FOM Hochschule für Ökonomie & Management

Studienzentrum Frankfurt am Main

Projektarbeit

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

Abbildungsverzeichnis

Tabellenverzeichnis

Inhaltsverzeichnis

1 Einleitung

1.1 Forschungsfragen

Die vorliegende Projektarbeit setzt sich mit insgesamt vier Forschungsfragen auseinander, die im Verlauf der Arbeit untersucht und beantwortet werden sollen. Die erste Forschungsfrage beschäftigt sich mit dem aktuellen Stand der Forschung im Bereich des Prunings von Large Language Models (LLMs). Dabei wird analysiert, welche Methoden derzeit für das Pruning angewendet werden, welche Unterschiede zwischen diesen bestehen und welche Vor- und Nachteile die jeweiligen Ansätze mit sich bringen. Das Ziel dieser Untersuchung ist es, einen Überblick über den aktuellen Stand der Forschung in diesem Bereich zu geben und die relevantesten Methoden zu beschreiben.

Ein weiterer Aspekt dieser Arbeit ist die Analyse der verschiedenen Frameworks, die Pruning-Methoden für LLMs unterstützen. Hierbei soll untersucht werden, welche dieser Frameworks welche Methoden implementieren und wie sie in der Praxis angewendet werden können. Basierend auf dieser Analyse wird eine geeignete Auswahl getroffen und eines dieser Framework für die praktische Umsetzung festgelegt.

Neben der theoretischen Auseinandersetzung mit dem Thema wird zudem ein eigener praktischer Versuch durchgeführt. Hierzu wird ein geeignetes Basismodell ausgewählt, das anschließend in verschiedenen Stufen geprunt wird. Die Auswirkungen dieses Prunings werden untersucht und dokumentiert. Im Rahmen dieser Analyse werden standardisierte Benchmarks erstellt, um die geprunten Modelle zu bewerten. Dabei wird nicht nur die Performance hinsichtlich der Ergebnisse in den Tests betrachtet, sondern auch der Einfluss des Prunings auf den Speicherverbrauch und die benötigte Rechenzeit. Ziel ist es, Erkenntnisse darüber zu gewinnen, wie sich das Pruning auf verschiedene Aspekte der Modelle auswirkt.

Zusammenfassend ergeben sich daraus die folgenden Forschungsfragen:

1. Wie ist der aktuelle Stand der Forschung im Bereich des Prunings von Large

Language Models, und welche Methoden werden derzeit eingesetzt?

2. Welche Frameworks bieten Unterstützung für das Pruning von LLMs, und wie lassen sich diese in der Praxis anwenden?

3. Welche Auswirkungen hat das Pruning eines Modells auf dessen Leistungsfähigkeit in Bezug auf standardisierte Benchmarks und Aufgaben?

4. Inwiefern beeinflusst das Pruning eines Modells dessen Speicherverbrauch sowie die benötigte Rechenzeit?

2 Theoretische Grundlagen

2.1 Übersicht über Large Language Models

Large Language Models (LLMs) stellen ein bedeutendes Forschungsfeld innerhalb der künstlichen Intelligenz (KI) dar. Sie basieren auf komplexen neuronalen Netzwerken, die auf die Generierung und Verarbeitung natürlicher Sprache spezialisiert sind, und gehören zur Klasse der Foundation Models. Bekannte Beispiele hierfür sind ChatGPT (OpenAI), Llama (Meta), Bard (Google) oder DeepSeek (Lian Wenfeng). Insbesondere Unternehmen zeigen großes Interesse an der Nutzung dieser Modelle, da sie ein erhebliches Potenzial zur Steigerung der Produktivität bieten. Der aktuelle Trend zeigt, dass LLMs zunehmend in bestehende Software integriert oder bei Neuentwicklungen unmittelbar berücksichtigt werden[1]

Das Training von LLMs erfolgt auf Basis des Generative Pretrained TransformerAnsatzes (GPT). Durch Mechanismen wie Gewichtung (Attention) und Self-Supervised Learning wird die Vorhersage des nächsten Tokens ermöglicht. Die zugrunde liegenden Trainingsdatensätze umfassen oft mehrere Petabytes, was den Modellen erlaubt, trotz breiter Themenvielfalt komplexe Zusammenhänge zu erkennen. Darüber hinaus entwickeln LLMs emergente Fähigkeiten wie In-Context Learning oder MultiStep Reasoning.[2]

Aufgrund der hohen Anzahl an Parametern und der großen Datenmengen erfordern sowohl das Training als auch der Betrieb von LLMs erhebliche Rechenressourcen. Zur Reduzierung dieser Anforderungen werden für spezifische Anwendungsbereiche Small Language Models (SLMs) entwickelt. Diese Modelle sind weniger ressourcenintensiv und neigen zu geringeren Halluzinationen, sofern die Gesprächsthematik mit ihrer trainierten Spezialisierung übereinstimmt. Ein weiterer vielversprechender Ansatz besteht darin, Halluzinationen zunächst durch ein SLM zu identifizieren

[1] Vgl. *Lu*; *L. Zhu*; *X. Xu* et al. (2024), S. 1-2, 5; *Minaee*; *Mikolov*; *Nikzad* et al. (2024), S. 1-2.

[2] Vgl. *Lu*; *L. Zhu*; *X. Xu* et al. (2024), S. 6-10; *Minaee*; *Mikolov*; *Nikzad* et al. (2024), S. 1-2; *Naveed*; *Khan*; *Qiu* et al. (2024), S. 1-2.

und anschließend mithilfe eines nachgelagerten LLMs sowie dessen Constraint-Based Reasoning-Funktion hinsichtlich Konsistenz und Logik zu verbessern. Huang, Yu, Ma et al. betonen, dass die Bereitstellung verfeinerter Halluzinationskategorien es LLMs ermöglicht, Halluzinationen zuverlässiger zu erkennen[3]

2.2 Effizienzoptimierung mittels Pruning

Bei Pruning handelt es sich um eine Methode zur Reduktion der Komplexität und Größe bei Modellen. Sie findet Anwendung im Bereich der Entscheidungsbäume und neuronalen Netzen. Die Effizienzsteigerung wird durch die Entfernung irrelevanter Teile erzielt. Pruning kann in drei Unterkategorien unterteilt werden: strukturelles, unstrukturelles und adaptives Pruning. Diese werden folgend weiter vorgestellt. Der Schwerpunkt wird auf strukturelles Pruning gelegt, da dieses auch später im Praxisteil Anwendung findet.

Das Kapitel wird durch die Vorstellung einer weiteren Unterscheidungsvariante aus einer Metastudie von Vadera und Ameen abgerundet.

2.2.1 Strukturelles Pruning

Beim strukturellen Pruning werden ganze Teileinheiten wie Filter oder Kanäle aus dem neuronalen Netz entfernt. Die Modelle werden kompakter und dadurch auch auf Standardhardware ausführbar. Ebenso die gute Kompatibilität zu Standardbibliotheken aus dem Deep Learning-Segment und die Möglichkeit das Pruning mit weiteren Kompressionsverfahren zu kombinieren, machen den Ansatz attraktiv. Beides wirkt sich positiv auf die entstehenden Kosten aus.[4]

Die Ausgangslage ist ein bereits trainiertes Netz. Die Herausforderung beim Pruning ist die Identifizierung der Teileinheiten, die ohne Verlust an Modellgenauigkeit löschbar sind. Zum einen kann die Selektion gewichtsabhängig erfolgen. Hierfür

[3]Vlg. *Kelbert*; *Siebert* und *Jöckel* (2023); *Huang*; *Yu*; *Ma* et al. (2024); *Hu*; *R. Xu*; *Lei* et al. (2024), S. 6-14.

[4]Vgl. *He* und *Xiao* (2023), S. 1; *Vadera* und *Ameen* (2022), S. 1-2.

werden z.B. anhand des geometrischen Medians redundante Filter identifiziert. Alternativ ist eine Betrachtung der L1-Norm (Summe aller Absolutwerte eines Vektors) und L2-Norm (Quadratwurzel der Summe der Quadrate der Einträge eines Vektors) möglich. Filter mit kleinen Summen werden entfernt. Hierbei spricht man von magnitudenbasierten Metriken. Neben der Betrachtung der Gewichte kann auch die Aktivierung als Kriterium genutzt werden. Der Rekonstruktionsfehler, der durchschnittliche Rang oder die Anzahl an Nullen auf den Aktivierungskarten werden als Kriterien für die Eliminierung genutzt. Über eine Kreuzkorrelation kann zudem die Unabhängigkeit eines Kanals bestimmt werden. Mit der Taylor-Expansion kann der entstehende Verlustwert nach Entfernung eines Filters möglichst annähernd ermittelt werden. He und Xian ordnen die Taylor-Expansion der Unterkategorie optimierungsbasierten Methoden zu, während Vadera und Ameen den Ansatz zur Gruppe der Sensitivity Analysis zählen. Unabhängig von der gewählten Methode ist es wichtig nach der Entfernung von Strukturen ein Finetuning vorzunehmen. Dieses dient dazu die Leistung zu optimieren und soll auftretende Genauigkeitsverluste auszugleichen.[5]

2.2.2 Unstrukturelles Pruning

Während beim strukturellen Pruning ganze Neuronen, Kanäle oder Filter entfernt werden, wird beim unstrukturellen Pruning jede Gewichtsverbindung analysiert und bewertet. Die Zielsetzung besteht erneut darin, überflüssige Informationen zu entfernen. Die Reduktion führt jedoch zu einer reduzierten Gewichtsmatrix mit unregelmäßigen Mustern. Die Umsetzung erfolgt meist über Maskierungstechniken. Diese deaktivieren durch binäre Masken unwichtige Gewichte.[6]

Da klassische GPUs auf dichte Matrizen optimiert sind, erfordert der Einsatz spezielle Hard- und Software. Es handelt sich somit um eine kostenintensivere Variante als das strukturierte Pruning. Die Bewertung jedes einzelnen Gewichts lässt zwar eine sehr feine Kontrolle über die Einsparung zu, jedoch kann das

[5] Vgl. *He* und *Xiao* (2023), S. 1-5; *Vadera* und *Ameen* (2022), S. 1-5.
[6] Vlg. *Cheng*; *M. Zhang* und *Shi* (2024), S. 3.

fehlerhafte Entfernen einer einzelnen kritischen Verbindung bereits zu Informationsverlust führen. Unter sorgfältiger Berücksichtigung dieser Gefahr kann jedoch ein hoher Kompressionsgrad erzielt werden. So zeigen Zhu und Gupta mittels Pruning verschiedener neuronaler Architekturen, dass eine Kompression ohne Leistungsbeeinträchtigung um bis 90% möglich ist.[7]

2.2.3 Adaptive Pruning Verfahren

Im Gegensatz zu traditionellen Pruning-Verfahren, bei denen die Kriterien im Voraus festgelegt werden, erfolgt das adaptive Pruning dynamisch während des Trainings. Das Verfahren passt sich dynamisch an Änderungen wie Gewichte, Gradienten oder Aktivierungen an und maximiert so die Effizienz. Manuelle Anpassungen werden auf ein Minimum beschränkt. Auch die komplexe Ermittlung von Parametern, die universell für das vollständige Modell gelten könnten, entfällt. Wie beim unstrukturierten Pruning muss die Hardware jedoch sparse Matrizen verarbeiten können. Im Gegenzug ist adaptives Pruning sehr flexibel und somit ideal für ressourcenbegrenzte Geräte. Beispielsbereiche sind das autonome Fahren sowie moderne Smartphones und Tablets mit dedizierten KI-Chips.

Prominente Beispiele sind das Layer-Adaptive-Pruning oder das Adaptive Channel-Pruning. Ersteres leitet die unterschiedlichen Pruning-Raten je Schicht aus der Sensitivität ab. Letzteres arbeitet mit der Entfernung von Kanälen in CNNs.[8]

2.2.4 Weitere Möglichkeiten der Kategorisierung von Pruning

In einer Metastudie haben Vadera und Ameen über 150 Paper zu Pruning untersucht. Hierbei gruppierten sie die Veröffentlichungen nicht nach strukturiert, unstrukturiert und adaptiv, sondern nach dem verwendeten (mathematischen) Ansatz. Es konnten acht Kategorien gebildet werden, die kurz benannt und vorgestellt werden:[9]

[7] Vgl. *Cheng*; *M. Zhang* und *Shi* (2024), S.3; *M. Zhu* und *Gupta* (2022), S. 6-8.
[8] Vgl. *Sakai*; *Eto* und *Teranishi* (2022), S. 295-296; *B. Wang*; *Pan*; *Diao* et al. (2025), S. 1-2.
[9] Vgl. *Vadera* und *Ameen* (2022), S. 2.

- **Magnitude-based Pruning Methods:** Relevanz von Gewichten und Neuronen wird über lokale Maße wie z. B. die Magnitude bestimmt.
- **Similarity and Clustering Methods:** Identifizierung und Entfernung redundanter oder sehr ähnlicher Gewichte.
- **Sensitivity Analysis Methods:** Entfernung der Gewichte, die geringen Einfluss auf das Endergebnis haben.
- **Knowledge Distillation Methods:** Ableitung eines neuen Modells („Student“) aus dem Ursprungsmodell („Teacher“).
- **Low-Rank Methods:** Zerlegung der Gewichtsmatrix in ein Produkt aus zwei kleineren Matrizen.
- **Quantization Methods:** Verwendung von Hashing, niedriger Präzision oder binärer Darstellung, um die Berechnungsaufwände zu reduzieren.
- **Architectural Design Methods:** Nutzung von Such- und Reinforcement-Learning zur Erstellung neuronaler Netzwerkarchitekturen.
- **Hybrid Methods:** Kombination mehrerer Ansätze zur Steigerung der Kompressionseffekte.

Erwähnenswert ist zudem, dass diese acht Kategorien teilweise noch feiner unterteilt werden konnten. Abbildung 1 zeigt eine Auswahl aus den ersten drei Kategorien der Metastudie.

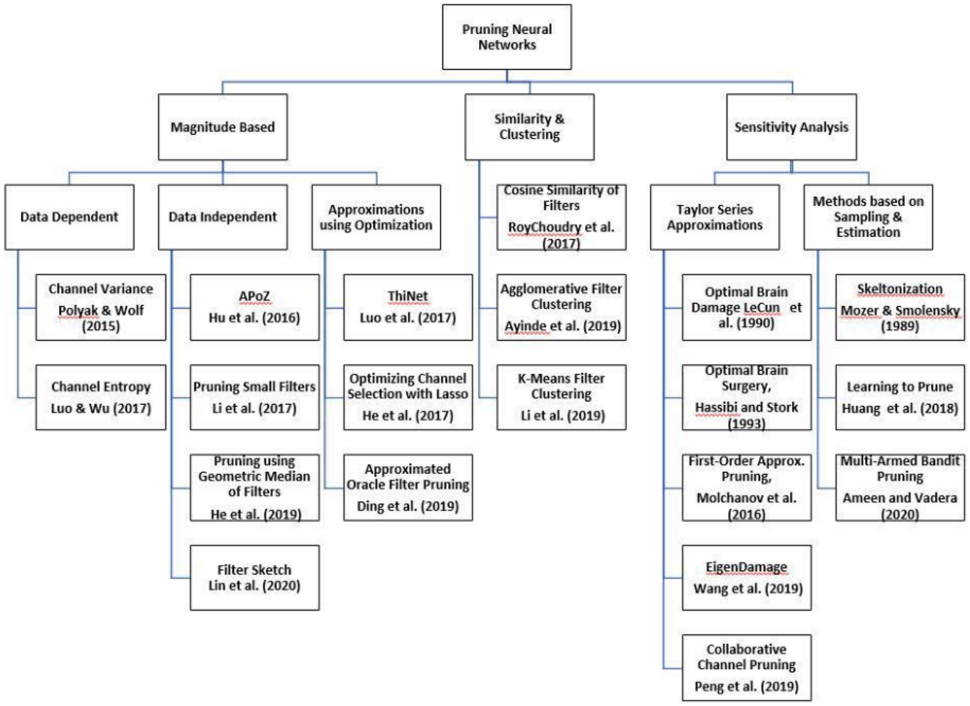

Abbildung 1: Aufteilung nach Vadera und Ameen

Quelle: *Vadera* und *Ameen* 2022

2.3 Bewertungskriterien

Um die Effizienz der unterschiedlichen Pruning-Ansätze zu ermitteln, ist das Heranziehen des Random-Pruning als Vergleichsmethode beliebt. Li, Adamczewski et al. unterscheiden dabei in drei verschiedene Wege, wie das Random-Pruning umgesetzt werden kann:

- **Vollständig zufällig:** Das Pruning findet zufällig und ohne weitere Vorgaben statt.
- **Eingeschränkt zufällig:** Das Pruning-Verhältnis wird je Schicht vorgegeben. Innerhalb der Schicht erfolgt das Pruning jedoch weiterhin zufällig.
- **Zufällige Kanalanzahlauswahl:** Das Pruning-Verhältnis wird je Schicht zufällig ausgewählt.[10]

Sowohl He und Xiao als auch Li, Adamczewski et al. konstatieren in ihren Untersuchungen ein überraschend gutes Abschneiden des Random-Prunings im Vergleich zu weiterentwickelten Methoden.[11]

Relevanter ist jedoch die Frage, inwieweit das Sprachmodell nach dem Pruning noch „funktionsfähig“ ist. Hierunter wird die Fähigkeit verstanden Eingaben in das Modell auf plausible Weise zu beantworten. Zur Untersuchung kommen extra auf diese Fragestellung zugeschnittene Benchmarks zum Einsatz. Diese enthalten Testfragen, welche an das Sprachmodell gestellt werden, sowie zugehörige Lösungen, die mit den Antworten des LLMs in Anschluss abgeglichen werden. Bei der Auswahl des Benchmark-Modells ist auf dessen Prüfziele zu achten. So unterscheiden sich die Modelle im Schwierigkeitsgrad und in der Breite der Themenabfrage. Die im Projekt eingesetzten Benchmark-Modelle werden im Folgenden kurz vorgestellt.

- **BoolQ (Boolean Questions):** Der Datensatz enthält 15.942 Ja-/Nein-Fragen. Die Fragen sind realen Kontexten entnommen und nicht generiert

[10] Vgl. *Y. Li*; *Adamczewski*; *W. Li* et al. (2022).
[11] Vgl. *He* und *Xiao* (2023); *Y. Li*; *Adamczewski*; *W. Li* et al. (2022).

worden. Neben der Fragestellung wird dem LLM ein Begleittext mitgegeben, welcher die Antwort auf die Frage bzw. Hilfestellungen enthält. Hierbei wird jedoch oft auf komplexere Schlussfolgerungen durch die KI gesetzt. Als einfaches Beispiel zum Verständnis sei folgender Begleittext gegeben: Spanien hat die Fußball-EM 2024 gewonnen. Die zugehörige Frage lautet: Hat Harry Kane die EM 2024 gewonnen? Die KI muss nun trotz ihres reduzierten neuronalen Netzes erkennen, dass Harry Kane zwar ein Fußballspieler ist, aber als Nationalspieler für England und nicht für Spanien im Finale antrat. Die Antwort muss daher Nein lauten.[12]

- **Arc-C (AI2 Reasoning Challenge):** Der Datensatz enthält 7.787 Multiple-Choice-Fragen aus dem naturwissenschaftlichen Bereich auf amerikanischem Grundschulniveau (Klassen 3 bis 9). 2.590 der Fragen werden als besonders herausfordernd klassifiziert. Sie gelten als nicht beantwortbar für abrufbasierte oder Wort-Koexistenz-Algorithmen. Optional wird ein Datensatz mit 14 Millionen wissenschaftlichen Sätzen angeboten, welcher der KI zur Beantwortung der Fragen als Hilfestellung gereicht werden kann.[13]

- **HellaSwag:** Bei HellaSwag handelt es sich um die Weiterentwicklung von Swag. Der Datensatz enthält 39.905 Trainingsdatensätze und 5.021 Datensätze für Test und Validierung. Bei HellaSwag ist für das LLM die Aufgabe, einen gegebenen Satz sinnvoll fortzusetzen. Dem LLM werden dabei 4 mögliche Fortsetzungen angeboten, wovon nur eine richtig ist. Das LLM wird somit auf richtige Schlussfolgerungen basierend auf Alltagswissen getestet. Die drei falschen Antwortmöglichkeiten wurden mittels des Adversarial-Filtering-Verfahrens generiert. Auf Basis von iterativ eingesetzten Diskriminatoren werden die negativen Antworten so gestaltet, dass sie für künstliche Intelligenz schwer zu erkennen sind. Der Mensch hingegen kann die falschen Antworten leicht erkennen.[14]

[12] Vgl. *C. Clark*; *Lee*; *Chang* et al. (2019), S. 1-2.
[13] Vgl. *P. Clark*; *Cowhey*; *Etzioni* et al. (2019), S. 1.
[14] Vgl. *Zellers*; *Holtzman*; *Bisk* et al. (2019), S. 1-3.

- **OpenBookQA:** Der Datensatz enthält 5.957 Multiple-Choice-Fragen mit je vier Antwortmöglichkeiten. Wie der Name bereits nahelegt, empfindet der Test Open-Book-Prüfungen nach, bei denen die Prüflinge freien Zugang zu Informationen haben. Das Modell enthält daher zu jeder Frage auch ein *core fact*, welches die unterstützende Wissensquelle imitieren soll. Die Fragen sind so gestaltet, dass eine mehrschrittige Schlussfolgerung unter Einbeziehung von externem Wissen notwendig ist.[15]

- **WinoGrande:** Bei WinoGrande handelt es sich um die Weiterentwicklung der Winograd Schema Challenge (WSC). Während das Original nur 273 Aufgaben beinhaltet, setzt die Erweiterung auf 44.000 Problemstellungen. Diese wurden auf Basis von Crowdsourcing erstellt und geprüft. Die Aufgaben sind als *fill-in-the-blank*-Probleme gestellt, wo Pronomen durch ihre richtige Referenz ersetzt werden müssen. Um den Schwierigkeitsgrad anzuheben, wurde zusätzlich AfLite (Adversarial Filtering) angewendet. Dies verhindert die Ausnutzung einfacher Wortassoziationen durch das LLM bei der Beantwortung der Fragen.[16]

[15] Vgl. *Mihaylov*; *P. Clark*; *Khot* et al. (2018), S. 1-2.

[16] Vgl. *Sakaguchi*; *Le Bras*; *Bhagavaula* et al. (2019), S. 1-3.

3 Methodik

3.1 Literaturrecherche

3.2 Arbeitsorganisation

Für die Zusammenarbeit und Organisation der anstehenden Aufgaben und zu verfassenden Kapitel wurde das Notizenprogramm *Notion* verwendet. Notion erlaubt das Erstellen kollaborativer Notizbücher, die parallel von mehreren Personen bearbeitet werden können. Über die Datenbankfunktion lassen sich sowohl Meilensteinpläne als auch Task-Boards erstellen, die zur Planung der Arbeit verwendet wurden.

3.3 Entwicklungsumgebung

Da sowohl das Pruning von Large Language Models als auch die anschließende Evaluierung der geprunten Modelle erhebliche Rechenressourcen erfordern, wurde entschieden, diese Prozesse auf einem leistungsstarken Server in der Cloud durchzuführen. Insbesondere das Testen der resultierenden Modelle stellt eine hohe Belastung für die verfügbaren Ressourcen dar und ist auf herkömmlicher Hardware nicht effizient durchzuführen. Zu diesen Zweck wurden EC2-Instanzen des Cloud-Providers AWS (Amazon Web Services) aus der *g*-Instanzfamilie verwendet. Diese Instanzfamilie ist speziell auf rechenintensive Anwendungen ausgelegt und bietet GPUs (Graphics Processing Units), die für parallele Berechnungen optimiert sind und den Einsatz von CUDA, dem von *NVIDIA* entwickelten Toolkit zur Verwendung von GPUs in allgemeinen Rechenaufgaben, ermöglichen.

Im Detail wurden Instanzen des Typs *gd4n.xlarge* verwendet, die mit 16 GiB Hauptspeicher, einem leistungsstarken Prozessor der *Intel Xeon Family* sowie einer *NVIDIA T4 Tensor Core* Grafikeinheit ausgestattet sind. Für den Start der Instanzen wurde das speziell auf Deep-Learning-Anwendungen zugeschnittene *Deep Learning OSS Nvidia Driver AMI GPU PyTorch 2.5.1 (Ubuntu 22.04) 20241208* Ubuntu-Image verwendet. In diesem Image ist das benötigte Python-Package

PyTorch bereits vorinstalliert und ist zusätzlich mit *CUDA* ausgestattet.

Verwendet wurden im Detail Instanzen des Types *gd4n.xlarge*, die mit 16GiB Hauptspeicher, einem physischen Prozessor der *Intel Xeon Family* und einer *NVIDIA T4 Tensor Core* Grafikeinheit kommen. Gestartet wurden die Instanzen mit dem *Deep Learning OSS Nvidia Driver AMI GPU PyTorch 2.5.1 (Ubuntu 22.04) 20241208* Ubuntu Image, das bereits das benötigte Python-Package *PyTorch* vorinstalliert hat. Diese, auf Deep Learning ausgerichteten, Images haben neben PyTorch bereits *CUDA* vorinstalliert und ermöglichen so einen einfachen Start in das Arbeiten mit Programmen, die die GPU benötigen.

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

Nachdem über SSH (Secure Shell) mit der jeweiligen Instanz verbunde wurde, wurde zunächst die vorinstallierte PyTorch-Umgebung gestartet und anschließenden das Repository des verwendeten Frameworks geklont.

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

Da die auf dem Image vorinstallierte PyTorch-Umgebung mit dem Conda-Package-Manager bereitgestellt wird, wurden die im Repository in der *requirements.txt*-Datei angebenen Package über Conda installiert.

```
1 $ conda install transformers sentencepiece datasets wandb ...
```

Damit ist die Versuchsumgebung identisch zu der in dieser Arbeit verwendeten Umgebung.

3.4 Verwendetes Large Language Model

Aufgrund von finanziellen und zeitlichen Einschränkungen wurde für diese Analyse das TinyLlama-Modell (*TinyLlama/TinyLlama-1.1B-Chat-v1.0*) als Basismodell gewählt. Mit 1,1 Milliarden Parametern ist es im Vergleich zu modernen, größeren Modellen wie *Llama 3.1* – das mit 405 Milliarden Parametern deutlich umfangreicher ist – relativ klein. Das verwendete TinyLlama wurde auf einem Datensatz von drei Milliarden Token vortrainiert und basiert auf der gleichen Architektur wie die *Llama 2*-Modelle. Diese Wahl ermöglichte es, innerhalb der verfügbaren Ressourcen eine Analyse durchzuführen, während gleichzeitig die Komplexität des Modells berücksichtigt wurde.

Der nachfolgende Auszug zeigt die detailierte Architektur des Modells:

```
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()
)
```

Die Architektur besteht aus einer Embedding-Schicht, über die die Eingabesequenzen in Vektoren der Dimension 2048 umwandelt werden. Die Token-Embeddings basieren auf einem Vokabular von 32.000 Wörtern. Darauf folgt eine Modul-Liste mit 22 LlamaDecoderLayer, die jeweils aus mehreren Submodulen bestehen. Die Decoder-Schicht enthält einen Self-Attention-Mechanismus (LlamaSdpaAttention).

Um die grundlegenden Funktionen der ersten und letzten Schichten nicht zu beeinflussen, wurden sowohl im Attention-Abschnitt als auch im MLP-Abschnitt ausschließlich die Layer 4 bis 18 für das Pruning verwendet. Die Schichten der Architektur außerhalb des Decoder-Layers wie bspw. die Embedding-Schichten werden grundsätzlich nicht berührt, da sie für die Umwandlung der Texteingaben in die korrekte Repräsentation nötig sind.

3.5 Pruning

Das nachfolgende Kapitel beinhaltet die Vorgehensweise und verwendeten Technologien, die für das Pruning des TinyLlama-Modells verwendet wurde.

3.5.1 Verwendetes Framework

Für die Durchführung des Prunings standen zwei Frameworks zur Auswahl: *LLM-Pruner* und *Wanda* (Pruning by Weights and Activations). Während der *LLM-Pruner* ausschließlich das im vorherigen Kapitel beschriebene strukturierte Pruning unterstützt, bietet *Wanda* zusätzlich die Möglichkeit, unstrukturiertes Pruning durchzuführen. Trotz dieser zusätzlichen Funktionalität wurde für die Umsetzung dieser Untersuchung aus verschiedenen Gründen ausschließlich der *LLM-Pruner* verwendet.

Ein wesentlicher Faktor für diese Entscheidung war die Aktualität der Projekte. Die letzten Updates im *Wanda*-Projekt wurden Ende 2023 vorgenommen. Dies lag zum Zeitpunkt des Verfassens dieser Arbeit bereits mehr als ein Jahr zurück. Im Gegensatz dazu wird der *LLM-Pruner* weiterhin aktiv weiterentwickelt, was eine aktuellere und besser unterstützte Basis bietet. Ein weiterer wichtiger Aspekt

ist die Unterstützung spezifischer Modelle. Während beim *LLM-Pruner* explizit die Kompatibilität mit dem *TinyLlama*-Modell hervorgehoben wird, fehlt eine solche Aussage im Fall von *Wanda*. Es ist zwar anzunehmen, dass die Methoden von *Wanda* aufgrund der Unterstützung von *Llama-2* – dessen Architektur dem *TinyLlama* ähnlich ist – ebenfalls auf das *TinyLlama*-Modell anwendbar sein könnten. Allerdings bleibt dies ungewiss, da eine direkte Unterstützung nicht garantiert wird.

Ein weiterer entscheidender Punkt war, dass trotz mehrerer Versuche mit *Wanda* kein erfolgreiches Pruning durchgeführt werden konnte. Angesichts dieser Schwierigkeiten und unter Berücksichtigung der bereits genannten Argumente fiel die Entscheidung, sich ausschließlich auf den *LLM-Pruner* zu fokussieren.

3.5.2 Durchführung des Prunings

Das Pruning wurde, wie bereits erläutert, anhand des *TinyLlama*-Modells getestet und untersucht. Dabei wurden drei unterschiedliche Pruning-Ratios gewählt, um die Auswirkungen verschiedener Verkleinerungsgrade des Modells zu analysieren. Da das *LLM-Pruner*-Projekt bereits eigene Ergebnisse für das *TinyLlama*-Modell mit einer Pruning-Ratio von 20% veröffentlicht hatte, wurde dieses Verhältnis in der vorliegenden Untersuchung nicht erneut evaluiert. Stattdessen wurden die bereits existierenden Ergebnisse mit den Verhältnissen von 30%, 40% und schließlich 70% verglichen. Obwohl eine Verkleinerung um 70% bei einem ohnehin bereits kompakten Modell wie *TinyLlama* als unrealistisch erscheint, wurde dennoch im Rahmen der Untersuchung überprüft, welche Ergebnisse das Modell bei einer solch extremen Reduktion in den Tests liefert.

Das Pruning erfolgt stets über den folgenden Befehl, der ausgeführt werden kann, sobald sich im LLM-Pruner-Repository auf der höchsten Ebene befindet:

```
$ python llama3.py
    --base_model TinyLlama/TinyLlama-1.1B-Chat-v1.0
    --pruning_ratio [PRUNING_RATIO]
    --device cuda
    --eval_device cuda
    --block_wise
```

```
    --block_mlp_layer_start [START_MLP_LAYER]
    --block_mlp_layer_end [END_MLP_LAYER]
    --block_attention_layer_start [START_ATTENTION_LAYER]
    --block_attention_layer_end [END_ATTENTION_LAYER]
    --save_ckpt_log_name [SAVE_PATH]
    --pruner_type [PRUNER_TYPE]
    [--taylor param_first]
    --save_model
    --max_seq_len 2048
    --test_after_train
```

Für das Pruning des *TinyLlama*-Modells wurde das Skript *llama3.py* verwendet, das speziell für das Pruning von *Llama3*-Modellen entwickelt wurde. In dieser Untersuchung diente stets das zuvor beschriebene Modell *TinyLlama/TinyLlama-1.1B-Chat-v1.0* als *base_model*. Wie bereits erwähnt, wurden für die Pruning-Ratio die Verhältnisse 0.3, 0.5 und 0.7 ausgewählt, um unterschiedliche Stufen der Modellreduktion zu analysieren.

Alle Prozesse wurden in einer CUDA-fähigen Umgebung ausgeführt, weshalb die Anweisung, die GPU für die Berechnungen zu verwenden, stets mitgegeben wurde. Von den insgesamt 22 verfügbaren MLP- und Attention-Layern des Modells wurden für das Pruning jeweils die Layer vier bis 18 berücksichtigt.

Bezüglich der verfügbaren Pruner-Typen bot der *LLM-Pruner* vier verschiedene Optionen an: *Taylor*, *L1*, *L2* und *random*. Jede dieser vier Methoden wurde für jede der drei gewählten Pruning-Ratios getestet und evaluiert, um ihre jeweiligen Auswirkungen auf die Modellleistung zu untersuchen.

Zusätzlich wurde bei allen Experimenten das Argument *test_after_train* verwendet. Dadurch wurde nach jedem Pruning automatisch die Perplexity des Modells ermittelt, basierend auf den beiden Testdatensätzen *wikidata2* und *ptb* (*Penn Treebank*).

Um das TinyLlama-Modell zu 30% mit der *Taylor*-Methode zu prunen sieht der Befehl demnach beispielhaft wie folgt aus:

```
$ 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_0616_prune_log
    --pruner_type taylor
    --taylor param_first
    --save_model
    --max_seq_len 2048
    --test_after_train
```

3.6 Fine-Tuning

Für das nach dem Pruning stattfindende Fine-Tuning wird vom LLM-Pruner *PEFT* (Parameter-Efficient Fine-Tuning) verwendet. PEFT stellt eine Methode dar, um große vortrainierte Modelle an spezifische Aufgaben anzupassen, ohne den gesamten Parameterraum des Modells zu optimieren. Stattdessen wird nur ein kleinerer Teil der Parameter während des Trainings modifiziert.

In den Evaluierungen der Modelle, die direkt im Anschluss an das Pruning durchgeführt wurden, hat sich bereits gezeigt, dass über die Pruning-Methode *Taylor* die vielversprechendsten Ergebnisse erzielt werden konnte. Diese geprunten Modelle konnten die höchsten Werte in den Evaluierungen erreichen. Das rechen- und kostenintensive Fine-Tuning wurde daher nur testweise für das Modell durchgeführt, das zu 30% mit der *Taylor*-Methode geprunt wurden.

Verwendet wurde dafür das im *LLM-Pruner* vorhandene Skript *post_training.py* über den folgenden Befehl:

```
$ python post_training.py
    --prune_model prune_log/tinyllama_30_0418_prune_log/pytorch_model.bin
    --data_path yahma/alpaca-cleaned \
    --lora_r 8 \
    --num_epochs 2 \
    --learning_rate 1e-4 \
    --batch_size 64 \
```

```
8       --output_dir tune_log/tinyllama_30_tuned \
9       --wandb_project tinyllama_30_tune
```

3.7 Evaluierung der Modelle

Als Basis jeglicher durchgeführter Tests dienten jeweils die Ergebnisse des selbst durchgeführten Benchmarks des TinyLlama-Basismodells. Alle erhobenen Werte und Ergebnisse werden in Relation dazu bewertet.

3.7.1 Modell-Benchmarks

Die Modelle, die durch das Pruning und das anschließende Fine-Tuning entstanden sind, wurden anschließend auf ihre verbliebenen Fähigkeiten hin untersucht. Zu diesem Zweck wurden sie anhand verschiedener Datensätze evaluiert, die jeweils unterschiedliche Aspekte der Leistungsfähigkeit von LLMs testen. Welche spezifischen Aspekte dabei geprüft wurden, wurde bereits in den vorherigen Abschnitten detailliert beschrieben.

- *openbookqa*
- *winogrande*
- *hellaswag*
- *arc_challenge*
- *boolq*

Die Evaluierung anhand dieser Datensätze wurde, wie beim Pruning, mit dem immer gleichen Befehl - angepasst an das jeweilige Modell - durchgeführt.

```
1 $ export PYTHONPATH='.'
2 $ python lm-evaluation-harness/main.py
3       --model hf-causal-experimental
4       --model_args checkpoint=[PATH_TO_MODEL]/pytorch_model.bin,
5               config_pretrained=TinyLlama/TinyLlama-1.1B-Chat-v1.0
6       --tasks openbookqa,winogrande,hellaswag,arc_challenge,boolq
7       --device cuda:0
8       --no_cache
9       --output_path [PATH_TO_RESULTS]
```

Wie im Befehl ersichtlich, wurde das *lm-evaluation-harness*-Framework von OpenAI verwendet, das bereits im Repository enthalten ist. Dieses Framework dient der Evaluierung von Sprachmodellen anhand verschiedener Benchmarks bzw. *Tasks*. Durch die Nutzung des Frameworks in Kombination mit den definierten *Tasks* wird eine standardisierte Bewertung der Modelle ermöglicht, was wiederum den Vergleich unterschiedlicher Modelle erleichtert.

Das Argument *–model hf-causal-experimental* wird übergeben, um die Nutzung der GPU während der Tests zu gewährleisten. Zusätzlich ist die Angabe des Pfads zum geprunten bzw. nachtrainierten Modell erforderlich, ebenso wie die Grundarchitektur des Basismodells. Die im Befehl spezifizierten *Tasks* entsprechen dabei den zuvor beschriebenen.

Ein beispielhafter Aufruf zur Evaluierung des Modells, das zu 30% geprunt wurde, sieht wie folgt aus:

```
$ export PYTHONPATH='.'
$ python lm-evaluation-harness/main.py
    --model hf-causal-experimental
    --model_args
        checkpoint=prune_log/tinyllama_30_0418_l1_prune_log/pytorch_model.bin,
        config_pretrained=TinyLlama/TinyLlama-1.1B-Chat-v1.0
    --tasks openbookqa,winogrande,hellaswag,arc_challenge,boolq
    --device cuda:0
    --no_cache
    --output_path results/tinyllama_30_0418_l1
```

3.7.2 Speicherverbrauch und Rechenleistung

Neben den verbliebenen Fähigkeiten der geprunten LLMs sollen zusätzlich deren Speicherverbrauch und deren benötigte Rechenleistung analysiert werden. Diese Messung erfolgt über das im *LLM-Pruner* integrierte Python-Skript *test_speedup.py*. Wie man dem Skript entnehmen kann, verwendet es intern das Python-Package *ptflops*, das sich selbst als "Flops counting tool for neural networks in pytorch framework"[17] beschreibt. Damit gemessen werden können die Anzahl

[17] Vgl. *Sovrasov* (2018).

an Parametern, die Rechenkomplexität in GMac und die GPU Speicheranforderungen. Die Zeit zu messen, die die Modelle für verschiedene Befehle benötigen, unterscheidet sich stark anhand der verwendeten Hardware und ist daher keine geeignete Größe anhand der die Modelle vergliche werden sollten.

Das Skript wird auf die einzelnen Modelle angewendet und diese anhand der daraus generierten Resultate verglichen.

```
$ export PYTHONPATH='.'
$ python test_speedup.py
    --model_type pruneLLM
    --ckpt [PATH_TO_PRUNED_MODEL]
```

4 Ergebnisse

Im folgenden Kapitel werden die Ergebnisse der Evaluierungen und Tests der geprunten sowie teilweise anschließend trainierten Modelle vorgestellt. Der Fokus liegt dabei insbesondere auf den ermittelten Perplexity-Werten sowie den Ergebnissen der Tests mit dem *lm-evaluation-harness*-Framework. Die verschiedenen Pruning-Stufen und die dabei verwendeten Methoden werden einzeln analysiert, bevor abschließend ein umfassender Gesamtüberblick über die Ergebnisse gegeben wird.

4.1 Evaluierung Basismodell

Um eine Vergleichsbasis zu schaffen, muss zunächst das Basismodell in allen Tests evaluiert werden. Als Basismodell dient hierbei das *TinyLlama*-Modell (*TinyLlama/TinyLlama-1.1B-Chat-v1.0*). Die Darstellung der Ergebnisse orientiert sich an den vom *LLM-Pruner* für andere Modelle bereitgestellten Resultaten, um eine einheitliche Vergleichbarkeit zu gewährleisten.

Die Spalte *Average* gibt den Durchschnitt der getesteten *Tasks* wieder. Die Ergebnisse für *WikiText2* und *PTB* werden dabei nicht in diese Berechnung einbezogen, da bei diesen Benchmarks ein niedrigerer Score eine bessere Leistung des Modells widerspiegelt.

Die zusammengefassten Ergebnisse sind in der folgenden Tabelle dargestellt:

Pruning Ratio	Method	WikiText2	PTB	BoolQ	HellaSwag	WinoGrande	ARC-c	OBQA	Average
Pruned 0%	–	7.97	20.76	61.31	46.15	60.30	30.12	24.20	39.58

Tabelle 2: Evaluierung des Basismodells

Wie hier zu erkennen ist, weist das Basismodell bereits einen vergleichsweise niedrigen Score von *39,58* auf. Zum Vergleich: In den Ergebnissen des *LLM-Pruners* für das Modell *Llama7B* ergibt sich ein Durchschnittswert von *68,59*. Allerdings wurden in diesen Tests noch weitere *Tasks* berücksichtigt, die in der vorliegenden Analyse nicht enthalten sind.

Da sich diese Untersuchung jedoch auf die relative Verschlechterung im Vergleich zum Basismodell konzentriert, stellt der niedrigere Ausgangswert hier kein Problem dar.

4.2 Tatsächliche Anzahl geprunter Parameter

Anhand der Modellevaluierungen, die nach dem Pruning über das Evaluierungsskript erstellt wurden, lässt sich schnell erkennen, dass die im Befehl angegebene Menge an zu prunenden Parametern bzw. das definierte Verhältnis nicht exakt vom *LLM-Pruner* umgesetzt wird.

Am Beispiel des *30%*-Prunings zeigt sich dies wie folgt: Bei einer angegebenen Pruning-Ratio von *30%* wäre zu erwarten, dass nach dem Pruning noch *70%* der ursprünglichen Parameter des Basismodells erhalten bleiben. Tatsächlich sind es in diesem Fall jedoch *73,06%*, also *3,06%* mehr als ursprünglich vorgesehen.

Noch deutlicher wird diese Abweichung bei einer Pruning-Ratio von *70%*. Hier sollten theoretisch nur noch *30%* der Parameter im Modell verbleiben, tatsächlich sind es jedoch *55,08%*, was einer Differenz von *25,08%* entspricht.

Diese Diskrepanz muss bei der Interpretation der nachfolgenden Ergebnisse berücksichtigt werden, da die angegebenen Verhältnisse nicht mit den tatsächlichen übereinstimmen.

Specified ratio	**#Parameters before**	**#Parameters after**	**Actual ratio**
30%	1261.53 Mio	921.65 Mio	73.06%
40%	1261.53 Mio	873.22 Mio	69.21%
70%	1261.53 Mio	694.83 Mio	55,08%

Tabelle 3: Anzahl der vorhandenen Parameter nach dem Pruning

4.3 Evaluierung der reduzierten Modelle

4.3.1 Evaluierung Pruning zu 30%

In Tabelle 4 sind die Ergebnisse der geprunten Modelle sowie die des Basismodells dargestellt. Getestet wurden – wie in Kapitel 3 beschrieben – die *Tasks* BoolQ, HellaSwag, WinoGrande, ARC-Challenge und OpenBookQA. Zusätzlich wurde die *Perplexity* anhand der Datensätze WikiText2 und PTB ermittelt.

Pruning Ratio	**Method**	**WikiText2**	**PTB**	**BoolQ**	**HellaSwag**	**WinoGrande**	**ARC-c**	**OBQA**	**Average**
Pruned 0%	–	7.97	20.76	61.31	46.15	60.30	30.12	24.20	39.58
Pruned 30%	Taylor	15.19	43.19	55.50	37.90	54.22	24.66	23.20	39.10
	L1	281.63	4992.16	46.54	28.53	48.46	21.33	14.00	31.77
	L2	42.19	167.51	60.89	35.97	54.30	22.95	18.40	38.50
	Random	40.89	166.21	59.63	33.18	53.99	20.99	18.00	37.17
Pruned 30% (tuned)	Taylor	/	/	45.75	39.29	56.67	25.26	22.00	37.79

Tabelle 4: Evaluierungen bis 30% Pruning

Es ist wichtig zu beachten, dass die tatsächliche Anzahl der Parameter im Vergleich zum Basismodell nur um *26,94%* reduziert wurde – und nicht, wie ursprünglich erwartet, um volle *30%*. Die Tests wurden unmittelbar nach dem Pruning durchgeführt, ohne dass ein erneutes Fine-Tuning erfolgte.

Betrachtet man ausschließlich die Ergebnisse der *Tasks* und deren Durchschnittswerte, so schneiden die *Taylor*- und *L2*-Methoden am besten ab, da ihre Werte am nächsten an denen des Basismodells liegen. Allerdings zeigt sich bei der *L2*-Methode eine deutlich höhere und damit schlechtere *Perplexity* in beiden Datensätzen im Vergleich zur *Taylor*-Methode. Hervorzuheben ist dennoch, dass sie in der *BoolQ*-Task um *5,39* Prozentpunkte besser abgeschnitten hat als die *Taylor*-Methode.

Die *Random*-Methode weist mit ihren Resultaten Ähnlichkeiten zur *L2*-Methode auf und schneidet somit ebenfalls schlechter als die *Taylor*-Methode ab. Hier sind keine weiteren Auffälligkeiten hervorzuheben.

Am schlechtesten hat die *L1*-Methode abgeschnitten: Das daraus resultierende

Modell weist extrem schlechte *Perplexity*-Werte auf, und auch der Durchschnitt der *Tasks* liegt im Vergleich zu den anderen Methoden deutlich niedriger.

In der letzten Zeile der Tabelle sind zusätzlich die Ergebnisse des nachtrainierten Taylor-Modells zu sehen. Leider konnten hier keine vergleichbaren Ergebnisse wie in der Dokumentation des *LLM-Pruners* erzielt werden. Nach einem dreistündigen Post-Training ist der Durchschnittswert von 39,85 auf 37,79 gesunken. Das Modell hat damit nach dem Post-Training schlechtere Ergebnisse geliefert als zuvor. Lediglich in den Evaluierungsaufgaben *HellaSwag* und *ARC-c* konnte eine leichte Steigerung erzielt werden.

4.3.2 Evaluierung Pruning zu 40%

In Tabelle 5 sind die Inhalte der Tabelle 4 ergänzt um die Ergebnisse der Tests zu den Modellen, die mit der Angabe *40%* geprunt wurden zu sehen. Es wurden erneut jeweils die vier möglichen Methoden angewendet und getestet.

Pruning Ratio	Method	WikiText2	PTB	BoolQ	HellaSwag	WinoGrande	ARC-c	OBQA	Average
Pruned 0%	–	7.97	20.76	61.31	46.15	60.30	30.12	24.20	39.58
Pruned 30%	Taylor	15.19	43.19	55.50	37.90	54.22	24.66	23.20	39.10
	L1	281.63	4992.16	46.54	28.53	48.46	21.33	14.00	31.77
	L2	42.19	167.51	60.89	35.97	54.30	22.95	18.40	38.50
	Random	40.89	166.21	59.63	33.18	53.99	20.99	18.00	37.17
Pruned 30% (tuned)	Taylor	/	/	45.75	39.29	56.67	25.26	22.00	37.79
Pruned 40%	Taylor	18.43	53.33	59.39	34.84	52.88	23.55	18.80	37.89
	L1	441.35	14000.87	48.99	28.13	49.96	21.42	14.80	32.66
	L2	86.23	229.87	60.83	34.18	51.62	21.08	19.60	37.59
	Random	86.91	364.46	57.89	30.82	52.33	20.56	17.00	35.72

Tabelle 5: Evaluierungen bis 40% Pruning

Auch hier zeigt sich, dass die *Taylor*-Methode die vielversprechendsten Ergebnisse erzielt. Mit einem durchschnittlichen Score von 37.89 übersteigt die Methode bei einer angegebenen Pruning-Ratio von 40% bzw. tatsächlichen Pruning-Ratio von 30,79% noch ein besseres Ergebnis als die nachtrainierte Version des 30%-Modells.

Ebenso erzielt die *L1* erneut im Vergleich die schlechtesten Ergebnisse. Mit einer Perplexity von ca. 14.000 erreicht diese Methode ein extrem schlechtes Ergebnis. Deutlich schlechter in den Tests schneidet die Methode insbesondere bei den beiden Tasks *BoolQ* und *HellaSwag* ab.

Die *L2*-Methode erzielt wiederholt ähnliche, wenn auch leicht schlechtere, Ergebnisse als die *Taylor*-Methode. Die *Random*-Methode liegt ebenso erneut auf dem 3. Platz.

4.3.3 Evaluierung Pruning zu 70%

Zuletzt wird das Modell betrachtet, das mit einer angegebenen Pruning-Ratio von 70% reduziert wurde (tatsächlich nur um 44,92%). Zu sehen in Tabelle 6 sind die vorherigen Ergebnisse erweitert um die Resultate der vier Methoden, mit denen das Modell reduziert wurde.

Pruning Ratio	Method	WikiText2	PTB	BoolQ	HellaSwag	WinoGrande	ARC-c	OBQA	Average
Pruned 0%	–	7.97	20.76	61.31	46.15	60.30	30.12	24.20	39.58
Pruned 30%	Taylor	15.19	43.19	55.50	37.90	54.22	24.66	23.20	39.10
	L1	281.63	4992.16	46.54	28.53	48.46	21.33	14.00	31.77
	L2	42.19	167.51	60.89	35.97	54.30	22.95	18.40	38.50
	Random	40.89	166.21	59.63	33.18	53.99	20.99	18.00	37.17
Pruned 30% (tuned)	Taylor	/	/	45.75	39.29	56.67	25.26	22.00	37.79
Pruned 40%	Taylor	18.43	53.33	59.39	34.84	52.88	23.55	18.80	37.89
	L1	441.35	14000.87	48.99	28.13	49.96	21.42	14.80	32.66
	L2	86.23	229.87	60.83	34.18	51.62	21.08	19.60	37.59
	Random	86.91	364.46	57.89	30.82	52.33	20.56	17.00	35.72
Pruned 70%	Taylor	83.25	274.04	52.39	28.43	48.70	19.11	17.00	33.12
	L1	37762.14	11607.13	46.91	25.98	50.43	20.39	16.40	32.02
	L2	394.08	783.72	60.49	26.84	51.62	20.31	13.20	34.49
	Random	4138.65	4074.48	54.74	26.64	51.38	20.99	14.20	33.59

Tabelle 6: Evaluierungen bis 70% Pruning

Die *Taylor*-Methode schneidet hierbei bezüglich des durchschnittlichen Scores – abgesehen von der Perplexity – zum ersten Mal nicht am besten unter den vier zur Verfügung stehenden Methoden ab, sondern belegt Platz drei. Unter Berücksichtigung der Perplexity auf den beiden Testdatensätzen, die im Vergleich deutlich besser ist, lässt sich jedoch zusammenfassend sagen, dass sie auch hier die vielversprechendste Methode ist.

Bezüglich der fünf Tests führt hierbei die *L2*-Methode, erzielt jedoch, wie bereits erwähnt, hinsichtlich der Perplexity deutlich schlechtere Resultate.

Interessant ist jedoch, dass bei einer so hohen Pruning-Ratio wie hier angegeben die Ergebnisse nicht wesentlich schlechter als die des Basismodells ausfallen.

4.4 Rechenanforderungen

Zuletzt werden die Auswirkungen des Prunings auf die Rechenkomplexität (engl. Computational Complexity) und die benötigten Speicheranforderungen an die GPU (engl. GPU Memory Requirement) betrachtet. Die Ergebnisse der Evaluierungen sind in Tabelle 7 dargestellt. Zu sehen ist, dass mit der Abnahme der tatsächlichen Parameter auch die Rechenkomplexität ungefähr proportional sinkt. Bei einem angegebenen Pruning-Verhältnis von 30% und tatsächlich verbleibenden 73,06% der Parameter konnte die Rechenkomplexität um 29,13% reduziert werden.

Gleiches gilt für die Speicheranforderungen an die GPU, die in etwa im selben Maße abnehmen, wie die Anzahl der Parameter im Modell reduziert wird.

Pruning ratio	**Parameters left**	**Computational Complexity**	**Ratio**	**GPU Memory Requirement**	**Ratio**
Pruned 0%	100%	77.32 GMac	100%	2427.31 MiB	100%
Pruned 30%	73.06%	54.8 GMac	70.87%	1793.30 MiB	73,86%
Pruned 40%	69.21%	51.7 GMac	66.86%	1709.30 MiB	70,31%
Pruned 70%	55,08%	40.28 GMac	52,09%	1337.05 MiB	55,08%

Tabelle 7: Speicheransprüche nach dem Pruning

Das ist insofern nicht weiter überraschend, als die Speicheranforderungen und die benötigte Rechenleistung von der Anzahl der Parameter abhängen, von der wiederum die Anzahl der durchzuführenden mathematischen Operationen bestimmt wird.

5 Diskussion

Für die Diskussion werden die zuvor gezeigten Tabellen 6: Evaluierungen bis 70% Pruning und Tabelle 7: Speicheransprüche nach dem Pruning zu Rate gezogen. Begonnen wird mit der Analyse der Parameter Ratio.

Wie bereits in Kapitel 4.2 dargelegt, weicht das tatsächliche Verhältnis von dem angegebenen Verhältnis ab. Die Differenz steigt dabei mit zunehmender Pruning-Rate an. Bei der höchsten Pruning-Ratio (70%) ist mit einer actual ratio von 55,08% eine Abweichung um 25,08 Prozentpunkte gegenüber dem Erwartungswert von 30% zu beobachten. Wie in Kapitel 3.4 erklärt, besteht das neuronale Netz von TinyLlama-1.1B-Chat-v1.0 aus 22 Schichten. Dem Pruning wurden nur die Layer vier bis 18 unterzogen. Sieben Layer, also ungefähr ein Drittel, blieben somit auch nach dem Pruning vollständig erhalten. Im Umkehrschluss bedeutet dies, dass zur Erreichung der gewünschten specified ratio des Gesamtmodells die geprunten Layer noch stärker hätten reduziert werden müssen. Weitere mögliche Erklärungen für die abweichende actual ratio können sein, dass die analysierten Schichten für die Performance entscheidende Informationen enthielten und daher nicht aggressiv gepruned werden konnten. Hierfür spricht zusätzlich, dass das gewählte LLM bereits eine reduzierte Variante von Llama ist. Ebenso bleibt trotz Pruning häufig ein Overhead für Datenstrukturen und Kontrollmechanismen in gleichem Umfang bestehen. Denkbar ist auch, dass weitere architekturelle Besonderheiten, welche für die Gesamtfunktionalität notwendig sind, dem Pruning entgegenstehen.

Bei den Benchmarks zeigt TinyLlama-1.1B bereits in seiner Urform ohne Pruning mit einem Durchschnittswert von 39,58% eher mäßige Leistung. Erwähnenswert ist jedoch der Unterschied zwischen den einzelnen Tests. So schneiden BoolQ und WinoGrande mit $\tilde{6}0\%$ korrekt gelöster Aufgaben ab. ARC-c und OpenBookQA bilden mit 30,12% und 24,2% das Schlusslicht. Diese Verteilung bei der Erreichung von Güte lässt sich auch bei den reduzierten Modellen feststellen. Auffällig ist, dass ARC-c und OpenBookQA im Testverfahren auf zusätzlich separat bereitgestellte Informationen angewiesen sind. Dies scheint eine größere Herausforderung

darzustellen. Um eine Fehlbedienung auszuschließen, wurden im Internet Testergebnisse zum Vergleich gesucht. Auch wenn die Ergebnisse in der Punktzahl besser ausfielen, stellten Zhang, Zeng et al. ebenfalls ein schlechteres Abschneiden bei ARC-c und OpenBookQA im Benchmarking fest.[18] Zhang stellte zudem auf seinem Github-Account Messergebnisse von Betaversionen des LLMs bereit, die dasselbe Phänomen aufweisen.[19]

Bereits bei einem moderaten Pruning mit einer Ratio von 30% zeigt sich die L1-Norm mit Ø 31,77% als ungeeignet. Die Werte fallen in allen Tests schlechter als beim Random-Pruning (Ø 37,17%) aus. Gerade die Perplexity-Metriken fallen besonders negativ auf. Die Prunings mittels Taylor (Ø 39,10%) und L2-Norm (Ø 38,50%) liegen leicht über dem Random-Verfahren. Bei einem stark aggressiven Pruning mit einer angesetzten Ration von 70% zeigen alle vier Pruningverfahren ähnlich niedrige Durchschnittswerte.

Bei einem leicht aggressiven Pruning von 40% Ratio liegt der Referenzwert des Random-Verfahrens bei Ø 35,72%. Der Taylor- und der L2-Ansatz platzieren sich in der Messung mit ca. zwei Prozentpunkten über dem Zufallsverfahren (Taylor: Ø 37,89%; L2: Ø 37,59%). Für die gesamte Untersuchung betrachtet weisen die zwei Verfahren hier eine vertretbare Balance zwischen Reduktion des Modells und Vermeidung von Wissensverlust auf. Aufgrund weiter fehlender Messpunkte kann jedoch nicht abschließend konstatiert werden, dass 40% Ratio in Verbindung mit dem Taylor-Verfahren die beste Pruning-Option für TinyLlama-1.1B-Chat-v1.0 darstellt. Es wird daher empfohlen, weitere Testungen mit Quotienten von 35%, 45% oder 50% vorzunehmen und so den idealen Punkt besser einzugrenzen. Die L1-Norm kann bei den weiteren Messungen außen vorgelassen werden, da diese über alle Szenarien hinweg schlechtere Ergebnisse als der Zufall liefert.

Erschwerend kommt für eine abschließende Bewertung hinzu, dass das mit Taylor um 30% geprunte LLM nach einem Posttraining schlechtere Werte aufweist. Dies

[18] Vgl. *P. Zhang*; *Zeng*; *T. Wang* et al. (2024), S. 6.
[19] Vgl. *P. Zhang* (2023).

entspricht nicht der erwarteten Veränderung. Warum das Posttraining die durchs Pruning entstandenen Defizite nicht glättet, sondern verstärkt, kann nicht erklärt werden und bedarf weiterer Analyse.

6 Fazit

Hier kommt das Fazit hin!

Literatur

Cheng, *Hongrong*; *Miao Zhang*, *Javen Qinfeng Shi* (2024): *A Survey on Deep Neural Network Pruning: Taxonomy, Comparison, Analysis, and Recommendations*, Version 2 vom 09.08.2024, abgerufen am 08.02.2025, 2024, URL: https://arxiv.org/pdf/2308.06767

Clark, *Christopher*; *Kenton Lee*; *Ming-Wei Chang*; *Tom Kwiatkowski*; *Michael Collins*, *Kristina Toutanova* (2019): *BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions*, vom 24.05.2019, entnommen am 21.02.2025, 2019, URL: https://arxiv.org/pdf/1905.10044

Clark, *Peter*; *Isaak Cowhey*; *Oren Etzioni*; *Tushar Khot*; *Ashish Sabharwal*; *Carissa Schoenick*, *Oyvind Tafjord* (2019): *Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge*, vom 14.03.2018, abgerufen am 21.02.2025, 2019, URL: https://arxiv.org/pdf/1803.05457

He, *Yang*, *Lingao Xiao* (2023): *Structured Pruning for Deep Convolutional Neural Networks: A Survey*, Version 2 vom 30.11.2023, abgerufen am 30.01.2025, 2023, URL: https://arxiv.org/pdf/2303.00566

Hu, *Mengya Mia*; *Rui Xu*; *Deren Lei*; *Yaxi Li*; *Mingyu Wang*; *Emily Ching*; *Eslam Kamal*, *Alex Deng* (2024): *SLM Meets LLM: Balancing Latency, Interpretability and Consistency in Hallucination Detection*, Version 1 vom 22.08.2024, abgerufen am 29.01.2025, 2024, URL: https://arxiv.org/pdf/2408.12748

Huang, *Lei*; *Weijiang Yu*; *Weitao Ma*; *Weihong Zhong*; *Zhangyin Feng*; *Haotian Wang*; *Qianglong Chen*; *Weihua Peng*; *Xiaocheng Feng*; *Bing Qin*, *Ting Liu* (2024): *A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions*, Version 2 vom 19.11.2024, abgerufen am 29.01.2025, 2024, URL: https://arxiv.org/pdf/2311.05232

Kelbert, *Patricia*; *Julien Siebert*, *Lisa Jöckel* (2023): *Was sind Large Language Models? Und was ist bei der Nutzung von KI-Sprachmodellen zu beachten?*, abgerufen am 29.01.2025, 2023, URL: https://www.iese.fraunhofer.de/blog/large-language-models-ki-sprachmodelle/

Li, *Yawei*; *Kamil Adamczewski*; *Wen Li*; *Shuhang Gu*; *Radu Timofte*, *Luc Van Gool* (2022): *Revisiting Random Channel Pruning for Neural Network Compression*, vom 11.05.2022, abgerufen am 02.02.2025, 2022, URL: https://arxiv.org/pdf/2205.05676

Lu, *Qinghua*; *Liming Zhu*; *Xiwei Xu*; *Yue Liu*; *Zhenchang Xing*, *Jon Whittle* (2024): *A TAXONOMY OF FOUNDATION MODEL BASED SYSTEMS THROUGH THE LENS OF SOFTWARE ARCHITECTURE*, Version 1 vom 22.01.2024, abgerufen am 29.01.2025, 2024, URL: https://arxiv.org/pdf/2305.05352v6

Mihaylov, *Todor*; *Peter Clark*; *Tushar Khot*, *Ashish Sabharwal* (2018): *Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering*, vom 08.09.2018, abgerufen am 22.02.2025, 2018, URL: https://export.arxiv.org/pdf/1809.02789v1.pdf

Minaee, *Shervin*; *Tomas Mikolov*; *Narjes Nikzad*; *Meysam Chenaghlu*; *Richard Socher*; *Xavier Amatriain*, *Jianfeng Gao* (2024): *Large Language Models: A Survey*, Version 6 vom 20.02.2024, abgerufen am 29.01.2025, 2024, URL: https://arxiv.org/pdf/2402.06196

Naveed, *Humza*; *Asad Ullah Khan*; *Shi Qiu*; *Muhammad Saqib*; *Saeed Anwar*; *Muhammad Usman*; *Naveed Akhtar*; *Nick Barnes*, *Ajmal Mian* (2024): *A Comprehensive Overview of Large Language Models*, Version 10 vom 17.10.2024, abgerufen am 29.01.2025, 2024, URL: https://arxiv.org/pdf/2307.06435

Sakaguchi, *Keisuke*; *Ronan Le Bras*; *Chandra Bhagavaula*, *Yejin Choi* (2019): *WINOGRANDE: An Adversarial Winograd Schema Challenge at Scale*, Version 2 vom 21.11.2019, abgerufen am 22.02.2025, 2019, URL: https://arxiv.org/pdf/1907.10641

Sakai, *Yasufumi*; *Yu Eto*, *Yuta Teranishi* (2022): „Structured Pruning for Deep Neural Networks with Adaptive Pruning Rate Derivation Based on Connection Sensitivity and Loss Function“, in: *Journal of Advances in Information Technology* 13.3 (2022), S. 295–300

Sovrasov, *Vladislav* (2018): *ptflops: Ein Tool zur Berechnung von FLOPs für neuronale Netze im PyTorch-Framework*, 2018, URL: https://pypi.org/project/ptflops/

Vadera, *Sunil*, *Salem Ameen* (2022): „Methods for Pruning Deep Neural Networks“, in: *IEEE Access* 10 (2022), abgerufen am 18.02.2025, URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9795013

Wang, *Boyao*; *Rui Pan*; *Shizhe Diao*; *Xingyuan Pan*; *Jipeng Zhang*; *Renjie Pi*, *Tong Zhang* (2025): *Adapt-Pruner: Adaptive structural pruning for efficient small language Model Training*, vom 05.02.2025, abgerufen am 08.02.2025, 2025, URL: https://arxiv.org/pdf/2502.03460v1

Zellers, *Rowan*; *Ari Holtzman*; *Yonatan Bisk*; *Ali Farhadi*, *Yejin Choi* (2019): *HellaSwag: Can a Machine Really Finish Your Sentence?*, vom 19.05.2019, abgerufen am 21.02.2021, 2019, URL: https://arxiv.org/pdf/1905.07830

Zhang, *Peiyuan* (2023): *Evaluate TinyLlama*, Abgerufen am 23.02.2025, vom 28.12.2023, 2023, URL: https://github.com/jzhang38/TinyLlama/blob/main/EVAL.md

Zhang, *Peiyuan*; *Guangtao Zeng*; *Tianduo Wang*, *Wei Lu* (2024): „TinyLlama: An Open-Source Small Language Model“, Version 2, in: *arXiv* 2401.02385 (2024), Version 2 vom 04.06.2024, abgerufen am 23.02.2025, URL: https://arxiv.org/pdf/2401.02385

Zhu, *Michael*, *Suyog Gupta* (2022): *To prune, or not to prune: exploring the efficacy of pruning for model compression*, Version 2 vom 13.11.2017, abgerufen am 08.02.2025, 2022, URL: https://arxiv.org/pdf/1710.01878