LLM-Pruner-On the Structural Pruning of Large Language Models

https://arxiv.org/abs/2305.11627

Accepted by NeurIPS 2023

2023.11.14

arXiv:2305.11627v1 [cs.CL] 19 May 2023

LLM-Pruner: On the Structural Pruning of Large Language Models

Xinyin Ma Gongfan Fang Xinchao Wang*
National University of Singapore
maxinyin@u.nus.edu, gongfan@u.nus.edu, xinchao@nus.edu.sg

Abstract

Large language models (LLMs) have shown remarkable capabilities in language understanding and generation. However, such impressive capability typically comes with a substantial model size, which presents significant challenges in both the deployment, inference, and training stages. With LLM being a general-purpose task solver, we explore its compression in a task-agnostic manner, which aims to preserve the multi-task solving and language generation ability of the original LLM. One challenge to achieving this is the enormous size of the training corpus of LLM, which makes both data transfer and model post-training over-burdensome. Thus, we tackle the compression of LLMs within the bound of two constraints: being task-agnostic and minimizing the reliance on the original training dataset. Our method, named LLM-Pruner, adopts structural pruning that selectively removes non-critical coupled structures based on gradient information, maximally preserving the majority of the LLM's functionality. To this end, the performance of pruned models can be efficiently recovered through tuning techniques, LoRA, in merely *3 hours*, requiring only *50K* data. We validate the LLM-Pruner on three LLMs, including LLaMA, Vicuna, and ChatGLM, and demonstrate that the compressed models still exhibit satisfactory capabilities in zero-shot classification and generation. The code is available at: `https://github.com/horseee/LLM-Pruner`

1 Introduction

Recently, Large Language Models (LLMs) [36, 48, 47, 41, 59, 4, 66] have demonstrated remarkable proficiency in language understanding and generation. With the increase in model size, they are better equipped to handle complex tasks [3, 5, 55] and even exhibit emergent abilities [54]. However, notwithstanding their impressive performance, LLMs pose challenges in deployment and inference. Their extensive scale engenders substantial computational demands, and the multitude of parameters involved can induce long latencies and other related issues. Several techniques are proposed to solve these problems, like model pruning [53, 56, 64, 20], knowledge distillation [43, 38, 44],quantization [1, 12] within the context of pre-trained language model (PLM).

While previous methods have effectively maintained model performance amidst parameter reduction, they primarily target compression within specialized domains or for designated tasks in the context of task-specific compression. For instance, a PLM is fine-tuned on a particular dataset, such as one of the classification tasks in the GLUE benchmark [50], after which these models are distilled into a smaller classification model [43, 17]. Although this paradigm could potentially be employed for LLM compression, it compromises the LLM's capacity as a versatile task solver, rendering it suited to a single task exclusively.

*Corresponding author

arXiv:2305.11627v3 [cs.CL] 28 Sep 2023

LLM-Pruner: On the Structural Pruning of Large Language Models

Xinyin Ma Gongfan Fang Xinchao Wang*
National University of Singapore
maxinyin@u.nus.edu, gongfan@u.nus.edu, xinchao@nus.edu.sg

Abstract

Large language models (LLMs) have shown remarkable capabilities in language understanding and generation. However, such impressive capability typically comes with a substantial model size, which presents significant challenges in both the deployment, inference, and training stages. With LLM being a general-purpose task solver, we explore its compression in a task-agnostic manner, which aims to preserve the multi-task solving and language generation ability of the original LLM. One challenge to achieving this is the enormous size of the training corpus of LLM, which makes both data transfer and model post-training over-burdensome. Thus, we tackle the compression of LLMs within the bound of two constraints: being task-agnostic and minimizing the reliance on the original training dataset. Our method, named LLM-Pruner, adopts structural pruning that selectively removes non-critical coupled structures based on gradient information, maximally preserving the majority of the LLM's functionality. To this end, the performance of pruned models can be efficiently recovered through tuning techniques, LoRA, in merely *3 hours*, requiring only *50K* data. We validate the LLM-Pruner on three LLMs, including LLaMA, Vicuna, and ChatGLM, and demonstrate that the compressed models still exhibit satisfactory capabilities in zero-shot classification and generation. The code is available at: `https://github.com/horseee/LLM-Pruner`

1 Introduction

Recently, Large Language Models (LLMs) [37, 49, 48, 42, 62, 4, 69] have demonstrated remarkable proficiency in language understanding and generation. With the increase in model size, they are better equipped to handle complex tasks [3, 5, 56, 58] and even exhibit emergent abilities [55]. However, notwithstanding their impressive performance, LLMs pose challenges in deployment and inference. Their extensive scale engenders substantial computational demands, and the multitude of parameters involved can induce long latencies and other related issues. Several techniques are proposed to solve these problems, like model pruning [54, 59, 67, 21], knowledge distillation [44, 39, 45],quantization [1, 13] within the context of pre-trained language model (PLM).

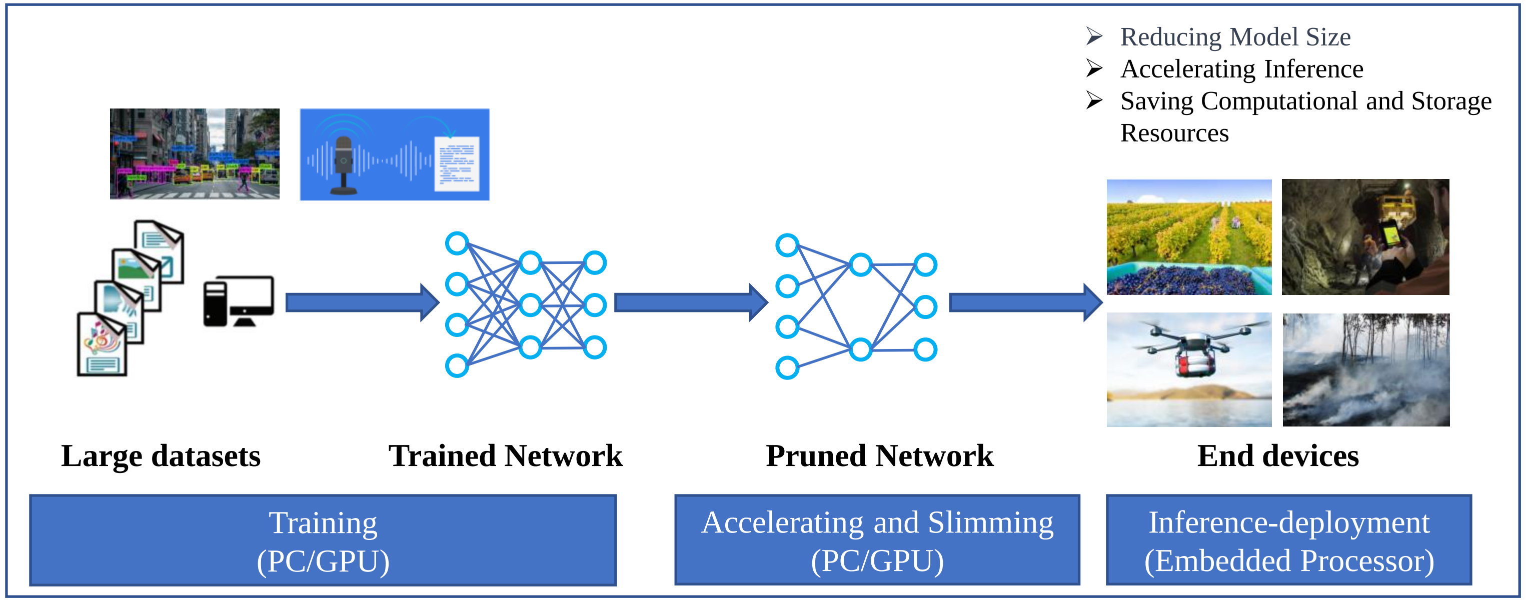
While previous methods have effectively maintained model performance amidst parameter reduction, they primarily target compression within specialized domains or for designated tasks in the context of task-specific compression. For instance, a PLM is fine-tuned on a particular dataset, such as one of the classification tasks in the GLUE benchmark [51], after which these models are distilled into a smaller classification model [44, 18]. Although this paradigm could potentially be employed for LLM compression, it compromises the LLM's capacity as a versatile task solver, rendering it suited to a single task exclusively.

*Corresponding author

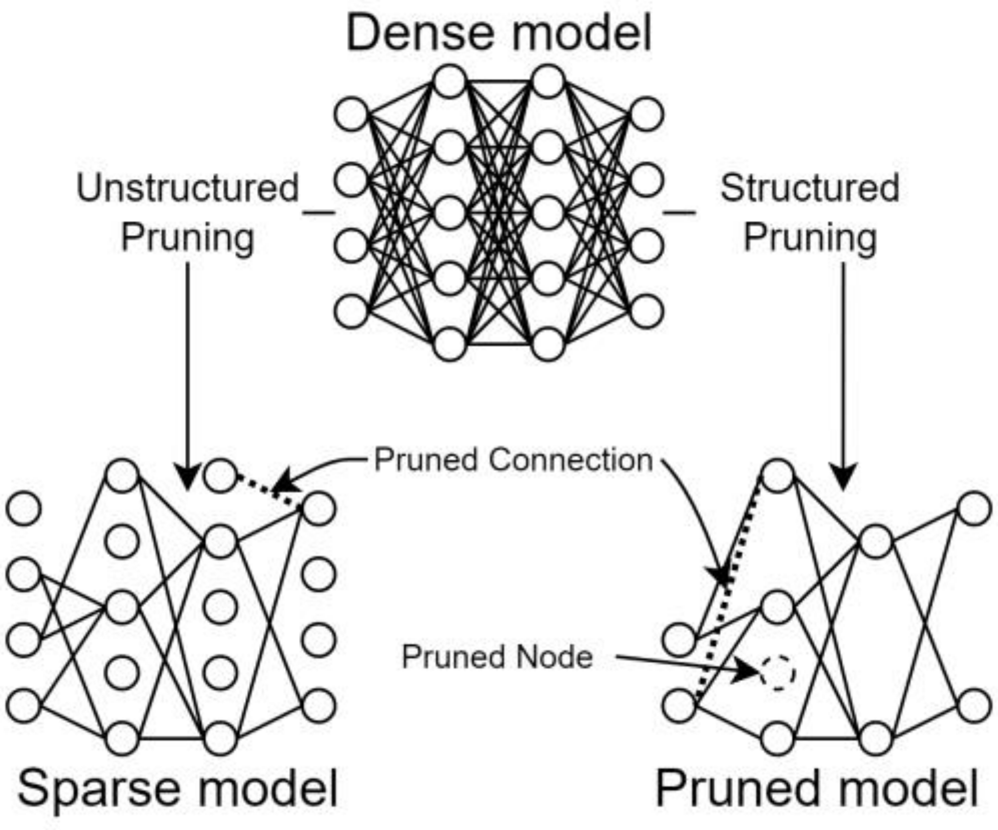
What is Pruning

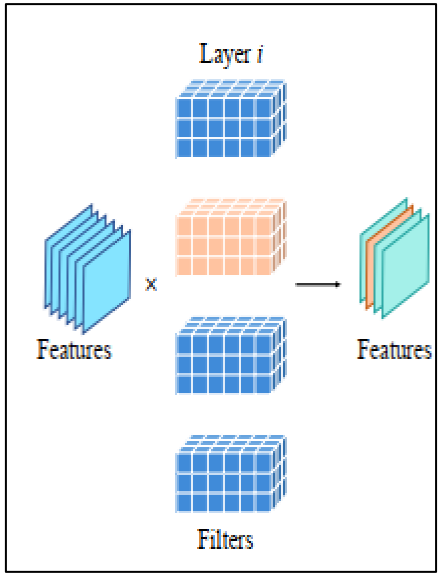
In machine learning and neural networks, “pruning” generally refers to the process of removing certain parts of the model to improve its efficiency without significantly affecting its performance.

It aims to **reduce** the **computational complexity**, **memory footprint**, and **power consumption** of the model and **accelerate inference time**, making it more suitable for deployment in resource-constrained environments.



Unstructured Pruning vs. Structured Pruning





from DNNShifter: An Efficient DNN Pruning System for Edge Computing, 2023

Examples of LLMs

LLMs are statistical models that predict the next word in a sequence. They trained on billions of words, which are scraped from multiple websites like Reddit, StackOverflow, Wikipedia, Books, ArXiv, Github, etc.

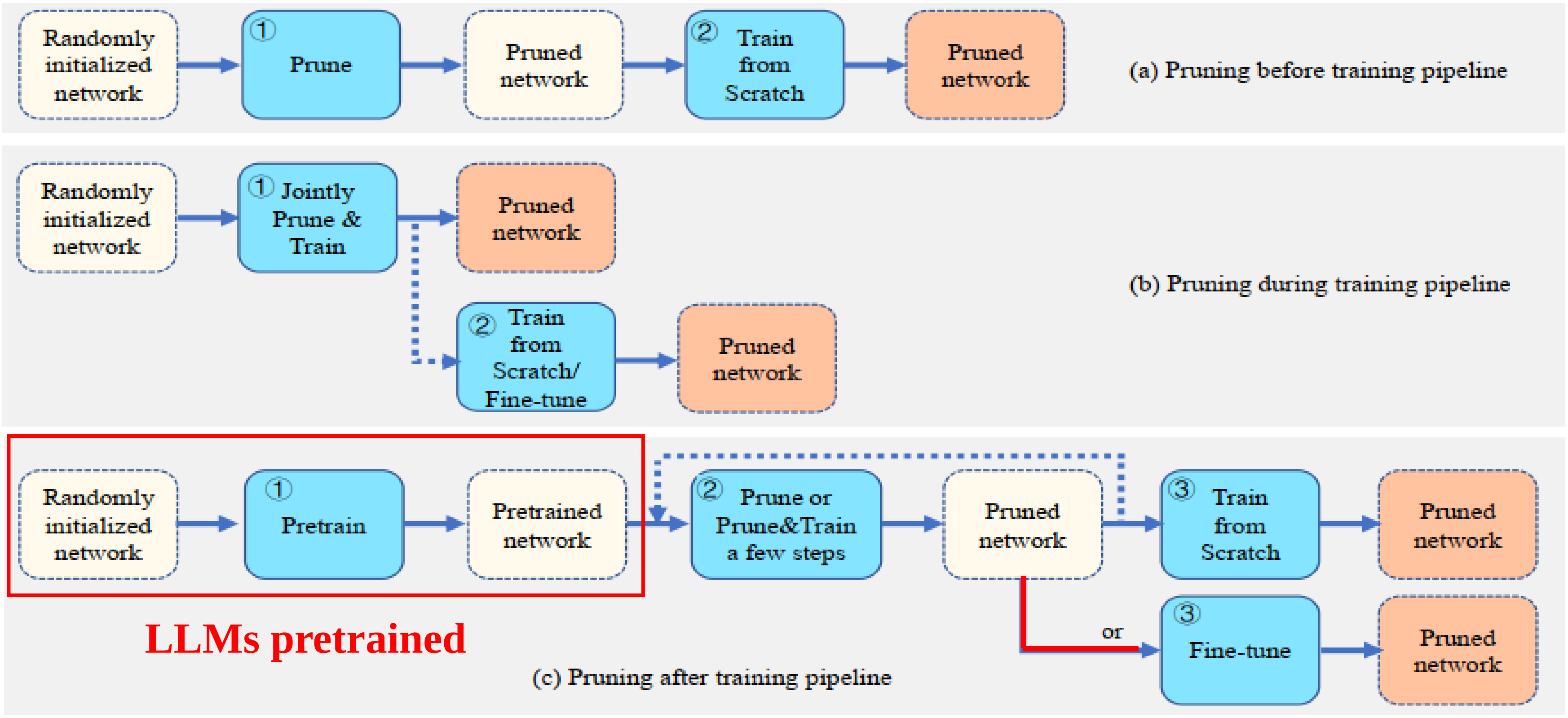
Model	Dataset (no. of words)	Paramters
GPT-3	0.5T	175B
LLaMA	1.2T	7B-65B

From https://www.analyticsvidhya.com/blog/2023/05/how-does-chatgpt-work-from-pretraining-to-rlhf/

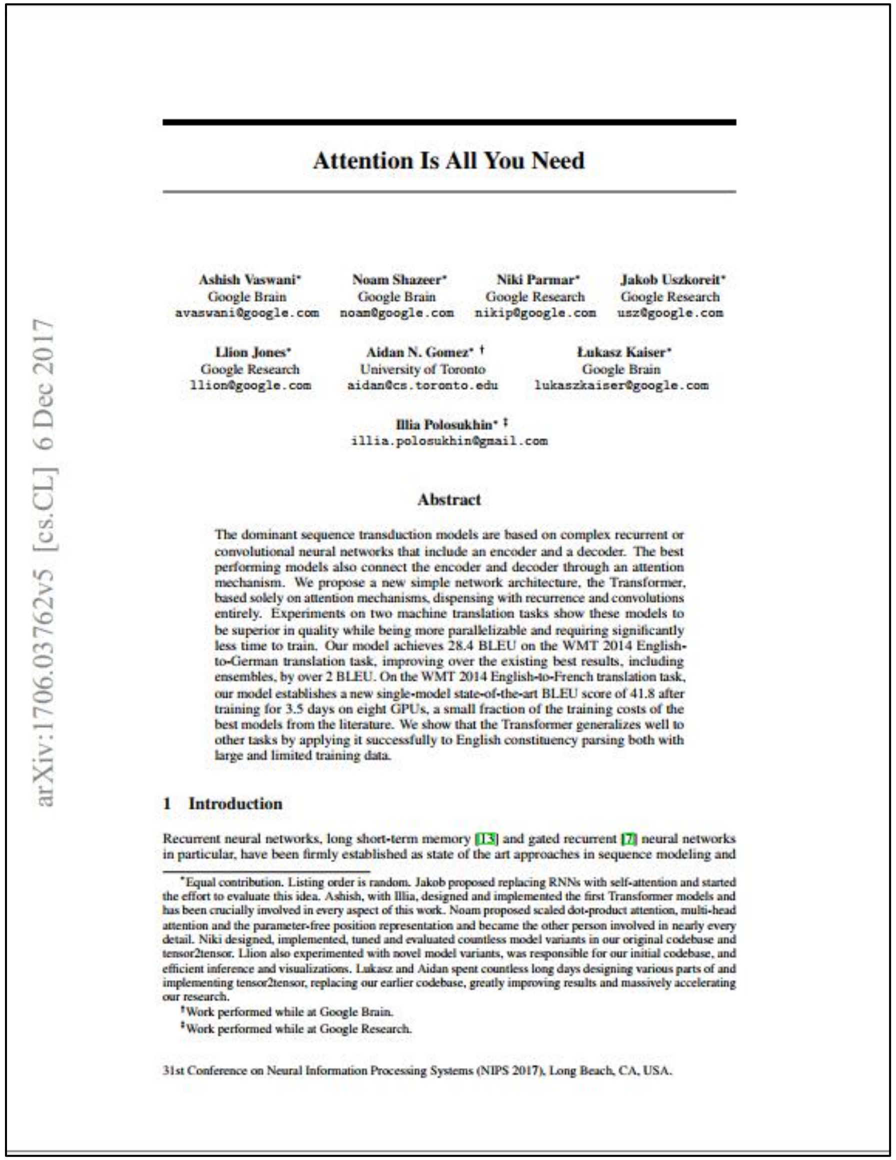
For example, **Llama-2-70B with 16-bit precision = 2 bytes * 70 billion = 140 GB of memory**

In practice, this means Llama-2-70B will need at least 2 A100 GPUs (80GB) for inference.

LLMs pruning vs. Traditional pruning



Attention is All You Need


arXiv:1706.03762v5 [cs.CL] 6 Dec 2017

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

1 Introduction

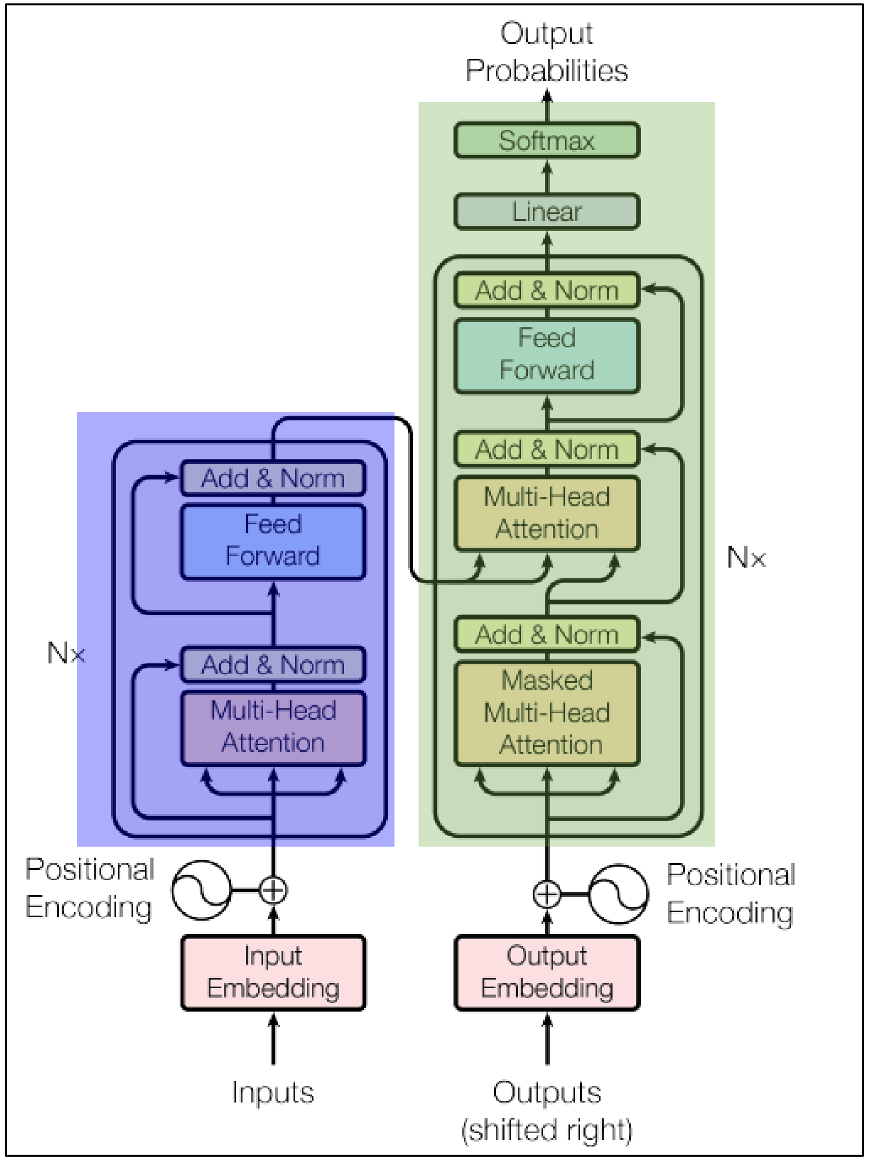
Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.



- **Self-attention**
- **Multi-head attention**
- **Layer normalization**

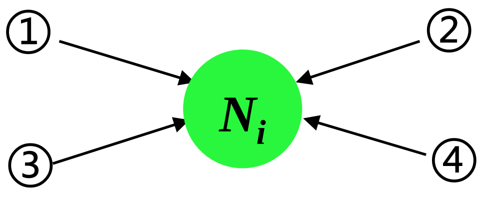
NIPS 2017

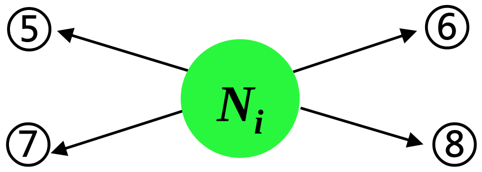
Method

- Step1. Discovery Stage
- Step2. Estimation Stage
- Step3. Recover Stage

Step1. Discover All Coupled Structured in LLMs

- In(N_i) and Out(N_i) represents all the neurons that point towards or point from N_i.

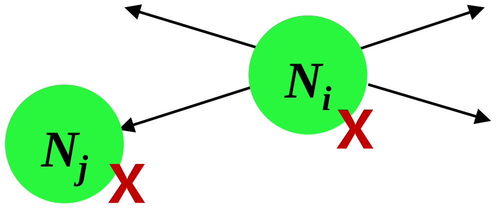




In(N_i)={①②③④} Out(N_i)={⑤⑥⑦⑧}

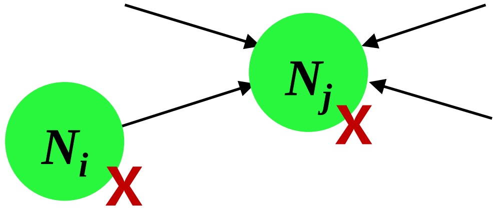
- The dependency between structures can be defined as:

$$N_j \in \mathrm{Out}(N_i) \wedge \mathrm{Deg}^{-}(N_j) = 1 \Rightarrow N_j \text{ is dependent on } N_i \quad (1)$$



where $\mathrm{Deg}^{-}(N_j)$ represents the in-degree of neuron N_j.

$$N_i \in \mathrm{In}(N_j) \wedge \mathrm{Deg}^{+}(N_i) = 1 \Rightarrow N_i \text{ is dependent on } N_j \quad (2)$$



where $\mathrm{Deg}^{+}(N_i)$ represents the out-degree of neuron N_i.

Step1. Discover All Coupled Structured in LLMs

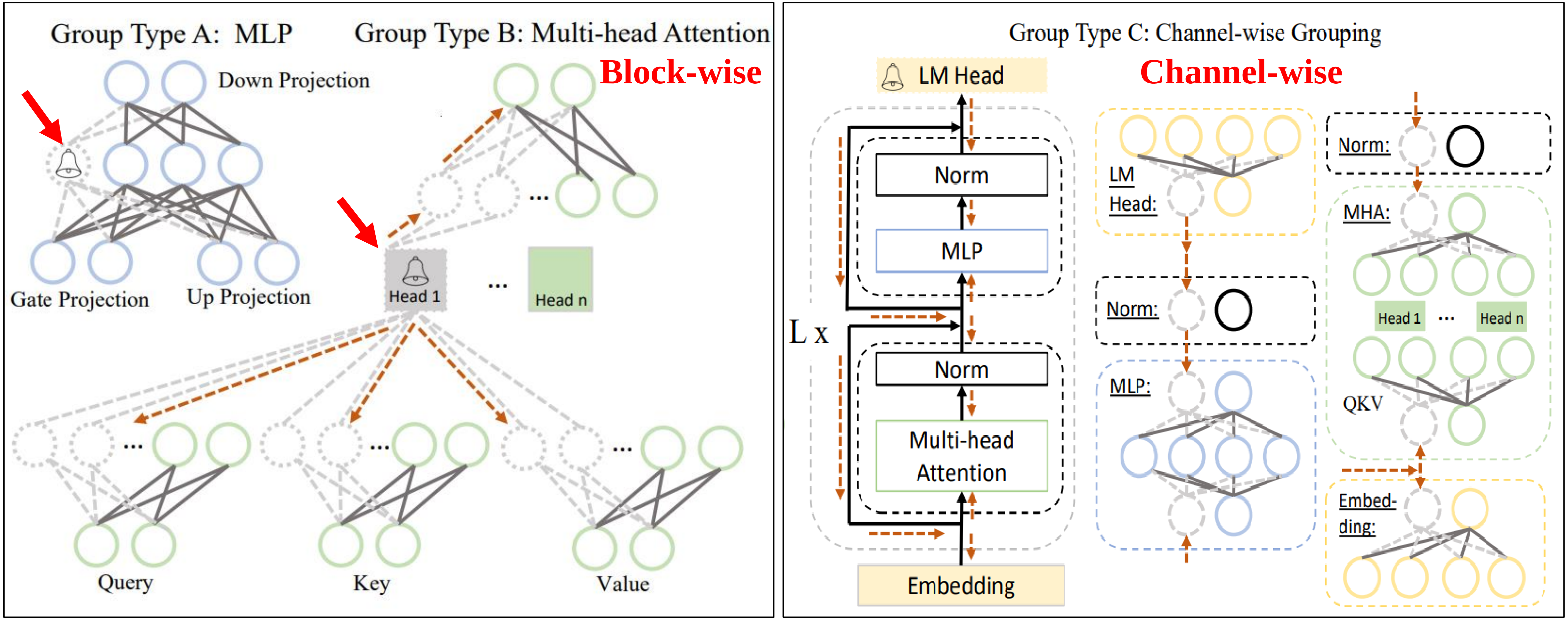


Figure 2: Illustration of the coupled structures in LLaMA. We simplify the neurons in each layer to make the dependent group clear. The trigger neuron, marked as a circle with a bell, cause weights with dependency pruned (dashed lines), which may propagate (red dashed lines) to coupled neurons (dashed circles). A group can be triggered by a variety of trigger neurons. Taking Group Type B as an example, the trigger for this group involves (i) the attention head, (ii) the output neuron in Query, Key or Value, and (iii) the input neuron in the final output projection.

DepGraph

DepGraph: Towards Any Structural Pruning

Gongfan Fang[1] Xinyin Ma[1] Mingli Song[2] Michael Bi Mi[3] Xinchao Wang[1*]
National University of Singapore[1] Zhejiang University[2] Huawei Technologies Ltd.[3]
gongfan@u.nus.edu, maxinyin@u.nus.edu, xinchao@nus.edu.sg
https://github.com/VainF/Torch-Pruning

arXiv:2301.12900v2 [cs.AI] 23 Mar 2023

Abstract

Structural pruning enables model acceleration by removing structurally-grouped parameters from neural networks. However, the parameter-grouping patterns vary widely across different models, making architecture-specific pruners, which rely on manually-designed grouping schemes, non-generalizable to new architectures. In this work, we study a highly-challenging yet barely-explored task, any structural pruning, to tackle general structural pruning of arbitrary architecture like CNNs, RNNs, GNNs and Transformers. The most prominent obstacle towards this goal lies in the structural coupling, which not only forces different layers to be pruned simultaneously, but also expects all removed parameters to be consistently unimportant, thereby avoiding structural issues and significant performance degradation after pruning. To address this problem, we propose a general and fully automatic method, Dependency Graph (DepGraph), to explicitly model the dependency between layers and comprehensively group coupled parameters for pruning. In this work, we extensively evaluate our method on several architectures and tasks, including ResNe(X)t, DenseNet, MobileNet and Vision transformer for images, GAT for graph, DGCNN for 3D point cloud, alongside LSTM for language, and demonstrate that, even with a simple norm-based criterion, the proposed method consistently yields gratifying performances.

1. Introduction

The recent emergence of edge computing applications calls for the necessity for deep neural compression [16, 22, 25, 33, 34, 61, 65–67, 69, 75]. Among the many network compression paradigms, pruning has proven itself to be highly

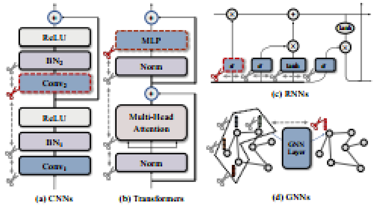


Figure 1. Parameters from different layers are inherently dependent on each other across network architectures, which forces several layers to be pruned simultaneously. For instance, to prune the Conv_2 in (a), all other layers $\{\text{Conv}_1, \text{BN}_1, \text{BN}_2\}$ within the block must be pruned as well. In this work, we introduce a generic scheme, termed as Dependency Graph, to explicitly account for such dependencies and execute the pruning of arbitrary architecture in a fully automatic manner.

be roughly categorized into two schemes, *structurual pruning* [4, 29, 71] and *unstructurual pruning* [8, 13, 44]. The core difference between the two lies in that, structural pruning changes the structure of neural networks by physically removing grouped parameters, while unstructural pruning conducts zeroing on partial weights without modification to the network structure. Compared to unstructural ones, structural pruning does not rely on specific AI accelerators or software to reduce memory consumption and computational costs, thereby finding a wider domain of applications in practice [38, 68].

Nevertheless, the nature of structural pruning *per se* makes itself a challenging task, especially for modern deep neural networks with coupled and complex internal structures. The rationale lies in that, deep neural networks are

Dependency Modeling. Leveraging this notation, we re-sketch the neural network as Equation 2, where two principal types of dependencies can be discerned, namely inter-layer dependency and intra-layer dependency, as demonstrated below:

$$\underbrace{(f_1^-, f_1^+) \leftrightarrow (f_2^-}_{\textit{Inter-layer Dep}}, f_2^+) \cdots \leftrightarrow \underbrace{(f_L^-, f_L^+)}_{\textit{Intra-layer Dep}} \tag{2}$$

The symbol $\leftrightarrow$ signifies the connectivity between two adjacent layers. Examination of these two dependencies yields straightforward but general rules for dependency modeling:

- *Inter-layer Dependency*: A dependency $f_i^- \Leftrightarrow f_j^+$ consistently arises in connected layers where $f_i^- \leftrightarrow f_j^+$.
- *Intra-layer Dependency*: A dependency $f_i^- \Leftrightarrow f_i^+$ exists if f_i^- and f_i^+ share identical pruning schemes, denoted by $sch(f_i^-) = sch(f_i^+)$.

```
Successful Pruning: 81 Models
 ['ssdlite320_mobilenet_v3_large', 'ssd300_vgg16', 'fasterrcnn_resnet50_fpn', 'fasterrcnn_resnet50_fpn_v2', 'fasterrcnn_mobilenet_v3_large_320_fpn', 'fasterrcnn_mobilenet_v3_large_fpn', 'fcos_re

Unsuccessful Pruning: 4 Models
 ['raft_large', 'swin_t', 'swin_s', 'swin_b']
```

Method

- Step1. Discovery Stage
- Step2. Estimation Stage
- Step3. Recover Stage

Step2. Grouped Importance Estimation of Coupled Structure

- Magnitude

$$\text{Score}(w_i) = |w_i|$$

- L_2 norm

$$\text{Score}(w_i) = w_i^2$$

- Loss Change

$$\Delta\mathcal{L} = \mathcal{L}(\mathbf{w} + \Delta\mathbf{w}) - \mathcal{L}(\mathbf{w}) \begin{cases} \textbf{Vector-wise Importance} \\ \\ \textbf{Element-wise Importance} \end{cases}$$

Step2. Grouped Importance Estimation of Coupled Structure

Vector-wise Importance. Suppose that given a dataset $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$, where N is the number of samples. In our experiments, we set N equal to 10 and we use some public datasets as the source of $\mathcal{D}$. A group (as previously defined as a set of coupled structures) can be defined as $\mathcal{G} = \{W_i\}_{i=1}^M$, where M is the number of coupled structures in one group and W_i is the weight for each structure. While pruning, our goal is to remove the group that has the least impact on the model's prediction, which can be indicated by the deviation in the loss. Specially, to estimate the importance of W_i, the change in loss can be formulated as [24]:

$$I_{W_i} = |\Delta\mathcal{L}(\mathcal{D})| = |\mathcal{L}_{W_i}(\mathcal{D}) - \mathcal{L}_{W_i=0}(\mathcal{D})| = |\boxed{\underbrace{\frac{\partial \mathcal{L}^\top(\mathcal{D})}{\partial W_i} W_i}_{\neq 0}} - \frac{1}{2} {W_i}^\top H W_i + \mathcal{O}\left(\|W_i\|^3\right)| \quad (3)$$

where H is the hessian matrix. Here, $\mathcal{L}$ represents the next-token prediction loss. The first term is

$G_1 = \{W_1, W_2, W_3, W_4\}$

Step2. Grouped Importance Estimation of Coupled Structure

Element-wise Importance. The above can be considered as an estimate for the weight W_i. We can derive another measure of importance at a finer granularity, where each parameter within W_i is assessed for its significance:

$$I_{W_i^k} = |\Delta\mathcal{L}(\mathcal{D})| = |\mathcal{L}_{W_i^k}(\mathcal{D}) - \mathcal{L}_{W_i^k=0}(\mathcal{D})| = |\frac{\partial\mathcal{L}(\mathcal{D})}{\partial W_i^k}W_i^k - \frac{1}{2}W_i^k \boxed{H_{kk}} W_i^k + \mathcal{O}\left(\|W_i^k\|^3\right)| \quad (4)$$

Here, k represents the k-th parameter in W_i. The diagonal of the hessian H_{kk} can be approximated by the <u>Fisher information matrix</u>, and the importance can be defined as:

$$I_{W_i^k} = |\mathcal{L}_{W_i^k}(\mathcal{D}) - \mathcal{L}_{W_i^k=0}(\mathcal{D})| \approx |\frac{\partial\mathcal{L}(\mathcal{D})}{\partial W_i^k}W_i^k - \frac{1}{2}\sum_{j=1}^{N}\left(\frac{\partial\mathcal{L}(\mathcal{D}_j)}{\partial W_i^k}W_i^k\right)^2 + \mathcal{O}\left(\|W_i^k\|^3\right)| \quad (5)$$

where N is the number of samples.

$G_1=\{W_1, W_2, W_3, W_4\}$

$w_1^1 \cdots w_1^m$

$G_1=\{\{ w_1^1 \cdots w_1^m \}, \ldots, \{ w_4^1 \cdots w_4^n \}\}$

Step2. Grouped Importance Estimation of Coupled Structure

```
# Linear out_channels
if prune_fn in [tp.prune_linear_out_channels, hf_linear_pruner.prune_out_channels]:
    if self.taylor == 'vectorize':
        local_norm = salience.sum(1).abs()
    elif 'param' in self.taylor:
        local_norm = salience.abs().sum(1)
```

vectorize: $|s_1 + s_2 + \cdots + s_n|$

element: $|s_1| + |s_2| + \cdots + |s_n|$

Step2. Grouped Importance Estimation of Coupled Structure

Group Importance. By utilizing either $I_{W_i^k}$ or I_{W_i}, we estimate the importance at the granularity of either a parameter or a weight. Remembering that our goal is to estimate the importance of $\mathcal{G}$, we aggregate the importance scores in four ways: (i) Summation: $I_{\mathcal{G}} = \sum_{i=1}^{M} I_{W_i}$ or $I_{\mathcal{G}} = \sum_{i=1}^{M} \sum_k I_{W_i^k}$, (ii) Production: $I_{\mathcal{G}} = \prod_{i=1}^{M} I_{W_i}$ or $I_{\mathcal{G}} = \prod_{i=1}^{M} \sum_k I_{W_i^k}$, (iii) Max: $I_{\mathcal{G}} = \max_{i=1}^{M} I_{W_i}$ or $I_{\mathcal{G}} = \max_{i=1}^{M} \sum_k I_{W_i^k}$; (iv) Last-Only: Since deleting the last executing structure in a dependency group is equivalent to erasing all the computed results within that group, we assign the importance of the last executing structure as the importance of the group: $I_{\mathcal{G}} = I_{W_l}$ or $I_{\mathcal{G}} = \sum_k I_{W_l^k}$, where l is the last structure. After assessing the importance of each group, we rank the importance of each group and prune the groups with lower importance based on a predefined pruning ratio.

Method

- Step1. Discovery Stage
- Step2. Estimation Stage
- Step3. Recover Stage

Step3. Recovery Stage

To facilitate this, we employ the low-rank approximation, LoRA[19], to post-train the pruned model. Each learnable weight matrix in the model, denoted as W, encompassing both pruned and unpruned linear projection in the LLM, can be represented as W. The update value ΔW for W can be decomposed as $\Delta W = PQ \in \mathbb{R}^{d^- \times d^+}$, where $P \in \mathbb{R}^{d^- \times d}$ and $Q \in \mathbb{R}^{d \times d^+}$. The forward computation can now be expressed as:

$$f(x) = (W + \Delta W)X + b = (WX + b) + (PQ)X \tag{6}$$

where b is the bias in the dense layer. Only training P and Q reduces the overall training complexity, reducing the need for large-scale training data. Besides, the extra parameters P and Q can be reparameterized into ΔW, which would not cause extra parameters in the final compressed model.

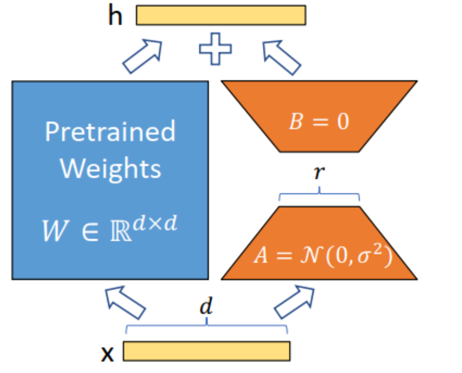


Figure 1: Our reparametrization. We only train A and B.

2021-LoRA-Low Rank Adaptation of Large Language Models

Step3. Recovery Stage

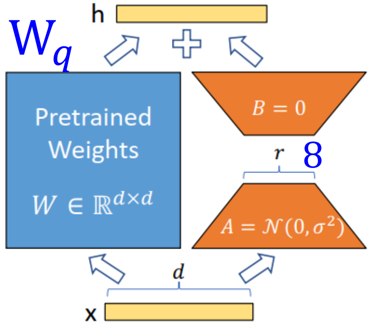


Figure 1: Our reparametrization. We only train A and B.

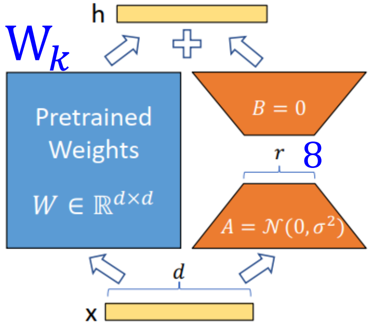


Figure 1: Our reparametrization. We only train A and B.

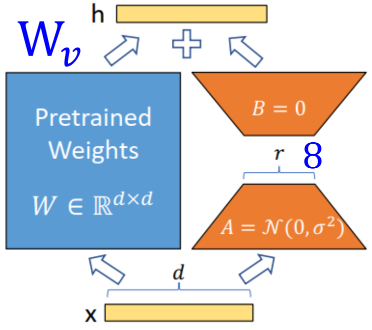


Figure 1: Our reparametrization. We only train A and B.

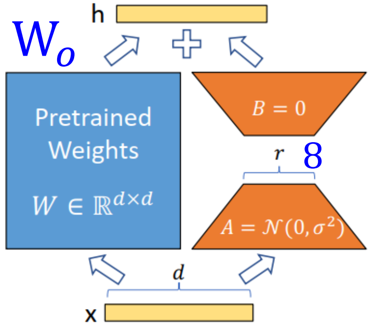


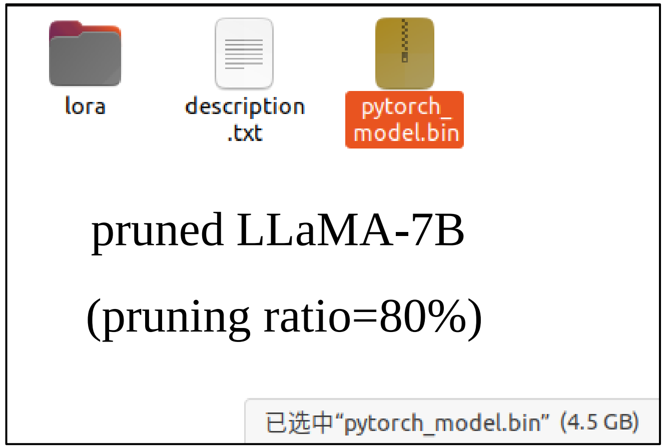
Figure 1: Our reparametrization. We only train A and B.

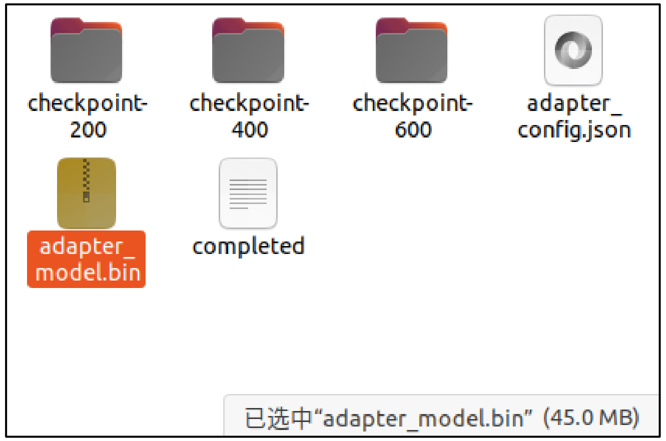
```
(layers): ModuleList(
  (0): LlamaDecoderLayer(
    (self_attn): LlamaAttention(
      (q_proj): Linear(
        in_features=4096, out_features=4096, bias=False
        (lora_dropout): ModuleDict(
          (default): Dropout(p=0.05, inplace=False)
        )
        (lora_A): ModuleDict(
          (default): Linear(in_features=4096, out_features=8, bias=False)
        )
        (lora_B): ModuleDict(
          (default): Linear(in_features=8, out_features=4096, bias=False)
        )
        (lora_embedding_A): ParameterDict()
        (lora_embedding_B): ParameterDict()
      )
      (k_proj): Linear(
        in_features=4096, out_features=4096, bias=False
        (lora_dropout): ModuleDict(
          (default): Dropout(p=0.05, inplace=False)
        )
        (lora_A): ModuleDict(
          (default): Linear(in_features=4096, out_features=8, bias=False)
        )
        (lora_B): ModuleDict(
          (default): Linear(in_features=8, out_features=4096, bias=False)
        )
        (lora_embedding_A): ParameterDict()
        (lora_embedding_B): ParameterDict()
      )
```

Step3. Recovery Stage

For example, finetuning a pruned LLaMA-7B (pruning ratio=50%) with Lora

```
Atrainable params: 14454784 || all params: 3919618048 || trainable%: 0.36878042255611126
```





Experiments

LLaMA-7B, Vicuna-7B, ChatGLM-6B

Structured pruning on bookcorpus

Datasets: bookcorpus

↓

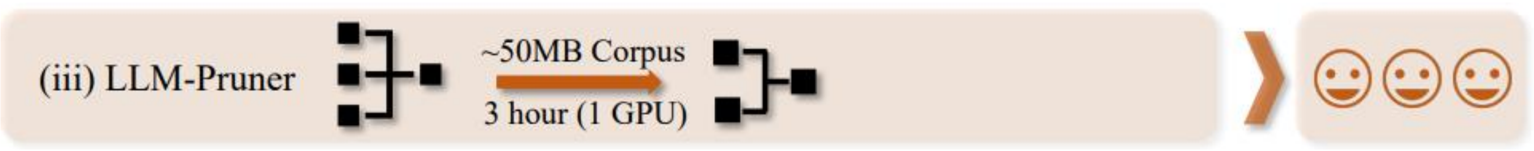
Finetuning on alpaca-cleaned

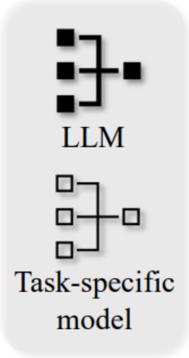
Datasets: yahma/alpaca-cleaned

↓

Inference on two kinds of tasks

- 7 common sense reasoning datasets, such as BoolQ, etc.
- wikitext2 and PTB





One-shot pruning + zero-shot compression

Experiments

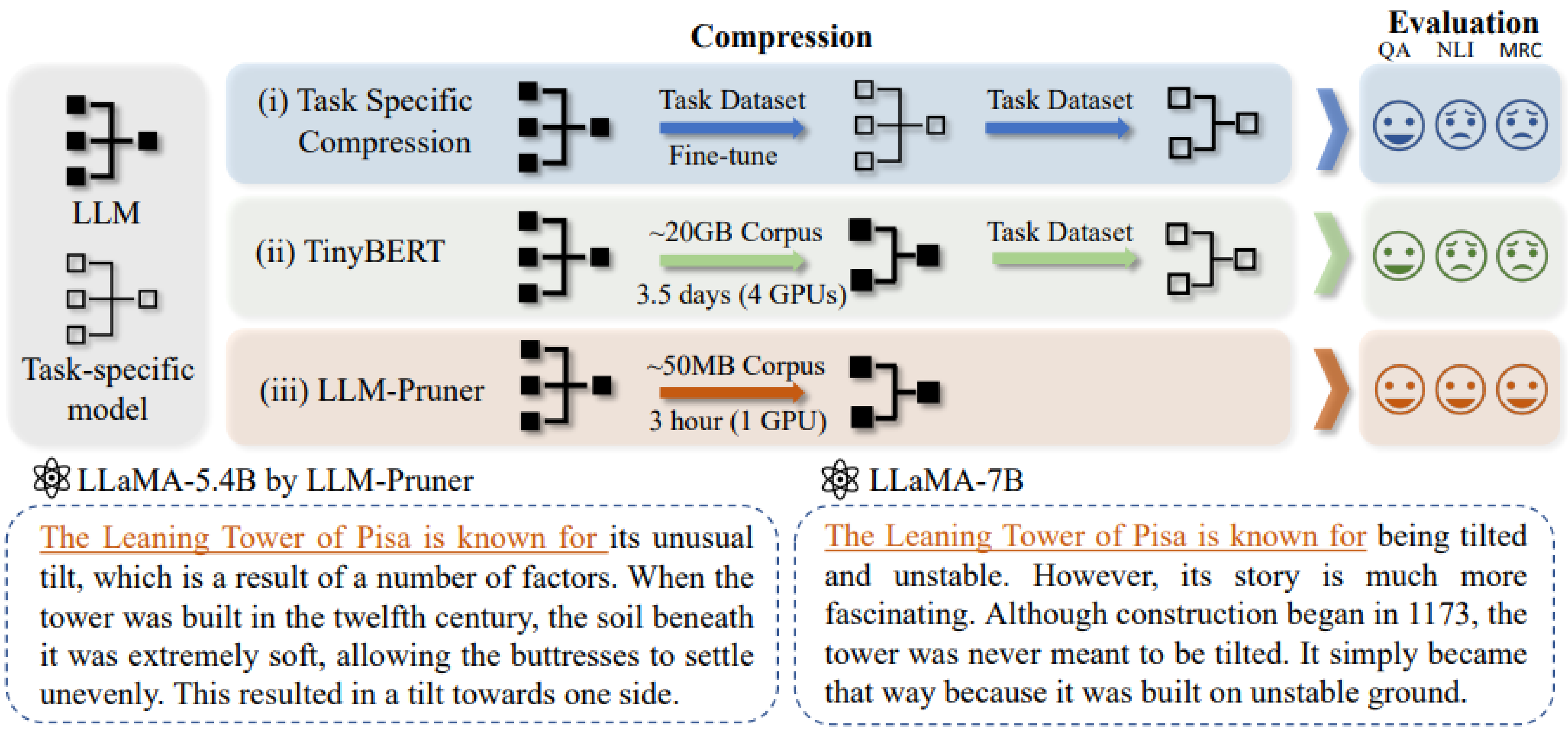


Figure 1: Illustration of LLM-Pruner. (i) Task-specific compression: the model is fine-tuned then compressed on a specific task. (ii) TinyBERT: First distill the model on unlabeled corpus and then fine-tune it on the specific task. (iii) LLM-Pruner: Task-agnostic compression within 3 hours.

Original LLaMA-7B

```
(3): LlamaDecoderLayer(
  (self_attn): LlamaAttention(
    (q_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (k_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (v_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (o_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (rotary_emb): LlamaRotaryEmbedding()
  )
  (mlp): LlamaMLP(
    (gate_proj): Linear(in_features=4096, out_features=11008, bias=False)
    (down_proj): Linear(in_features=11008, out_features=4096, bias=False)
    (up_proj): Linear(in_features=4096, out_features=11008, bias=False)
    (act_fn): SiLUActivation()
  )
  (input_layernorm): LlamaRMSNorm()
  (post_attention_layernorm): LlamaRMSNorm()
)
(4): LlamaDecoderLayer(
  (self_attn): LlamaAttention(
    (q_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (k_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (v_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (o_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (rotary_emb): LlamaRotaryEmbedding()
  )
  (mlp): LlamaMLP(
    (gate_proj): Linear(in_features=4096, out_features=11008, bias=False)
    (down_proj): Linear(in_features=11008, out_features=4096, bias=False)
    (up_proj): Linear(in_features=4096, out_features=11008, bias=False)
    (act_fn): SiLUActivation()
  )
  (input_layernorm): LlamaRMSNorm()
  (post_attention_layernorm): LlamaRMSNorm()
)
```

Pruned LLaMA-7B (pruning ratio=50%)

```
(3): LlamaDecoderLayer(
  (self_attn): LlamaAttention(
    (q_proj): Linear(in_features=4096, out_features=2048, bias=False)
    (k_proj): Linear(in_features=4096, out_features=2048, bias=False)
    (v_proj): Linear(in_features=4096, out_features=2048, bias=False)
    (o_proj): Linear(in_features=2048, out_features=4096, bias=False)
    (rotary_emb): LlamaRotaryEmbedding()
  )
  (mlp): LlamaMLP(
    (gate_proj): Linear(in_features=4096, out_features=5504, bias=False)
    (down_proj): Linear(in_features=5504, out_features=4096, bias=False)
    (up_proj): Linear(in_features=4096, out_features=5504, bias=False)
    (act_fn): SiLUActivation()
  )
  (input_layernorm): LlamaRMSNorm()
  (post_attention_layernorm): LlamaRMSNorm()
)
(4): LlamaDecoderLayer(
  (self_attn): LlamaAttention(
    (q_proj): Linear(in_features=4096, out_features=2048, bias=False)
    (k_proj): Linear(in_features=4096, out_features=2048, bias=False)
    (v_proj): Linear(in_features=4096, out_features=2048, bias=False)
    (o_proj): Linear(in_features=2048, out_features=4096, bias=False)
    (rotary_emb): LlamaRotaryEmbedding()
  )
  (mlp): LlamaMLP(
    (gate_proj): Linear(in_features=4096, out_features=5504, bias=False)
    (down_proj): Linear(in_features=5504, out_features=4096, bias=False)
    (up_proj): Linear(in_features=4096, out_features=5504, bias=False)
    (act_fn): SiLUActivation()
  )
  (input_layernorm): LlamaRMSNorm()
  (post_attention_layernorm): LlamaRMSNorm()
)
```

Experiments

Table 1: Zero-shot performance of the compressed LLaMA-7B. The average is calculated among seven classification datasets. 'Underline' indicates the best pruning-only performance, while 'bold' represents the overall best performance with the same pruning ratio, considering both pruning and post-training. The 'Channel' strategy only prunes the dependent group of Type C, while all other methods employ the 'Block' strategy to prune dependent groups in both Type A and Type B. Since [49] did not provide its prompt, the evaluation of the result with * is performed under different prompts, which is lower than the official results.

Pruning Ratio	Method	WikiText2↓	PTB↓	BoolQ	PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Average
Ratio = 0%	LLaMA-7B[49]	-	-	76.5	79.8	76.1	70.1	72.8	47.6	57.2	68.59
	LLaMA-7B*	12.62	22.14	73.18	78.35	72.99	67.01	67.45	41.38	42.40	63.25
Ratio = 20% w/o tune	L2	582.41	1022.17	59.66	58.00	37.04	52.41	33.12	28.58	29.80	42.65
	Random	27.51	43.19	61.83	71.33	56.26	54.46	57.07	32.85	35.00	52.69
	Channel	74.63	153.75	62.75	62.73	41.40	51.07	41.38	27.90	30.40	45.38
	Vector	22.28	41.78	61.44	71.71	57.27	54.22	55.77	33.96	38.40	53.25
	Element[2]	19.77	36.66	59.39	75.57	65.34	61.33	59.18	37.12	39.80	56.82
	Element[1]	19.09	34.21	57.06	75.68	66.80	59.83	60.94	36.52	**40.00**	56.69
Ratio = 20% w/ tune	Channel	22.02	38.67	59.08	73.39	64.02	60.54	57.95	35.58	38.40	55.57
	Vector	18.84	33.05	65.75	74.70	64.52	59.35	60.65	36.26	39.40	57.23
	Element[2]	**17.37**	30.39	**69.54**	76.44	68.11	**65.11**	63.43	**37.88**	**40.00**	**60.07**
	Element[1]	17.58	**30.11**	64.62	**77.20**	**68.80**	63.14	**64.31**	36.77	39.80	59.23

Experiments

Table 3: Statistics of the base model and the compressed model.

Model	Strategy	Ratio	#Params	#MACs	Memory	Latency
LLaMA-7B Vicuna-7B	-	-	6.74B	424.02G	12884.5MiB	69.32s
	Channel	20%	5.39B	339.36G	10363.6MiB	61.50s
	Block	20%	5.42B	339.60G	10375.5MiB	58.55s
	Channel	50%	3.37B	212.58G	6556.3MiB	40.11s
	Block	50%	3.35B	206.59G	6533.9MiB	37.54s

LLaMA-5.4B by LLM-Pruner

The Leaning Tower of Pisa is known for its unusual tilt, which is a result of a number of factors. When the tower was built in the twelfth century, the soil beneath it was extremely soft, allowing the buttresses to settle unevenly. This resulted in a tilt towards one side.

LLaMA-7B

The Leaning Tower of Pisa is known for being tilted and unstable. However, its story is much more fascinating. Although construction began in 1173, the tower was never meant to be tilted. It simply became that way because it was built on unstable ground.

Experiments

B.2 For Recovery Stage

We follow [19] in our recovery stage. We set the rank d to 8 in our experiment. The learning rate is set to 1e-4 with 100 warming steps. The batch size of training is selected from {64, 128} and the AdamW optimizer is employed in our experiment. The best training epoch we found is 2 epochs, as training with more epochs even has a negative impact on the model performance. We run our experiment on a single GPU with 24GB memory, using approximately 2.5 hours if RTX4090 is utilized. All the linear module is taken into account for efficient tuning. An ablation experiment for this is shown in Table 11.

Table 11: Ablation: Tuning different modules in the recovery stage

Module	WikiText ↓	PTB ↓
ALL	17.36	29.99
- MLP	17.64	30.63
- MHA	17.62	30.23

Experiments

Channel Strategy vs. Block Strategy. From the results presented in Table 2, it is evident that pruning 'Channel' significantly deteriorates performance compared to pruning 'Block'. This discrepancy arises because the layers within the stacked transformer do not evenly distribute their importance. As shown in Figure 3, the first and last layers have a profound impact on the model's performance, and pruning them results in more substantial performance degradation compared to other layers. However, due to the uniform treatment of the 'Channel' group across all layers, it becomes inevitable to prune the first and last layers, leading to a significant decline in performance.

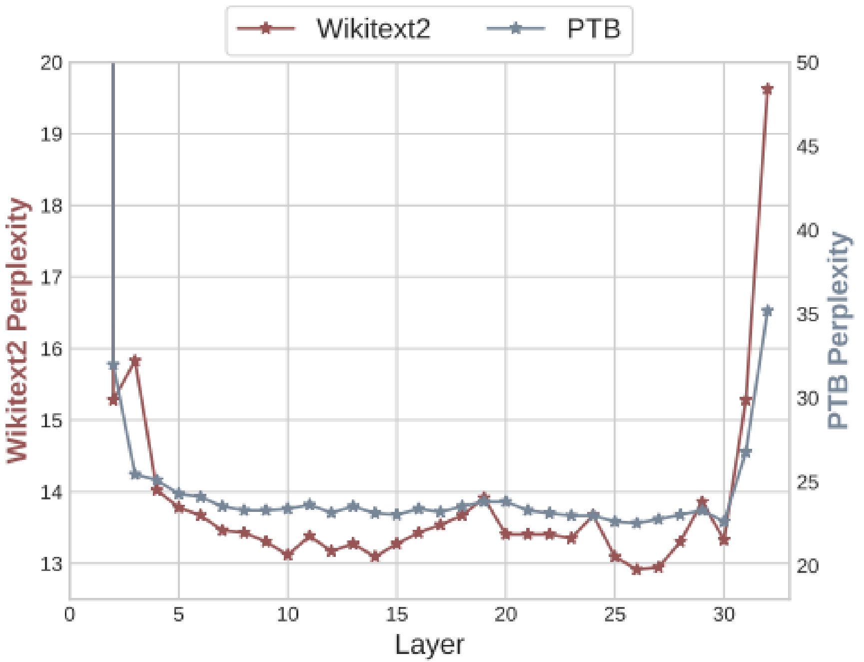


Figure 3: Layer sensitivity for Pruning: Removing Groups in only one layer.

Experiments

Impact of Different Aggregation Strategies. We conduct tests on the aggregation algorithms proposed in Section 3.2. Our experimental results unveil notable discrepancies in model performance across different aggregation strategies, with particular emphasis on the 'Last-only' strategy. Among the evaluated approaches, the 'Max' strategy attains the most favorable outcomes in terms of perplexity, signifying enhanced coherence and fluency in sentence generation. However, it is important to note that the 'Max' strategy exhibits the poorest zero-shot classification results compared to all four strategies. Conversely, the 'Last-only' strategy showcases superior classification performance but

Table 7: Impact of different aggregation strategies on group importance estimation. Experiments are performed on LLaMA-7B.

Method	WikiText2↓	PTB↓	ARC-e↑	PIQA↑	OBQA↑
Summation	66.13	164.25	40.70	63.49	34.80
Max	62.59	144.38	39.60	63.71	34.60
Production	77.63	192.88	37.84	62.08	35.00
Last-only	130.00	170.88	41.92	64.75	35.20

Experiments

C.2 Pruning vs. Quantization

Here, we conduct a comparative analysis of different compression techniques and illustrate that these techniques can be effectively combined with little performance degradation. We have chosen LLM.int8() [8] as a representative example of quantization methods. Our results show that LLM.int8() outperforms LLM-Pruner while LLM-Pruner enhances latency, reduces parameter size. When these two techniques are applied in tandem, they collectively reduce memory consumption and expedite inference, offering a balanced approach that combines the benefits of both methods.

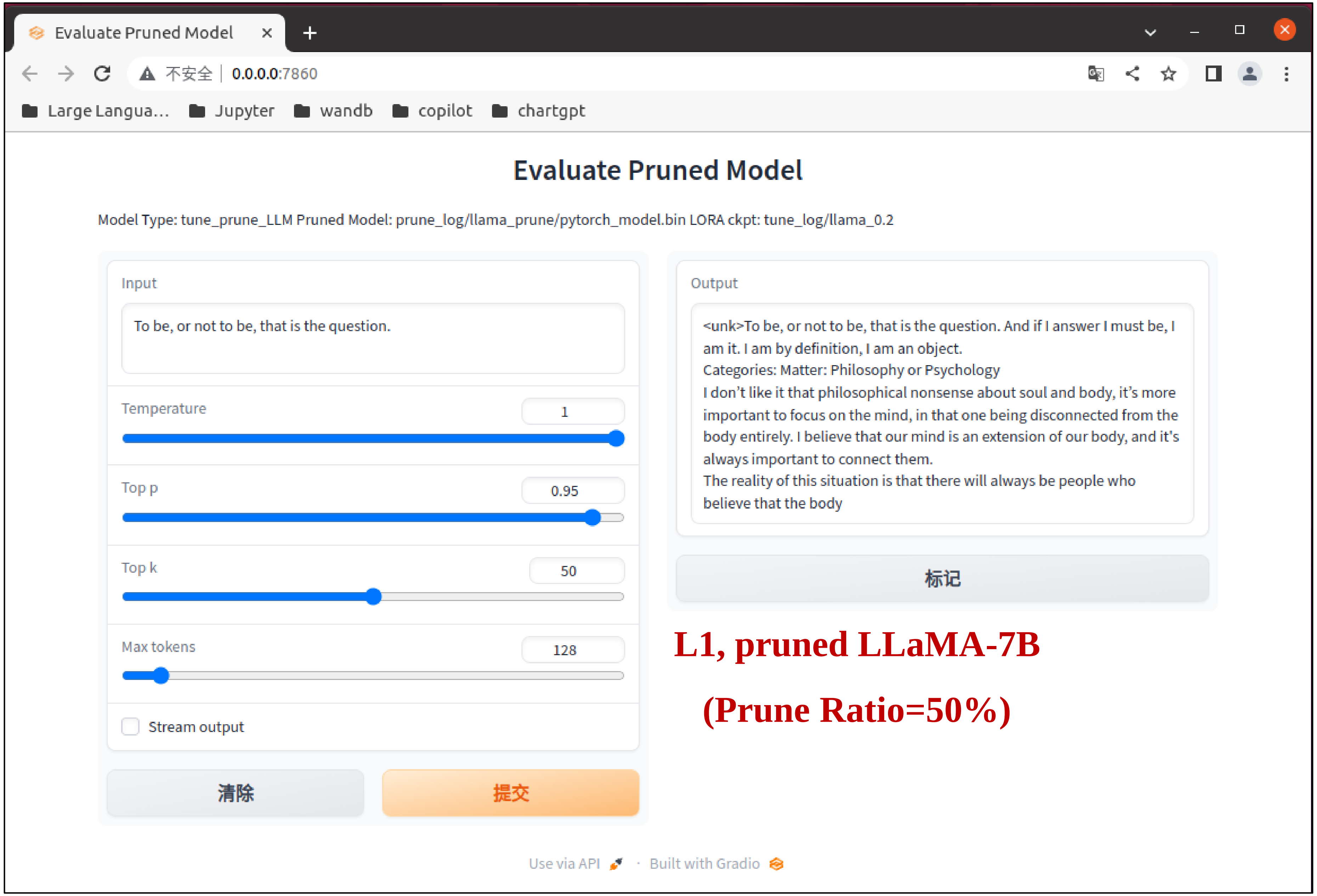
Table 13: Pruning and Quantization on LLaMA-7B

Pruning Ratio	#Param	Memory	Latency	BoolQ	PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Average
LLaMA-7B	6.74B	12884.5MiB	69.32s	73.18	78.35	72.99	67.01	67.45	41.38	42.40	63.25
LLM.int8()	6.74B	6777.7MiB	76.20s	73.36	78.18	73.01	66.93	67.47	40.87	41.80	63.09
LLaMA-5.4B	5.47B	10488.4MiB	58.55s	76.57	77.37	66.60	65.82	70.62	40.70	38.80	62.36
LLaMA-5.4B + LLM.int8()	5.47B	5444.37MiB	63.10s	76.39	76.71	66.62	66.46	70.54	40.19	39.20	62.30

Experiments

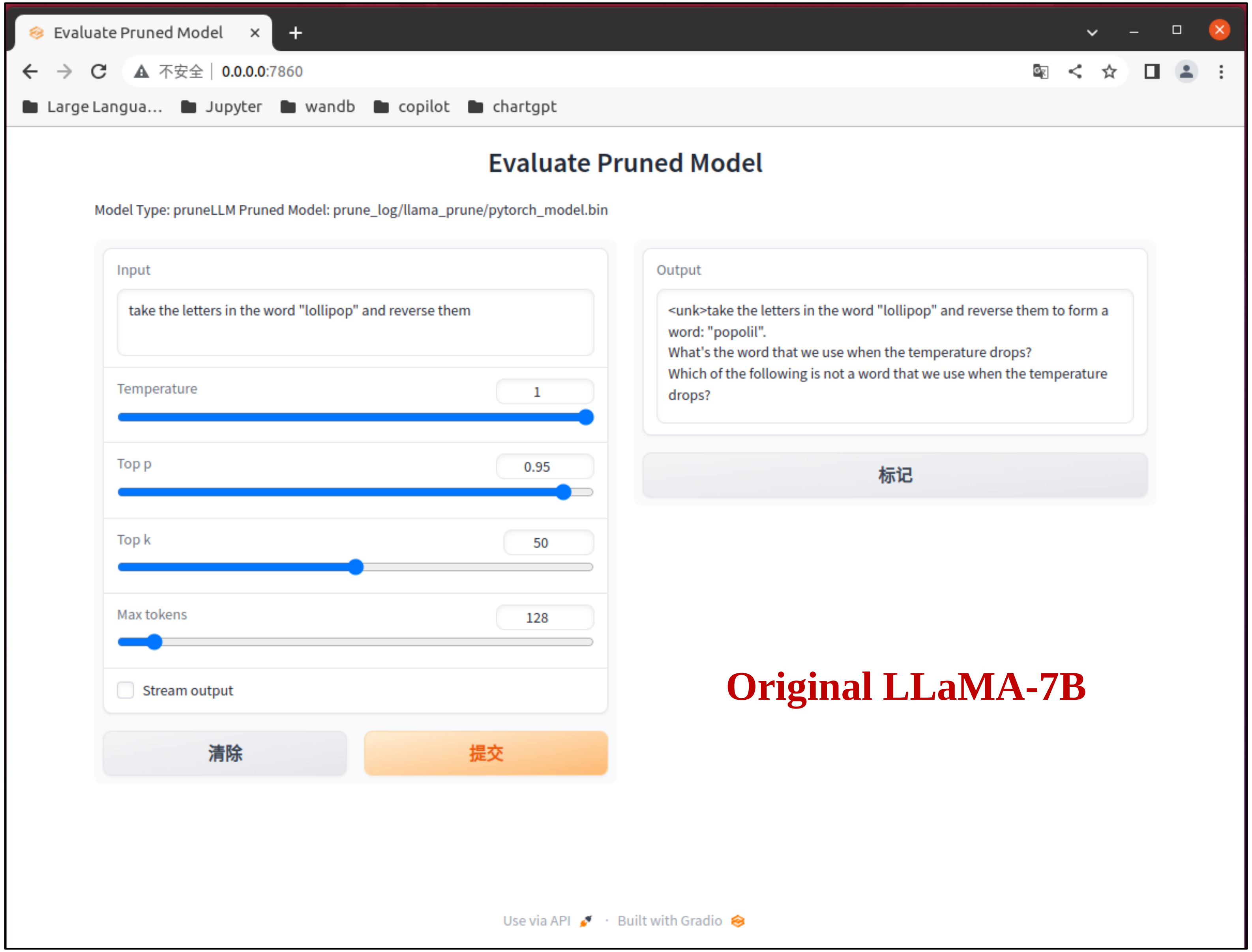
Table 10: Some examples of generations obtained with the original model and the pruned model.

Model	Original Model	Pruned Model
LLaMa	The universe is the entirety of space, time, matter, and energy that exists. It includes all matter and energy, whether visible or not, and in fact the whole universe is invisible. There are many theories regarding its origin and ultimate fate...	The universe is the entirety of space, time, matter, and energy that exists. The laws that make up physics — electrons orbiting atoms, magnets attracting other objects, planets orbiting the Sun—have been consistent over ...
Vicuna	10 steps to build an ios app: 1. Research and develop the idea. 2. Decide on the platform you want to use (iOS, Android, or both). 3. Create a plan and budget. 4. Hire a team of developers. 5. Design the user interface (UI). 6. Develop the user interface (UI). 7. Test and iterate on the app. 8. Submit the app to the app store. 9. Optimize the app for performance and user experience (UX). 10. Market and promote the app.	10 steps to build an ios app: 1. Plan your app: decide what it should do, what it will look like, and what it will cost to build it. 2. Hire a developer: A good iOS developer will help you design the user interface, create the app, and make sure it's bug-free. 3. Create a user interface: This is where the magic happens. Your app's user interface will be designed and mocked up at this point...
ChatGLM	Write a poetry about love and peace. In the midst of life's hustle and bustle, I find your love a soothing peace. It is a feeling that cannot be conveyed, A connection that cannot be distanced. With your smile, your eyes, your embrace, I feel a sense of belonging that only you can give. It is a love that lasts forever, A love that cannot be defeat...	Write a poetry about love and peace. Love and peace, two such wonderful things\ A pure and endless desire in my heart\ And both of them, I must seek for\ A long, long time, I know..\ Love, I know, is a feeling of being\ A perfect partner, in every sense\ And peace, I need it, so much, one day\ A long, long way, my heart will go..

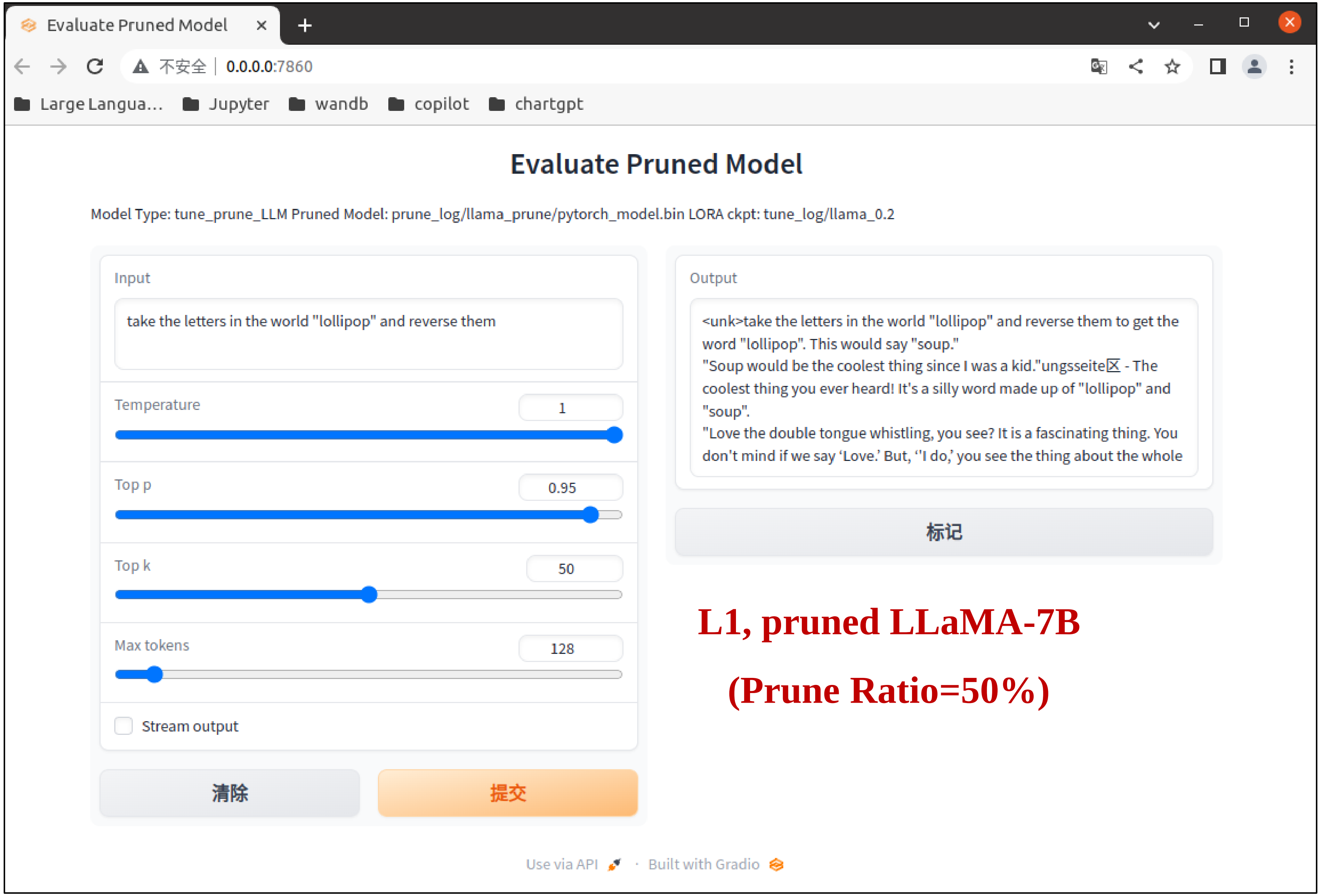


L1, pruned LLaMA-7B

(Prune Ratio=50%)



Original LLaMA-7B



L1, pruned LLaMA-7B

(Prune Ratio=50%)

ChatGPT3.5 / ChatGPT4

ChatGPT 3.5

You
take the letters in the word "lollipop" and reverse them

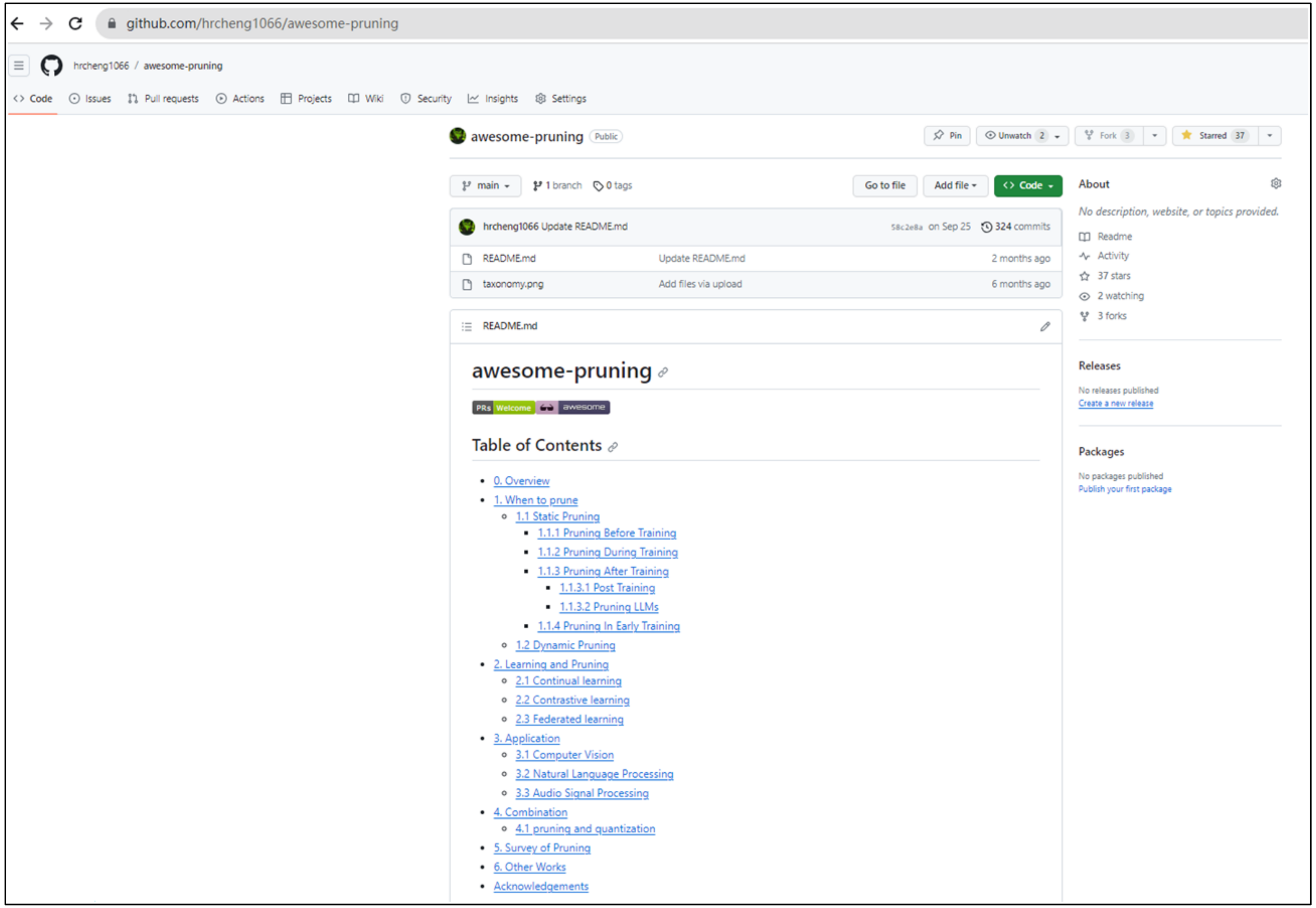
ChatGPT
Sure, if we reverse the letters in the word "lollipop," it becomes "poppilol."

ChatGPT 4

You
take the letters in the word "lollipop" and reverse them

ChatGPT
The word "lollipop" reversed is "popillol".

Ours Survey on Pruning



https://github.com/hrcheng1066/awesome-pruning

Thanks