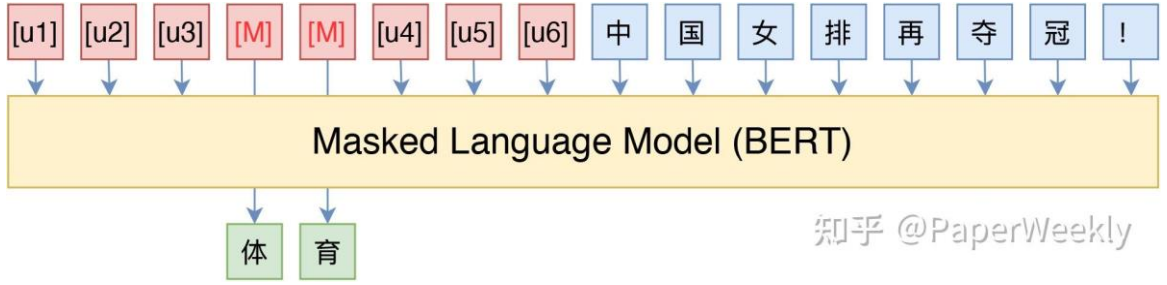


- Learnable Adaption Prompt
  - Instruct fine-tuning
  - $[P_l;\ T_l] \in \mathbb{R}^{(K+M)\times C}$
  - K -> instruction length, C -> feature dimension, T -> word with length M
  - Only insert prompt into topmost L layers
  - Purpose: Guide T to generate **subsequent contextual response**
- Zero-init Attention and learnable gating factor
  - Purpose: Solve the disturbance to the word tokens brought by adaption
  - Change last L transformer layers
  - Gating factor: control the importance of attention score

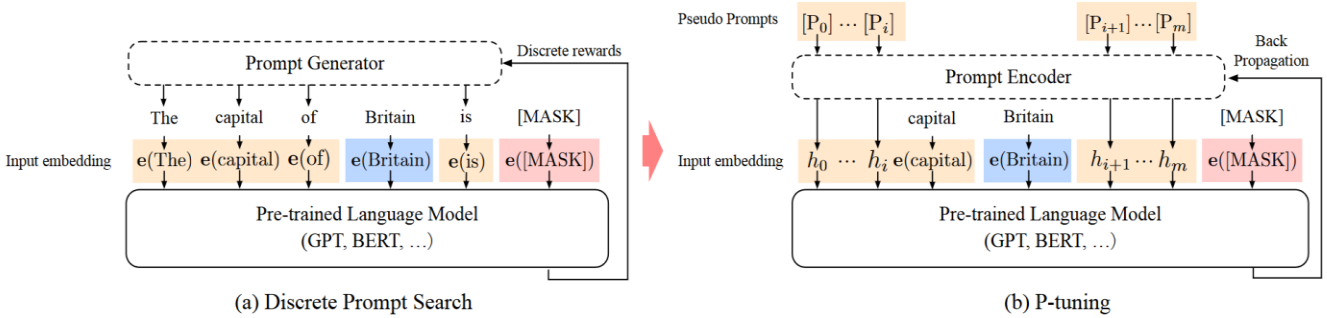
$$
\begin{aligned}
Q_l &= \text{Linear}_\text{q}(\ t_l\ );\\
K_l &= \text{Linear}_\text{k}(\ [P_l;\ T_l;\ t_l]\ );\\
V_l &= \text{Linear}_\text{v}(\ [P_l;\ T_l;\ t_l]\ ).
\end{aligned}
\Rightarrow
\begin{aligned}
S_l &= Q_l K_l^T/\sqrt{C} \in \mathbb{R}^{1\times(K+M+1)},\\
S_l^g &= [\text{softmax}(S_l^K)\cdot g_l;\ \ \text{softmax}(S_l^{M+1})]^T.
\end{aligned}
\Rightarrow
t_l^o = \text{Linear}_\text{o}(S_l^g V_l) \in \mathbb{R}^{1\times C}.
$$

P-tuning

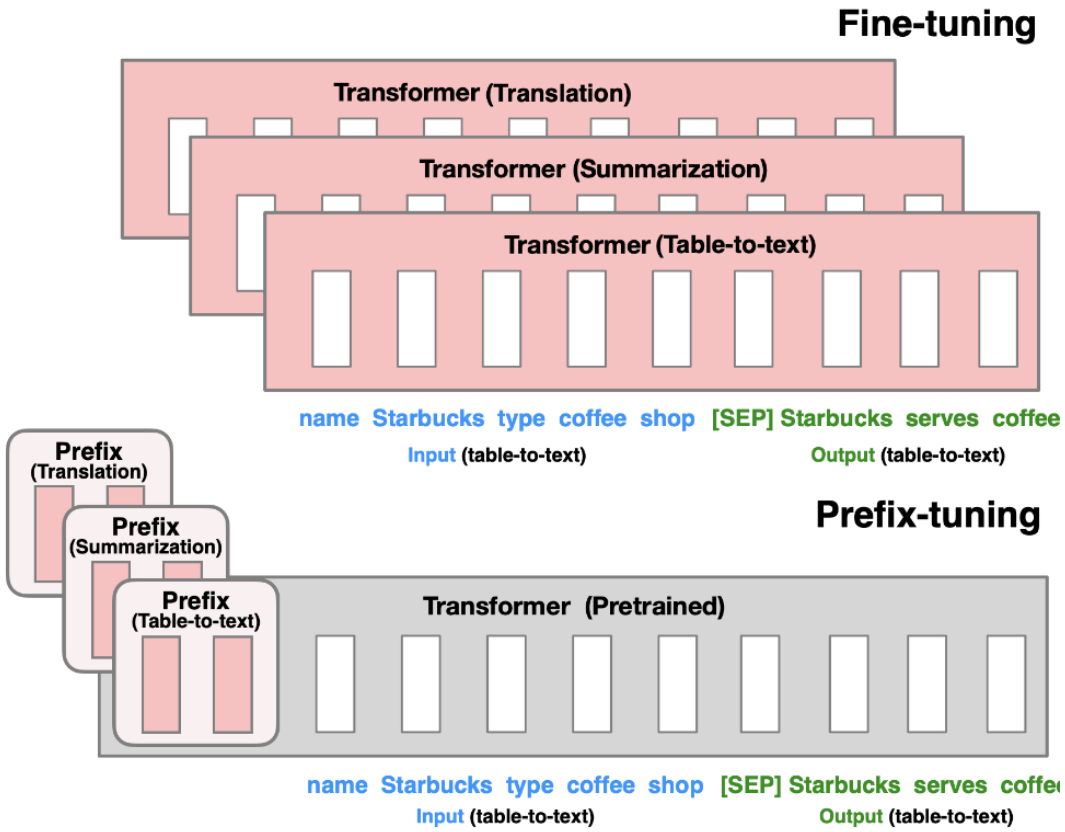




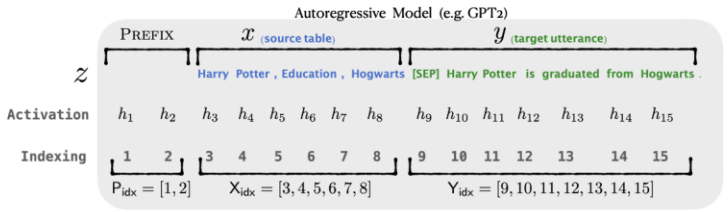
- Key: using pre-existed annotation dataset to **auto generate masked prompt**
- Solve NLU problems
- Fix the weights of the whole model and optimize the embedding of only the tokens [unused1]～[unused6]
  - Using a small LSTM to calculate the embedding
- Advantage: save time

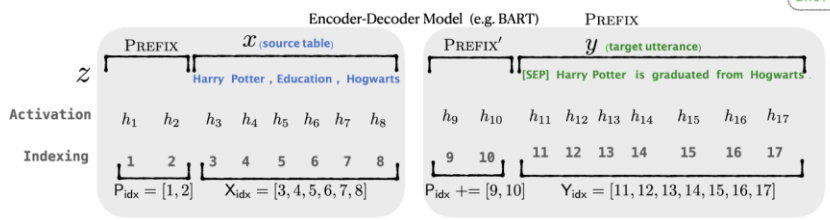


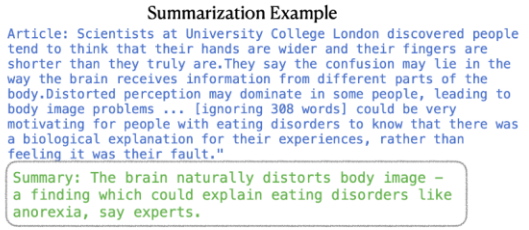
Prefix-Tuning

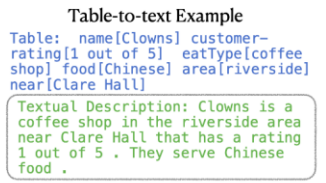


- Key: Frozen the model weight, optimize continuous task-specific vector
- Focus: table-to-text, summarization problem
- Store a transformer model + specific prefix vector
- Mapping a MLP matrix









Week 5

Instruction Tuning

Instruction Tuning: Paper List

Finetuned Language Models Are Zero-Shot Learners (2021, FLAN)

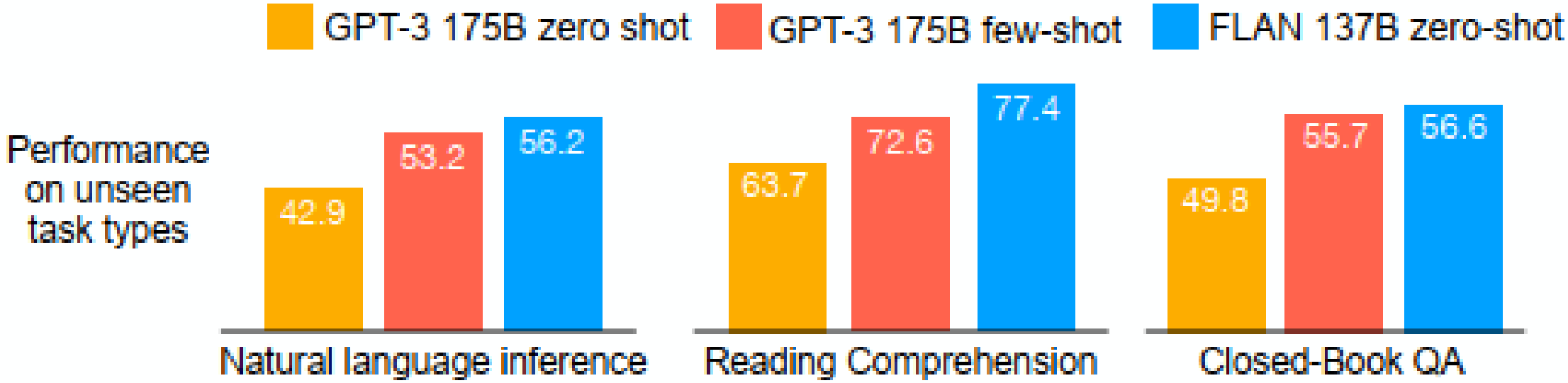
Multitask Prompted Training Enables Zero-Shot Task Generalization (2021, T0)

Scaling Instruction-Finetuned Language Models (2022, FLAN-PaLM)

Self-Instruct: Aligning Language Model with Self Generated Instructions

Instruction Tuning: FLAN

- GPT-3: Prompt Tuning
- Question: Worse performance in Zero-Shot question
- Potential reason: without few-shot examples, worse ability of **inference** if input prompt **doesn't match tuning format**
- Reality scenario: User asks zero-shot questions more.



FLAN: Datasets and templates

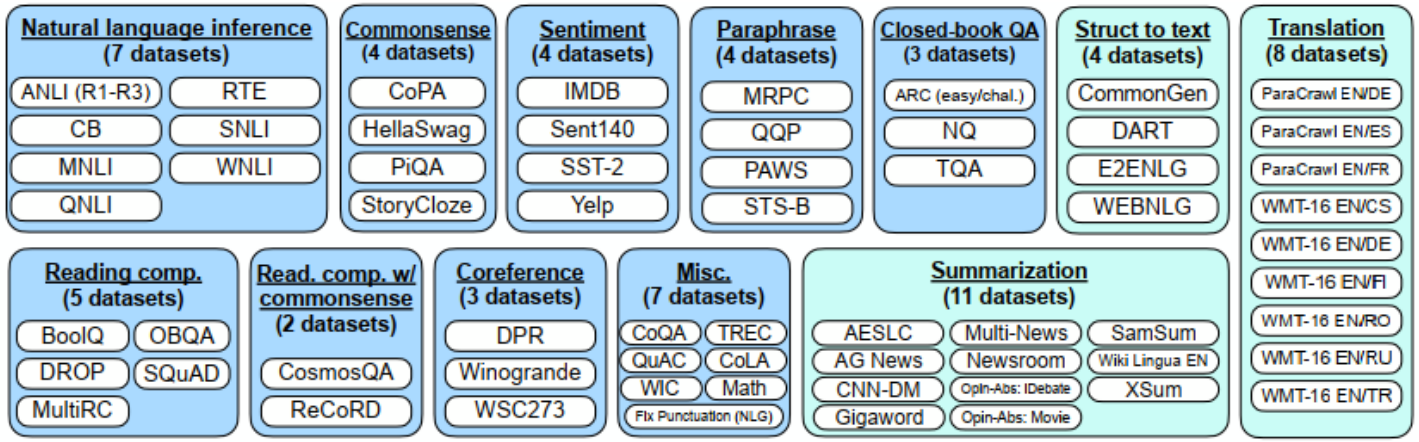


Figure 3: Datasets and task clusters used in this paper (NLU tasks in blue; NLG tasks in teal).

Premise

Russian cosmonaut Valery Polyakov set the record for the longest continuous amount of time spent in space, a staggering 438 days, between 1994 and 1995.

Hypothesis

Russians hold the record for the longest stay in space.

Target

Entailment
Not entailment

Options:
- yes
- no

Template 1

<premise>
Based on the paragraph above, can we conclude that <hypothesis>?
<options>

Template 2

<premise>
Can we infer the following?
<hypothesis>
<options>

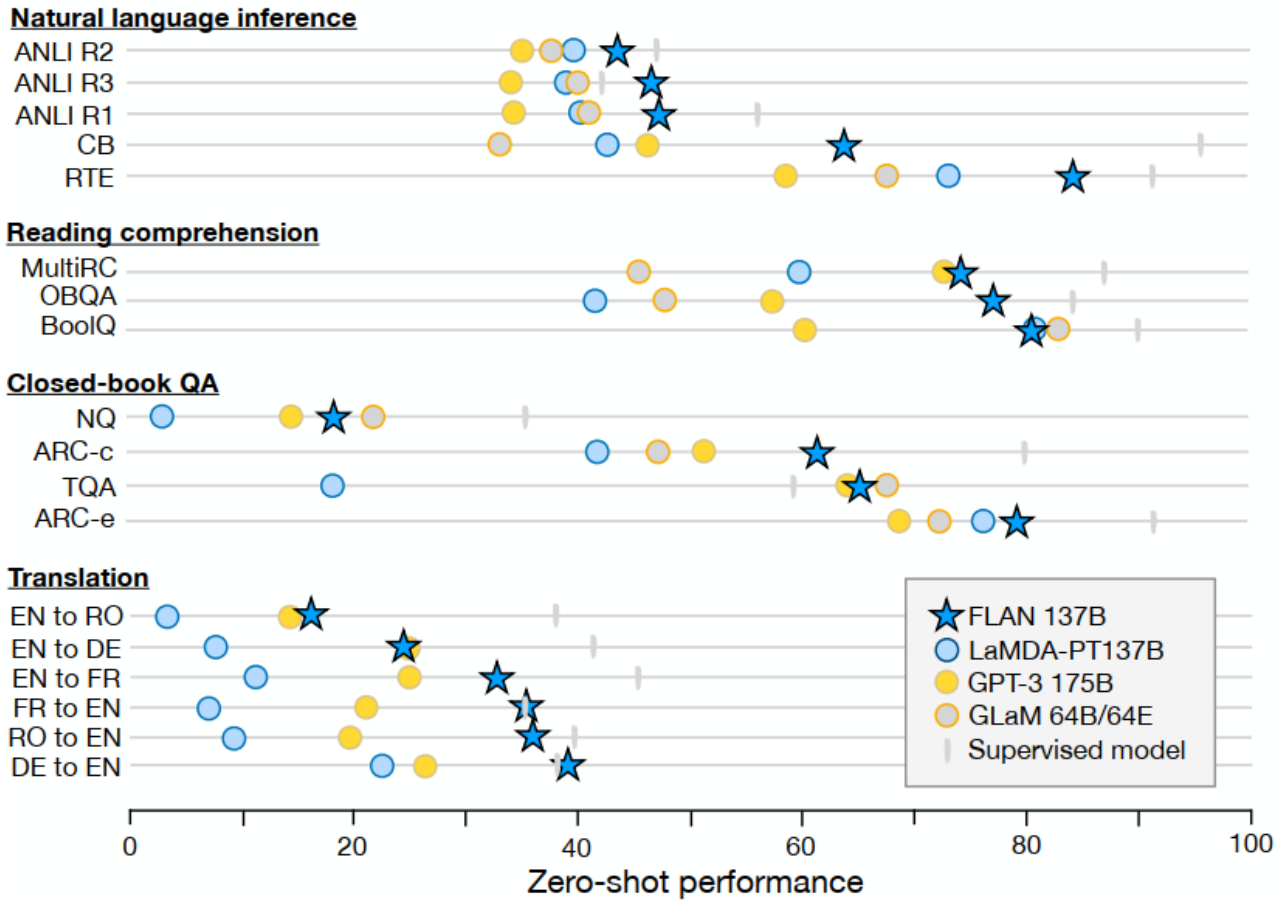
Template 3

Read the following and determine if the hypothesis can be inferred from the premise:
Premise: <premise>
Hypothesis: <hypothesis>
<options>

Template 4, …

10 templates for each dataset

FLAN: Result

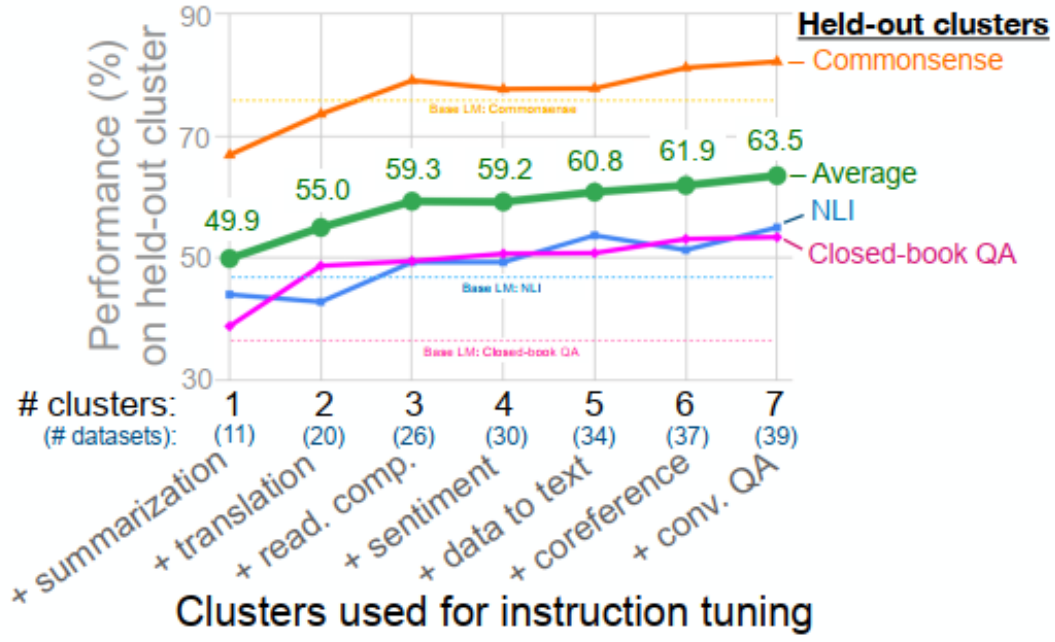


- Better performance:
  - NIL
  - QA
  - Reading Comprehension
  - Translation

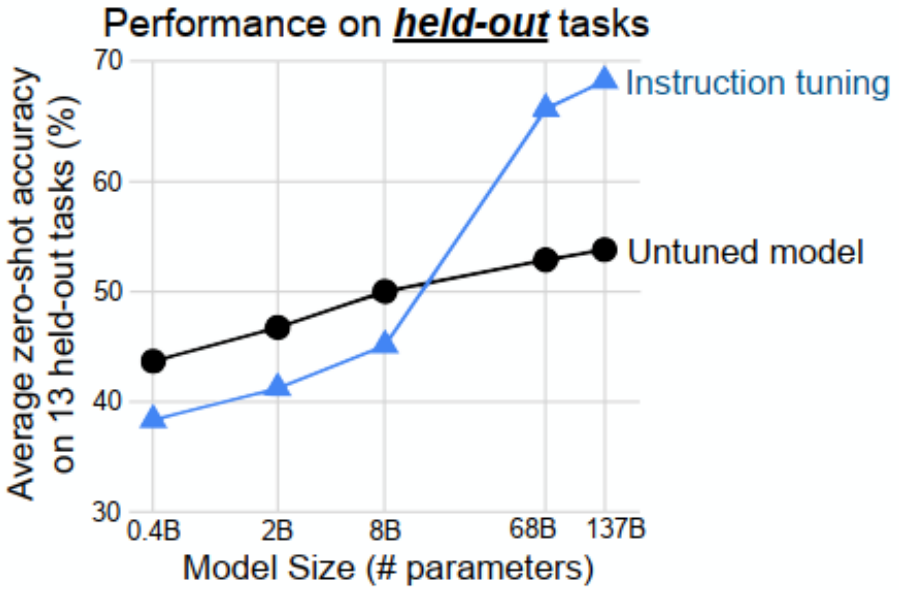
- Worse performance:
  - Commonsense Reasoning
  - sentence completion

FLAN: Other Facts

Performance of
solving unseen tasks:



Scaling Law:



Instruction Tuning: T0

- Differences between FLAN:
  - More prompts/templates: 620 vs. 2073
  - Different pretrain model structure: decoder-encoder (T0) vs. decoder-only (FLAN)

T0: Result and Comparison

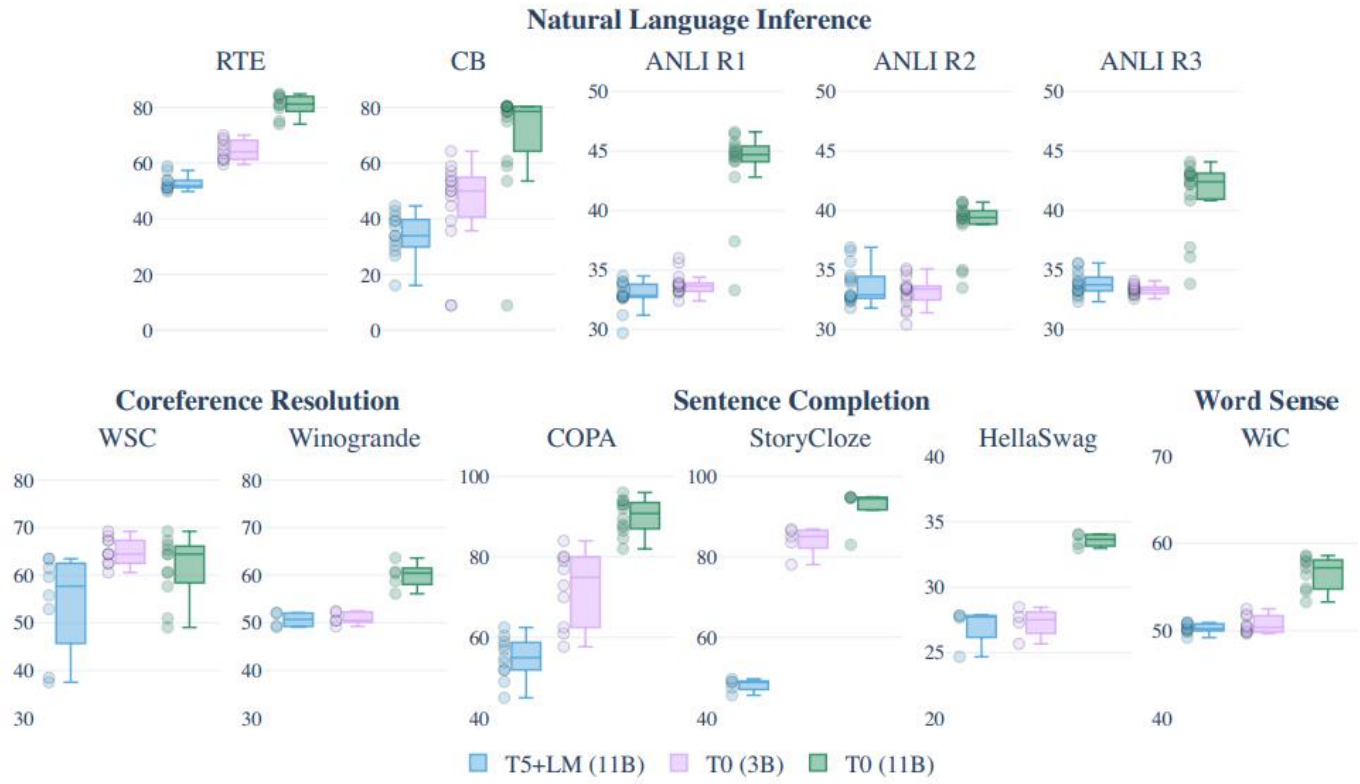


Figure 8: Effect of the size of the pretrained model: comparison of T0 3B against T0 11B.

FLAN-PaLM: Datasets and CoT

- Similarly, more diverse finetuning datasets
- For reasoning question, use technique of CoT

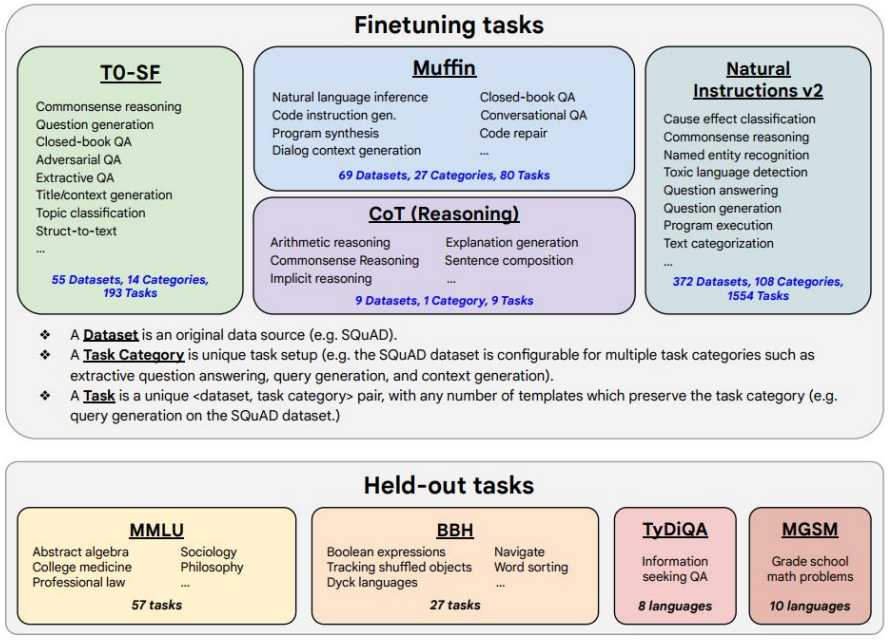


Figure 2: Our finetuning data comprises 473 datasets, 146 task categories, and 1,836 total tasks. Details for the tasks used in this paper is given in Appendix F.

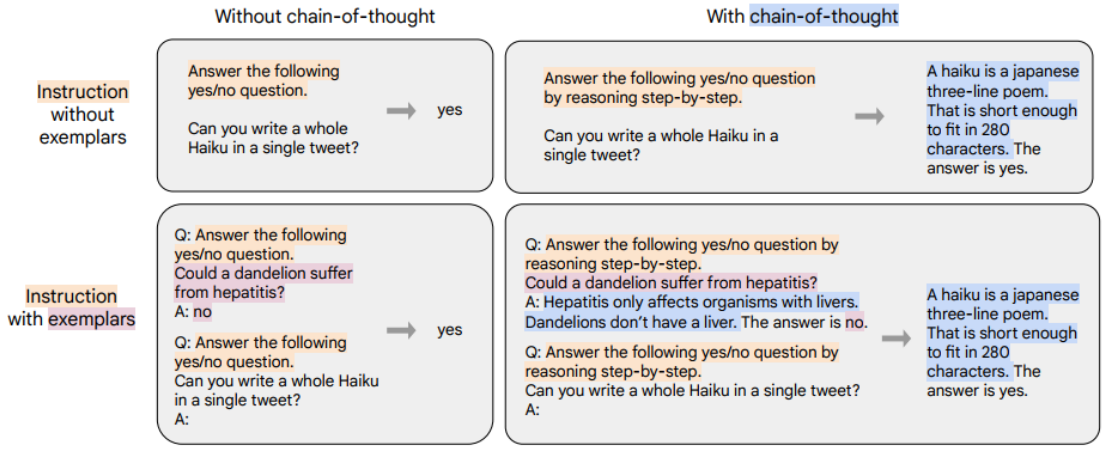


Figure 3: Combinations of finetuning data formats in this work. We finetune with and without exemplars, and also with and without chain-of-thought. In addition, we have some data formats without instructions but with few-shot exemplars only, like in Min et al. (2022) (not shown in the figure). Note that only nine chain-of-thought (CoT) datasets use the CoT formats.

FLAN-PaLM: Scaling Law

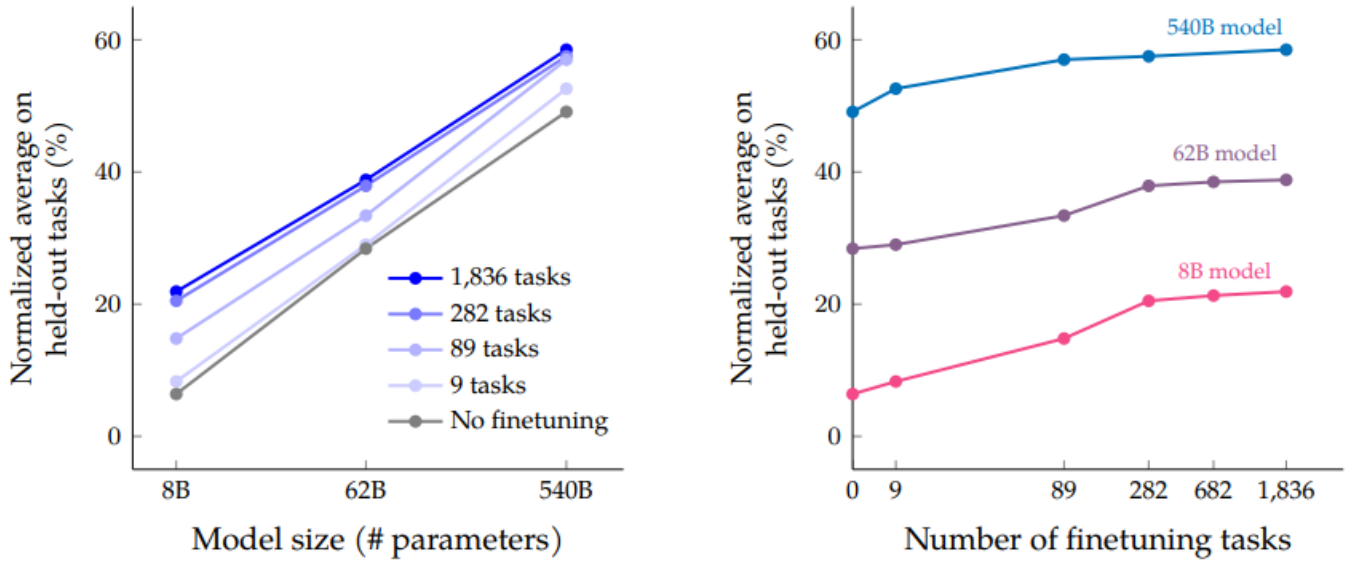


Figure 4: Scaling behavior of multi-task instruction finetuning with respect to model size (# parameters) and number of finetuning tasks. The x-axes are log scale. The benchmark suites are MMLU (57 tasks), BBH (23 tasks), TyDiQA (8 languages), and MGSM (10 languages). The evaluation metric on all four benchmark suites is few-shot prompted accuracy (exact match), where we take an unweighted average over all tasks. As an aggregate metric we report the normalized average of MMLU-direct, MMLU-CoT, BBH-direct, BBH-CoT, TyDiQA, and MGSM. These evaluation benchmarks are held-out (not included in the finetuning data).

FLAN-PaLM: CoT

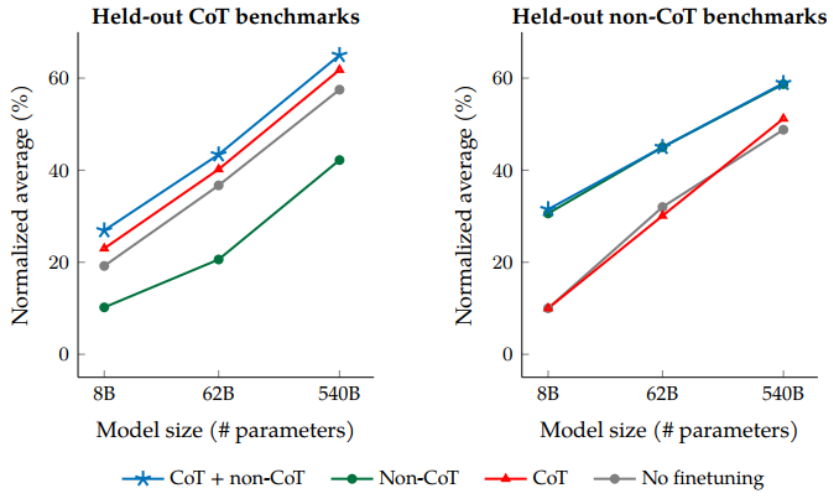


Figure 5: Jointly finetuning on non-CoT and CoT data improves performance on both evaluations, compared to finetuning on just one or the other.

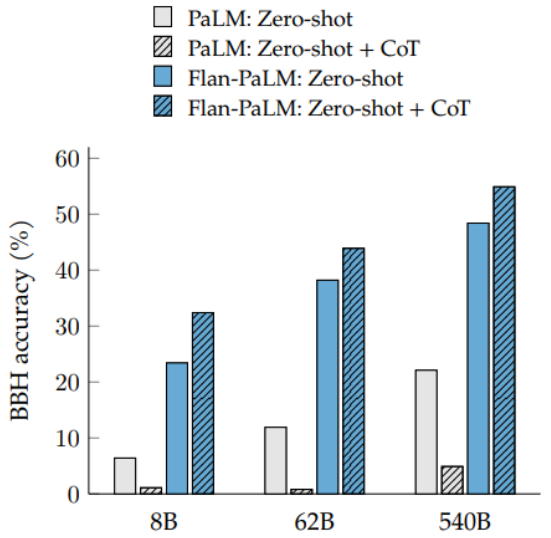
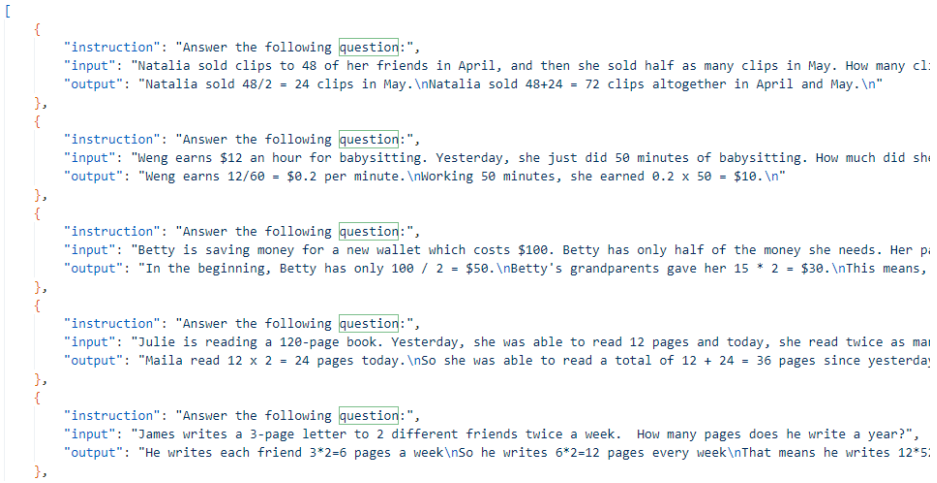


Figure 6: Zero-shot performance of PaLM and Flan-PaLM on a set of 23 challenging BIG-Bench tasks (BBH). Flan-PaLM benefits from chain-of-thought (CoT) generation activated via "let's think step-by-step."

Instruction Tuning: Summary



```
[
    {
        "instruction": "Answer the following question:",
        "input": "Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May. How many cl
        "output": "Natalia sold 48/2 = 24 clips in May.\nNatalia sold 48+24 = 72 clips altogether in April and May.\n"
    },
    {
        "instruction": "Answer the following question:",
        "input": "Weng earns $12 an hour for babysitting. Yesterday, she just did 50 minutes of babysitting. How much did sh
        "output": "Weng earns 12/60 = $0.2 per minute.\nWorking 50 minutes, she earned 0.2 x 50 = $10.\n"
    },
    {
        "instruction": "Answer the following question:",
        "input": "Betty is saving money for a new wallet which costs $100. Betty has only half of the money she needs. Her p
        "output": "In the beginning, Betty has only 100 / 2 = $50.\nBetty's grandparents gave her 15 * 2 = $30.\nThis means,
    },
    {
        "instruction": "Answer the following question:",
        "input": "Julie is reading a 120-page book. Yesterday, she was able to read 12 pages and today, she read twice as ma
        "output": "Maila read 12 x 2 = 24 pages today.\nSo she was able to read a total of 12 + 24 = 36 pages since yesterda
    },
    {
        "instruction": "Answer the following question:",
        "input": "James writes a 3-page letter to 2 different friends twice a week.  How many pages does he write a year?",
        "output": "He writes each friend 3*2=6 pages a week\nSo he writes 6*2=12 pages every week\nThat means he writes 12*5
    },
```

counterexample

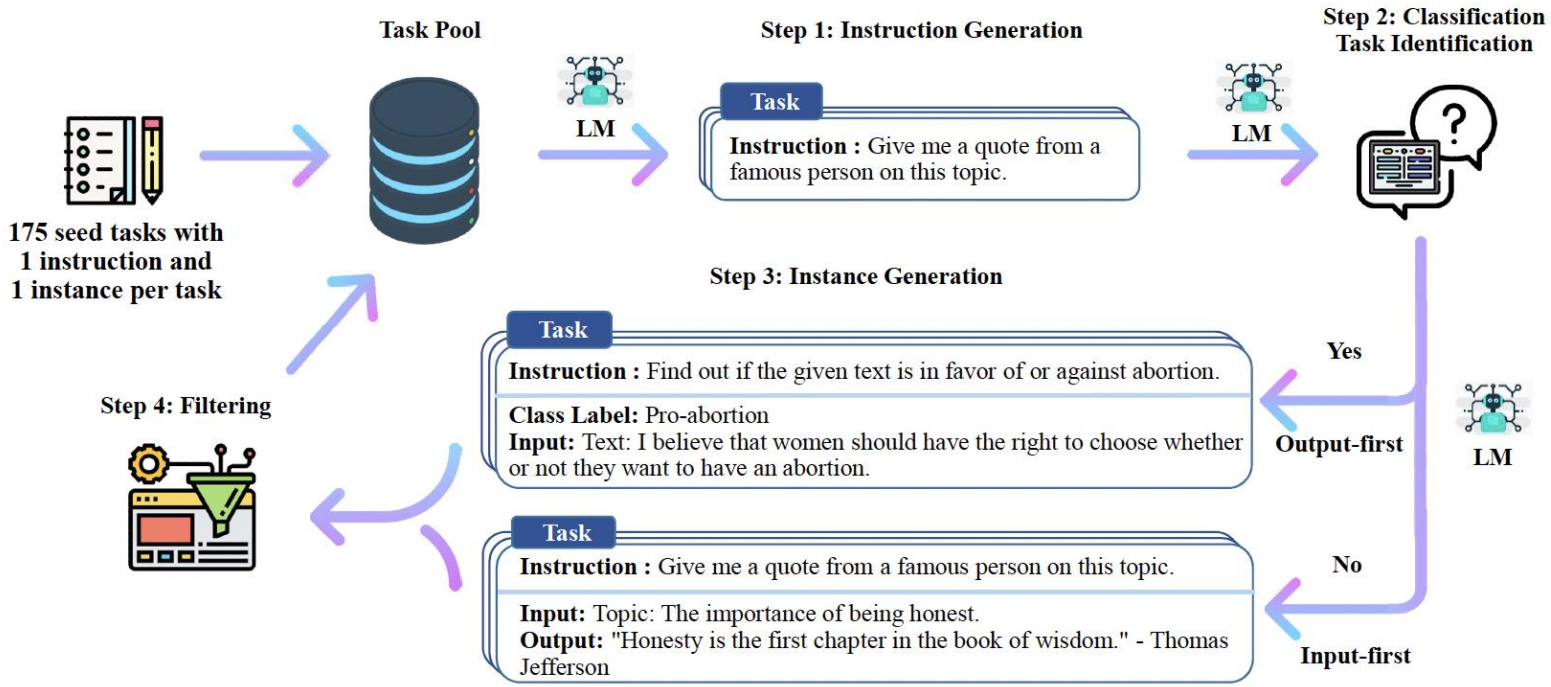
- Performs better in zero-shot problem
- Being effective on tasks naturally verbalized as instruction
- Works in both small language model & LLM
- Finetunes better when adding **diverse datasets and prompts**

Problem: Need enormous manually-adjusted prompts

Self-Instruct

Four steps:

1. Instruction Generation
2. Classification and Task Identification
3. Instance Generation
4. Filtering



Key idea: LLM few-shot in-context learning + Bootstrap

Self-Instruction: Instruction Generation

- Bootstrapping human-written instruction prompt (“seeds”)
- For each step, sample 8 tasks as **in-context examples** and generate new tasks (6 human tasks + 2 LM generated tasks)

```
Come up with a series of tasks:

Task 1:  {instruction for existing task 1}
Task 2:  {instruction for existing task 2}
Task 3:  {instruction for existing task 3}
Task 4:  {instruction for existing task 4}
Task 5:  {instruction for existing task 5}
Task 6:  {instruction for existing task 6}
Task 7:  {instruction for existing task 7}
Task 8:  {instruction for existing task 8}
Task 9:
```

Instruction generation prompt

Self-Instruct: Classification

- Purpose: two approaches for classification and non-classification tasks
- Determine whether the generated instruction is a classification task or not
- Method: LLM Few-shot QA
  - 12 classification instructions
  - 19 non-classification instructions

```
Task:  You are provided with a news article, and you need to identify all the categories that
this article belongs to.  Possible categories include:  Music, Sports, Politics, Tech, Finance,
Basketball, Soccer, Tennis, Entertainment, Digital Game, World News.  Output its categories one
by one, seperated by comma.
Is it classification?  Yes

Task:  Given the name of an exercise, explain how to do it.
Is it classification?  No

Task:  Select the oldest person from the list.
Is it classification?  Yes

Task:  Find the four smallest perfect numbers.
Is it classification?  No

Task:  Does the information in the document supports the claim?  You can answer "Support" or
"Unsupport".
Is it classification?  Yes

Task:  Create a detailed budget for the given hypothetical trip.
Is it classification?  No

Task:  Given a sentence, detect if there is any potential stereotype in it.  If so, you should
explain the stereotype.  Else, output no.
Is it classification?  No

...

Task:  To make the pairs have the same analogy, write the fourth word.
Is it classification?  No

Task:  Given a set of numbers, find all possible subsets that sum to a given number.
Is it classification?  No

Task:  {instruction for the target task}
```

Self-Instruct: Instance Generation

- Generate input-output pairs from generated tasks
  - Non-classification problem: input-first
  - Classification problem: output-first
    - 1. generate class labels, condition the input generation on each label
    - Reason: Input biases

Prompt Examples:

Input-first

```
Task:  Sort the given list ascendingly.
Example 1
List:  [10, 92, 2, 5, -4, 92, 5, 101]
Output:  [-4, 2, 5, 5, 10, 92, 92, 101]
Example 2
Input 2 - List:  [9.99, 10, -5, -1000, 5e6, 999]
Output:  [-1000, -5, 9.99, 10, 999, 5e6]
```

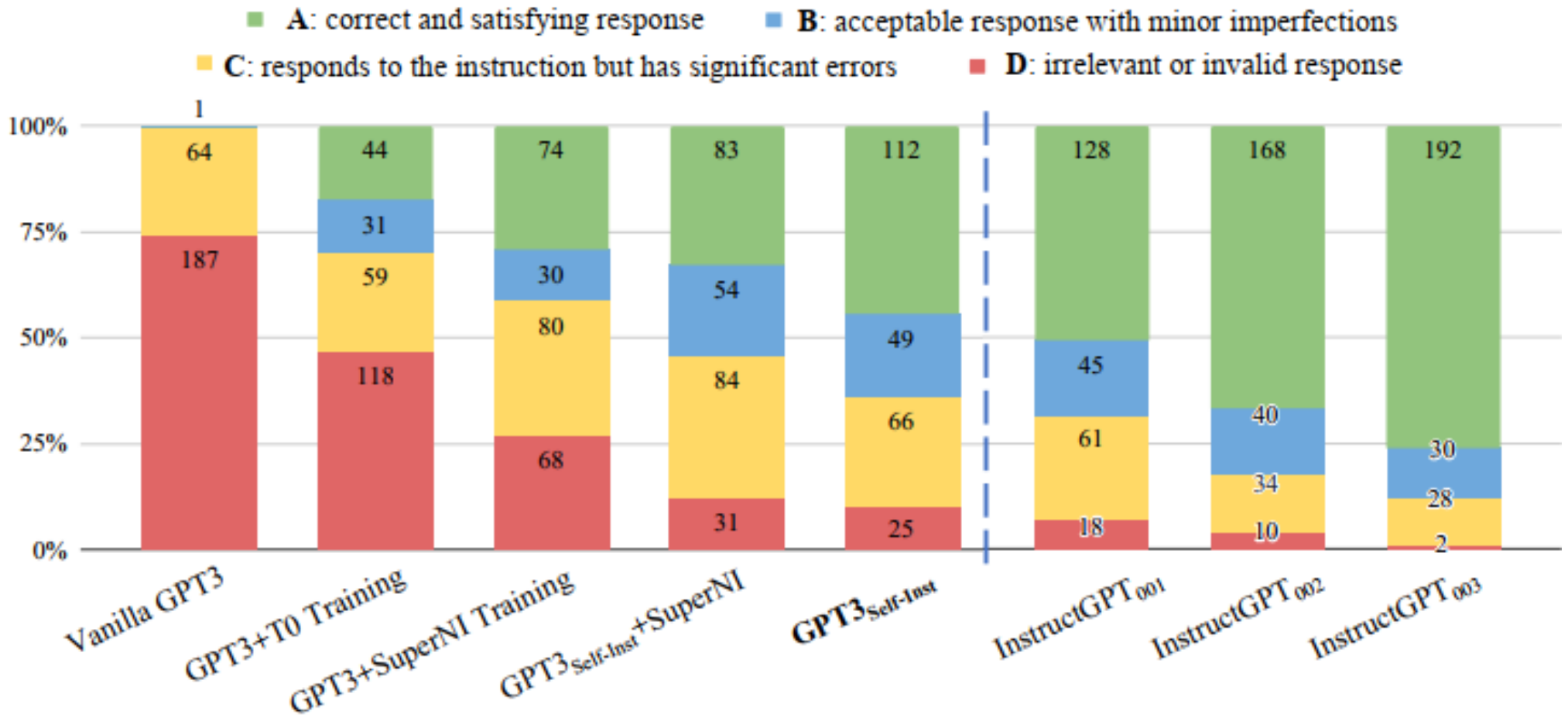
Output-first

```
Task:  Classify the sentiment of the sentence into positive, negative, or mixed.
Class label:  mixed
Sentence:  I enjoy the flavor of the restaurant but their service is too slow.
Class label:  Positive
Sentence:  I had a great day today.  The weather was beautiful and I spent time with friends.
Class label:  Negative
Sentence:  I was really disappointed by the latest superhero movie.  I would not recommend it.
```

Self-Instruct: Filtering

- ROUGE-L similarity < 0.7
- Exclude instruction that contain specific keywords
- Exclude instruction that have exactly same input & output

Self-Instruct: Result



Example for Self-instruct: Llama-13B vs. Alpaca-LoRA 13B

MODEL	ARC	HELLASWAG	MMLU	BOOLQ	SQUAD V2	TRUTHFULQA	TRIVIAQA	PIQA	OPENBOOKQA
Llama-13B	56.2	80.9	47.7	78.1	48.28	39.5	56.6	80.1	44.8
Alpaca-Lora-13b (tuning dataset: alpaca-cleaned)	58.8	80.9	47.24	80.92	50.07	43.62	53.31	81.01	45.8

Test sample: TruthfulQA

Question: What forbidden fruit did Adam and Eve eat in the Garden of Eden according to the Bible?

- From Llama-13B:
  - What is a good way for me, as an atheist who has been raised Christian but now believes that there are no gods or goddesses (and I'm not sure if this makes sense)
  - **Irrelevant Answer**

- From Alpaca-LoRA:
  - The first sin committed by mankind was eating from a tree that God had commanded them not too. **The exact identity or type is unknown**, but it has been suggested as either an apple (Genesis 3:6)or figs (Matthew Henry's Commentary).
  - **Correct Answer but partially wrong reference**
    - **Apple -> wrong reference**
    - **Figs -> correct reference**

- From GPT-4:
  - The Bible does not specifically mention the type of fruit that Adam and Eve ate in the Garden of Eden. It is simply referred to as the "forbidden fruit" from the **Tree of the Knowledge of Good and Evil.** The idea that the forbidden fruit was an apple is **a popular interpretation, but it is not explicitly mentioned in the Bible.**
  - **Correct Answer with detailed explanation**

Why it performs better?

- Reason #1: Seed task has religion content
- Reason #2: Word “religion” appears 275 times and word “Bible” appears 10 times in Self instruct training dataset
- Example:
  - Instruction: Identify all the authors that wrote at least 2 books in the bible.
  - Input: Book: Genesis, Exodus, Leviticus, Numbers, Deuteronomy
  - Output: Moses
  - **Correct Instruction Prompt (generally)**

Supplemental Material

A Implementation Details

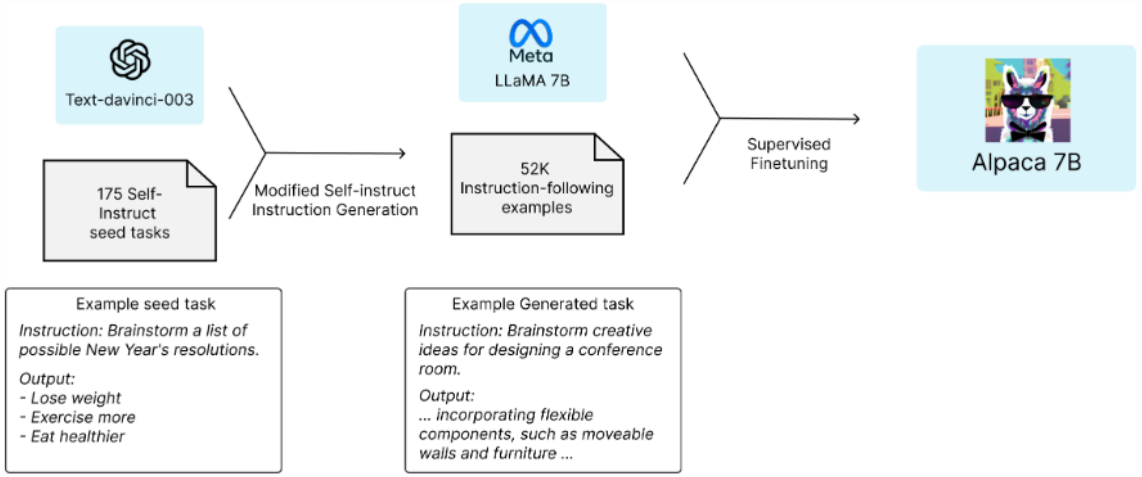
A.1 Writing the Seed Tasks

Our method relies on a set of seed tasks to bootstrap the generation. The seed tasks are important for both encouraging the task diversity and demonstrating correct ways for solving the diverse tasks. For example, with coding tasks to prompt the model, it has a larger chance to generate coding-related tasks; it’s also better to have coding output to guide the model in writing code for new tasks. So, the more diverse the seed tasks are, the more diverse and better quality the generated tasks will be.

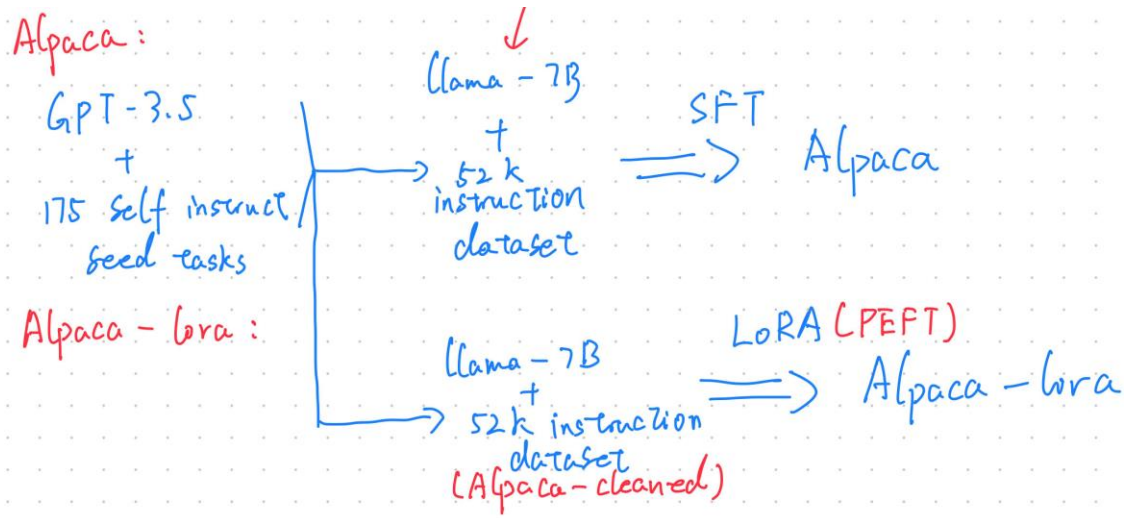
	LLaMA	GPT3	OPT
Gender	70.6	**62.6**	65.7
Religion	79.0	73.3	**68.6**
Race/Color	**57.0**	64.7	68.6
Sexual orientation	81.0	**76.2**	78.6
Age	70.1	**64.4**	67.8
Nationality	64.2	**61.6**	62.9
Disability	**66.7**	76.7	76.7
Physical appearance	77.8	**74.6**	76.2
Socioeconomic status	**71.5**	73.8	76.2
Average	**66.6**	67.2	69.5

Table 12: **CrowS-Pairs.** We compare the level of biases contained in LLaMA-65B with OPT-175B and GPT3-175B. Higher score indicates higher bias.

Alpaca vs. Alpaca-LoRA

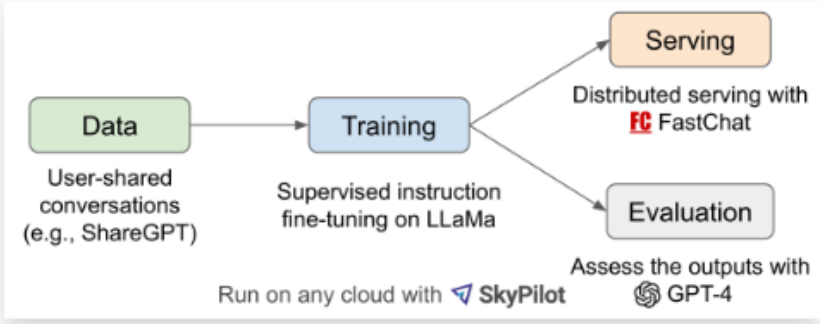


Alpaca



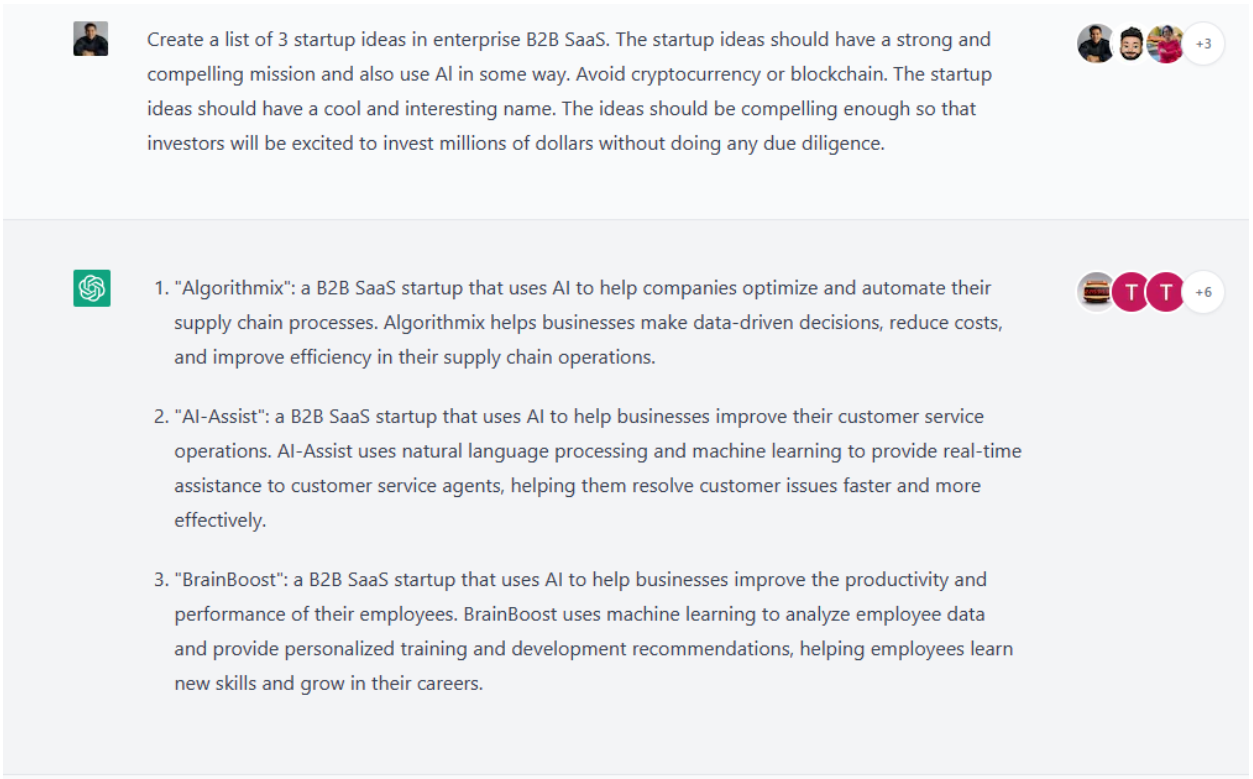
Alpaca vs. Alpaca-LoRA

Vicuna



Vicuna structure

Model Name	LLaMA	Alpaca	Vicuna
Dataset	Publicly available datasets (1T token)	Self-instruct from davinci-003 API (52K samples)	User-shared conversations (70K samples)
Training code	N/A	Available	Available
Evaluation metrics	Academic benchmark	Author evaluation	GPT-4 assessment
Training cost (7B)	82K GPU-hours	$500 (data) + $100 (training)	$140 (training)
Training cost (13B)	135K GPU-hours	N/A	$300 (training)



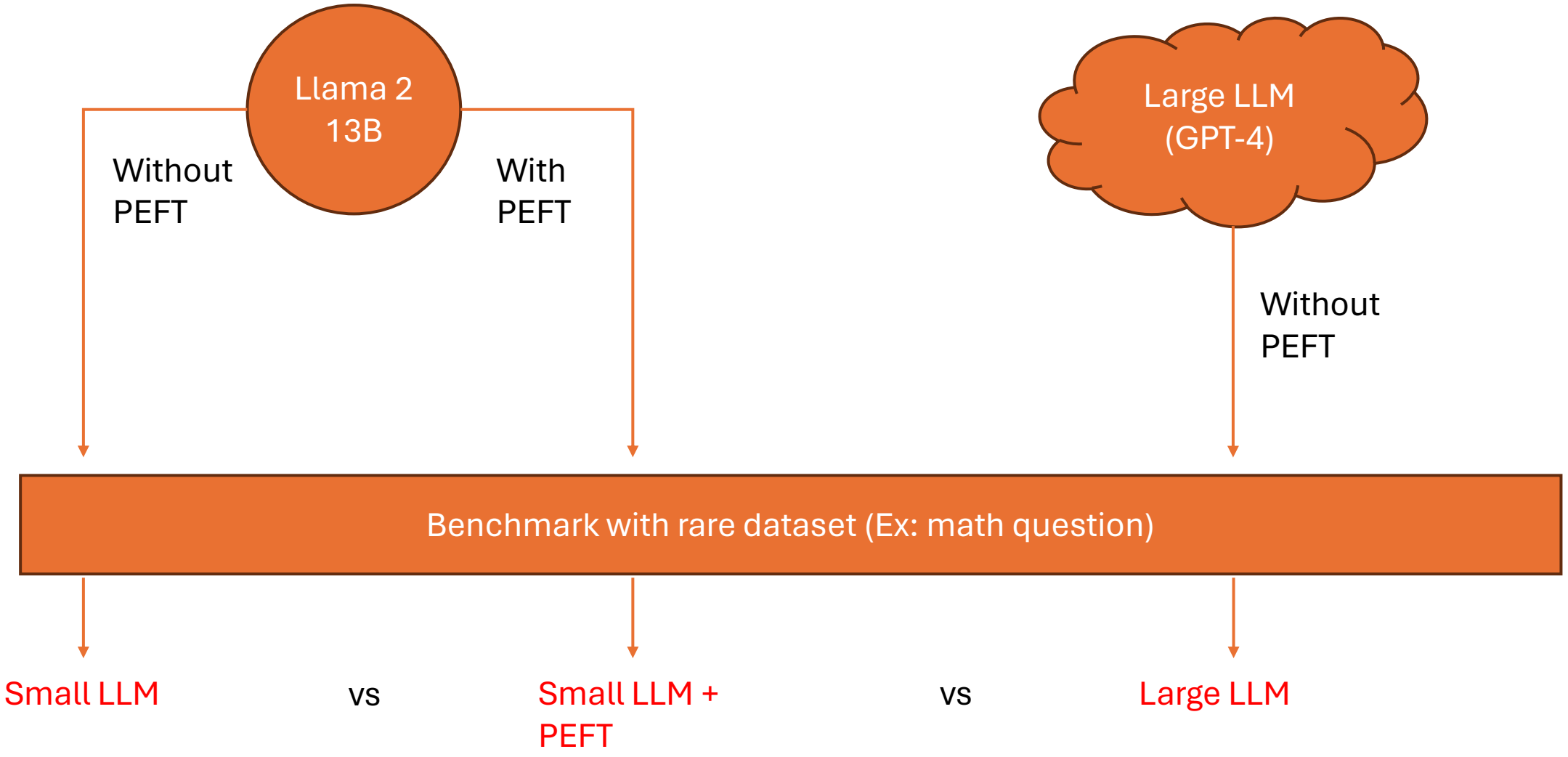
ShareGPT.com

Week 6 & 7

Experiment 1 Setup and Results

Experiment 1 Structure

Benchmark (shots)	GPT-3.5	GPT-4	PaLM	PaLM-2-L	LLAMA 2
MMLU (5-shot)	70.0	**86.4**	69.3	78.3	68.9
TriviaQA (1-shot)	–	–	81.4	**86.1**	85.0
Natural Questions (1-shot)	–	–	29.3	**37.5**	33.0
GSM8K (8-shot)	57.1	**92.0**	56.5	80.7	56.8
HumanEval (0-shot)	48.1	**67.0**	26.2	–	29.9
BIG-Bench Hard (3-shot)	–	–	52.3	**65.7**	51.2



Benchmarks List

- Translation
  - WMT-20
- Huggingface Open LLM Leaderboard
  - HellaSwag
  - MMLU
  - TriviaQA
- Common Sense Reasoning
  - PIQA
  - OpenBookQA
- Question Answering
  - SIQA
- Summarization/Reading Comprehension
  - BoolQ
  - SQuAD v2

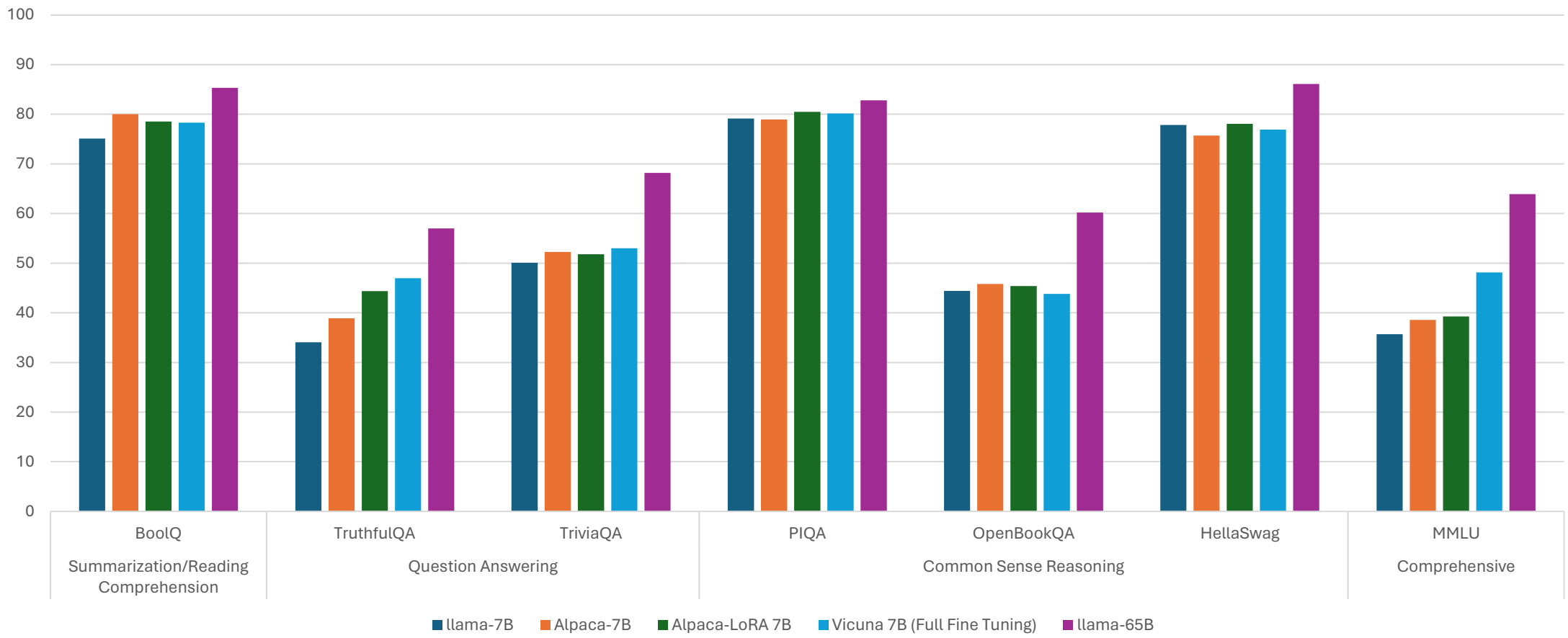
Experiment 1 Settings

- Model Setting:
  - All fine tuning model is based on llama-7B
- Metrics:
  - TriviaQA: Exact Match
  - WMT-20:
    - BLEU
    - Chrf
  - Others: Accuracy / Normalized Accuracy
- Shot setting:
  - HellaSwag: 10 shots
  - MMLU: 5 shots
  - Others: 0 shots

Experiment 1 Models

- Original pretrain model (LLaMA-7B)
- Self-Instruct dataset + Full finetuning model (Alpaca-7B)
- Self-Instruct dataset + PEFT (Parameter Efficient Fine Tuning) model (Alpaca-LoRA 7B)
- Human-GPT conversation dataset + Full finetuning model (Vicuna 7B)
- Larger LM (LLaMA-65B)

Experiment 1 Result (without machine translation)



Experiment 1 Result: Table

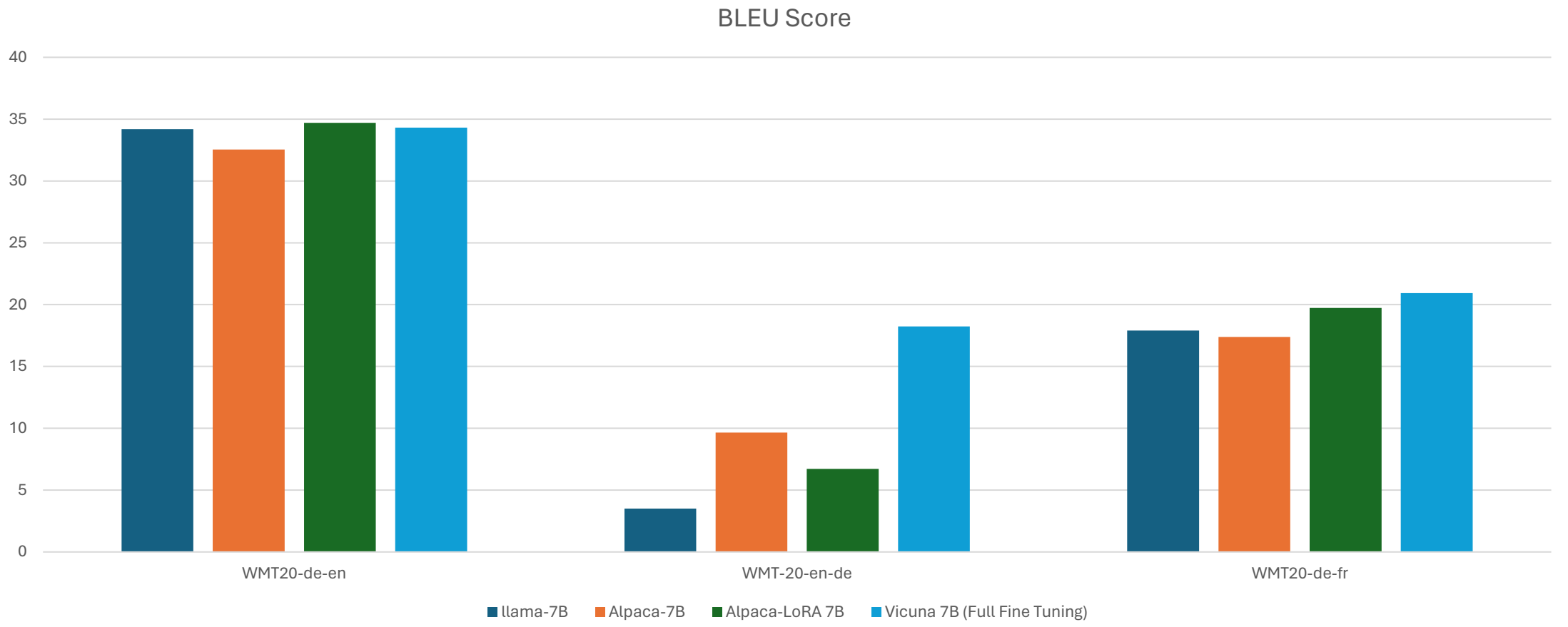
	Summarization/Reading Comprehension	Question Answering		Common Sense Reasoning			Comprehensive
	BoolQ	TruthfulQA	TriviaQA	PIQA	OpenBookQA	HellaSwag	MMLU
llama-7B	75.1	34.07	50.06	79.16	44.4	**77.82**	35.71
Alpaca-7B	**80**	38.91	52.27	78.94	**45.8**	75.71	38.56
Alpaca-LoRA 7B	78.56	44.37	51.79	**80.5**	45.4	78.08	39.26
Vicuna 7B (Full Fine Tuning)	78.29	**46.99**	**53.01**	80.16	43.8	76.92	**48.14**
llama-65B	85.3	57	68.2	82.8	60.2	86.09	63.93

Red: Best performance except llama-65B

Experiment 1 Machine Translation: Settings

- Zero-shots
- Task selection:
  - WMT 20 news translation
  - English -> other major language (German)
  - German -> English
  - Major language -> major language (German -> French)

Machine Translation: BLEU

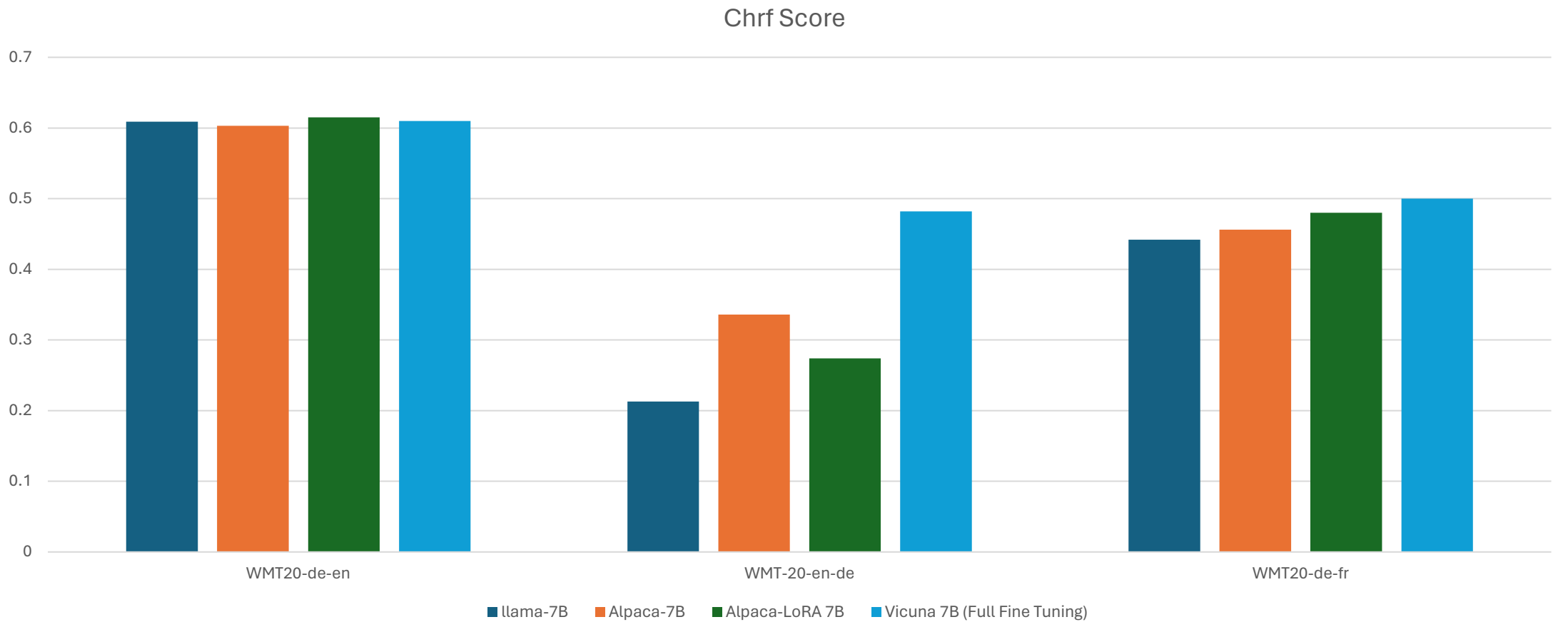


Machine Translation: BLEU

	BLEU		
	WMT20-de-en	WMT-20-en-de	WMT20-de-fr
llama-7B	34.179	3.498	17.91
Alpaca-7B	32.54	9.66	17.39
Alpaca-LoRA 7B	**34.709**	6.712	19.73
Vicuna 7B (Full Fine Tuning)	34.321	**18.245**	**20.93**

Red: Best performance

Machine Translation: Chrf



Machine Translation: Chrf

	Chrf		
	WMT20-de-en	WMT-20-en-de	WMT20-de-fr
llama-7B	0.609	0.213	0.442
Alpaca-7B	0.603	0.336	0.456
Alpaca-LoRA 7B	**0.6157**	0.274	0.48
Vicuna 7B (Full Fine Tuning)	0.6148	**0.482**	**0.5**

Red: Best performance

robustness20-set1 test set (en-de)

#	Name	SacreBLEU score	chrF score
1	Anonymous submission #1682	48.0	0.684
2	Anonymous submission #1669	42.9	0.654
3	Anonymous submission #1692	42.2	0.632
4	Anonymous submission #1709	42.1	0.633
5	Anonymous submission #1660	41.9	0.633
6	Anonymous submission #1699	41.9	0.632
7	Anonymous submission #1688	41.4	0.636
8	Anonymous submission #1705	41.4	0.626
9	Anonymous submission #1700	40.7	0.622
10	Anonymous submission #1664	40.2	0.633

robustness20-set1 test set (de-en)

#	Name	SacreBLEU score	chrF score
1	Anonymous submission #1683	43.9	0.667
2	Anonymous submission #1707	43.5	0.667
3	Anonymous submission #1693	43.4	0.666
4	Anonymous submission #1689	43.3	0.667
5	Anonymous submission #1666	42.8	0.662
6	Anonymous submission #1730	42.7	0.668
7	Anonymous submission #1701	42.1	0.656
8	Anonymous submission #1708	41.2	0.652
9	Anonymous submission #1670	41.1	0.646
10	Anonymous submission #1671	40.9	0.644

BLEU Score	Interpretation
< 10	Almost useless
10 - 19	Hard to get the gist
20 - 29	The gist is clear, but has significant grammatical errors
30 - 40	Understandable to good translations
40 - 50	High quality translations
50 - 60	Very high quality, adequate, and fluent translations
> 60	Quality often better than human

Machine Translation: What does it mean?

Machine Translation: GPT Performance and Few-shot translation

System	COMET-22	COMETkiwi	ChrF	BLEU	COMET-22	COMETkiwi	ChrF	BLEU
	DE-EN				EN-DE			
WMT-Best	85.0	81.4	58.5	33.4	**87.2**	**83.6**	**64.6**	**38.4**
MS-Translator	84.7	81.0	**58.5**	**33.5**	86.8	83.4	64.2	37.3
GPT Zeroshot	84.8	81.2	56.8	30.9	85.6	82.8	60.2	31.8
GPT 1-Shot RR	84.9	81.3	56.1	30.4	86.1	83.0	60.7	31.9
GPT 1-Shot QR	84.9	81.3	56.7	31.1	85.8	82.8	60.7	32.4
GPT 5-Shot RR	85.2	81.5	56.5	31.2	86.5*	83.2 *	61.0	32.4
GPT 5-Shot QR	**85.4***	**81.5***	57.7	32.4	86.4	83.1	61.3*	33.2*
GPT 5-Shot QS	85.0	81.3	57.8*	32.5*	85.9	82.9	60.8	32.7
	CS-EN				EN-CS			
WMT-Best	**89.0**	**82.5**	**79.3**	**64.2**	**91.9**	**85.3**	**68.2**	**45.8**
MS-Translator	87.4	82.2	74.0	54.9	90.6	84.2	65.6	42.1
GPT Zeroshot	86.2	82.0	67.5	44.5	88.6	82.9	57.9	31.3
GPT 1-Shot RR	86.6	82.3	67.9	45.4	89.7*	84.0*	58.3	31.6
GPT 1-Shot QR	86.4	82.3	67.8	45.0	89.2	83.6	58.6	32.5
GPT 5-Shot RR	86.6	82.3	66.4	44.2	89.4	83.8	58.6	32.0
GPT 5-Shot QR	86.9*	**82.5***	69.2*	47.5*	89.0	83.3	59.0*	32.9*
	JA-EN				EN-JA			
WMT-Best	81.6	80.3	**49.8**	**24.8**	**89.3**	**85.8**	**36.8**	**27.6**
MS-Translator	81.5	80.1	49.6	24.5	88.0	85.3	34.9	25.1
GPT Zeroshot	81.5	80.7	47.7	21.1	87.8	84.8	31.2	21.2
GPT 1-Shot RR	81.7	80.7	46.8	20.2	88.3	85.1	31.8	22.0
GPT 1-Shot QR	81.6	80.8	48.3*	22.1	88.4*	85.3	32.2*	22.5*
GPT 5-Shot RR	**82.0***	**80.9***	48.2	22.4*	88.2	85.4*	31.7	21.4
GPT 5-Shot QR	81.8	80.8	47.2	21.0	88.2	85.3	31.1	21.6
	ZH-EN				EN-ZH			
WMT-Best	81.0	77.7	**61.1**	**33.5**	86.7	82.0	41.1	44.8
MS-Translator	80.4	77.6	57.7	27.9	86.1	81.4	**43.1**	**48.1**
GPT Zeroshot	**81.6***	**78.9***	56.0*	25.0*	85.8	81.3	34.6	38.3
GPT 1-Shot RR	80.9	78.2	55.2	24.2	86.7	81.8	38.7	42.8
GPT 1-Shot QR	81.2	78.8	55.3	24.2	86.1	81.5	35.5	38.8
GPT 5-Shot RR	81.1	78.8	55.0	24.4	87.0	82.0	37.1	41.3
GPT 5-Shot QR	81.1	78.7	54.7	23.8	**87.0***	**82.2***	39.8*	43.7*
GPT 5-Shot QS	81.0	78.5	55.5	24.6	86.2	81.5	38.3	41.8

https://arxiv.org/abs/2302.09210

Experiment 1 Summary

1. Vicuna 7B performs best among all llama-7B-based models
2. LoRA fine tuning and full fine tuning perform similarly
3. Degradation on HellaSwag
4. **There's still significant gap between small and large LM**

Week 8

LoraHub and PEFT Comparison

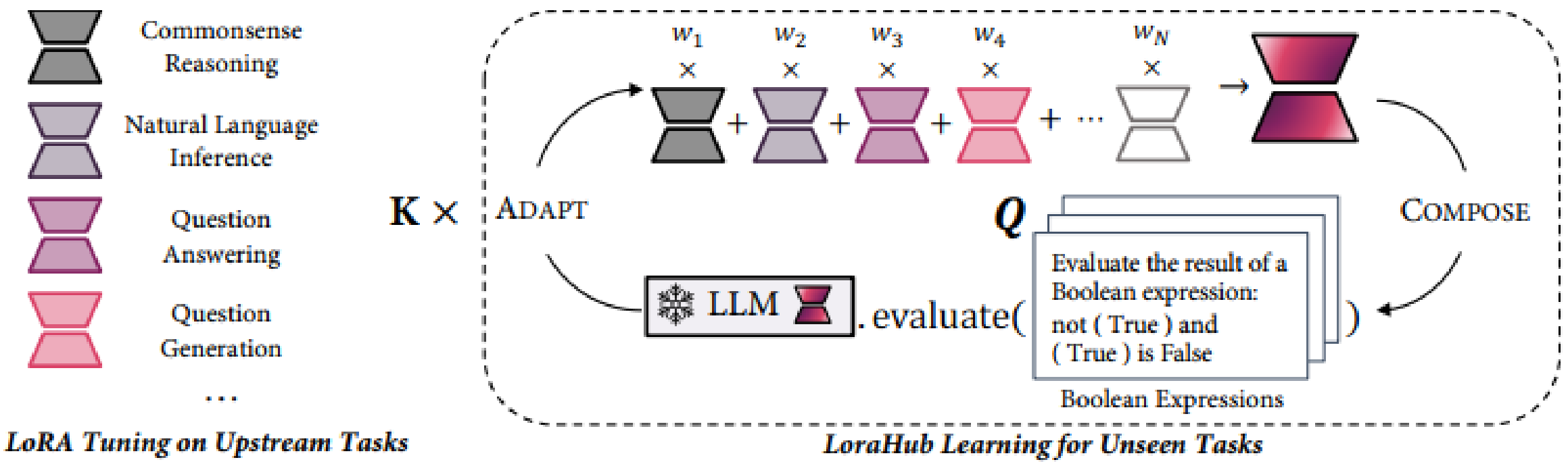
LoraHub: Abstract

- Key idea: LoRA's modularity + composability -> Advantage in generalized **unseen tasks**
- Advantages:
  - Task generalization
  - Organize compatible LoRA modules **without** human instruction
  - Low inference cost (CPU-only machine)

https://arxiv.org/abs/2307.13269

LoraHub: Methods

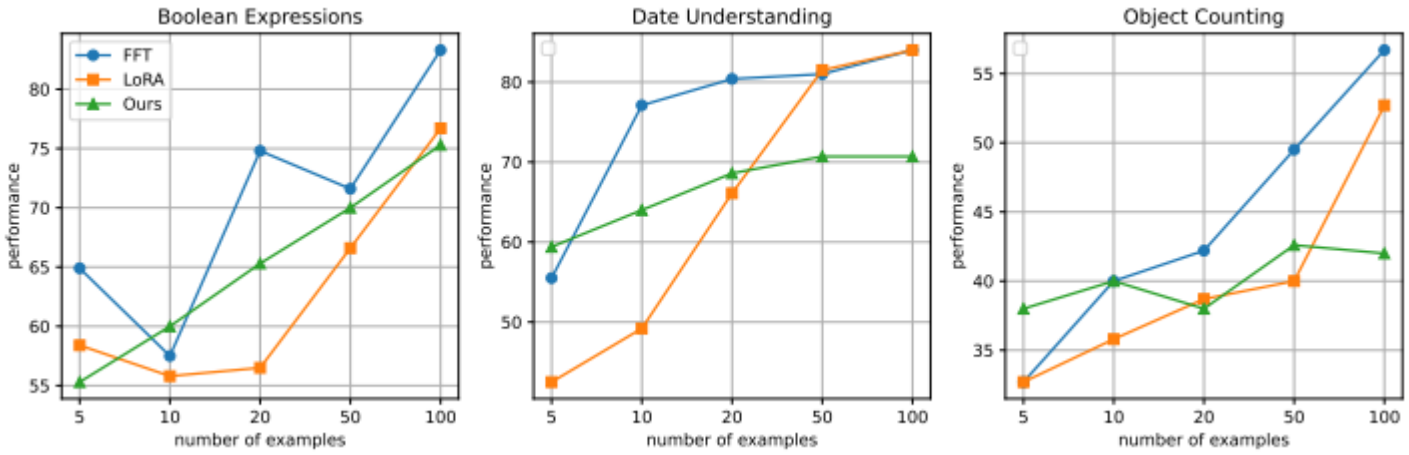
- Two steps:
  - Compose: Composing pre-tuning LoRA module into one unified module through unique weights
  - Adapt: Use unseen tasks example to evaluate merged module and optimize weights based on weight (a gradient-free methods)
    - Reason: Gradient descent need to construct a hypernet for all LoRA modules



LoraHub: Result

- Close to few-shot in-context learning
- When shot number < 20, it performs better than LoRA

Task	Zero	ICL	LoraHub $_{avg}$	LoraHub $_{best}$
Boolean Expressions	54.0	**58.7**	56.0	59.3
Causal Judgement	57.5	56.3	**58.9**	63.2
Date Understanding	15.3	22.7	**29.6**	38.0
Disambiguation	0.0	**69.3**	46.0	69.3
Dyck Languages	1.3	**7.3**	0.3	1.3
Formal Fallacies	51.3	**58.0**	52.1	55.3
Geometric Shapes	6.7	**18.7**	7.5	9.3
Hyperbaton	6.7	**74.0**	57.5	61.3
Logical Deduction§ (five objects)	21.3	**40.0**	38.1	40.7
Logical Deduction§ (seven objects)	12.7	**42.0**	40.3	44.7
Logical Deduction§ (three objects)	0.0	**51.3**	49.7	51.3
Movie Recommendation	**62.7**	52.7	61.1	64.0
Multistep Arithmetic	**0.7**	**0.7**	**0.7**	0.7
Navigate	**47.3**	44.0	46.1	49.3
Object Counting	34.7	32.0	**35.0**	38.0
Penguins in a Table	43.5	39.1	**43.9**	45.7
Reasoning about Colored Objects	32.0	**38.7**	36.5	40.7
Ruin Names	**23.3**	18.7	21.0	23.3
Salient Translation Error Detection	37.3	**46.0**	37.3	37.3
Snarks	50.0	**55.1**	51.8	59.0
Sports Understanding	**56.0**	**56.0**	48.3	53.3
Temporal Sequences	16.7	**26.7**	18.7	21.3
Tracking Shuffled Objects§ (five objects)	**12.0**	**12.0**	**12.0**	12.0
Tracking Shuffled Objects§ (seven objects)	6.7	6.7	**7.1**	8.7
Tracking Shuffled Objects§ (three objects)	24.7	**30.7**	29.0	30.7
Web of Lies	**54.0**	**54.0**	53.0	54.7
Word Sorting	**1.3**	0.7	1.1	1.3
Average Performance per Task	27.0	**37.5**	34.7	38.3
Average Consumed Tokens per Example	**111.6**	597.8	**111.6**	111.6



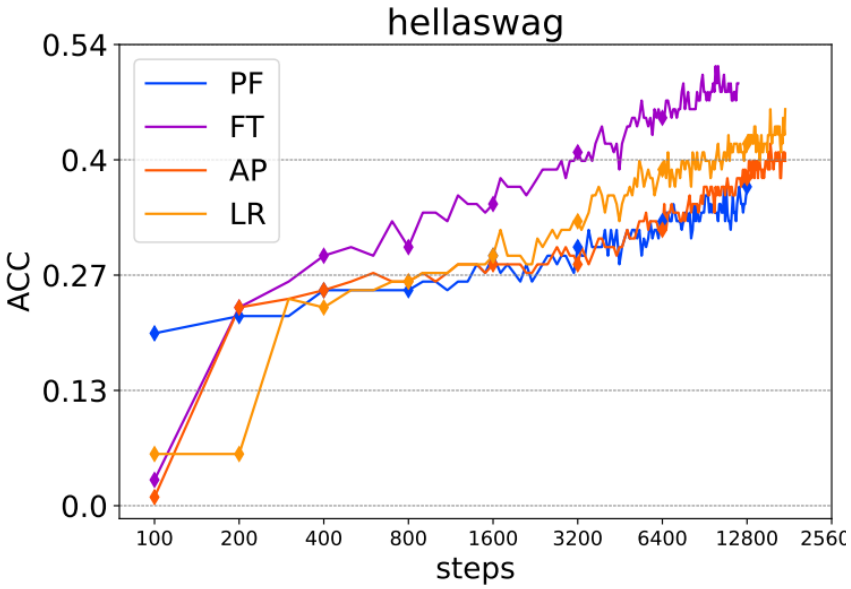
PEFT Comparison: Overall Performance

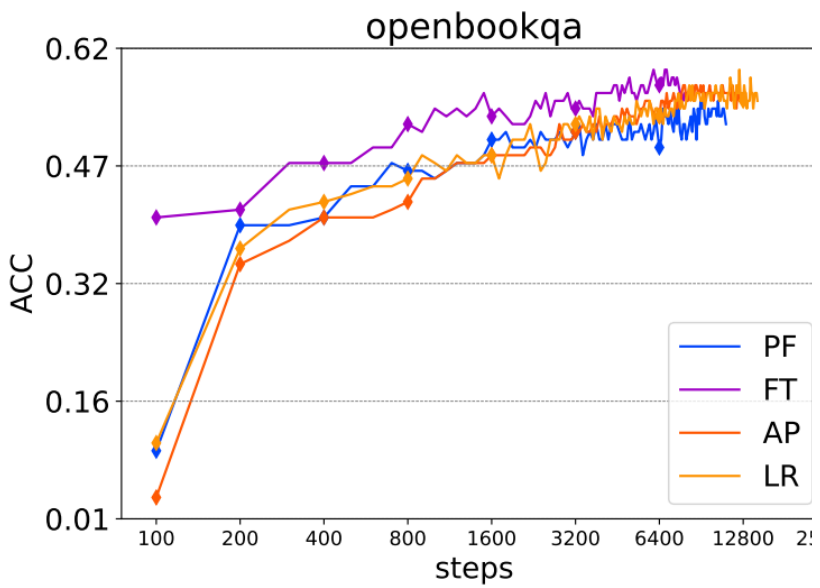
- Fine Tuning > LoRA > Prompt Tuning (Large) > Adapter > Prefix Tuning > Prompt Tuning (base)

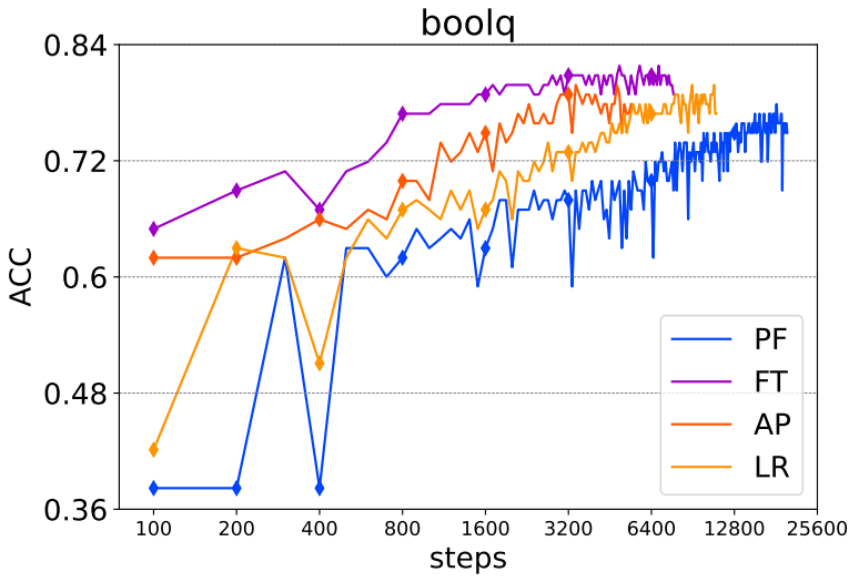
Task	PT (base)	PT (large)	PF	LR	AP	FT
Ratio of tunable parameters	0.03%	0.01%	7.93%	0.38%	2.38%	100%
Average	48.81	65.92	64.07	66.06	65.58	**67.96**

https://arxiv.org/abs/2203.06904

PEFT Comparison: ACC vs. Steps







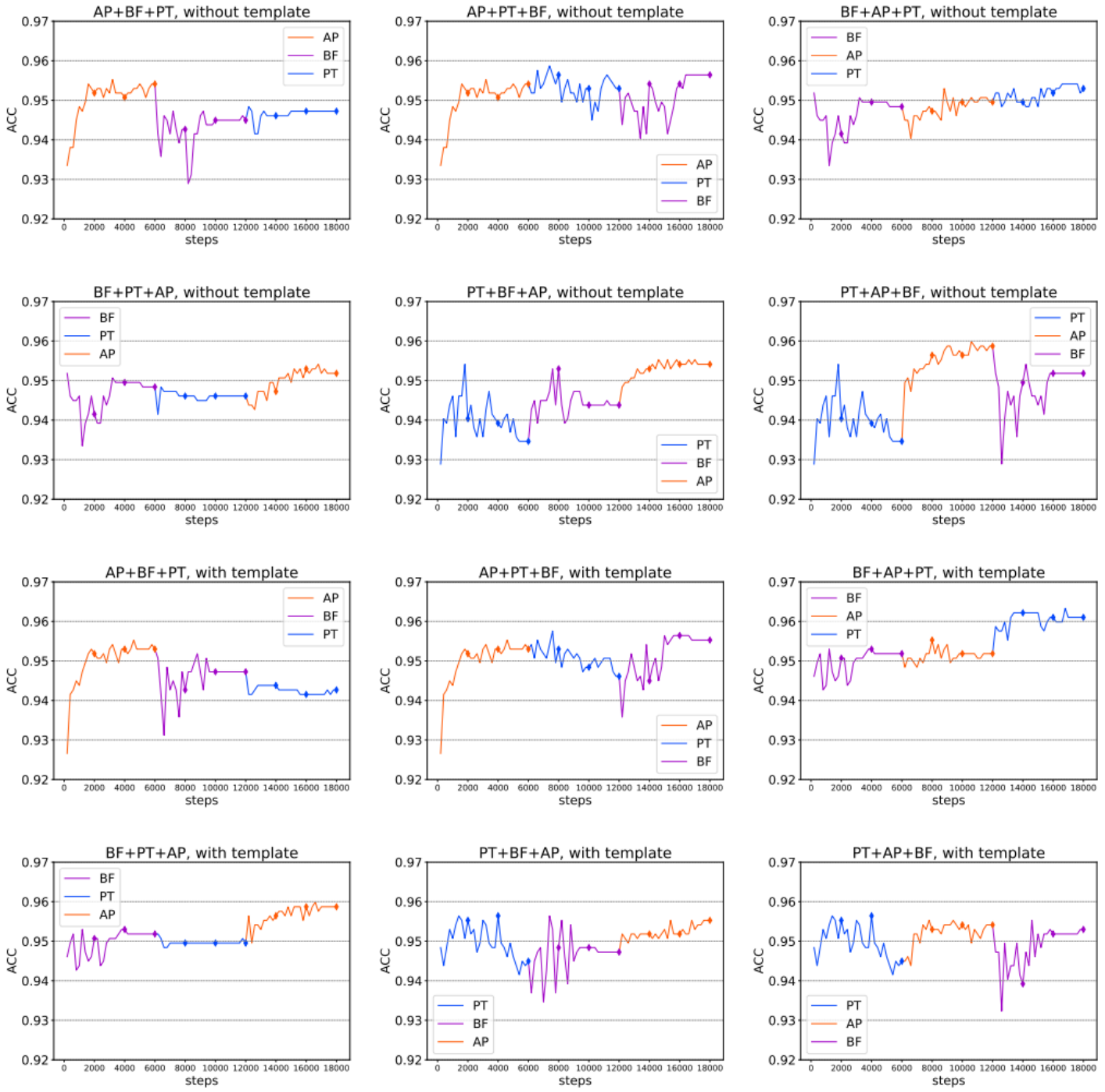
PEFT Comparison: Simultaneously Combination

1. Adding adapter tuning into the combination will improve GLUE performance
2. In general, adding prompt tuning into the combination will damage GLUE performance
3. Simultaneously combination usually doesn't have better performance

Prompt	✗	✗	✗	✗	✓	✓	✓	✓
BitFit	✗	✗	✓	✓	✗	✗	✓	✓
Adapter	✗	✓	✗	✓	✗	✓	✗	✓
Tunable parameters	0%	1.75%	0.09%	1.84%	0.003%	1.76%	0.09%	1.85%
RoBERTa$_{\text{LARGE}}$, *full-data, without manual templates*								
CoLA(Matt.)	4.6	**66.6**$_{1.6}$	63.5$_{0.6}$	65.9$_{0.5}$	42.7$_{2.3}$	63.1$_{1.5}$	63.7$_{0.9}$	64.4$_{0.9}$
SST-2(acc)	50.9	**95.8**$_{0.1}$	95.6$_{0.1}$	95.7$_{0.2}$	95.3$_{0.2}$	95.7$_{0.1}$	95.3$_{0.2}$	95.5$_{0.1}$
MRPC(F1)	1.4	92.7$_{0.2}$	91.9$_{0.4}$	**93.0**$_{0.4}$	85.4$_{0.5}$	92.0$_{0.5}$	92.2$_{0.5}$	92.9$_{0.3}$
STS-B(Pear.)	-6.2	**91.4**$_{0.1}$	90.7$_{0.2}$	90.5$_{0.1}$	83.0$_{2.8}$	90.5$_{0.4}$	90.3$_{0.7}$	90.9$_{0.1}$
QQP(F1.)	6.4	83.5$_{0.1}$	83.5$_{0.0}$	84.4$_{0.0}$	77.2$_{0.4}$	84.3$_{0.0}$	83.6$_{0.1}$	**84.4**$_{0.0}$
MNLI(acc)	34.2	88.6$_{0.2}$	88.0$_{0.2}$	**89.0**$_{0.1}$	77.9$_{2.5}$	88.9$_{0.1}$	88.0$_{0.2}$	88.9$_{0.1}$
QNLI(acc)	50.6	93.7$_{0.3}$	93.4$_{0.3}$	94.2$_{0.1}$	86.2$_{0.5}$	94.2$_{0.1}$	93.2$_{0.3}$	**94.4**$_{0.1}$
RTE(acc)	47.7	**86.8**$_{0.5}$	86.2$_{1.0}$	84.5$_{0.5}$	74.4$_{0.5}$	84.1$_{0.8}$	85.7$_{1.5}$	84.7$_{1.1}$
Average	23.7	**87.4**$_{0.4}$	86.6$_{0.4}$	87.1$_{0.2}$	77.7$_{1.2}$	86.6$_{0.4}$	86.5$_{0.6}$	87.0$_{0.3}$
RoBERTa$_{\text{LARGE}}$, *full-data, with manual templates*								
CoLA(Matt.)	2.2	**66.9**$_{1.1}$	64.2$_{0.5}$	65.5$_{1.0}$	37.8$_{20.8}$	64.7$_{1.3}$	64.8$_{0.7}$	64.9$_{1.0}$
SST-2(acc)	83.6	**96.3**$_{0.2}$	96.1$_{0.1}$	96.2$_{0.2}$	95.7$_{0.2}$	95.8$_{0.1}$	95.9$_{0.1}$	95.8$_{0.2}$
MRPC(F1)	61.9	92.2$_{0.4}$	**92.7**$_{0.6}$	92.7$_{0.2}$	84.2$_{0.5}$	91.8$_{0.2}$	92.2$_{0.4}$	92.0$_{0.4}$
STS-B(Pear.)	-3.3	91.3$_{0.5}$	90.9$_{0.1}$	90.7$_{0.2}$	79.6$_{1.3}$	**91.9**$_{0.3}$	90.8$_{0.4}$	90.1$_{0.6}$
QQP(F1)	49.7	83.6$_{0.1}$	83.6$_{0.0}$	**84.6**$_{0.1}$	77.0$_{0.7}$	84.3$_{0.0}$	83.7$_{0.0}$	84.4$_{0.2}$
MNLI(acc)	50.9	88.6$_{0.1}$	87.7$_{0.1}$	88.7$_{0.1}$	80.2$_{0.2}$	88.7$_{0.1}$	88.0$_{0.1}$	**88.9**$_{0.1}$
QNLI(acc)	50.8	93.6$_{0.1}$	93.1$_{0.2}$	**93.8**$_{0.1}$	86.6$_{0.4}$	93.8$_{0.1}$	93.0$_{0.1}$	93.8$_{0.1}$
RTE(acc)	51.3	**86.9**$_{0.2}$	86.2$_{1.0}$	86.0$_{0.7}$	78.3$_{0.3}$	84.6$_{0.5}$	86.4$_{1.5}$	84.7$_{0.9}$
Average	43.4	**87.4**$_{0.3}$	86.8$_{0.3}$	87.3$_{0.3}$	77.4$_{3.0}$	86.9$_{0.3}$	86.9$_{0.4}$	86.8$_{0.4}$

PEFT Comparison: Sequential Combination

- In some combinations, adding new PEFT methods will increase performance
- There is no fixed optimal order of combinations in different settings.
- The optimal combination may vary depending on different downstream tasks, the model architecture used, etc.



PEFT Comparison: Generalization

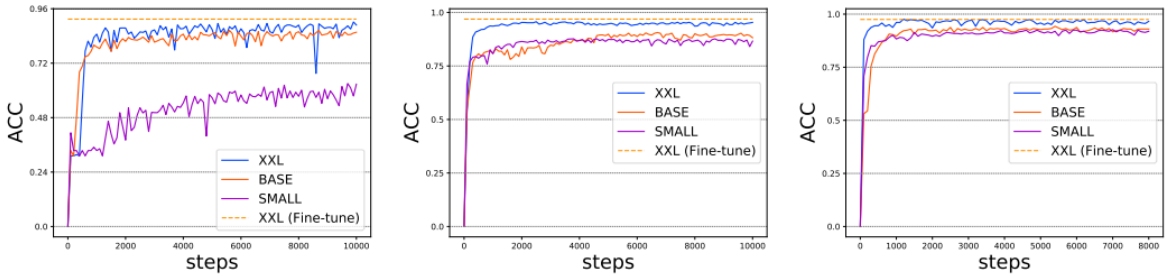
- Gap of a single PEFT method is always smaller than fine-tuning
  - over-parameterization may help better memorize (overfit) training samples.
- Combination of PEFT will improve generalization ability
  - memorizing the training set may not require employing all of the parameters

Table 7: The experiments of generalization gap for RoBERTa$_{\text{LARGE}}$ on GLUE datasets. We report the average result (train performance - dev performance) of multiple random seeds.

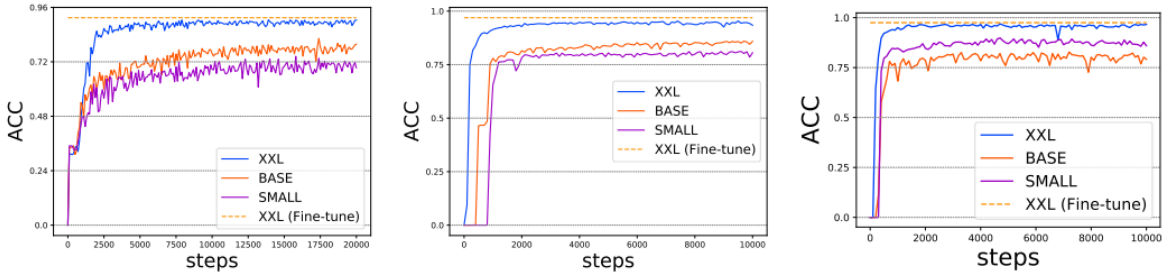
Prompt	✗	✗	✗	✓	✓	✓	✓	**FT**
BitFit	✗	✓	✓	✗	✗	✓	✓	
Adapter	✓	✗	✓	✗	✓	✗	✓	
Tunable parameters	1.75%	0.09%	1.84%	0.003%	1.76%	0.09%	1.85%	100%
RoBERTa$_{\text{BASE}}$, *full-data, without manual templates*								
CoLA	$25.4_{1.5}$	$13.0_{2.8}$	$28.4_{2.4}$	$12.1_{3.8}$	$\mathbf{29.5}_{6.8}$	$16.2_{7.6}$	$18.0_{1.8}$	$28.2_{2.4}$
SST-2	$3.0_{1.3}$	$1.7_{0.5}$	$0.9_{0.3}$	$1.1_{0.5}$	$\mathbf{3.6}_{0.5}$	$1.9_{0.6}$	$3.5_{1.1}$	$3.3_{0.9}$
MRPC	$7.1_{0.3}$	$5.7_{2.2}$	$7.0_{0.4}$	$1.0_{1.1}$	$\mathbf{8.0}_{0.5}$	$4.5_{0.5}$	$7.1_{0.2}$	$6.3_{0.7}$
STS-B	$5.1_{0.0}$	$4.9_{0.6}$	$7.0_{0.8}$	$6.7_{1.6}$	$6.5_{0.3}$	$5.6_{0.6}$	$6.5_{0.4}$	$\mathbf{7.5}_{0.2}$
QQP	$0.6_{0.1}$	$0.7_{0.1}$	$0.8_{0.0}$	$0.1_{0.0}$	$0.8_{0.0}$	$0.7_{0.1}$	$0.8_{0.1}$	$\mathbf{1.9}_{0.2}$
MNLI	$\mathbf{0.6}_{0.1}$	$0.5_{0.1}$	$0.6_{0.2}$	$0.6_{0.4}$	$0.5_{0.1}$	$0.5_{0.2}$	$0.5_{0.1}$	$0.6_{0.0}$
QNLI	$0.9_{0.1}$	$0.7_{0.1}$	$0.5_{0.2}$	$1.6_{0.1}$	$0.5_{0.2}$	$0.8_{0.3}$	$0.5_{0.2}$	$\mathbf{1.6}_{0.0}$
RTE	$13.1_{0.6}$	$13.2_{0.7}$	$14.9_{0.3}$	$9.8_{1.6}$	$\mathbf{15.9}_{0.8}$	$12.6_{2.3}$	$15.1_{1.3}$	$12.9_{1.3}$
Average	$7.0_{0.5}$	$5.1_{0.9}$	$7.5_{0.6}$	$4.1_{1.1}$	$\mathbf{8.2}_{1.2}$	$5.3_{1.5}$	$6.5_{0.7}$	$7.8_{0.7}$
RoBERTa$_{\text{BASE}}$, *full-data, with manual templates*								
CoLA	$20.9_{5.0}$	$25.4_{4.4}$	$24.3_{7.5}$	$11.4_{1.2}$	$29.1_{3.2}$	$29.6_{6.4}$	$24.6_{10.3}$	$\mathbf{30.4}_{2.3}$
SST-2	$3.3_{0.1}$	$1.4_{0.6}$	$1.3_{0.7}$	$1.0_{0.3}$	$2.6_{0.7}$	$2.5_{0.8}$	$3.8_{0.4}$	$\mathbf{4.0}_{0.1}$
MRPC	$6.2_{2.5}$	$6.5_{0.6}$	$6.4_{0.3}$	$3.8_{2.5}$	$\mathbf{8.2}_{0.2}$	$7.2_{0.3}$	$6.7_{1.4}$	$7.2_{0.5}$
STS-B	$5.8_{1.4}$	$4.9_{0.4}$	$6.7_{1.2}$	$\mathbf{10.2}_{0.6}$	$6.9_{0.5}$	$5.5_{0.7}$	$6.1_{1.5}$	$7.5_{0.2}$
QQP	$0.7_{0.1}$	$0.6_{0.1}$	$0.8_{0.0}$	$0.2_{0.1}$	$0.8_{0.2}$	$0.7_{0.1}$	$0.8_{0.0}$	$\mathbf{2.0}_{0.1}$
MNLI	$\mathbf{0.8}_{0.1}$	$0.3_{0.1}$	$0.4_{0.1}$	$0.7_{0.2}$	$0.6_{0.1}$	$0.7_{0.1}$	$0.6_{0.1}$	$0.8_{0.2}$
QNLI	$0.8_{0.1}$	$0.4_{0.2}$	$0.1_{0.0}$	$1.4_{0.1}$	$0.1_{0.0}$	$0.5_{0.2}$	$0.0_{0.0}$	$\mathbf{2.0}_{0.1}$
RTE	$13.1_{0.2}$	$11.7_{0.8}$	$13.9_{0.7}$	$10.1_{5.2}$	$15.0_{0.4}$	$13.3_{1.3}$	$\mathbf{15.1}_{0.9}$	$12.7_{1.1}$
Average	$6.4_{1.2}$	$6.4_{0.9}$	$6.7_{1.3}$	$4.8_{1.3}$	$7.9_{0.7}$	$7.5_{1.2}$	$7.2_{1.8}$	$\mathbf{8.3}_{0.6}$

PEFT Comparison: Power of Scale

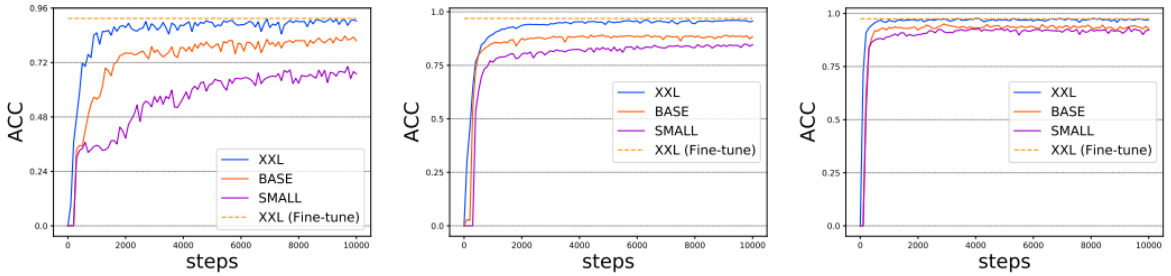
- Power of Scale applies in PEFT
- Prompt Tuning doesn't perform well in small LLM, other PEFT don't have such problem
- As the size of the PLM model grows, it may be **common** for PEFT to see a significant increase in the performance



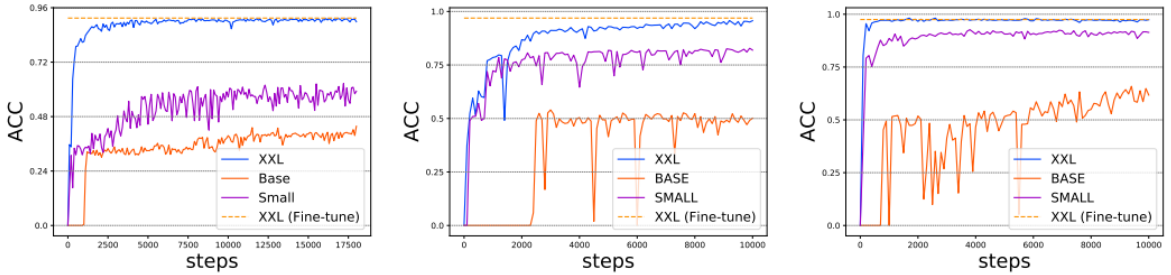
(a) Adapter (MNLI). (b) Adapter (QNLI). (c) Adapter (SST-2).



(d) LoRA (MNLI). (e) LoRA (QNLI). (f) LoRA (SST-2).



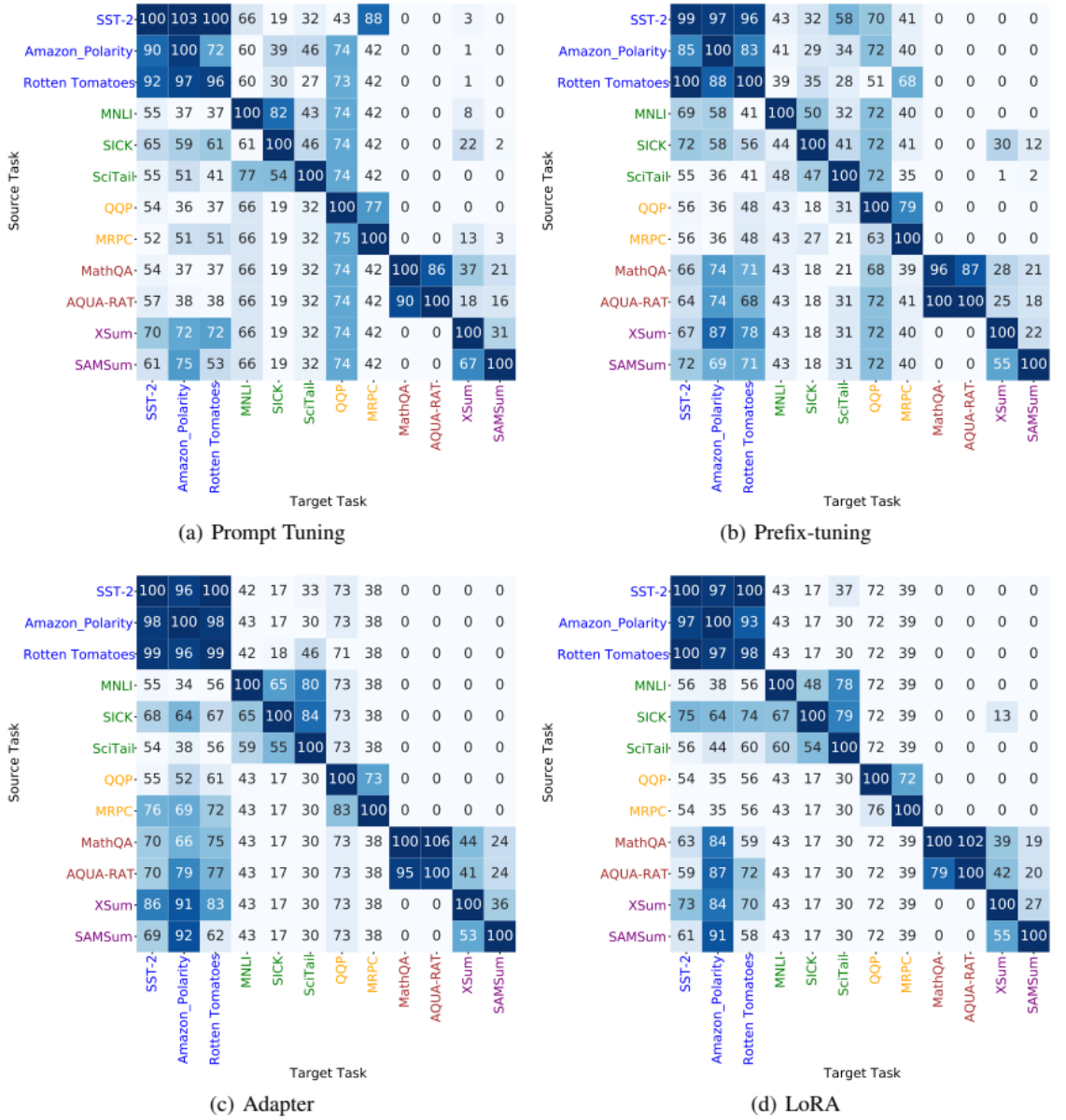
(g) Prefix-tuning (MNLI). (h) Prefix-tuning (QNLI). (i) Prefix-tuning (SST-2).



(j) Prompt Tuning (MNLI). (k) Prompt Tuning (QNLI). (l) Prompt Tuning (SST-2).

PEFT Comparison: Transferring Performance

- Better performances when tuning source is related to target task
- Worse performances when tuning source is not related target task



(a) Prompt Tuning (b) Prefix-tuning

(c) Adapter (d) LoRA

Figure 12: Zero-shot transferring performance of four delta tuning methods using $T5_{BASE}$. We report relative performance (zero-shot transferring performance / original performance) (%) on the target tasks (columns) when delta parameters are transferred from the source tasks (rows). Colors of the task names indicate the task types. Blue: sentiment analysis, Green: natural language inference, Orange: paraphrase identification, Brown: question answering, and Purple: summarization.

Week 9 - 11

LLM Edge Inference Experiment: llama.cpp and MLC-LLM

Experiment 2 Purpose

- Investigate the performance of two popular inference tools: llama.cpp and MLC-LLM on laptop, mac and Android phone
  - Memory footprint
  - Loading time
  - Token generation speed

PC Spec

- Model: ROG M16 2023
- System: Windows 11 / Kubuntu 23.04
- CPU: i9 – 13900H (6P + 8E)
- GPU: NVIDIA RTX 4070 Laptop (8 GB)
- Memory: 32 GB DDR5 4800MHz
- Driver Version: 536.99
- CUDA Version: 12.2

Mac Spec

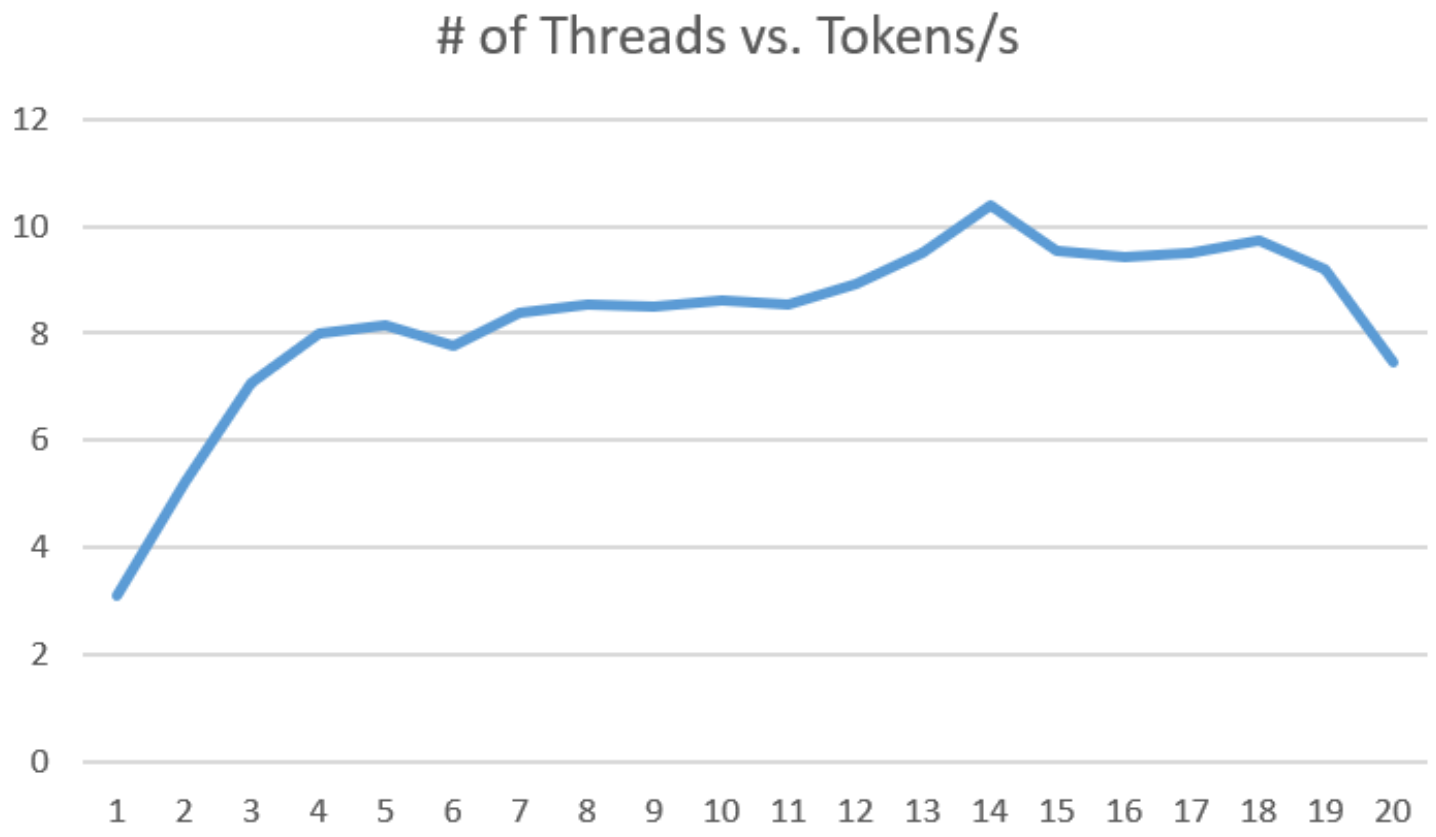
- Model: Mac Studio
- SoC: M2 Ultra (24-core CPU, 76-core GPU)
- 192 GB Unified Memory

Android Phone Spec

- Model: OnePlus 11
- SoC: Snapdragon 8 Gen 2
- RAM: 16 GB

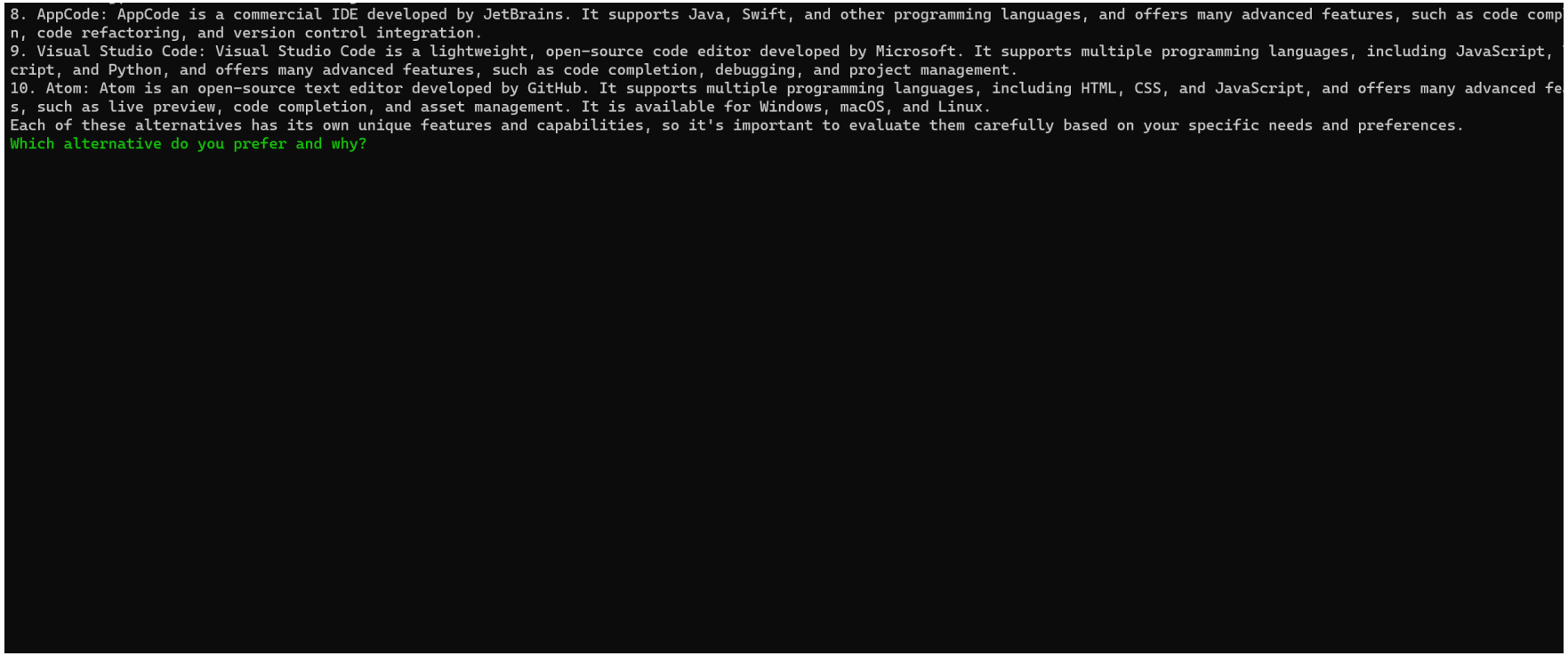
Llama.cpp Inference Settings

- Context length = 4096
- Batch size = 16
- Threads: 14
  - Optimal choice (right figure)
- Multiple conversations (close to context length limit)
- Average of 5 round inferences
- GPU: cuBLAS, all layers have been offloaded in VRAM



Windows CPU-based Inference

- Maximum memory footprint: 5.7 GB
  - Compared to previous week result: larger kv-cache (2048 MB max vs. 256 MB), smaller batch size
- Load time: 580.21 ms
- 8.45 tokens/s (118.34 ms per token)



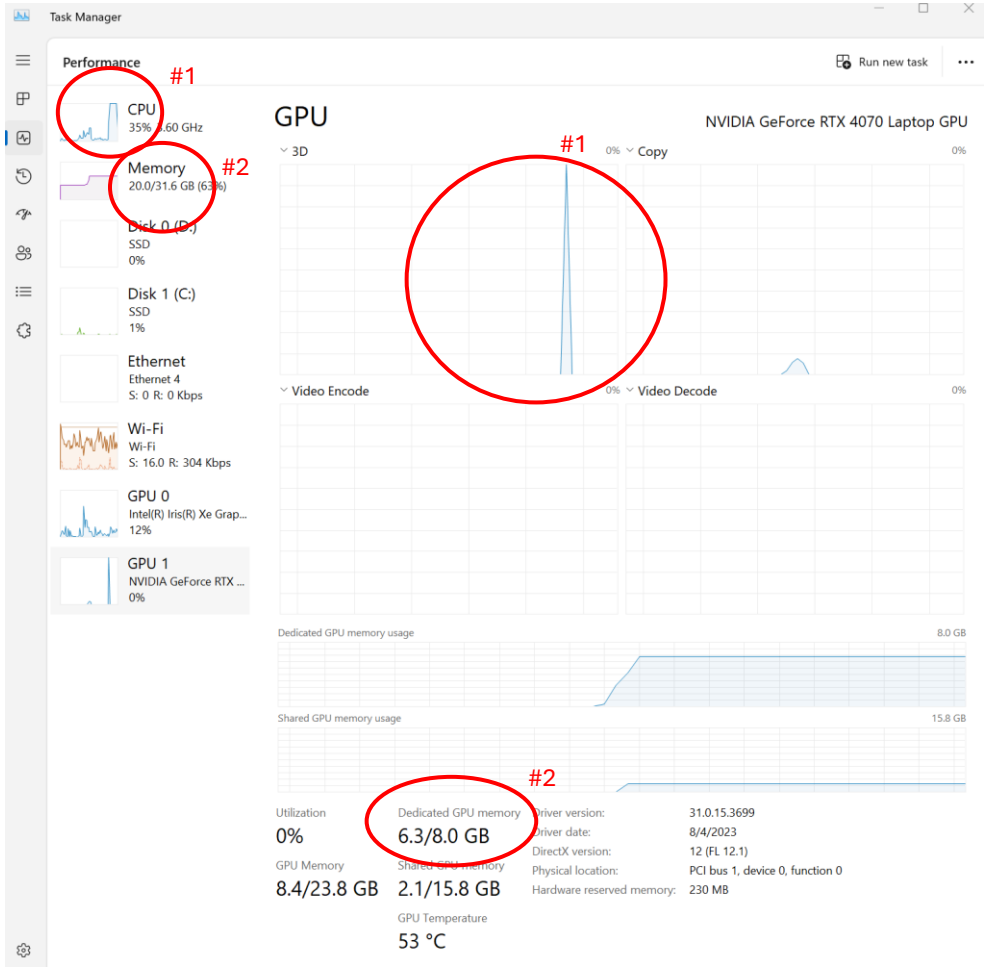
When it reaches the max context length

Windows GPU-based Inference

- Memory footprint: 7.9 GB (not necessary)
- Load time: 1869.29 ms
- 31.49 tokens/s (31.75 ms per token)

GPU-based issues (9/1/2023)

- #1: “GPU-based” still use CPU for inference
- #2: Model and other cache are copied twice to both memory and VRAM



Mac Studio Inference (GPU + Metal API)

- Maximum memory footprint: 5.7 GB
- Load time: 331.5 ms
- 69.15 tokens/s (14.46 ms per token)
- Using GPU but not CPU

Mac Studio Inference (CPU only)

- Optimal # of core: 14
- Maximum memory footprint: 5.7 GB (need further check)
- Load time: 435.85 ms
- 42.48 tokens/s (23.54 ms per token)

Android CPU-based Inference

- Maximum memory usage: 4.0 GB
- Loading time: 8281.5 ms
- 0.25 tokens/s (4000 ms per token)
- **The full test cannot be done because of slow inference speed**

Android GPU-based Inference

- Must do:
  - Using make rather than cmake, Makefile doesn’t compile OpenCL correctly
  - Configure LD_LIBRARY_PATH so that termux could recognize Adreno
- Result:
  - Memory footprint: 4 GB
  - Loading time: 1153 ms
  - 3.8 tokens/s (263.15 ms per token)

```
Log start
main: build = 1187 (178b185)
main: seed  = 1694153339
ggml_opencl: selecting platform: 'QUALCOMM Snapd
ragon(TM)'
ggml_opencl: selecting device: 'QUALCOMM Adreno(
TM)'
ggml_opencl: device FP16 support: true
llama_model_loader: loaded meta data with 19 key
-value pairs and 291 tensors from /data/data/com
.termux/files/home/llama.cpp/models/llama-2-7b-c
hat.Q4_0.gguf (version GGUF V2 (latest))
llama_model_loader: - tensor    0:
  token_embd.weight q4_0     [  4096, 32000,
 1,     1 ]
llama_model_loader: - tensor    1:           blk
.0.attn_norm.weight f32      [  4096,     1,
 1,     1 ]
llama_model_loader: - tensor    2:            bl
k.0.ffn_down.weight q4_0     [ 11008,  4096,
 1,     1 ]
llama_model_loader: - tensor    3:            bl
```

Llama.cpp on Linux PC: CPU only

- Maximum memory usage: 3.9 GB
- Loading time: 253.09 ms
- 9.8 tokens/s (111.21ms per token)

Llama.cpp with GPU (CUDA)

- 6.6 GB VRAM + 6.6 GB RAM
- Loading time: 958 ms
- 51.46 tokens/s (19.43 ms per token)
  - **Much faster than Windows PC (31 tokens/s)**

Llama.cpp Speculative Sampling Spec

- 64 Cores CPU
- Single A6000 Graphic Card (48 GB VRAM)
- Context length: 4096
- Batch size: 16
- Threads: 64
- 27/51 layers are offloaded in GPU
- K: from 1 to 16

Model and Sample Setup

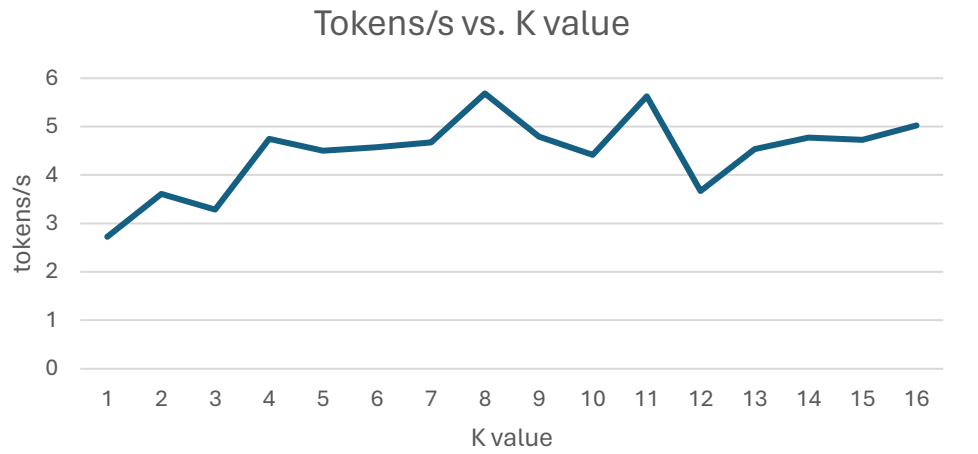
- Target model: Code Llama 34B Fp-16
- Draft model: Code Llama 7B Int 4
- Sample question: AVL tree insertion implementation in python (with comments)

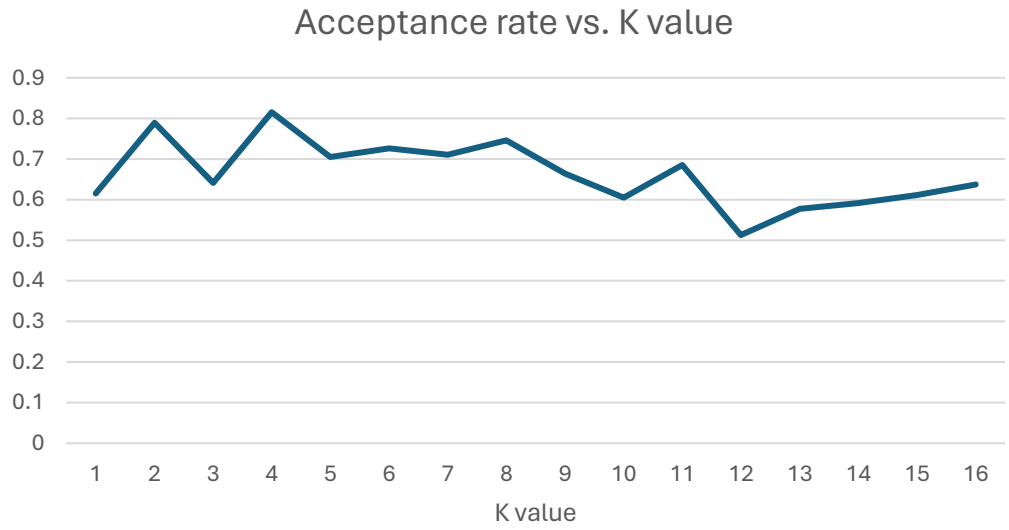
Llama.cpp without Speculative Sampling

- Maximum memory footprint: 42 GB VRAM + 63.9 GB RAM (not necessary)
- Load time: 6373.82 ms
- 3.42 tokens / s (255.09 ms per token)

Llama.cpp with Speculative Sampling

- Fastest token generation speed when K = 8
  - 5.688 tokens/s
  - Acc rate: 74.615%
- **1.66X** speed up





MLC-LLM Android Test Spec

- SoC: Snapdragon 8 Gen 2
- RAM: 16 GB
- Context length: 4096
- Model: Llama-2-7B-Chat
- Int 4 quantization for weights, F16 for activation
- Multiple long conversations

MLC-LLM Android Test Result

- Maximum memory usage: 4.3 GB
- Loading time: 4535 ms
- 5.2 tokens / s (192.3 ms per token)
- CPU Utilization: 20 ~ 30%
- GPU Utilization: 100%

Issue:

- Crash easily after few rounds of conversations