FOM Hochschule für Ökonomie & Management

Studienzentrum Frankfurt am Main

Zusätzliche Beteiligung

zum Thema

Analyse der Auswirkungen von Pruning an *Large Language Models*

von

Kai Daniel Herbst und Sebastian Viet

28.02.2025

Gutachter:	Prof. Dr. Rüdiger Buchkremer
Zeitraum:	01.09.2024 - 28.02.2025
Semester:	3
Modul:	Big-Data-Analyseprojekt

Inhaltsverzeichnis

1 Einleitung (Kai Herbst)

Im Rahmen des Moduls *Big-Data-Analyseprojekt* wird das *Pruning* von *Large Lanugage Models* (LLM) theoretisch untersucht und anschließend in einem praktisch Versuch mit Hilfe des Python-Packages *LLM-Pruner* selbst am Llama-Modell *TinyLlama/TinyLlama-1.1B-Chat-v1.0* durchgeführt. Die aus dem Pruning resultierenden Modelle werden anschließend den Ergebnissen verschiedenen Tests evaluiert und mit dem originalen Modell verglichen.

Die dafür verwendeten Aufrufe der Skripte des *LLM-Pruners* und dazugehörige Konsolenausgabe sind in dieser Arbeit dokumentiert.

2 Versuchsumgebung (Kai Herbst)

Das Prungin wurde aufgrund der erforderlichen Rechenleistung auf einer EC2-Instanz in AWS (Amazon Web Services) durchgeführt. Dabei verwendet wurde eine EC2-Instanz vom Instanz-Typ *gd4n.xlarge* mit 16GiB Hauptspeicher und vier vCPU.

Zusätzlich wurde das von AWS bereitgestellte *Pytorch Deep Learning* AMI verwendet. Darauf vorinstalliert ist bereits eine Conda-Umgebung mit PyTorch und alle benötigten CUDA-Treiber, um PyTorch die Möglichkeit zu bieten die GPU zu verwenden.

In Tabelle 1 sind die genauen Spezifikationen der verwendeten Instanz aufgelistet.

Verwendete EC2-Instanz	
Instanz-Typ	gd4n.xlarge
Hauptspeicher	16GiB
vCPU	4
Clock Speed	2.5GHz
GPU	nvidia t4 tensor core
CPU	Intel Xeon Family
AMI	Deep Learning OSS Nvidia Driver AMI GPU PyTorch 2.5.1
AMI-ID	ami-0fa5e5fd27b3e163a

Tabelle 1: Attribute der verwendeten Hardware

Über einen in der AWS-Konsole für die Instanz erstellten Schlüssel kann sich anschließend über ssh mit der Instanz verbunden werden.

Anschließend wird die PyTorch-Umgebung aufgerufen und das GitHub-Repository des *LLM-Pruners* geklont.

```
1    $ source activate pytorch
2    $ git clone https://github.com/horseee/LLM-Pruner.git
```

Die in der *requirements.txt* spezifizierten Packages können sowohl mit Conda als

auch mit Pip installiert werden.

3 Basismodell (Kai Herbst)

Für das Pruning wurde das LLM *TinyLlama/TinyLlama-1.1B-Chat-v1.0* verwendet.

Mit folgendem Skript wurden genauere Informationen über die Architektur des Modells generiert:

```
from transformers import AutoModel

model = AutoModel.from_pretrained("TinyLlama/TinyLlama-1.1B-Chat-v1.0")
print(model)
```

Das Skript erzeugt folgende Ausgabe:

```
$ python printModel.py

LlamaModel(
  (embed_tokens): Embedding(32000, 2048)
  (layers): ModuleList(
    (0-21): 22 x LlamaDecoderLayer(
      (self_attn): LlamaSdpaAttention(
        (q_proj): Linear(in_features=2048, out_features=2048, bias=False)
        (k_proj): Linear(in_features=2048, out_features=256, bias=False)
        (v_proj): Linear(in_features=2048, out_features=256, bias=False)
        (o_proj): Linear(in_features=2048, out_features=2048, bias=False)
        (rotary_emb): LlamaRotaryEmbedding()
      )
      (mlp): LlamaMLP(
        (gate_proj): Linear(in_features=2048, out_features=5632, bias=False)
        (up_proj): Linear(in_features=2048, out_features=5632, bias=False)
        (down_proj): Linear(in_features=5632, out_features=2048, bias=False)
        (act_fn): SiLU()
      )
      (input_layernorm): LlamaRMSNorm((2048,), eps=1e-05)
      (post_attention_layernorm): LlamaRMSNorm((2048,), eps=1e-05)
    )
  )
  (norm): LlamaRMSNorm((2048,), eps=1e-05)
  (rotary_emb): LlamaRotaryEmbedding()
)
```

4 Pruning (Kai Herbst)

Im Nachfolgenden sind die Befehle und Teile der Ausgabe aufgelistet, die für das Pruning des Basismodells verwendet wurde. Zu beachten ist, dass beim Pruning selbst in der Konsole bereits Angaben zu der verbleibenden Anzahl an Parametern generiert werden. Wie in der Hauptarbeit jedoch beschrieben, wurde die Anazahl an Parametern nach dem Pruning mit dem Evaluierungsskript im *LLM-Pruner* gemessen, das selbst die Anzahl an Parametern ausliest. Für die weitere Evaluierung wurden die Anzahl der Parameter verwendet, die durch das Evaluierungsskript generiert wurden, da dieses Skript anhand dieser Angabe die benötigte Rechenleistung berechnet.

Aufgelistet sind hier die verwendeten Befehle zum Prunen des Modells mit der Angabe das Modell um 30% mit den vier möglichen Methoden reduzieren. Für die 40% und 70% sind die Befehle identisch, außer, dass die *–prungin_ratio* entsprechend auf 0.3 bzw. 0.7 gesetzt und der Speicherpfad angepasst wurde.

Pruning des Modells mit der Taylor-Methode:

```
$ python llama3.py
        --base_model TinyLlama/TinyLlama-1.1B-Chat-v1.0
        --pruning_ratio 0.3
        --device cuda
        --eval_device cuda
        --block_wise
        --block_mlp_layer_start 4
        --block_mlp_layer_end 18
        --block_attention_layer_start 4
        --block_attention_layer_end 18
        --save_ckpt_log_name tinyllama_30_0418_prune_log
        --pruner_type taylor
        --taylor param_first
        --save_model
        --max_seq_len 2048
        --test_after_train
```

```
2025-01-04 15:57:27 - INFO :       We will use 90% of the memory on device 0
    for storing the model, and 10% for the buffer to avoid OOM. You can set
    ‘max_memory‘ in to a higher value to use more memory (at your own risk).
```

```
2025-01-04 15:57:29 - INFO :        Use taylor pruner...
2025-01-04 15:57:29 - INFO :        Pruning Attention Layer = [4, 5, 6, 7, 8, 9,
    10, 11, 12, 13, 14, 15, 16, 17]
2025-01-04 15:57:29 - INFO :        Pruning MLP Layer = [4, 5, 6, 7, 8, 9, 10,
    11, 12, 13, 14, 15, 16, 17]
2025-01-04 15:57:29 - INFO :        Start Pruning
2025-01-04 15:57:32 - INFO :        Start Backwarding in iterative steps = 0...
2025-01-04 15:57:32 - INFO :        Loss = 3.7723865509033203
2025-01-04 15:57:33 - INFO :        After Iter 1/1, #parameters: 921651200
2025-01-04 15:57:33 - INFO :        #Param before: 1100048384, #Param after:
    921651200, Ratio = 83.7828%
...
2025-01-04 15:59:37 - INFO :        PPL after pruning: {'wikitext2':
    15.190999662372043, 'ptb': 43.19069160819034}
2025-01-04 15:59:37 - INFO :        Memory Requirement: 4130.3818359375 MiB
```

Pruning des Modells mit der L1-Methode:

```
    $ python llama3.py
            --base_model TinyLlama/TinyLlama-1.1B-Chat-v1.0
            --pruning_ratio 0.3
            --device cuda
            --eval_device cuda
            --block_wise
            --block_mlp_layer_start 4
            --block_mlp_layer_end 18
            --block_attention_layer_start 4
            --block_attention_layer_end 18
            --save_ckpt_log_name tinyllama_30_0418_l1_prune_log
            --pruner_type l1
            --save_model
            --max_seq_len 2048
            --test_after_train
```

```
2025-01-04 14:31:44 - INFO :        We will use 90% of the memory on device 0
    for storing the model, and 10% for the buffer to avoid OOM. You can set
    'max_memory' in to a higher value to use more memory (at your own risk).
2025-01-04 14:31:46 - INFO :        Use l1 pruner...
2025-01-04 14:31:46 - INFO :        Pruning Attention Layer = [4, 5, 6, 7, 8, 9,
    10, 11, 12, 13, 14, 15, 16, 17]
2025-01-04 14:31:46 - INFO :        Pruning MLP Layer = [4, 5, 6, 7, 8, 9, 10,
    11, 12, 13, 14, 15, 16, 17]
2025-01-04 14:31:46 - INFO :        Start Pruning
2025-01-04 14:31:47 - INFO :        After Iter 1/1, #parameters: 921651200
2025-01-04 14:31:47 - INFO :        #Param before: 1100048384, #Param after:
    921651200, Ratio = 83.7828%
```

```
...
2025-01-04 14:35:14 - INFO :       PPL after pruning: {'wikitext2':
    281.63868765896893, 'ptb': 4992.165187767088}
2025-01-04 14:35:14 - INFO :       Memory Requirement: 1783.6884765625 MiB
```

Pruning des Modells mit der L2-Methode:

```
    $ python llama3.py
            --base_model TinyLlama/TinyLlama-1.1B-Chat-v1.0
            --pruning_ratio 0.3
            --device cuda
            --eval_device cuda
            --block_wise
            --block_mlp_layer_start 4
            --block_mlp_layer_end 18
            --block_attention_layer_start 4
            --block_attention_layer_end 18
            --save_ckpt_log_name tinyllama_30_0418_l2_prune_log
            --pruner_type l2
            --save_model
            --max_seq_len 2048
            --test_after_train
```

```
2025-01-04 14:38:13 - INFO :       We will use 90% of the memory on device 0
    for storing the model, and 10% for the buffer to avoid OOM. You can set
    'max_memory' in to a higher value to use more memory (at your own risk).
2025-01-04 14:38:15 - INFO :       Use l2 pruner...
2025-01-04 14:38:15 - INFO :       Pruning Attention Layer = [4, 5, 6, 7, 8, 9,
    10, 11, 12, 13, 14, 15, 16, 17]
2025-01-04 14:38:15 - INFO :       Pruning MLP Layer = [4, 5, 6, 7, 8, 9, 10,
    11, 12, 13, 14, 15, 16, 17]
2025-01-04 14:38:16 - INFO :       Start Pruning
2025-01-04 14:38:16 - INFO :       After Iter 1/1, #parameters: 921651200
2025-01-04 14:38:16 - INFO :       #Param before: 1100048384, #Param after:
    921651200, Ratio = 83.7828%
...
2025-01-04 14:41:14 - INFO :       PPL after pruning: {'wikitext2':
    42.190180314697045, 'ptb': 167.51849280886478}
2025-01-04 14:41:14 - INFO :       Memory Requirement: 1783.6904296875 MiB
```

Pruning des Modells mit der Random-Methode:

```
    $ python llama3.py
            --base_model TinyLlama/TinyLlama-1.1B-Chat-v1.0
            --pruning_ratio 0.3
```

```
    --device cuda
    --eval_device cuda
    --block_wise
    --block_mlp_layer_start 4
    --block_mlp_layer_end 18
    --block_attention_layer_start 4
    --block_attention_layer_end 18
    --save_ckpt_log_name tinyllama_30_0418_random_prune_log
    --pruner_type random
    --save_model
    --max_seq_len 2048
    --test_after_train
```

```
2025-01-04 15:50:55 - INFO :        We will use 90% of the memory on device 0 for storing the model, and 10% for the buffer to avoid OOM. You can set `max_memory` in to a higher value to use more memory (at your own risk).
2025-01-04 15:50:57 - INFO :        Use random pruner...
2025-01-04 15:50:57 - INFO :        Pruning Attention Layer = [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
2025-01-04 15:50:57 - INFO :        Pruning MLP Layer = [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
2025-01-04 15:50:57 - INFO :        Start Pruning
2025-01-04 15:50:57 - INFO :        After Iter 1/1, #parameters: 921651200
2025-01-04 15:50:57 - INFO :        #Param before: 1100048384, #Param after: 921651200, Ratio = 83.7828%
...
2025-01-04 15:53:04 - INFO :        PPL after pruning: {'wikitext2': 40.892124929554, 'ptb': 166.21485355527523}
2025-01-04 15:53:04 - INFO :        Memory Requirement: 1793.31396484375 MiB
```

5 Performance-Evaluierung (Kai Herbst)

Auch hier wurde für die entsprechenden Evaluierungen die nahezu identischen Aufrufe verwendet. Im folgenden aufgelistet sind daher der Befehl zur Evaluierung des Basismodells und der Befehl zur Evaluierung eines geprunten Modells.

```
$ export PYTHONPATH='.'
$ python lm-evaluation-harness/main.py
            --model hf-causal-experimental
            --model_args pretrained=TinyLlama/TinyLlama-1.1B-Chat-v1.0
            --tasks openbookqa,winogrande,hellaswag,arc_challenge,boolq
            --no_cache
            --output_path results/tinyllama-1.1B-Chat-v1.0
            --device cuda:0
```

Dazugehöriger relevanter Ausschnitt der Konsolenausgabe:


```
{
  "results": {
    "arc_challenge": {
      "acc": 0.30119453924914674,
      "acc_stderr": 0.01340674176784762,
      "acc_norm": 0.3250853242320819,
      "acc_norm_stderr": 0.013688147309729119
    },
    "boolq": {
      "acc": 0.6131498470948012,
      "acc_stderr": 0.00851818834084475
    },
    "openbookqa": {
      "acc": 0.242,
      "acc_stderr": 0.019173085678337164,
      "acc_norm": 0.376,
      "acc_norm_stderr": 0.021683827539286125
    },
    "winogrande": {
      "acc": 0.6029992107340174,
      "acc_stderr": 0.013751092519806702
    },
    "hellaswag": {
      "acc": 0.4614618601872137,
      "acc_stderr": 0.004974937803907464,
      "acc_norm": 0.5827524397530373,
      "acc_norm_stderr": 0.004920967192255295
```

```
    }
  },
  "versions": {
    "arc_challenge": 0,
    "boolq": 1,
    "openbookqa": 0,
    "winogrande": 0,
    "hellaswag": 0
  },
  "config": {
    "model": "hf-causal-experimental",
    "model_args": "pretrained=TinyLlama/TinyLlama-1.1B-Chat-v1.0",
    "num_fewshot": 0,
    "batch_size": null,
    "device": "cuda:0",
    "no_cache": true,
    "limit": null,
    "bootstrap_iters": 100000,
    "description_dict": {}
  }
}
hf-causal-experimental (pretrained=TinyLlama/TinyLlama-1.1B-Chat-v1.0), limit:
    None, provide_description: False, num_fewshot: 0, batch_size: None
|     Task      |Version| Metric |Value |   |Stderr|
|---------------|------:|--------|-----:|---|-----:|
|arc_challenge|      0|acc     |0.3012|+-  |0.0134|
|               |       |acc_norm|0.3251|+-  |0.0137|
|boolq          |      1|acc     |0.6131|+-  |0.0085|
|openbookqa     |      0|acc     |0.2420|+-  |0.0192|
|               |       |acc_norm|0.3760|+-  |0.0217|
|winogrande     |      0|acc     |0.6030|+-  |0.0138|
|hellaswag      |      0|acc     |0.4615|+-  |0.0050|
|               |       |acc_norm|0.5828|+-  |0.0049|
```

Aufruf zur Evaluierung eines geprunten Modells:

```
    $ export PYTHONPATH='.'
    $ python lm-evaluation-harness/main.py
        --model hf-causal-experimental
        --model_args
            checkpoint=prune_log/tinyllama_30_0418_prune_log/pytorch_model.bin,
            config_pretrained=TinyLlama/TinyLlama-1.1B-Chat-v1.0
        --tasks
            openbookqa,arc_easy,winogrande,hellaswag,arc_challenge,piqa,boolq
        --device cuda:0
        --no_cache
```

```
        --output_path results/tinyllama_30_0418
```


```
{
  "results": {
    "hellaswag": {
      "acc": 0.379008165704043,
      "acc_stderr": 0.00484148671685576,
      "acc_norm": 0.48376817367058356,
      "acc_norm_stderr": 0.004987151381091178
    },
    "arc_challenge": {
      "acc": 0.24658703071672355,
      "acc_stderr": 0.012595726268790125,
      "acc_norm": 0.29180887372013653,
      "acc_norm_stderr": 0.013284525292403511
    },
    "winogrande": {
      "acc": 0.5422257300710339,
      "acc_stderr": 0.014002284504422438
    },
    "openbookqa": {
      "acc": 0.232,
      "acc_stderr": 0.01889619359195206,
      "acc_norm": 0.362,
      "acc_norm_stderr": 0.021513662527582404
    },
    "boolq": {
      "acc": 0.555045871559633,
      "acc_stderr": 0.008691897543539214
    }
  },
  "versions": {
    "hellaswag": 0,
    "arc_challenge": 0,
    "winogrande": 0,
    "openbookqa": 0,
    "boolq": 1
  },
  "config": {
    "model": "hf-causal-experimental",
    "model_args":
        "checkpoint=prune_log/tinyllama_10_0616_prune_log/pytorch_model.bin,
␣␣␣␣␣␣␣␣config_pretrained=TinyLlama/TinyLlama-1.1B-Chat-v1.0",
    "num_fewshot": 0,
    "batch_size": null,
    "device": "cuda:0",
```

```
    "no_cache": true,
    "limit": null,
    "bootstrap_iters": 100000,
    "description_dict": {}
  }
}
hf-causal-experimental (checkpoint=prune_log/tinyllama_30_0418_prune_log/pytorch_model.bin,
config_pretrained=TinyLlama/TinyLlama-1.1B-Chat-v1.0), limit: None,
    provide_description: False, num_fewshot: 0, batch_size: None
|     Task     |Version| Metric |Value |   |Stderr|
|--------------|------:|--------|-----:|---|-----:|
|hellaswag     |      0|acc     |0.3790|+-  |0.0048|
|              |       |acc_norm|0.4838|+-  |0.0050|
|arc_challenge|      0|acc     |0.2466|+-  |0.0126|
|              |       |acc_norm|0.2918|+-  |0.0133|
|winogrande    |      0|acc     |0.5422|+-  |0.0140|
|openbookqa    |      0|acc     |0.2320|+-  |0.0189|
|              |       |acc_norm|0.3620|+-  |0.0215|
|boolq         |      1|acc     |0.5550|+-  |0.0087|
```

6 Tuning (Kai Herbst)

Wie in der Hausarbeit beschrieben, wurde eines der Modelle nach dem Pruning nachtrainiert. Dazu wurde folgender Befehl verwendet. Damit die Verbindung mit Wandb funktioniert, muss sich entsprechend vorher ein Account erstellt werden und sich mit diesem über das Terminal angemeldet werden.

```
1    python post_training.py --prune_model
         prune_log/tinyllama_30_0418_prune_log/pytorch_model.bin
2          --data_path yahma/alpaca-cleaned
3          --lora_r 8
4          --num_epochs 2
5          --learning_rate 1e-4
6          --batch_size 64
7          --output_dir tune_log/tl30
8          --wandb_project llama_tune
```

7 Evaluierung Rechenanforderung (Kai Herbst)

Im folgenden Text sind erneut die Befehle für die Evaluierung des Basismodells aufgelistet und für eines der 30% Modelle sowie den dazugehörigen Output im Terminal.

Evaulierung des Basismodells:

```
$ python test_speedup.py
    --model_type pretrain
    --base_model TinyLlama/TinyLlama-1.1B-Chat-v1.0
```

Dazugehörige Ausgabe:

```
LlamaForCausalLM(
  1195.99 M, 94.805% Params, 76.94 GMac, 99.509% MACs,
  (model): LlamaModel(
    1130.46 M, 89.610% Params, 72.75 GMac, 94.084% MACs,
    (embed_tokens): Embedding(0, 0.000% Params, 0.0 Mac, 0.000% MACs, 32000,
        2048)
    (layers): ModuleList(
      (0-21): 22 x LlamaDecoderLayer(
        51.38 M, 4.073% Params, 3.31 GMac, 4.277% MACs,
        (self_attn): LlamaAttention(
          16.78 M, 1.330% Params, 1.09 GMac, 1.411% MACs,
          (q_proj): Linear(4.19 M, 0.332% Params, 268.44 MMac, 0.347% MACs,
              in_features=2048, out_features=2048, bias=False)
          (k_proj): Linear(4.19 M, 0.332% Params, 268.44 MMac, 0.347% MACs,
              in_features=2048, out_features=2048, bias=False)
          (v_proj): Linear(4.19 M, 0.332% Params, 268.44 MMac, 0.347% MACs,
              in_features=2048, out_features=2048, bias=False)
          (o_proj): Linear(4.19 M, 0.332% Params, 268.44 MMac, 0.347% MACs,
              in_features=2048, out_features=2048, bias=False)
          (rotary_emb): LlamaRotaryEmbedding(0, 0.000% Params, 0.0 Mac, 0.000%
              MACs, )
        )
        (mlp): LlamaMLP(
          34.6 M, 2.743% Params, 2.21 GMac, 2.865% MACs,
          (gate_proj): Linear(11.53 M, 0.914% Params, 738.2 MMac, 0.955% MACs,
              in_features=2048, out_features=5632, bias=False)
          (down_proj): Linear(11.53 M, 0.914% Params, 738.2 MMac, 0.955% MACs,
              in_features=5632, out_features=2048, bias=False)
          (up_proj): Linear(11.53 M, 0.914% Params, 738.2 MMac, 0.955% MACs,
              in_features=2048, out_features=5632, bias=False)
```

```
22            (act_fn): SiLU(0, 0.000% Params, 360.45 KMac, 0.000% MACs, )
23          )
24          (input_layernorm): LlamaRMSNorm(2.05 k, 0.000% Params, 262.14 KMac,
                0.000% MACs, )
25          (post_attention_layernorm): LlamaRMSNorm(2.05 k, 0.000% Params, 262.14
                KMac, 0.000% MACs, )
26        )
27      )
28      (norm): LlamaRMSNorm(2.05 k, 0.000% Params, 262.14 KMac, 0.000% MACs, )
29    )
30    (lm_head): Linear(65.54 M, 5.195% Params, 4.19 GMac, 5.425% MACs,
          in_features=2048, out_features=32000, bias=False)
31  )
32  Computational complexity:       77.32 GMac
33  Number of parameters:           1261.53 M
34  GPU Memory Requirement: 2427.3115234375 MiB
```

```
1     $ python test_speedup.py
2         --model_type pruneLLM
3         --ckpt prune_log/tinyllama_30_0418_prune_log/pytorch_model.bin
```

Dazugehörige Ausgabe:

```
1   LlamaForCausalLM(
2     856.02 M, 92.879% Params, 54.79 GMac, 99.988% MACs,
3     (model): LlamaModel(
4       790.49 M, 85.769% Params, 50.6 GMac, 92.334% MACs,
5       (embed_tokens): Embedding(0, 0.000% Params, 0.0 Mac, 0.000% MACs, 32000,
            2048)
6       (layers): ModuleList(
7         (0-3): 4 x LlamaDecoderLayer(
8           44.04 M, 4.778% Params, 2.82 GMac, 5.144% MACs,
9           (self_attn): LlamaAttention(
10            9.44 M, 1.024% Params, 603.98 MMac, 1.102% MACs,
11            (q_proj): Linear(4.19 M, 0.455% Params, 268.44 MMac, 0.490% MACs,
                  in_features=2048, out_features=2048, bias=False)
12            (k_proj): Linear(524.29 k, 0.057% Params, 33.55 MMac, 0.061% MACs,
                  in_features=2048, out_features=256, bias=False)
13            (v_proj): Linear(524.29 k, 0.057% Params, 33.55 MMac, 0.061% MACs,
                  in_features=2048, out_features=256, bias=False)
14            (o_proj): Linear(4.19 M, 0.455% Params, 268.44 MMac, 0.490% MACs,
                  in_features=2048, out_features=2048, bias=False)
15          )
16          (mlp): LlamaMLP(
17            34.6 M, 3.754% Params, 2.21 GMac, 4.042% MACs,
```

```
18          (gate_proj): Linear(11.53 M, 1.251% Params, 738.2 MMac, 1.347% MACs,
                in_features=2048, out_features=5632, bias=False)
19          (up_proj): Linear(11.53 M, 1.251% Params, 738.2 MMac, 1.347% MACs,
                in_features=2048, out_features=5632, bias=False)
20          (down_proj): Linear(11.53 M, 1.251% Params, 738.2 MMac, 1.347% MACs,
                in_features=5632, out_features=2048, bias=False)
21          (act_fn): SiLU(0, 0.000% Params, 360.45 KMac, 0.001% MACs, )
22        )
23        (input_layernorm): LlamaRMSNorm(0, 0.000% Params, 0.0 Mac, 0.000% MACs,
              (2048,), eps=1e-05)
24        (post_attention_layernorm): LlamaRMSNorm(0, 0.000% Params, 0.0 Mac,
              0.000% MACs, (2048,), eps=1e-05)
25      )
26      (4-17): 14 x LlamaDecoderLayer(
27        31.3 M, 3.396% Params, 2.0 GMac, 3.656% MACs,
28        (self_attn): LlamaAttention(
29          7.08 M, 0.768% Params, 452.98 MMac, 0.827% MACs,
30          (q_proj): Linear(3.15 M, 0.341% Params, 201.33 MMac, 0.367% MACs,
                in_features=2048, out_features=1536, bias=False)
31          (k_proj): Linear(393.22 k, 0.043% Params, 25.17 MMac, 0.046% MACs,
                in_features=2048, out_features=192, bias=False)
32          (v_proj): Linear(393.22 k, 0.043% Params, 25.17 MMac, 0.046% MACs,
                in_features=2048, out_features=192, bias=False)
33          (o_proj): Linear(3.15 M, 0.341% Params, 201.33 MMac, 0.367% MACs,
                in_features=1536, out_features=2048, bias=False)
34        )
35        (mlp): LlamaMLP(
36          24.22 M, 2.628% Params, 1.55 GMac, 2.829% MACs,
37          (gate_proj): Linear(8.07 M, 0.876% Params, 516.69 MMac, 0.943% MACs,
                in_features=2048, out_features=3942, bias=False)
38          (up_proj): Linear(8.07 M, 0.876% Params, 516.69 MMac, 0.943% MACs,
                in_features=2048, out_features=3942, bias=False)
39          (down_proj): Linear(8.07 M, 0.876% Params, 516.69 MMac, 0.943% MACs,
                in_features=3942, out_features=2048, bias=False)
40          (act_fn): SiLU(0, 0.000% Params, 252.29 KMac, 0.000% MACs, )
41        )
42        (input_layernorm): LlamaRMSNorm(0, 0.000% Params, 0.0 Mac, 0.000% MACs,
              (2048,), eps=1e-05)
43        (post_attention_layernorm): LlamaRMSNorm(0, 0.000% Params, 0.0 Mac,
              0.000% MACs, (2048,), eps=1e-05)
44      )
45      (18-21): 4 x LlamaDecoderLayer(
46        44.04 M, 4.778% Params, 2.82 GMac, 5.144% MACs,
47        (self_attn): LlamaAttention(
48          9.44 M, 1.024% Params, 603.98 MMac, 1.102% MACs,
49          (q_proj): Linear(4.19 M, 0.455% Params, 268.44 MMac, 0.490% MACs,
                in_features=2048, out_features=2048, bias=False)
```

```
50            (k_proj): Linear(524.29 k, 0.057% Params, 33.55 MMac, 0.061% MACs,
                  in_features=2048, out_features=256, bias=False)
51            (v_proj): Linear(524.29 k, 0.057% Params, 33.55 MMac, 0.061% MACs,
                  in_features=2048, out_features=256, bias=False)
52            (o_proj): Linear(4.19 M, 0.455% Params, 268.44 MMac, 0.490% MACs,
                  in_features=2048, out_features=2048, bias=False)
53          )
54          (mlp): LlamaMLP(
55            34.6 M, 3.754% Params, 2.21 GMac, 4.042% MACs,
56            (gate_proj): Linear(11.53 M, 1.251% Params, 738.2 MMac, 1.347% MACs,
                  in_features=2048, out_features=5632, bias=False)
57            (up_proj): Linear(11.53 M, 1.251% Params, 738.2 MMac, 1.347% MACs,
                  in_features=2048, out_features=5632, bias=False)
58            (down_proj): Linear(11.53 M, 1.251% Params, 738.2 MMac, 1.347% MACs,
                  in_features=5632, out_features=2048, bias=False)
59            (act_fn): SiLU(0, 0.000% Params, 360.45 KMac, 0.001% MACs, )
60          )
61          (input_layernorm): LlamaRMSNorm(0, 0.000% Params, 0.0 Mac, 0.000% MACs,
                (2048,), eps=1e-05)
62          (post_attention_layernorm): LlamaRMSNorm(0, 0.000% Params, 0.0 Mac,
                0.000% MACs, (2048,), eps=1e-05)
63        )
64      )
65      (norm): LlamaRMSNorm(0, 0.000% Params, 0.0 Mac, 0.000% MACs, (2048,),
            eps=1e-05)
66      (rotary_emb): LlamaRotaryEmbedding(0, 0.000% Params, 0.0 Mac, 0.000% MACs, )
67    )
68    (lm_head): Linear(65.54 M, 7.111% Params, 4.19 GMac, 7.654% MACs,
          in_features=2048, out_features=32000, bias=False)
69  )
70  Computational complexity:       54.8 GMac
71  Number of parameters:           921.65 M
72  GPU Memory Requirement: 1793.30126953125 MiB
```

8 Fazit (Sebastian Viet)

Mittlerweile gibt es verschiedene Packages um Large Language Models zu prunen. Leider funktionieren viele dieser Packages nicht einwandfrei und sind teilweise auf bestimme Modelle zugeschnitten/eingeschränkt. Mit Hilfe des *LLM-Pruners* lassen sich zumindest die Llama-Modelle vergleichsweise einfach prunen, wenn auch hier zahlreiche Komplikationen in Bezug auf Package-Versionen entstanden sind, die zu verschiedenen Fehlermeldungen geführt haben. Dennoch lässt sich, unter der Verwendung der richtigen Package-versionen, das TinyLlama-Modell unkompliziert prunen und anschließend auf verschiedene Aspekte hin evaluieren.