Effiziente Reduktion großer Sprachmodelle: Methodenvergleich und Anwendungsstrategien für Energieoptimierung und Hardware-Kompatibilität

Efficient reduction of Large Language Models: Comparing methods and develop strategies to optimize energy footprint and hardware comatitibility

Masterarbeit

Dr. Thomas Schmitt

Fachhochschule Südwestfalen

3. September 2024

Autor:	Dr. Thomas Schmitt
Referent:	Prof. Dr. Heiner Giefers
Korreferent:	Max Kuhmichel
Eingereicht:	3. September 2024

Zusammenfassung

Große Sprachmodelle (English: Large Language Models oder kurz LLMs) wie „Chat-GPT“oder „Gemini“haben erstaunliche Fähigkeiten entwickelt und Einzug in unseren Alltag gefunden. Zur Entwicklung und dem Betrieb solcher LLMs benötigt man erhebliche Ressourcen, insbesondere leistungsfähige Grafikkarten (GPU) und große Speicherkapazitäten. Daher werden diese Modelle z.Zt. fast ausschließlich von kommerziellen Anbietern in großen Rechenzentren betrieben und setzen somit eine Internet Verbindung zur Nutzung voraus.
In dieser Arbeit wird die Möglichkeit untersucht LLMs zu komprimieren, ohne dabei wesentliche Verluste ihrer Fähigkeiten in Kauf nehmen zu müssen. Existierende Technologien zur Kompression werden untersucht und kombiniert, um möglichst kleine Modelle zu erhalten, die im Idealfall auf normaler Consumer-Hardware kostengünstig betrieben werden können und so neue (Offline-)Einsatzszenarien erschließen.

Inhaltsverzeichnis

1 Einleitung

1.1 Einführung zu Natural Language Processing

Schon seit den 1950er Jahren gibt es Bestrebungen natürliche Sprache für Maschinen verständlich zu machen. Das zugehörige Wissensfeld bezeichnet man dabei als „Natural Language Processing (kurz NLP)“[1]. In dieser Disziplin wurden Techniken aus Linguistik und Informatik kombiniert, um die Struktur der Sprache programmatisch abbilden zu können. Ein erstes eindrucksvolles Beispiel erstellte Joseph Weizenbaum mit der Simulation ELIZA, die einen Psychotherapeuten nachahmte. Immerhin jeder fünfte Proband glaubte mit einem echten Therapeuten zu interagieren, obwohl nur recht simple Grammatik Regeln verwendet wurden [2]. ELIZA wird gerne auch als der erste Chatbot bezeichnet.

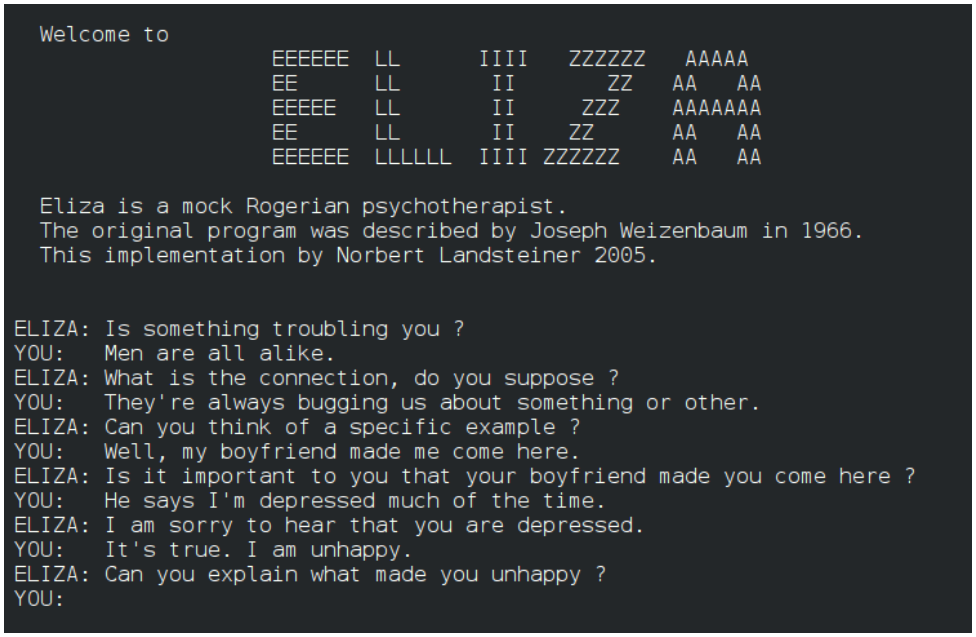

Abbildung 1.1: Beispiel einer ELIZA Konversation [1].

Einige Jahrzehnte waren die Fortschritte in diesem Gebiet gering. Erst in den letzten Jahren wurden, durch die rasante Entwicklung des Machine Learnings und speziell des Deep Learnings, auch im Bereich der Sprachadaption für Computer, erhebliche Verbesserungen erzielt.
Das Aufkommen des Deep Learnings markierte einen Paradigmenwechsel in der Entwicklung von Sprachmodellen. Neuronale Netze, insbesondere rekurrente neuronale Netze (RNN) und Netze mit langem Kurzzeitgedächtnis (LSTMs), brachten bemerkenswerte Verbesserungen bei der Verarbeitung sequenzieller Daten.
Diese frühen Deep Learning Modelle ermöglichten ein besseres Sprachverständnis, ihre Skalier-

barkeit war jedoch begrenzt, da die Eingangssequenz bei diesen Modellen sequenziell, also Wort für Wort, abgearbeitet wurde, was ihr Verständnis komplexerer Texte erheblich einschränkte.

1.1.1 Transformers als Kerntechnologie moderner Sprachmodelle

Ein technologisch bedeutender Schritt zu leistungsfähigeren Sprachmodellen war die Entwicklung der sogenannten Transformer Modelle, die in [3] erstmalig 2017 beschrieben wurden. Da alle aktuellen LLMs auf dieser Technologie oder Varianten davon basieren erfolgt hier eine kurze Vorstellung dieser Technologie.
Im Gegensatz zu den zuvor genannten Verfahren heben sich Transformer dadurch ab, dass sie in der Lage sind ganze Absätze bzw. Abschnitte parallel zu verarbeiten und somit Beziehungen zwischen weiter entfernten Textteilen erfassen können. Dieser Unterschied ist im wesentlichen für die stark verbesserten Fähigkeiten, der auf dieser Technologie basierenden Transformer Modelle verantwortlich.
Grundlage für die Erfassung weiter entfernter textlicher Abhängigkeiten ist der Aufmerksamkeitsmechanismus (engl. Attention), welcher in der Lage ist, textliche Beziehungen zu bewerten und mathematisch abzubilden. 1.2 zeigt den Aufbau einer Transformer Architektur, wie er in [3] diskutiert wird. Ein Transformer besteht in diesem Beispiel aus einer Encoder und einer Decoder Struktur die über einen zweiten Aufmerksamkeitsmechanismus gekoppelt sind.

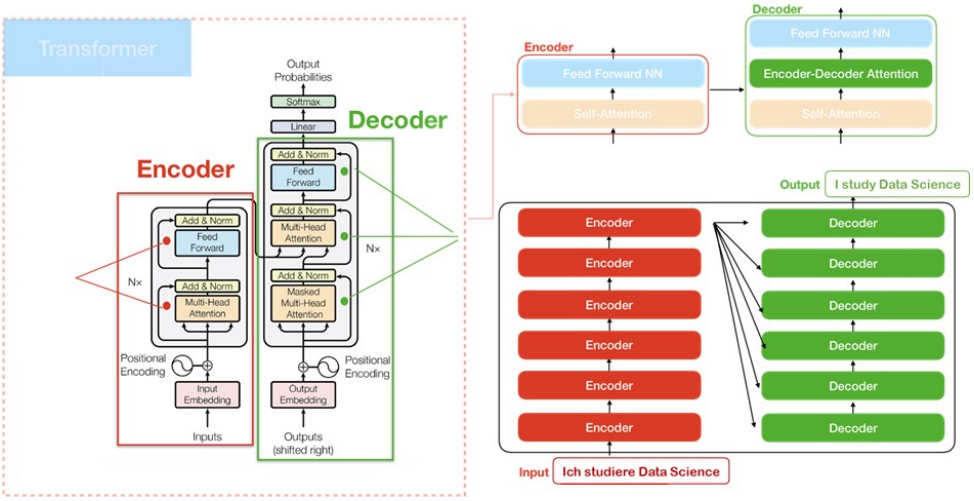

Abbildung 1.2: Schematische Darstellung der Transformer Architektur mit einem Beispiel [4]

Zunächst werden im Embedding Layer des Encoders die einzelnen Wörter der Eingabe in einen Vektor abgebildet. Diese Vektoren fließen dann durch die Self-Attention und Feed Forward Schicht des Encoders. Entscheidend ist dabei, dass jedes Wort in jeder Position durch verschiedene Pfade des Encoders fließt.
Die Self-Attention Schicht ist dadurch in der Lage die relative Bedeutung eines Wortes, im Verhältnis zu den anderen Worten des Textes, zu erfassen und bildet diese relativen Positionsinformationen in weiteren Vektoren ab.
Auf Seite des Decoders wird derselbe Prozess durchgeführt, wobei der einzige Unterschied dabei die Cross-Attention Schicht ist, welche die erlernten Attention Signale des Encoders auf den Decoder überträgt. Diese, hier als Multi-Head-Attention bezeichnete, Schicht unterstützt den Decoder bei der Konzentration auf die relevanten Wörter des Eingabesatzes. Letztlich

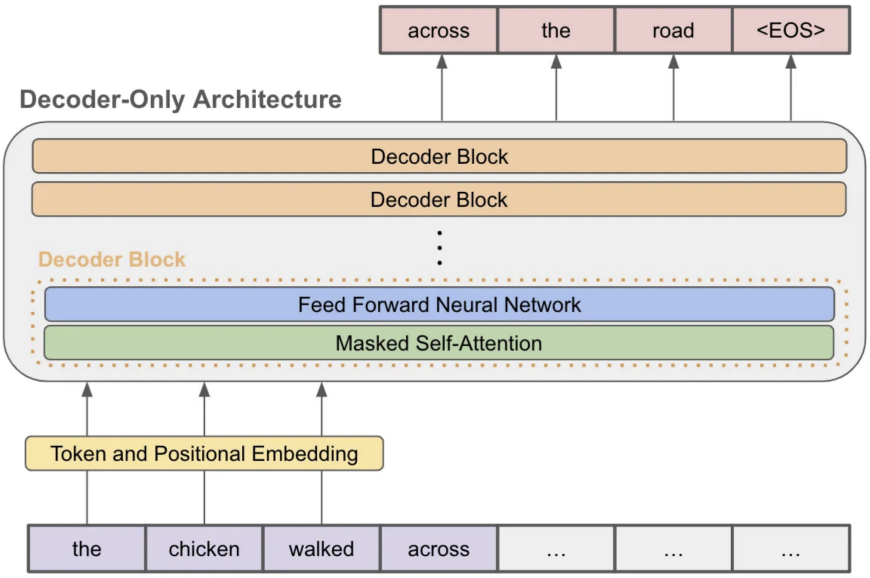

Abbildung 1.3: Schematische Darstellung einer Decoder Architektur [7].

bestimmt der lineare Layer eines neuronalen Netzes, dann das wahrscheinliche nächste Wort in der Ausgabesequenz.

Eines der ersten Modelle, welches die Fähigkeiten dieser Technologie demonstrierte war BERT [5] dessen Transformer Architektur als „Encoder only Transformer“Struktur bezeichnet wird. Bei den GPT Modellen von Open AI [6] wird hingegen eine „Decoder only Struktur“verwendet. Welche Struktur gewählt wird ergibt sich vor allem aus dem Anforderungsprofil für das Modell. Bei einem bidirektionalen Übersetzungsmodell verspricht eine „Encoder only Struktur“deutlich bessere Ergebnisse im Sprachverständnis und damit in der Übersetzungsqualität. Für das einfachere Generieren von Text durch einen Chatbot, überwiegen die Vorteile des „Decoder only“Ansatzes.

1.2 Entstehung der Large Language Models (LLMs)

Die neueren Arbeiten im Bereich Maschine Learning führten zu Sprachmodellen, die auf eine Frage oder Aufforderung mit einer Antwort reagieren, wie sie auch durch einen Menschen gegeben würde. Solche Sprachmodelle entstehen i.d.R. durch das Training neuronaler Netze mit großen Textmengen aus verschiedenen Disziplinen. Dies geschieht durch die Nutzung leicht zugänglicher textlicher Informationen, wie Beispielsweise von Wikipedia Inhalten. Man unterscheidet grundsätzlich die Trainingsphase von der Nutzungsphase der Modelle. Die Nutzungsphase wird dabei meist als „Inference“, was im Deutschen mit „Schlussfolgerung“übersetzt werden könnte, bezeichnet.

Eines der auf diese Weise generierten Sprachmodelle, ist das von OpenAI erstellte Chat-GPT, das in der Version 3.5 am 30 November 2022 einer größeren Öffentlichkeit zugänglich gemacht wurde Das Kürzel GPT steht dabei für „Generative Pretrained Transformer“. Durch die einfache Interaktion, in Form eines Chatbots, konnten auch Laien leicht mit

dem Sprachmodell interagieren und sich von den Möglichkeiten der KI einen Eindruck verschaffen.

Die Entwicklung von GPT-3.5 stellte auch vom Aufwand eine neue Dimension des Trainings für ein Sprachmodell dar, welcher in einem Modell mit 175 Milliarden Parametern resultierte. Spätestens seit diesem Zeitpunkt werden solche Modelle als Large Language Models (LLMs) bezeichnet [8].
Getrieben durch die eindrucksvollen Resultate, erfolgte in den nächsten Monaten eine intensive Auseinandersetzung von Medien, Gesellschaft und Politik mit unterschiedlichsten Aspekten, die der Einsatz von weit entwickelten KIs und insbesondere den Sprachmodellen mit sich bringt. Der Produktname Chat-GPT in der Allgemeinheit zu einem Synonym für Large Language Models und den breiten Einsatz solcher Modelle in verschiedene Lebensbereiche geworden. Chat-GPT wurde seitdem mehrfach aktualisiert und liegt z.Zt (1.8.2024) in der Version 4.0 vor. [9]

1.2.1 Entwicklung seit 2022

Die Entwicklung neuer Sprachmodelle wurde in den letzten Jahren stark vorangetrieben. Insbesondere die amerikanischen Technologie Konzerne wie OpenAI, Google, Anthropic und andere, aber auch europäische Firmen wie Mistral erstellen neue Modelle mit einer immer größeren Parameteranzahl.
Besteht Chat-GPT 3.5 noch aus 175 Milliarden Parametern vermutet man, das bereits GPT-4.0 (OpenAI) und Gemini 1.5 (Google) die Billionen Grenze überschritten haben. In einigen Fällen - wie bei Google Gemini - handelt es sich dabei nicht mehr um reine Sprache, sondern um multimodale Modelle, die verschiedene Arten von Daten parallel verarbeiten können. Dazu gehören neben Text auch Sprache, Videos, Bilder und Programmcode. [10].

Daneben haben einige der großen genannten Technologie Konzerne auch Modelle veröffentlicht, die von der Allgemeinheit kostenfrei genutzt werden können. Hier seien insbesondere der Meta Konzern mit seinen Llama Modellen und das relativ neue französische Unternehmen Mistral AI mit seinem gleichnamigen Modell Mistral 7B [11] genannt. Modelle beider Unternehmen werden in dieser Arbeit genutzt und im Teil 2.1 noch genauer beschrieben.
Natürlich unterliegen auch diese Modelle durch ihre Lizenzbedingungen gewissen Beschränkungen, die auch im Bereich der Lehre beachtet werden müssen. Z.Bsp. muss bei der Weitergabe von den auf Meta Llama basierenden Modellen, deren Herkunft angegeben werden. In Summe bilden solche Modelle allerdings eine gute Möglichkeit für Forschung, Lehre und sogar die kommerzielle Nutzung im kleinen Rahmen.

Die in diesem Kapitel beschriebenen Modelle werden auch als „Foundation Models“(Deutsch in etwa: Grundlagenmodell) bezeichnet, da sie auf einer sehr breiten Themenpalette trainiert werden und daher in vielen Bereichen einsetzbar sind. Darüber hinaus besitzen sie ein gutes Sprachverständnis, allgemeine Fähigkeit zu logischen Schlussfolgerungen und eine gute Lernkompetenz für weitere Inhalte. In Ergänzung dazu werden viele speziellere Modelle von Unternehmen, Hochschulen und Enthusiasten erstellt, die häufig auf den „Foundation Modellen“basieren, diese aber für spezifische Zwecke abwandeln.
Eine wichtige Richtung dieser Spezialisierung ist auch die Anpassung solcher Modelle auf einen geringeren Bedarf an GPU und Speicherkapazität, so dass sie in Umfeldern genutzt werden können, die über limitierte Ressourcen verfügen.

1.2.2 Ressourcenbedarf

Das Training großer Sprachmodelle, aber auch deren Nutzung, geht mit einem erheblichen Ressourcenbedarf einher. Insbesondere sind die Grafikkarten (GPUs) des Herstellers Nvidia z.Zt. eine notwendige Voraussetzung für das Training und die Nutzung großer Sprachmodelle. Andere Hersteller von Grafikprozessoren, wie AMD oder Intel, schließen technologisch erst langsam zu Nvidia auf, so dass dieser eine Beinah-Monopolstellung für das Training von LLMs einnimmt.
Wie viele GPUs für das Training eines Modells in der Größe von GPT-3.5 verwendet wurden ist leider nicht veröffentlicht. Es wird aber gemeinhin davon ausgegangen, dass ca. 10.000 GPUs parallel für das Training des Modells benötigt wurden und zur Zeit ca. 30.000 GPUs für die Inference durch Anwender genutzt werden.
Die Firma Meta mit ihren Llama Modellen ist etwas offener in der Kommunikation, und gewährt in [12] einen Einblick in ihr Rechenzentrum inklusive des geplanten weiteren Ausbaus. Wurde Llama 3 noch auf „nur“zwei Clustern a 24.000 GPUs gerechnet, plant Meta einen Ausbau der Kapazitäten bis Ende 2024 auf 350.000 GPUs der neuesten, leistungsfähigeren Generation.

Alleine die finanziellen Aufwände für die benötigten GPUs bei der Entwicklung eines neuen Modells, stellen einen erheblichen Kostenblock dar, der in [13] mit ca. einem Drittel der Gesamtkosten beziffert wird.
Die dabei benötigte Hardware wird nicht immer, wie z.Bsp. bei Meta, in eigenen Rechenzentren betrieben, sondern in den Rechenzentren der großen Cloud Computing Anbieter, wie Google oder Microsoft, implementiert und dann als Service bezogen. Dies verringert den Anlaufaufwand für die Entwicklung des Modells und begrenzt die Kapitalbindung durch anfängliche Hardwareinvestitionen.

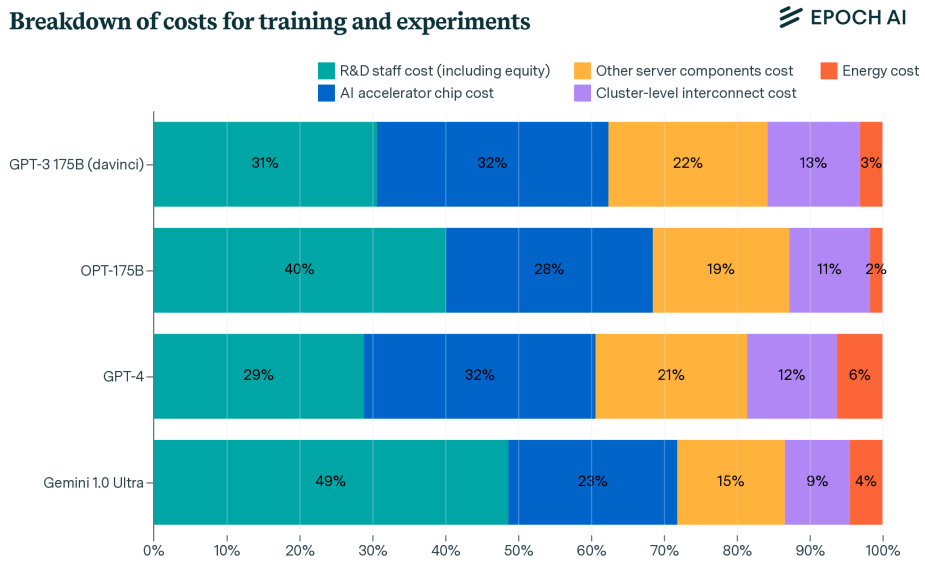

Abbildung 1.4: Kostenanteile bei der Entwicklung ausgewählter KIs [13]

Die Quelle [13] extrapoliert auch die Kostenentwicklung für zukünftige KI Modelle, die wie oben bereits erwähnt, deutlich mehr Parameter als die Aktuellen enthalten werden [14]. Der Kostenanstieg wird dabei im Durchschnitt mit dem 2,4 fachen pro Jahr für die nächsten Jahre prognostiziert. Siehe auch Abbildung 1.5
Daher ist zu erwarten, dass neue Foundation Modelle in den kommenden Jahren hauptsächlich

von sehr finanzkräftigen Konzernen entwickelt werden, welche in der Lage sind die notwendigen Investitionen zu tätigen. Hochschulen und öffentliche Forschungsinstitute haben i.d.R. kaum die Mittel in ähnlichen Größenordnungen zu agieren.
Um so bemerkenswerter ist die Initiative der EU den „Exascale Rechner“Jupiter im Forschungszentrum Jülich zu etablieren, der unter anderem auch für die Entwicklung von KI Foundation Modellen genutzt werden soll [15].

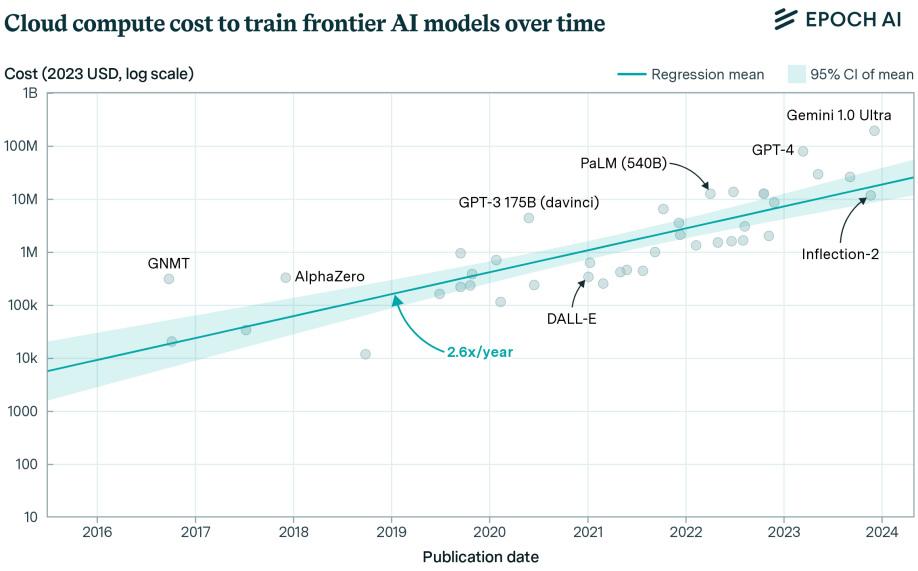

Abbildung 1.5: Kostenentwicklung neue KI Modelle [13]

Wie weiter oben bereits erwähnt, ist eine Nutzung der nativen Foundation Modelle ohne leistungsfähige und entsprechend teure GPUs aus dem Hause Nvidia kaum sinnvoll möglich. Als kritisches Maß gilt hier vor allem die Latenzzeit bei der Inferenz, d.h. die Zeitspanne bis das Modell eine Antwort liefert. Eine Abschätzung der Firma Nvidia besagt, dass aktuelle Modelle mit 16 bit Fließkomma Zahlen ca. 2 Gigabit GPU Speicher pro einer Milliarde Parameter benötigen. Dazu kommt noch weiterer Speicherbedarf, der für Eingabe, Ausgabe und Inferenz allokiert wird. Dies bedeutet, dass selbst die kleinsten Modelle von Meta und Mistral, die aus 7 bzw. 8 Milliarden Parametern bestehen, schon eine leistungsfähige GPUs mit mindestens 16 Gigabyte (bzw. eher mehr) Hauptspeicher für eine flüssige Nutzung benötigen. Geeignete GPUs liegen preislich z. Zt. zwischen 10.000 und 20.000 € und sind damit für Privatpersonen, Schulen, kleine Firmen usw. kaum erschwinglich. Alternativ können Cloud Ressourcen genutzt werden, was zumindest bei zeitlich begrenzter Nutzung eine finanziell tragbare Option darstellt.

Ein weiterer Aspekt ist auch die Inferenz Nutzung von LLMs auf sogenannten „Edge Devices“. Als „Edge Devices“bezeichnet man neben persönlichen IT Geräten wie Laptops, Workstations oder Smartphones, auch Smart TVs, Fahrzeuge, Sensorik und v.a.m.. In der Regel handelt es sich also um Geräte mit begrenzten Rechenressourcen und einer Kostenstruktur, die den Einbau kostspieliger Zusatzchips nicht erlaubt.
Wünschenswert wären daher Modelle, die auch ohne permanente Online Verbindung, auf solchen Geräten autonom in der Lage sind eine Inferenz mit qualitativ vernünftigen Ergebnissen zu ermöglichen.

Ein Ansatz dies zu erreichen, ist die Verkleinerung der Modelle bei weitestgehend gleichbleibender Qualität.
In den letzten Jahren wurden verschiedene Verfahren entwickelt, die eine Reduktion der Ressourcenanforderungen von LLMs potentiell ermöglichen. Ziel dieser Arbeit ist den jeweiligen Effekt dieser Methoden auf gängige LLMs zu ermitteln, die Verfahren zu vergleichen und ggf. sinnvoll zu kombinieren, um einen maximalen Effekt zu erreichen.

1.3 Verfahren zur Verringerung der Ressourcenanforderung von LLMs

Es existieren unterschiedliche Ansätze, um den Ressourcenbedarf von LLMs zu reduzieren. Zunächst werden die direkten Ansätze zur Größen und Komplexitätsreduzierung näher besprochen. Dabei handelt sich um das „Pruning“und die „Quantization“von LLMs.
Das Knowledge Distillation Verfahren hingegen versucht die Fähigkeiten eines größeren Modells (Teacher) auf ein kleineres Modell (Student) zu übertragen, was im Resultat ebenfalls den Ressourcenbedarf im Verhältnis zu den Fähigkeiten des resultierenden Modells gering halten soll.
Im weiteren wird mit „Low Rank Adaption“eine weitere Technologie charakterisiert die helfen solche verkleinerten Modelle qualitativ erneut zu verbessern, so dass sie bei den Inferenz Ergebnissen wieder möglichst nah an den Ursprungsmodellen liegen.

1.3.1 Pruning von LLMs

Der Begriff "Pruningümfasst verschiedene Verfahren, die zum Ziel haben, für die Qualität wenig relevante Anteile eines deep neuronal networks zu entfernen. Letztlich resultiert dies in einem ausgedünntem Netzwerk von geringer Größe. Positive Effekte sind im günstigen Fall ein verringerter Speicherbedarf und eine schneller Inferenz bei ähnlicher Qualität im Vergleich zur Ausgangssituation. [16] klassifiziert in einem umfangreichen Survey von 2023, drei Kategorien von Pruning die im folgenden besprochen werden.

Unstrukturiertes Pruning In dieser Kategorie werden Methoden gesammelt, die rein auf die Bedeutung einzelner Gewichte abzielen unabhängig von der Gesamtstruktur des DNNs.
Zunächst wird die Bedeutung der einzelnen Gewichte im Netzwerk bewertet. Dies geschieht typischerweise durch die Analyse des Betrags der Gewichte, wobei Gewichte mit kleineren Beträgen als weniger wichtig angesehen werden. Es gibt aber auch andere Methoden, die die Sensitivität des Verlusts gegenüber den Gewichten betrachten.
Die als weniger wichtig eingestuften Gewichte werden auf null gesetzt, also "gepruned". Im Gegensatz zum strukturierten Pruning werden beim unstrukturierten Pruning einzelne Verbindungen zwischen Neuronen (also Gewichte) entfernt, ohne dabei Rücksicht auf die zugrunde liegende Struktur des Netzwerks zu nehmen [17].
Die resultierenden Netzwerkstrukturen können sehr unregelmäßig sein, was die Hardware-Optimierung (wie z.B. auf GPUs) erschweren kann. Außerdem ist oft ein intensives Retraining notwendig um eine befriedigende Qualität zu erreichen. Dem gegenüber steht der Vorteil einer feineren Kontrolle und potenziell höhere Kompressionsraten, als bei strukturiertem Pruning.

Strukturiertes Pruning Beim strukturierten Pruning werden ganze Strukturen innerhalb des Modells entfernt, wie z.B. ganze Neuronen, Heads in einem Attention-Mechanismus oder sogar ganze Layer. Im Unterschied zum unstrukturierten Pruning bleibt das Modell dadurch strukturell regelmäßig und die resultierenden Matrizen sind dicht. Die Notwendigkeit zum Retraining des Modells ist durch die größere Homogenität geringer und es werden keine zusätzlichen Software Komponenten für die Inferenz benötigt.

Semi-strukturiertes Pruning In den letzten Jahren etablieren sich Verfahren, die versuchen Vorteile beider Ansätze zu kombinieren, d.h. sowohl die Flexibilität des Unstructured Pruning als auch die Effizienz des Structured Pruning zu nutzen.
Beim semi-strukturierten Pruning werden zusammenhängende Gruppen von Gewichten innerhalb einer bestimmten Struktur entfernt. Diese Gruppen können Teilstrukturen innerhalb einer größeren Struktur sein, wie z.B. bestimmte Reihen oder Spalten einer Gewichtsmatrix, aber nicht die gesamte Struktur (wie im strukturierten Pruning).

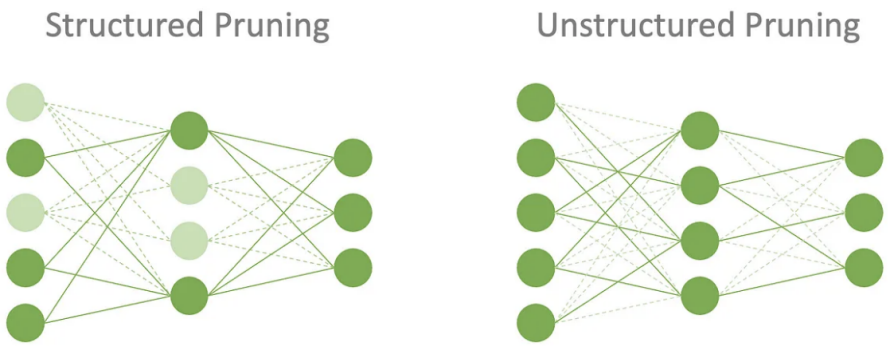

Abbildung 1.6: Strukturiertes und Unstrukturiertes Pruning.
Die linke Grafik zeigt wie komplette Strukturbereiche aus dem Netzwerk genommen werden. Die übrigen Strukturen und ihre Beziehungen bleiben unverändert erhalten. Beim unstrukturierten Pruning hingegen wird ein Großteil der Beziehungen verschlankt, während die Struktur vollständig erhalten bleibt.

1.3.2 Quantisierung von LLMs

Quantisierung ist ein weit verbreitetes Verfahren zur Verkleinerung der Speicherauslastung durch LLMs und zur Beschleunigung der Interferenz, ohne dabei signifikant an Genauigkeit zu verlieren Damit werden sowohl die Kosten reduziert, wie auch der Energieaufwand verringert.
Grundlage des Prozesses ist es, die numerische Präzision der Weights in einem Modell zu verringern. Typischerweise werden LLMs mit 32-Bit-Fließkommazahlen (FP32) trainiert, was eine hohe Präzision gewährleistet. Bei der Quantisierung wird diese Präzision auf 16-Bit-Fließkommazahlen (FP16), 8-Bit-Integer (INT8) oder 4-Bit-Integer, und in extremen Fällen sogar auf 2-Bit-Integer Zahlen, reduziert. Alleine die Nutzung von Integer Variablen im Gegensatz zu Fließkomma Variablen (Engl: Floating Point oder kurz FP)vermindert schon erheblich den Speicherbedarf, da nicht für jeden Wert der zugehörige Exponent gespeichert werden muss. Da die meisten Werte in einem Modell sich in einer ähnlichen Größenordnung bewegen, ist der Genauigkeitsverlust durch die Transformation in Integer Variablen sehr gering.

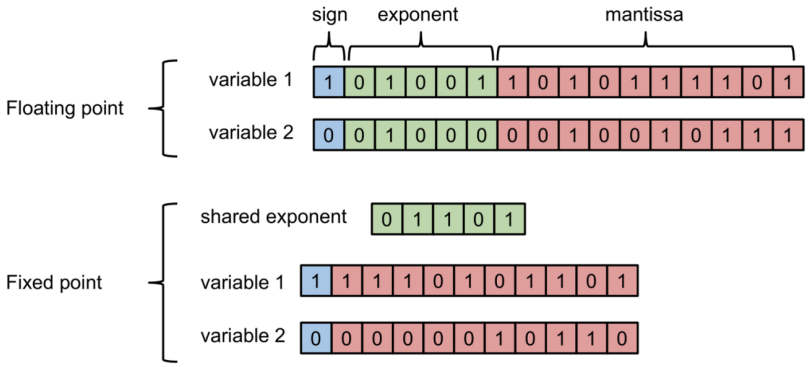

Abbildung 1.7: Speicherbedarf von Floating Point und Integer Variablen [18]

Darüber hinaus lässt sich noch mehr Speicherplatz einsparen, wenn Die Zahlen in Integer Variablen mit geringer Bitbreite abgebildet werden. Der Speicherbedarf einer 4Bit- Integer Variable ist um den Faktor 8 geringer als ein 32-Bit-FP Variable. In erster Näherung gilt dieses Verhältnis von 1:8 auch für den Speicherbedarf eines gesamten auf diese Weise quantisierten Modells.

xxx BILD

Zeitpunkt der Quantisierung

Es gibt generell zwei unterschiedliche Zeitpunkte im Lebenszyklus eines Modells, an denen eine Quantisierung sinnvoll ist.

Quantization-Aware Training (QAT) Bei dieser Methode wird das Modell während des Trainings so angepasst, dass es die Einschränkungen der Quantisierung berücksichtigt. Dadurch kann das Modell lernen, robust gegenüber der niedrigeren Präzision zu sein, was oft zu besseren Ergebnissen führt als das zweite Verfahren PTQ.

Post-Training-Quantisierung (PTQ) Diese Methode wird nach dem Training des Modells angewendet. Hierbei wird das Modell von einer höheren Präzision (z. B. FP32) auf eine niedrigere Präzision (z. B. INT8) umgewandelt. PTQ ist einfach und erfordert keine Änderung am Trainingsprozess, kann aber in manchen Fällen zu einem gewissen Verlust an Modellgenauigkeit führen.

Da in dieser Arbeit mit bereits existierenden Foundation Modellen gearbeitet wird, kommen hier Verfahren für die Post-Training-Quantisierung (PTQ) zum Einsatz.

1.3.3 Verfahren für Post-Training-Quantisierung

Es gibt verschiedene Implementierungen von PTQ die sich in der Plattform Unterstützung, der Flexibilität bei den möglichen Bit Raten, der Ergebnisqualität und anderen Kriterien unterscheiden [19]. In dieser Arbeit steht die Ergebnisqualität im Vordergrund, weswegen vor allem zwei neuere Verfahren vorgestellt werden sollen, die in dieser Hinsicht besonders überzeugen können.

Post-Training Quantization For Genertaive Pre-Trained Transformers (GPTQ) [20] nutzt eine gradientenbasierte Methode zur Minimierung der Quantisierungsfehler, um ein optimiertes Modell, mit nur geringe Präzisionsverlusten, zu generieren.
Die Gewichte des Modells werden dabei schrittweise quantisiert. Dies geschieht in einem „greedy“ Verfahren, bei dem in jedem Schritt das Gewicht oder die Gruppe von Gewichten ausgewählt wird, deren Quantisierung den geringsten Einfluss auf die Gesamtleistung des Modells hat. Dabei wird die inverse Hessian-Matrix verwendet, um die Sensitivität der Verlustfunktion gegenüber Änderungen der Gewichte zu bewerten. Nach jedem Schritt wird die verbleibende Menge an zu quantisierenden Gewichten neu bewertet, und der Prozess wird iterativ fortgesetzt.

Activation Aware Weight Quantization (AWQ) [21] ist ein fortschrittlicher Ansatz zur Quantisierung von neuronalen Netzen, der nicht nur die Gewichte, sondern auch die Aktivierungen des Modells während des Quantisierungsprozesses berücksichtigt. Die Autoren beschreiben, dass ein relativ geringer Anteil von 0,1 - 1 % der sogenannten 'salient weights' (Deutsch in etwa: bedeutende Gewichte) den größten Einfluss auf die Modell Performance besitzen.
AWQ schützt diese Gewichte bei der Quantisierung indem es die zugehörige Gruppe (bzw. den "Channel") mit einem Multiplikationsfaktor versieht. Empirisch ließ sich zeigen, dass dieser Ansatz sehr effektiv ist.
Insbesondere hebt er sich ab, von einem sogenannten 'Mixed precison Ansatz' bei dem die 'salient weights' in einem anderen Variablentyp (FP16) als die restlichen Gewichte (int 4) gespeichert werden. Dieser führt zwar zu einer vergleichbaren Präzision, kann aber auf normaler Hardware nur ineffizient umgesetzt werden.

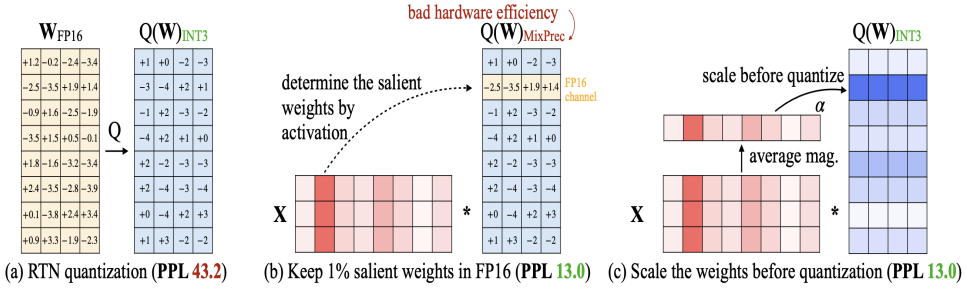

Abbildung 1.8: Vergleich der Quantisierungsmethoden, wie im Text beschrieben [21]
(a) Zeigt eine einfache Quantisierung mittels eines Rundungsverfahrens (RTN quantization), während (b) 'salient weights' im 'Mixed precision Ansatz' und (c) den AWQ Ansatz darstellt. Der Perplexity Wert (PPL) zeigt die Qualität, der auf diese Weise generierten Modelle.

In dieser Arbeit wird vor allem das AWQ Verfahren angewendet, da sich in Vorversuchen seine Überlegenheit gegenüber den anderen Verfahren bereits angedeutet hatte.

1.3.4 Knowledge Distillation

Knowledge Distillation ist eine Technik des maschinellen Lernens, die darauf abzielt, das Wissen eines großen, leistungsstarken Modells (genannt Teacher Model) auf ein kleineres, effizienteres Modell (genannt Student Model) zu übertragen. Dieses Verfahren ermöglicht es, ein kleines Modell zu trainieren, das nahezu die gleiche Leistung wie das große Modell erbringt, jedoch einen geringeren Speicherbedarf hat und eine schnellere Inferenz ermöglicht.
Unterschieden wird dabei zwischen „white box“und „black box“Knowledge Distillation:

- White Box - In der Regel bezeichnet dies die Arbeit mit open source Modellen, bei denen auf die einzelnen Antwortwahrscheinlichkeiten zugegriffen werden kann, die sich in der Form von Logits (s.u.) darstellen lassen.

- Black Box - In der Regel kommerzielle Modelle, bei denen nur die Frage/Antwort Paare für die Knowledge Distillation zur Verfügung stehen.

Im Folgenden wird nur noch der „White Box“Ansatz betrachtet, da in dieser Arbeit Open Source Modelle verwendet werden.
Für die Distillation wird zunächst das Teacher Modell mit einem Datensatz trainiert. Anstatt nur die harten Klassenvorhersagen (z.B. eine 0 oder 1 bei einer Klassifikationsaufgabe) zu verwenden, nutzt Knowledge Distillation die Wahrscheinlichkeitsverteilungen, die das Teacher Model für jede Klasse ausgibt. Diese Wahrscheinlichkeiten enthalten mehr Informationen als nur die korrekte Klasse, weil sie auch anzeigen, wie sicher das Modell in Bezug auf alle Klassen ist. Diese Ergebnisse bezeichnet man als SSoft Targets".
Das Student Model wird dann trainiert, um die Soft Targets des Teacher Models zu imitieren, anstatt nur die harten Labels der Trainingsdaten. Dadurch lernt das Student Model nicht nur, welche Klasse korrekt ist, sondern auch, wie sicher das Teacher Model bei seiner Antwort ist.

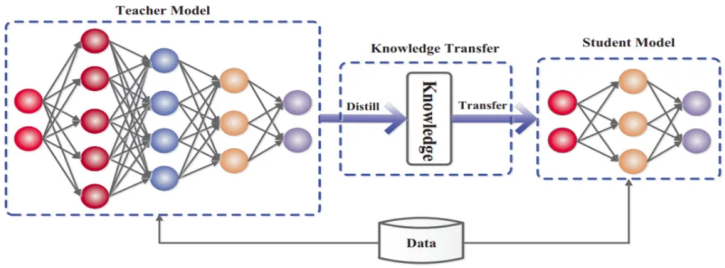

Abbildung 1.9: Schematische Darstellung von Knowledge Distillation. [22].

Response based Knowledge Distillation ist das bedeutendste Verfahren, das zur Durchführung verwendet wird. Als Trainingsgrundlage werden dabei die sogenannten Logits des Teacher Models verwendet. Logits sind die Ausgaben des neuronalen Netzes als reelle Zahlen und können damit sowohl positive wie negative Werte annehmen.
Durch die Anwendung der Softmax-Funktion auf die Logits entstehen daraus die oben erwähnten Soft Targets, welche dann zum Training des Student Models genutzt werden. Als Maß für den Trainingserfolg werden Logits von Teacher und Student Model miteinander verglichen, um daraus den 'Distillation Loss' zu ermitteln. Das Ziel ist somit eine möglichst starke Annäherung der Logits von Teacher und Student Model durch den Knowledge Distillation Prozess.

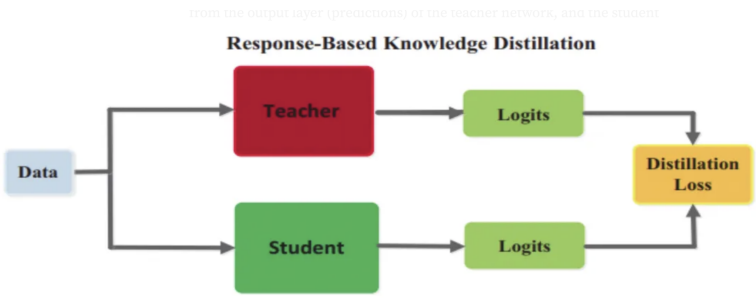

Abbildung 1.10: Schematische Darstellung der Response based Knowledge Distillation. [22].

1.3.5 Low Rank Adaption (LoRA)

Diese Technologie hat nicht, im Unterschied zu den vorher gehenden, das primäre Ziel den Ressourcenbedarf von fertigen LLMs zu verringern und/oder deren Effizienz zu steigern [23]. Vielmehr geht es hier im Wesentlichen darum das LLM mit anderen (neuen) Inhalten zu trainieren bzw. spezielle Fähigkeiten zu verbessern. Der Vorteil ist, dass mit LoRA da bereits fertige Modell nicht verändert werden muss, sondern die neuen Fähigkeiten über einen 'Adapter' dem Modell hinzugefügt werden.
Dies ermöglicht eine große Flexibilität zur Anpassung der oben genannten Foundation Modelle. Darüber hinaus auch kommt diese Technologie auch mit wenig (zusätzlichen) Ressourcen aus, so das der Effizienz Gedanke sekundär auch eine Rolle spielt.
Diese Technologie wird hier angesprochen, da sie sich auch gut zum Re-Training von Modellen eignet, die zuvor mittels Pruning ausgedünnt wurden. Vorversuche haben gezeigt, dass auf diese Art verkleinerte Modelle erheblich von einem Re-Training mit LoRA profitieren können.

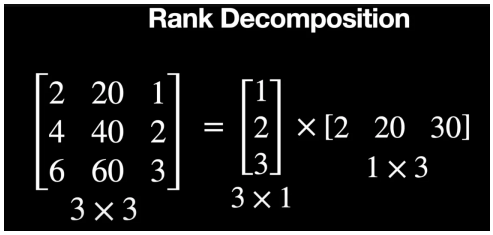

Abbildung 1.11: Zeigt die Dekomposition einer höherrangigen 3×3 Matrize in zwei niederrangigere 3×1 bzw. 1×3 Matrizen.

LoRA benutzt die Zerlegung von komplexeren Matrizen in niederrangigere Matrizen [24] . Anstatt alle Gewichte des Modells zu aktualisieren, wie es bei einem herkömmlichen Fine-Tuning der Fall wäre, führt LoRA eine „Low-Rank"-Dekomposition der Gewichtsmatrizen durch. Konkret werden die ursprünglichen Gewichtsmatrizen W in zwei kleinere Matrizen A und B zerlegt, wobei die beiden neuen Matrizen eine niedrigere Rangordnung (daher 'low-rank') haben. Die Anpassung wird dann nicht auf allen Parametern des Modells durchgeführt, sondern nur auf einem Teilraum, der durch „low-rank"-Matrizen beschrieben wird.
Die Idee ist, dass man W durch $W + A \times B$ ersetzen kann wobei A und B die erlernten Parameter beinhalten und deutlich weniger Parameter besitzen als W.
Dadurch bildet man einen relativ kleinen Adapter der mit dem ursprünglichen Modell fusioniert wird.

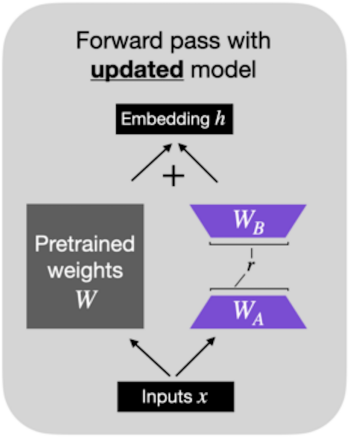

Abbildung 1.12: Zeigt die Kombination des ursprünglichen Modells W mit den niederrangigeren Matrizen W_A und W_B, welche die neu erlernten Gewichte enthalten [25].

LoRA eignet sich nicht nur für Szenarien, in denen große vortrainierte Modelle auf spezifische Aufgaben oder Daten angepasst werden müssen, wie z.B. bei der Verarbeitung von speziellen Textarten, Domänen oder Sprachen. Die Technologie ist auch geeignet um Modelle, die durch Kompression verschlankt wurden zu „re-trainieren“, damit ihre Fähigkeiten auf dem Niveau der Ursprungsmodells bleiben.

1.3.6 Beurteilung der Leistungsfähigkeit eines LLMs

Für die Beurteilung der Leistungsfähigkeit eines LLMs gibt es aufgrund ihrer Komplexität, eine große Anzahl von Kriterien und Metriken. In der Arbeit von (Guo et. al) [26] wird eine Einteilung der relevanten Metriken in drei hauptsächliche Bereiche vorgenommen. Diese sind:

- Die Bewertung der Kenntnisse und Fähigkeiten (knowledge and capability evaluation) eines Modells. Hierzu zählt u.a. die Fähigkeit zu Schlussfolgerungen, Wissens Vervollständigung und Antwortkompetenz.
- Die Bewertung der Anpassungsfähigkeit (alignment evaluation) konzentriert sich auf die Prüfung der Leistung von LLMs in kritischen Dimensionen. Diese umfassen ethische Erwägungen, moralische Implikationen, Erkennung von Verzerrungen, Vermeidung von Toxizität und die Bewertung der Wahrhaftigkeit von Aussagen.
- Bewertung der Sicherheit (safety evaluation) eines Modellls umfasst vor allem die Robustheit. Damit ist sowohl die Konstanz der Leistungen auf hohem Niveau, wie auch die Bewältigung von Störungen durch z.Bsp. böswillige Angriffe oder vorsätzliche Täuschung.

Die vorliegende Arbeit konzentriert sich praktisch ausschließlich auf den erstgenannten Block dieser Einteilung, die Ermittlung von Kenntnissen und Fähigkeiten des LLMs. Ziel ist es die verschieden komprimierten Modelle miteinander auf diese Kriterien hin zu vergleichen und aus diesen Informationen Rückschlüsse auf die Eignung des Vorgehens zu ziehen. Daher wurden für diese Arbeit Benchmarks ausgewählt, die diesen Komplex exemplarisch abdecken können. Außerdem wurde darauf geachtet Benchmarks zu verwenden, die in aktuellen Veröffentlichungen häufigere Nutzung erfahren, damit ein gewisse Vergleichbarkeit entsteht und die Einordnung der Ergebnisse erleichtert wird. Alle ausgewählten wurden daher auch bis Mai 2024 im HuggingFace

Leaderboard verwendet [27], auf dem viele Vergleiche von Modellen in den letzten Monaten beruhen.

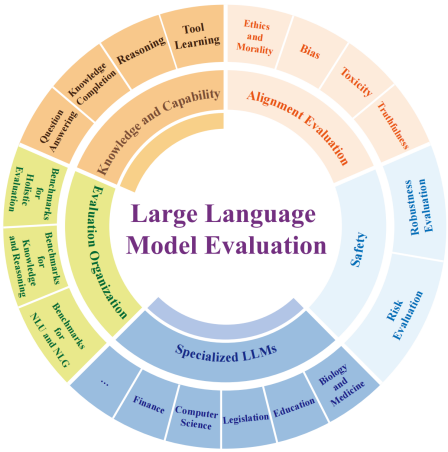

Abbildung 1.13: Darstellung der Kriterien und Metriken nach (Guo et al.)[26]. Neben den im Text behandelten drei Hauptbereichen sind noch Metriken für spezielle LLMs (dunkleres blau) angegeben.

Die, für die Experimente verwendeten Benchmarks werden im Teil 2.3.5 vorgestellt.

2 Material und Methoden

2.1 Verwendete Modelle

Für den Experimentalteil werden die Llama Sprachmodelle der amerikanischen Firma Meta und das Sprachmodell Mistral der gleichnamigen französischen Firma verwendet. Die Modelle werden in den weiteren Abschnitten charakterisiert. Folgende Argumente führten zur Auswahl dieser Modelle:

- Im Bereich der frei verfügbaren Modelle sehr gute Leistungen im Anwender Dialog.
- Liberale Lizenzpolitik, so dass sie weitgehend ohne Einschränkungen in der Forschung genutzt werden können.
- Llama 3 und Mistral wurden kurz vor Start dieser Arbeit veröffentlicht.
- Der Aufbau der Modelle und das Trainingsvorgehen sind dokumentiert.

Bei allen Modellen wurde immer die kleinstmögliche Variante gewählt. Dies passierte primär, um den Ressourcenbedarf gering und die Laufzeiten der Experimente mit den einzelnen Modelle nicht zu lang zu gestalten. Ein weitere Grund war das Bestreben, möglichst kompakte Modelle zu generieren. Daher war es sinnvoll, schon ein möglichst nicht zu großes Grundmodell als Ausgangspunkt zu benutzen.

2.1.1 Meta Llama 2

Am 18.7.2023 wurde Llama 2 von Meta veröffentlicht [28]. Das Modell liegt in drei verschiedenen Parametergrößen vor: 7 Milliarden, 13 Milliarden und 70 Milliarden Parameter. Alle Modelle liegen darüber hinaus zusätzlich noch in einer finegetunten, für den Anwender Dialog optimierten, „Chat“Variante vor, so dass sich in Summe sechs verschiedene Versionen ergeben. Für die Experimente in dieser Arbeit wurde das kleinste Modell Llama 7B in der Chat Variante ausgewählt.
Llama 2 folgt einem eher evolutionären Einsatz bei seiner Entstehung. Die Basis Struktur wurde von Llama (1) [29] übernommen, welches wiederum den in 1.1.1 beschriebenen Transformer Ansatz verwendet. Wesentliche Neuerungen gegenüber dem Vorgänger sind ein neuer größerer Trainingsdatensatz, eine Verdopplung der Prompt-Größe, um mehr Kontext zu ermöglichen und die Nutzung von „grouped-query attention“für eine schnelle und qualitativ stabile Inferenz [30]. Wie oben erwähnt war Llama 2 das erste Modell der Firma Meta, dass zusätzlich in einer finegetunten Variante für einen verbesserten Anwenderdialog zur Verfügung gestellt wurde. Dazu passieren im wesentlichen zwei Schritte. Im ersten Schritt dem „supervised fine-tuning“werden dem Modell Frage/Antwort Paare vorgelegt, damit es Muster für die Ausgaben besser erfasst.
In einem weiteren Schritt, der als „reinforcement learning with human feedback (RLHF)“bezeichnet wird, nehmen Menschen eine qualitative Bewertung der Antworten des Modells

vor. Mittels „rejection sampling “und „Proximal Policy Optimization “wird das Modell iterativ mit diesen Antworten verbessert und so die Kompetenz für den Anwender Dialog erhöht 2.1.

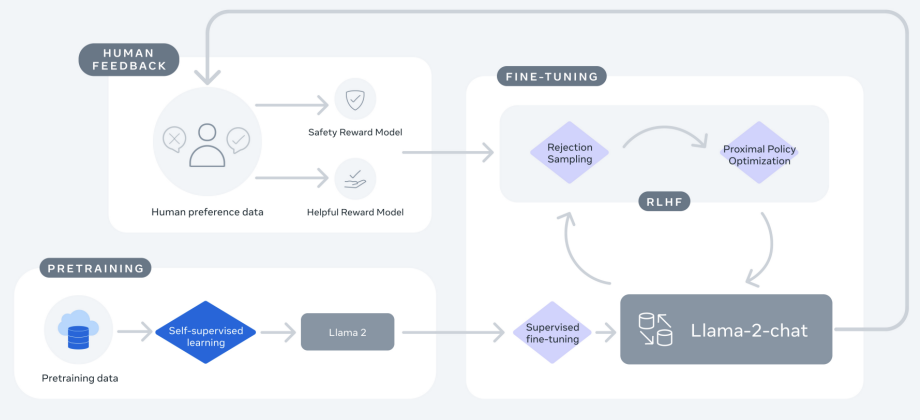

Abbildung 2.1: Erstellung des Chat Modells aus dem Grundmodell [28]. Erklärung siehe Text.

2.1.2 Meta Llama 3

Das Modell Llama 3 wurde am 18.4.2024 von der Firma Meta zwei Versionen veröffentlicht [31]. Im Gegensatz zum Vorgängermodell wurde die mittlere Version nicht mehr angeboten, so dass nur eine Variante mit 8 Milliarden und eine mit 70 Milliarden Parameter zur Verfügung steht. Wie bei Llama 2 existiert auch hier jeweils eine, für den Dialog mit dem Anwender optimierte, Chat Variante. Ähnlich wie Llama 2 verfolgt auch Llama 3 einen evolutionären Ansatz, indem es das alte Modell durch neuere Methoden und Erkenntnisse verbessert. Leider ist die Dokumentation zu Llama 3 deutlich rudimentärer, als die umfangreichen Veröffentlichungen zu den beiden Vorgänger Modellen. Bekannt ist, dass ein neuer 128 K Tokenizer etabliert und noch einmal deutlich mehr Trainingsdaten verwendet wurden.

2.1.3 Mistral

Mistral ist ein Transformer Modell mit 7 Milliarden Parametern, welches von der französischen Firma Mistral entwickelt und im September 2023 veröffentlicht wurde [11]. Ähnlich wie bei Metas Llama gibt es auch hier Schwester Modelle mit deutlich mehr Parametern die unter den Namen Mixtral (8x7B) und (8x22B) veröffentlicht wurden und 56 bzw. 176 Milliarden Parameter beinhalten. Ebenfalls gibt es hier auf Anwender Kommunikation spezialisierte Varianten, die als „Instruct“bezeichnet werden.
Obwohl ebenfalls auf einer Transformer Struktur basierend, unterscheiden sich die Mistral Modelle durch den Einsatz anderer Technologien von den zuvor genannten Meta Modellen. So benutzt Mistral u.a. einen „Sliding window attention mechanism“, der die Rechenkomplexität reduziert und damit der Performance zu gute kommt. Obwohl das Modell dabei nur ein begrenztes Fenster auf die Eingabesequenz legt und in diesem die Aufmerksamkeit berechnet, können, durch ein bewegen des Fensters schrittweise entlang der Sequenz, auch längere Abhängigkeiten modelliert werden.
Eine weiteres besonderes Merkmal ist die Nutzung der „FlashAttention“Technologie für die Berechnungen. Diese kombiniert sehr geschickt Matrixmultiplikationen und Softmax-Berechnungen

in einem speicherfreundlicheren Algorithmus, was ebenfalls zu einer Steigerung der Gesamtperformance führt.

2.2 Python Bibliotheken

In vielen der Verfahren werden einige zentrale Python Bibliotheken bzw. Frameworks verwendet, von denen die wichtigsten hier kurz vorgestellt werden.

2.2.1 Bibliothek: Transformers

Die aus der Huggingface Community hervorgegangene Transformers Bibliothek [32] ist eine zentrales Werkzeug zu Arbeit mit den gleichnamigen Modellen. Einige der Kernelemente sind:

- Einfacher Zugriff auf vortrainierte Modelle zugegriffen werden.
- Feinabstimmung (engl: FineTuning) der Modelle.
- Erstellen von Trainings Pipelines.

Der Funktionsumfang der Bibliothek wächst ständig, getrieben durch die Hugging Face Community. Dadurch bietet sie einen umfassenden, aktuellen und benutzerfreundlichen Zugang zu modernen LLMs, und stellt für Forschung, Entwicklung und Produktion ein wichtiges Werkzeug dar.

2.2.2 Framework torch (pytorch)

Pytorch ist eine Weiterentwicklung der seit 2002 bestehenden torch bibliothek. Der Name repräsentiert mittlerweile ein Ökosystem von Bibliotheken, mit einer Vielzahl von Funktionen, welche für das Machine Learning und insbesondere die Erstellung und das Training neuronaler Netzwerke genutzt werden. Ursprünglich entwickelt wurde die Bibliothek von Facebooks (jetzt Meta) AI Research Group k (FAIR), die sie im Jahr 2017 als Open Source der Entwicklergemeinde übergab. Seit 2022 steht die weitere Entwicklung unter der Obhut der PyTorch Foundation. [33] Mit Hilfe von Pytorch können insbesondere Tensor Operationen durch die Nutzung von GPUs stark beschleunigt und somit neuronale Netze effizient trainiert werden.

2.2.3 Bibliothek PEFT

Die PEFT-Bibliothek (Parameter-Efficient Fine-Tuning) von Hugging Face ist eine spezialisierte Bibliothek, die sich auf das effiziente Fine-Tuning großer vortrainierter Modelle konzentriert. Ein wesentlicher Aspekt dabei ist der effektive Umgang mit Speicherbedarf und Rechenleistung.
Wesentliche Funktionen sind:

- LoRA Fine Tuning mittels der Erstellung geeigneter Adapter.
- Optimierung der Eingabe (Prompt) bzw. der Präfixe.
- Kompression der Modelle durch Quantisierung.

Die PEFT Bibliothek integriert sich nahtlos mit der zuvor erwähnten Transformers Bibliothek und erlaubt es so den Entwicklern eine effiziente Arbeit mit den Modellen aus der zugehörigen Huggingface Bibliothek[34]

2.3 Kompressionsverfahren

2.3.1 Pruning

Für das Pruning gibt es wie schon in 1.3.1 beschrieben mehrere Varianten und Verfahren. Für diese Arbeit wurde eine „depth pruning“implementation mit der Bezeichnung (shortened-llm) verwendet [35]. Dabei werden „unwichtige“transfomer blocks aus dem Modelll entfernt. Die Auswahl der weniger wichtige Teile erfolgt durch eine Metric in der Taylor+ und Perplexity Kriterien als Diskriminatoren genutzt werden. Führt die Entfernung eines Blocks (bzw. von Blöcken) nur zu einer relativ geringen Verschlechterung dieser Werte, werden diese Blöcke als „unwichtig “angesehen. Die auf dieser Weise identifizierten Teile werden in einem zweiten Schritt aus dem Modell mittels „splitting “entfernt. Ein wichtige Ergänzung ist das Retraining der so geprunten Modelle mittels der LoRA Technologie, dass die notwendige Neubalancierung des neuronalen Netzes effektiv ermöglicht.
In der genannten Veröffentlichung wurde gezeigt, dass dieses Verfahren anderen Pruning Methoden mindestens ebenbürtig ist, bzw. diese zum Teil auch übertrifft. Bei moderatem Pruning (20 - 45%) können Benchmark Werte erreicht werden, die im Bereich des Ausgangsmodells liegen, aber weniger Speicher benötigen und die Inferenz beschleunigen.

2.3.2 Quantisierung

Wie bereits in 1.3.2 beschrieben, wird in dieser Arbeit das AWQ Verfahren zur Quantisierung genutzt [21]. Zur Erstellung der quantisierten Modelle wurde das zugehörige Verfahren unter [36] genutzt. Die Umsetzung wird im Kapitel Implementation beschrieben.

2.3.3 Knowledge Distillation

Als Verfahren für die Knowledge Distillation wurde das „Dual Space Knowledge Distillation (kurz: DSKD)“gewählt [37]. Die Wahl fiel auf dieses Verfahren, da es mit diesem relativ leicht möglich ist Student Modelle zu trainieren, die eine abweichende Struktur vom Teacher Modell besitzen (daher Dual Space). Da beide Modelle ihre eigenen unterschiedlichen Vorhersage Mechanismen (engl. „Prediction Heads“) nutzen, wird im DSKD Verfahren eine „cross-model attention (CMA)“implementiert, welche die verschiedenen Token Sequenzen automatisch miteinander abgleicht.
Das Verfahren entspricht ansonsten dem unter 1.3.4 beschriebenen.

2.3.4 Low Rank Adaption

Für die LoRA Experimente wurden Programmblöcke aus [38] verwendet und diese für die Bedürfnisse dieser Arbeit angepasst bzw. entsprechend erweitert. Der Prozess wurde aufgrund seiner guten Adaptierbarkeit und dem verwendeten Datensatz „Ms_Marco“ausgewählt. „Ms_Marco“ist ein von Microsoft entwickelter Datensatz [39], welcher aus Suchanfragen der Suchmaschine

„Bing“zusammengestellt wurde. Die Fragen sind mit einer unterschiedlichen Anzahl von Antworten versehen und überspannen ein breites Themenspektrum.
Technisch basiert der Prozess im wesentlichen auf den Möglichkeiten der Transformers und PEFT Bibliothek.

2.3.5 Benchmarks

Verwendet wurde das Framework lmėval [40].

- **Winogrande** [41] Eine Auswahl von 273 Problemen die vor allem auf eine Lösung durch „Gesunden Menschenverstand“abzielen, dienen als Kriterium für die die Lösungsfähigkeiten der KIs.
- **arc_challeng**e [42] Die Autoren der ARC challenges entwickelten eine Definition von Intelligenz, die u.a. Effizienz beim Erwerb von Fähigkeiten und die Fähigkeit zur Generalisierung als Kernelemente enthält. Die zugehörige Benchmark zielt darauf ab diese Elemente zu quantifizieren.
- **truthfulqa_mc2**, [43] Diese Benchmark besteht aus 817 Fragen die zu 38 Kategorien gehören. Ziel ist es zu quantifizieren wie oft das Sprachmodell wahrheitsgemäße Antworten auf diese Fragen generiert.
- **hellaswag** [44] Diese Benchmark zielt auf die Fähigkeit der LLMs zur sinnvollen Vervollständigung von Satzfragmenten.
- **perplexity mit wikitext2** [45] Die Perplexity (Deutsch: Verwirrtheit, Unsicherheit) eines LLMs gibt an wie unsicher es bei der Voraussage des nächsten Wortes einer Antwortsequenz ist. Um so niedriger die „Verwirrtheit“des Modells, um so besser versteht das Modell die ihm vorgelegten Daten.

3 Implementation

In diesem Kapitel werden die Implementationen der einzelnen Verfahren dargestellt. Der auszuführende Programmcode ist hellgrau unterlegt und wird an manchen Stellen zum besseren Verständnis zusätzlich kommentiert. Die zugehörigen Jupyter Notebooks zur Ausführung der Verfahren findet man unter [46].

3.1 Pruning: Shortened-LLM

Implementationen zum in 2.3.1 dargestellten Verfahren. Die Implementation basiert auf den in [36] genannten Github Repository.

3.1.1 Download der Dateien und erstellen des conda envrionments (Nur einmalig notwendig)

```
conda create -n shortened-llm python=3.9
conda activate shortened-llm
git clone https://github.com/Nota-NetsPresso/shortened-llm.git
cd shortened-llm
pip install -r requirement.txt
```

Installation zusätzlicher Pakete, die in dem Setup des GitHub Repositories nicht enthalten waren, aber benötigt werden

```
pip install typing_extensions packaging pygments psutil
```

3.1.2 Workflow zur Modell Compression

Das Verfahren wird in einer Bash Shell gestartet. Dazu wird die conda Umgebung „shortened-llm“geladen.

```
conda activate shortened-llm
```

```
cd /home/thsch026/masterarbeit/experiment/shortened-llm
```

Festlegen einer nutzbaren GPU, die manuell festgelegt werden muss

```
export CUDA_VISIBLE_DEVICES="2"
echo $CUDA_VISIBLE_DEVICES
```

3.1.3 Durchführung des Workflows

- In dem Verzeichnis „script“der Installation findet man vorgefertigte Scripte für die Nutzung des Workflows.
  - In diesen Scripten müssen ggf. Anpassungen vorgenommen werden (z.Bsp. wie viele Schichten des Modells entfernt werden sollen)
- In den Scripten müssen einige Variablen entsprechend der verwendeten Umgebung angepasst werden.
- Das jeweilige Script für das spezifisch zu komprimierende Modell wird ausgeführt.

3.1.4 Llama2-7B-Chat

```
script/prune_llama2-7b_crit-taylor.sh
```

3.1.5 Llama3-8B-Instruct

```
script/prune_llama3-8b-instruct_crit-taylor.sh
```

3.1.6 Mistral-7B-Instruct-v0.2

```
script/prune_Mistral_crit-taylor.sh
```

Ergebnisse des Workflows

- Im Ordner „output_prune“findet man die geprunten modelle
- Im zweiten Schritt wird das Model noch mittels der LoRA Verfahrens trainiert. Die fertigen Modelle (pruned + tuned) befinden sich im Ordner „output_tune“

3.2 Quantization: AWQ

Implementation für die Quantisierung der Modelle mit dem in 1.3.2 beschriebenen Verfahren.

3.3 Quantisierung mit der autoawq library

- Es wird die Bibliothek autoawq 0.1.0. genutzt. Diese setzt eigentlich die transformers >= 4.36 voraus, welche aber andere Probleme verursacht.
- Lösung: Installation von transformers 4.35 durch ignorieren der Abhängigkeit. Funktioniert. Fehler waren nicht feststellbar.
- huggingface_hub Bibliothek muss mindestens die Version 0.21 haben

3.3.1 Erstellen der Umgebung - Nur einmalig notwendig

```
!pip install autoawq nvidia-ml-py3 numpy==1.26.4 sentencepiece torch==2.0.1
 protobuf
```

Login für Hugging Face - Nur notwendig wenn HF Modelle genutzt werden

```
notebook_login()
```

3.3.2 Auswählen der GPU und laden der Bibliotheken

```
import os

os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "max_split_size_mb:256"
os.environ["CUDA_VISIBLE_DEVICES"] = "1"
!echo $CUDA_VISIBLE_DEVICES
```

```
from awq import AutoAWQForCausalLM
from transformers import AutoTokenizer
from huggingface_hub import notebook_login
import safetensors
```

Konfiguration der Parameter für die AWQ Quantisierung

```
quant_config = {"zero_point": True, "q_group_size": 128, "w_bit":4}
```

3.3.3 Pfad zum Modell und Speicherort für die quantisierten Basis Modelle

Llama3-8B-Instruct

```
model_path = "meta-llama/Meta-Llama-3-8B-Instruct"
quant_path = "/home/thsch026/masterarbeit/models/generated/llm-awq/
Meta-Llama-3-8B-Instruct-AWQ"
```

Mistral-7B Instruct v0.2

```
model_path = "mistralai/Mistral-7B-Instruct-v0.2"
quant_path = "/home/thsch026/masterarbeit/models/generated/llm-awq/
Mistral-7B-Instruct-v0.2-AWQ-4bit"
```

Llama2-7B-chat-HF

```
model_path = "meta-llama/Llama-2-7b-chat-hf"
quant_path = "/home/thsch026/masterarbeit/models/generated/llm-awq/
Llama-2-7b-chat-HF-AWQ-4bit"
```

3.3.4 Pfad zum Modell und Speicherort für die vorab veränderten Modelle

Mistral-7B-Instruct-v02-prune-lora

```
model_path = "/home/thsch026/masterarbeit/models/generated/
prune_plus_lora/shortened-llm/Mistral-7B-Instruct-v02_prune_lora"
quant_path = "/home/thsch026/masterarbeit/models/generated/
prune_plus_lora_plus_awq/Mistral-7B-Instruct-v02_prune_lora_awq_256groupsize"
```

Llama 3 8B Instruct prune-lora

```
model_path = "/home/thsch026/masterarbeit/models/generated/
prune_plus_lora/shortened-llm/Meta-Llama-3-8B_prune_lora"
quant_path = "/home/thsch026/masterarbeit/models/generated/
prune_plus_lora_plus_awq/Meta-Llama-3-8B_prune_lora_awq"
```

Llama-2-7b-chat-hf_prune_lora_awq

```
model_path = "/home/thsch026/masterarbeit/models/generated/prune_plus_lora/shortened-llm/Lla
quant_path = "/home/thsch026/masterarbeit/models/generated/prune_plus_lora_plus_awq/Llama-2-
```

Laden des Modells und des zuhgeörigen tokenizers

```
#load the model and tokenizer
model = AutoAWQForCausalLM.from_pretrained(model_path, safetensors=True)
tokenizer = AutoTokenizer.from_pretrained(model_path, use_fast=True)
```

Durchführung der Quantisierung

- Benötigt ca. 10 - 20 Minuten

```
model.quantize(tokenizer, quant_config=quant_config)
```

Abspeichern des Modells

```
import os

model.save_quantized(quant_path, safetensors=True)
tokenizer.save_pretrained(quant_path)
```

Freigeben des GPU Speichers

```
import torch
del model
del tokenizer
torch.cuda.empty_cache()
print(torch.cuda.memory_reserved(0))
print(torch.cuda.memory_allocated(0))
```

4 Ergebnisse

4.1 Ergebnisse der Experimente

4.1.1 Optimierungstechnologien für LLMs

In dem folgenden Kapitel werden die Ergebnisse zu den verschiedenen Verfahren der Modell Optimierung vorgestellt. Eine Übersicht bietet die folgende Tabelle.

Verfahren	Llama 3 8B Instruct	Mistral 7B Instruct v0.2	Llama 2 7B chat
Low rank adaption (Lora)	X	X	X
Quantized low rank adaption (Qlora)	X	X	X
Activation-aware Weight Quantization (AWQ)	X	X	X
Depth pruning (Layer Extraction)	X	X	X
Knowledge distillation	X	X	X
Kombinierte Verfahren			
Pruning + Lora	X	X	X
Pruning + Lora + AWQ	X	X	X

Tabelle 4.1: Übersicht über die verwendeten Verfahren und Modelle

Für eine Verringerung des Speicherbedarfs eignen sich bevorzugt die Verfahren Quantization und Pruning, während Lora und Knowledge Distillation für das erneute Tuning solcher Modelle eingesetzt werden.

Quantization

Benchmark Ergebnisse der AWQ 4bit Quantization im Vergleich zum jeweiligen Referenz Modell. vspace0.8 cm **Speicherbedarf der AWQ Modelle im Vergleich zur**

Benchmarks:	winogrande	truthfulqa_mc2	hellaswag	arc_challenge	Durchschnitt
Mistral 7 Referenz	0,7395	0,6682	0,8365	0,5606	0,7012
Standard Fehler	*0,0123*	*0,0152*	*0,0037*	*0,0145*	
Mistral 7 AWQ	0,7403	0,6752	0,8310	0,5700	0,7041
Standard Fehler	*0,0123*	*0,0151*	*0,0037*	*0,0145*	
Llama 3 Referenz	0,7206	0,5165	0,7582	0,5674	0,6407
Standard Fehler	*0,0126*	*0,0152*	*0,0043*	*0,0145*	
Llama 3 AWQ	0,7356	0,5099	0,7532	0,5572	0,6390
Standard Fehler	*0,0124*	*0,0152*	*0,0043*	*0,0145*	
Llama 2 Referenz	0,6646	0,4532	0,7547	0,4420	0,5786
Standard Fehler	*0,0133*	*0,0156*	*0,0043*	*0,0145*	
Llama 2 AWQ	0,6456	0,4509	0,7481	0,4462	0,5727
	0,0134	*0,0156*	*0,0043*	*0,0145*	

Tabelle 4.2: Die Tabelle zeigt die quantisierten Modellen zusammen mit den korrespondierenden Referenzen. Für die Quantisierung wurde das Verfahren AutoAWQ verwendet.

jeweiligen Referenz

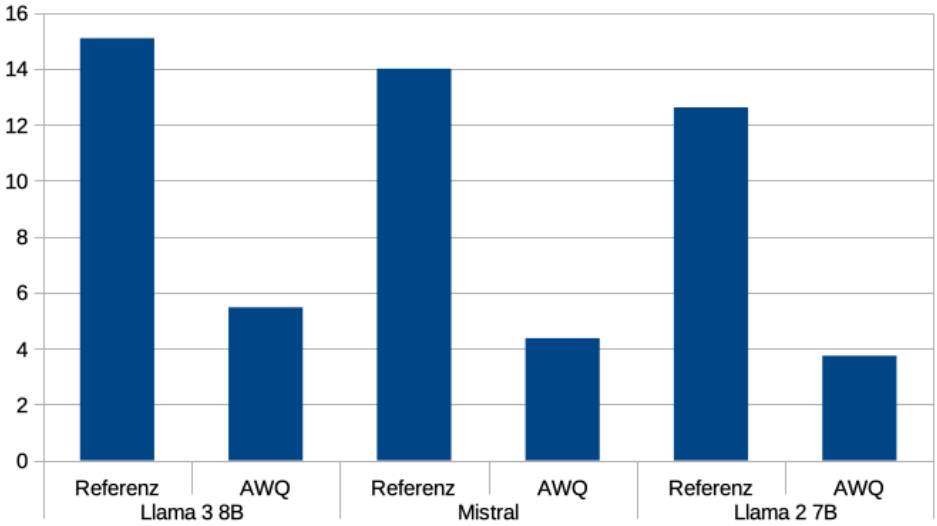

Abbildung 4.1: Auslastung des GPU Speichers durch die jeweiligen Modelle auf einer Nvidia A100 80GB GPU.

Pruning

Die Tabelle zeigt die Benchmark Ergebnisse für das Pruning der Referenz Modelle mittels PrunMe in jeweils zwei verschiedenen Ausprägungen

Benchmarks:	Layers Pruned	winogrande	truthfulqa_mc2	hellaswag	arc_challenge	Durchschnitt
Llama 3 8B	**0**	0,7206	0,5165	0,7582	0,5674	0,6407
	Standard Fehler	*0,0126*	*0,0152*	*0,0043*	*0,0145*	
	8	0,6369	0,5211	0,5897	0,4283	0,5440
	Standard Fehler	*0,0135*	*0,0159*	*0,0049*	*0,0145*	
	12	0,5706	0,5201	0,4204	0,3208	0,4580
	Standard Fehler	*0,0139*	*0,0164*	*0,0049*	*0,0136*	
Mistral 7 B	**0**	0,7395	0,6682	0,8365	0,5606	0,7012
	Standard Fehler	*0,0123*	*0,0152*	*0,0037*	*0,0145*	
	8	0,6811	0,6313	0,6646	0,4044	0,5954
	Standard Fehler	*0,0131*	*0,0158*	*0,0047*	*0,0143*	
	12	0,5943	0,5543	0,3239	0,3677	0,4601
	Standard Fehler	*0,0138*	*0,0167*	*0,0047*	*0,0141*	

Tabelle 4.3: Die Tabelle zeigt Modelle bei denen eine unterschiedliche Anzahl von Schichten mittels Pruning entfernt wurde.

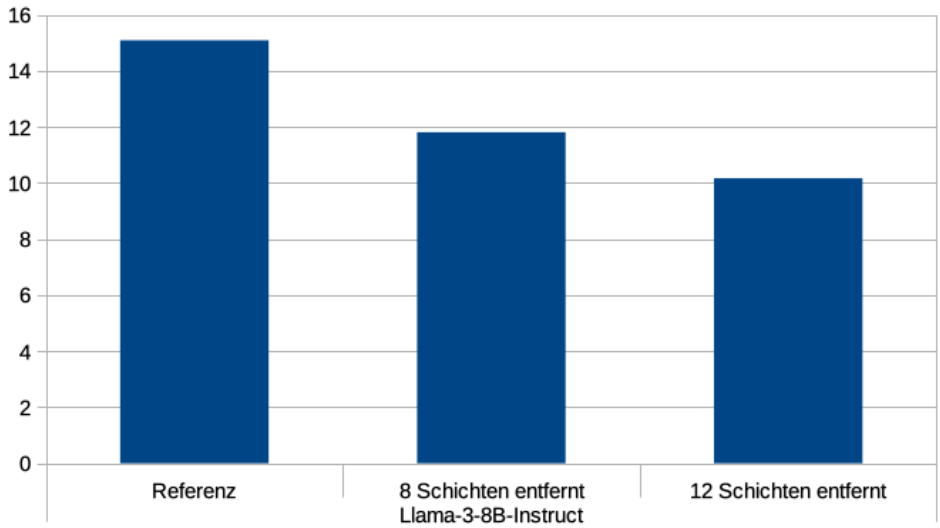

Abbildung 4.2: Auslastung des GPU Speichers der jeweiligen Modelle auf einer Nvidia A100 80GB GPU.

4.1.2 Kombination von Verfahren zur Kompression von LLM Modellen

Tuning eines geprunten Modells mittels Lora

Die Tabelle zeigt die Benchmark Ergebnisse für die Anwendung des Lora Verfahrens auf geprunte Modelle.

Modell	Verfahren	winogrande	truthfulqa_mc2	hellaswag	arc_challenge	Durchschnitt
Llama 3 8B Instruct	pruned	0,7111	0,5000	0,6255	0,4625	0,5748
		0,0127	*0,0157*	*0,0048*	*0,0146*	
	pruned + lora	0,7316	0,5189	0,7269	0,4974	0,6187
		0,0125	*0,0151*	*0,0044*	*0,0146*	
Mistral 7B Instruct v0.2	pruned	0,7072	0,6370	0,7298	0,4599	0,6335
		0,0128	*0,0158*	*0,0044*	*0,0146*	
	pruned + lora	0,7174	0,5774	0,7457	0,4855	0,6315
		0,0127	*0,0154*	*0,0043*	*0,0146*	
Llama 2 7B chat	pruned	0,6409	0,4955	0,6384	0,3968	0,5429
		0,0135	*0,0159*	*0,0048*	*0,0143*	
	pruned + lora	0,6732	0,4886	0,7082	0,4206	0,5727
		0,0132	*0,0154*	*0,0045*	*0,0144*	

Tabelle 4.4: Die Tabelle zeigt den Effekt der Anwendung des Lora Verfahrens auf die geprunten Modelle.

Tuning eines geprunten Modells mittels Knowledge Distillation

Einfluss von Knowledge Distillation auf die Ergebnisse.

Benchmarks:	winogrande	truthfulqa_mc2	hellaswag	arc_challenge	Durchschnitt
LLama 3 8B pruned lora	0,7316	0,5189	0,7269	0,4974	0,6187
Fehler	*0,0125*	*0,0151*	*0,0044*	*0,0146*	
LLama 3 8B pruned lora dist	0,6535	0,4425	0,6645	0,4386	0,5498
Fehler	*0,0134*	*0,0166*	*0,0047*	*0,0145*	
Mistral pruned lora	0,7174	0,5774	0,7457	0,4855	0,6315
Fehler	*0,0127*	*0,0154*	*0,0043*	*0,0146*	
Mistral pruned lora dist	0,6330	0,4654	0,6694	0,4437	0,5529
Fehler	*0,0135*	*0,0162*	*0,0047*	*0,0145*	

Tabelle 4.5: Optimierungsversuch mit Knowledge Distillation auf Modelle die mit pruning verkleinert und mit lora wieder traniert wurden

Vergleichende Darstellung der erzeugten Varianten am Beispiel von Llama 3 8B Instruct

Die nachfolgende Tabelle zeigt die Benchmark Ergebnisse der zuvor beschrieben Llama 3 Modelle.

Benchmarks:	winogrande	truthfulqa_mc2	hellaswag	arc_challenge	Durchschnitt
Llama 3 8B instruct referenz	0,7210	0,5160	0,7580	0,5670	0,6400
Standard Fehler	*0,0130*	*0,0150*	*0,0040*	*0,0150*	
Llama 3 8B instruct pruned	0,7110	0,5000	0,6250	0,4620	0,5700
Standard Fehler	*0,0130*	*0,0160*	*0,0050*	*0,0150*	
Llama 3 8B instruct awq	0,7370	0,5099	0,7529	0,5580	0,6395
Standard Fehler	*0,0124*	*0,0152*	*0,0043*	*0,0145*	
Llama 3 8B instruct pruned + lora	0,7320	0,5190	0,7270	0,4970	0,6200
Standard Fehler	*0,0130*	*0,0150*	*0,0040*	*0,0150*	
Llama 3 8B instruct pruned + lora + awq	0,7230	0,5270	0,7320	0,4870	0,6200
Standard fehler	*0,0130*	*0,0150*	*0,0040*	*0,0150*	

Tabelle 4.6: Die Tabelle zeigt die Benchmark Ergebnisse der verschiedenen Llama 3 8B Instruct Modelle im Vergleich.

Perplexity Ergebnisse

Die nachfolgende Tabelle zeigt die Perplexity Ergebnisse für verschiedene vom Referenz Modell abgeleitet Varianten.

Modell: Llama 3 8B instruct	bits_per_byte	byte_perplexity	word_perplexity
Referenz	0,62	1,54	9,94
AWQ	0,63	1,55	10,30
pruned	1,08	2,11	53,86
prune + lora	0,71	1,64	13,96
prune + lora + awq	0,70	1,62	13,24

Tabelle 4.7: Perplexity Messungen für die verschiedenen Varianten des LLama 3 8B Instruct Modells

Inference Ergebnisse

Die Tabelle zeigt die Inference Ergebnisse ermittelt auf einer Nvidia A100 80GB GPU (CUDA 11.7)

Modell	Referenz	awq*	pruned	lora	qlora	Pruned + lora	Pruned + lora +awq*
Llama-3-8B-Instruct	27,35	2,40	34,42	15,99	29,39	36,08	3,03
Mistral-7B-Instruct-v0.2	35,03	2,47	41,35	33,86	31,60	38,50	3,01
Llama-2-7b-chat	31.42	1.40	38.22	17.01	28.41	40.03	1.84

Tabelle 4.8: Inference Performance von nativen zu bearbeiteten Modellen. Die mit einem * gekennzeichneten Modell wurde in 4 Bit Genauigkeit geladen, da die verwendete T4 GPU nicht genug Speicherkapazität für die native Größe besaß.

Die Tabelle zeigt die Inference Ergebnisse ermittelt auf einer Nvidia T4 15GB GPU (CUDA 12.x)

Modell Name	Verfahren	Token pro Sekunde
Llama 3 8B	Referenz	10,22
	prune + lora + awq	33,34
Mistral-7B-Instruct-v02	Referenz	11,10
	prune + lora + awq	38,99

Tabelle 4.9: Inference Performance von nativen zu AWQ quantisierten Modellen. Die Messungen wurden in einer Google Colab Umgebung mit einer Nvidia T4 15 GB durchgeführt. Hier konnten Cuda 12.x Treiber genutzt werden, die eine Verwendung des AutoAWQ python Moduls ermöglichten.

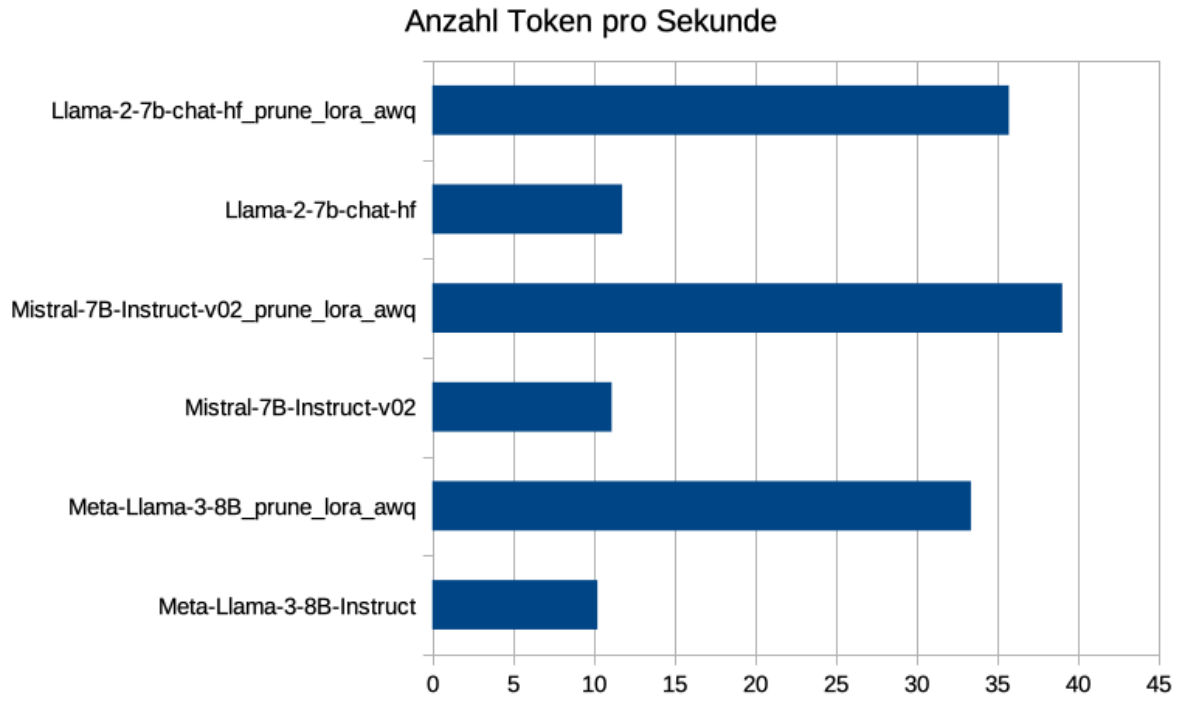

Abbildung 4.3: Token pro Sekunde ermittelt in einer Colab Umgebung mit CUDA 12.x und eine Nvidia T4 15GB GPU. Die Referenz Modelle wurden im 4bit mode geladen, da ansonsten der Speicher der GPU nicht ausgereicht hätte.

Systemresourcen

Speicherbedarf

Die Grafik zeigt die Speicherauslastung aller Varianten der drei Referenz Modelle

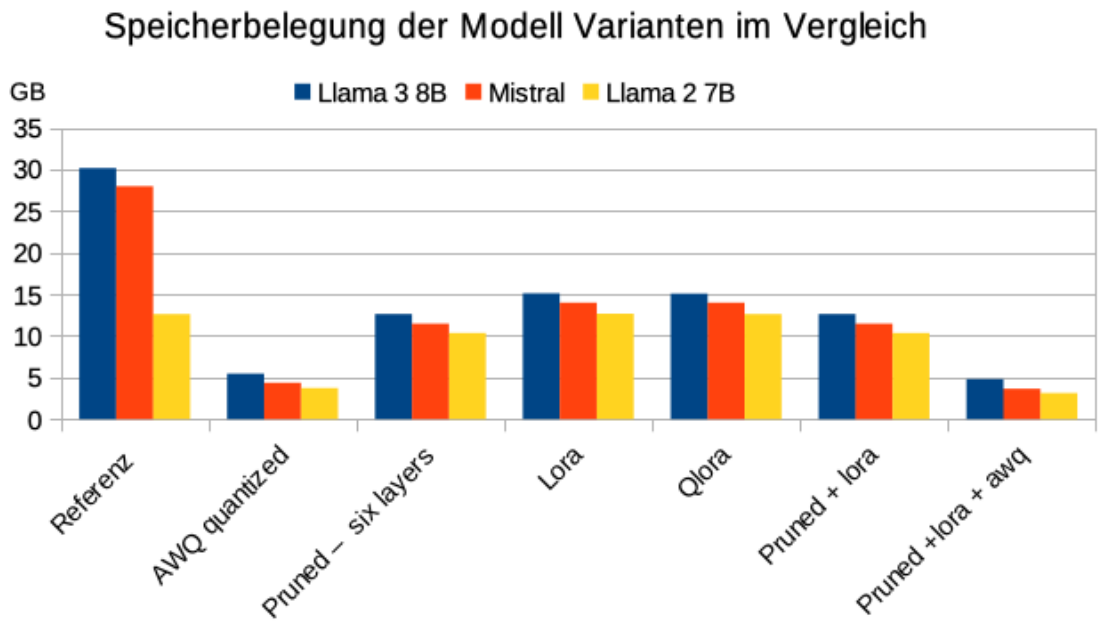

Abbildung 4.4: Speicherbedarf der Modelle auf einer Nvidia A100 80GB GPU. Aus dem jeweiligen Referenz Modell wurden alle weiteren Modelle abgeleitet. Die X-Achse zeigt die verwendeten Verfahren.

Literatur

[1] Wikipedia contributors. *History of natural language processing — Wikipedia, The Free Encyclopedia*. [Online; accessed 5-August-2024]. 2024.

[2] Cameron R. Jones und Benjamin K. Bergen. „People cannot distinguish GPT-4 from a human in a Turing test“. In: (2024). arXiv: 2405.08007 [cs.HC]. URL: https://arxiv.org/abs/2405.08007.

[3] Ashish Vaswani u. a. „Attention Is All You Need“. In: (2023). arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[4] Matin Emrich. *Transfer Learning & BERT Transformer Funktionsweise*. 2020. URL: https://www.alexanderthamm.com/de/blog/transfer-learning-bert-funktionsweise/.

[5] Jacob Devlin u. a. „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding“. In: (2019). arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[6] Xu Han u. a. „Pre-Trained Models: Past, Present and Future“. In: (2021). arXiv: 2106.07139 [cs.AI]. URL: https://arxiv.org/abs/2106.07139.

[7] Cameron R.Wolfe. *Towards Data Science "Language Models: GPT and GPT-2"*. URL: https://towardsdatascience.com/language-models-gpt-and-gpt-2-8bdb9867c50a.

[8] OpenAI. *Introducing ChatGPT*. 2022. URL: https://openai.com/index/chatgpt/.

[9] OpenAI u. a. „GPT-4 Technical Report“. In: (2024). arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774.

[10] Wikipedia contributors. *Gemini (language model) — Wikipedia, The Free Encyclopedia*. [Online; accessed 5-August-2024]. 2024.

[11] Albert Q. Jiang u. a. „Mistral 7B“. In: (2023). arXiv: 2310.06825 [cs.CL]. URL: https://arxiv.org/abs/2310.06825.

[12] Mathew Oldham Kevin Lee Adi Gangidi. *Building Meta's GenAI Infrastructure*. 2024. URL: https://engineering.fb.com/2024/03/12/data-center-engineering/building-metas-genai-infrastructure/.

[13] Ben Cottier u. a. „The rising costs of training frontier AI models“. In: (2024). arXiv: 2405.21015 [cs.CY].

[14] Heise (emw). *Zuckerberg: Llama 4 braucht zehnmal mehr Rechenleistung als Vorgänger*. 2024. URL: https://www.heise.de/news/Zuckerberg-Llama-4-braucht-zehnmal-mehr-Rechenleistung-als-Vorgaenger-9831779.html.

[15] Forschungszentrum Jülich. *JUPITER | Der Start von Exascale in Europa*. 2024. URL: https://www.fz-juelich.de/de/ias/jsc/jupiter.

[16] Hongrong Cheng, Miao Zhang und Javen Qinfeng Shi. „A Survey on Deep Neural Network Pruning-Taxonomy, Comparison, Analysis, and Recommendations“. In: (2023). arXiv: 2308.06767 [cs.LG]. URL: https://arxiv.org/abs/2308.06767.

[17] Song Han, Huizi Mao und William J. Dally. „Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding“. In: (2016). arXiv: 1510.00149 [cs.CV]. URL: https://arxiv.org/abs/1510.00149.

[18] Matthieu Courbariaux, Yoshua Bengio und Jean-Pierre David. „Training deep neural networks with low precision multiplications“. In: (2015). arXiv: 1412.7024 [cs.LG]. URL: https://arxiv.org/abs/1412.7024.

[19] Huggingface. *Huggingface - Quantization Overview*. 2024. URL: https://huggingface.co/docs/transformers/main/quantization/overview#.

[20] Elias Frantar u. a. „GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers“. In: (2023). arXiv: 2210.17323 [cs.LG].

[21] Ji Lin u. a. „AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration“. In: (2024). arXiv: 2306.00978 [cs.CL]. URL: https://arxiv.org/abs/2306.00978.

[22] Amit S. *Everything You Need To Know About Knowledge Distillation, aka Teacher-Student Model*. 2023. URL: https://amit-s.medium.com/everything-you-need-to-know-about-knowledge-distillation-aka-teacher-student-model-d6ee10fe7276.

[23] Edward J. Hu u. a. „LoRA: Low-Rank Adaptation of Large Language Models“. In: (2021). arXiv: 2106.09685 [cs.CL].

[24] Yangyang Xu u. a. „Parallel matrix factorization for low-rank tensor completion“. In: *Inverse Problems Imaging* 9.2 (2015), S. 601–624. ISSN: 1930-8345. DOI: 10.3934/ipi.2015.9.601. URL: http://dx.doi.org/10.3934/ipi.2015.9.601.

[25] Sebastian Raschka. *Parameter-Efficient LLM Finetuning With Low-Rank Adaptation*. 2023. URL: https://lightning.ai/pages/community/tutorial/lora-llm/.

[26] Zishan Guo u. a. „Evaluating Large Language Models: A Comprehensive Survey“. In: *arXiv preprint arXiv:2310.19736* (2023).

[27] Community. *Open LLM Leaderboard*. 2024. URL: https://huggingface.co/open-llm-leaderboard.

[28] Hugo Touvron u. a. „Llama 2: Open Foundation and Fine-Tuned Chat Models“. In: (2023). arXiv: 2307.09288 [cs.CL].

[29] Hugo Touvron u. a. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL]. URL: https://arxiv.org/abs/2302.13971.

[30] Joshua Ainslie u. a. *GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints*. 2023. arXiv: 2305.13245 [cs.CL]. URL: https://arxiv.org/abs/2305.13245.

[31] Meta. *Introducing Meta Llama 3: The most capable openly available LLM to date*. 2024. URL: https://ai.meta.com/blog/meta-llama-3/.

[32] Thomas Wolf u. a. *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. 2020. arXiv: 1910.03771 [cs.CL]. URL: https://arxiv.org/abs/1910.03771.

[33] Wikipedia contributors. *PyTorch — Wikipedia, The Free Encyclopedia*. [Online; accessed 25-August-2024]. 2024.

[34] Hugging Face. *PEFT Documentation*. https://huggingface.co/docs/peft. Accessed: 2024-09-02. 2023.

[35] Bo-Kyeong Kim u. a. „Shortened LLaMA: Depth Pruning for Large Language Models with Comparison of Retraining Methods“. In: (2024). arXiv: 2402.02834 [cs.LG]. URL: https://arxiv.org/abs/2402.02834.

[36] Bo-Kyeong Kim u. a. *Shortened LLM*. 2024. URL: https://github.com/Nota-NetsPresso/shortened-llm.

[37] Songming Zhang u. a. „Dual-Space Knowledge Distillation for Large Language Models“. In: *arXiv preprint arXiv:2406.17328* (2024).

[38] Prakhar Saxena. *Fine Tuning Mistral (or ANY LLM) using LoRA*. 2024. URL: https://medium.com/@prakharsaxena11111/a-general-approach-to-fine-tune-any-llm-using-lora-29d24e47a345.

[39] Payal Bajaj u. a. *MS MARCO: A Human Generated MAchine Reading COmprehension Dataset*. 2018. arXiv: 1611.09268 [cs.CL]. URL: https://arxiv.org/abs/1611.09268.

[40] Leo Gao u. a. *A framework for few-shot language model evaluation*. Version v0.4.0. Dez. 2023. DOI: 10.5281/zenodo.10256836. URL: https://zenodo.org/records/10256836.

[41] Keisuke Sakaguchi u. a. *WinoGrande: An Adversarial Winograd Schema Challenge at Scale*. 2019. arXiv: 1907.10641 [cs.CL]. URL: https://arxiv.org/abs/1907.10641.

[42] François Chollet. *On the Measure of Intelligence*. 2019. arXiv: 1911.01547 [cs.AI]. URL: https://arxiv.org/abs/1911.01547.

[43] Stephanie Lin, Jacob Hilton und Owain Evans. *TruthfulQA: Measuring How Models Mimic Human Falsehoods*. 2022. arXiv: 2109.07958 [cs.CL]. URL: https://arxiv.org/abs/2109.07958.

[44] Rowan Zellers u. a. *HellaSwag: Can a Machine Really Finish Your Sentence?* 2019. arXiv: 1905.07830 [cs.CL]. URL: https://arxiv.org/abs/1905.07830.

[45] Stephen Merity u. a. *Pointer Sentinel Mixture Models*. 2016. arXiv: 1609.07843 [cs.CL].

[46] Thomas Schmitt. *Jupyter Notebooks Masterarbeiti*. https://github.com/fhswf/Slim/tree/main/notebooks. Accessed: 2024-09-03. 2024.

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und dabei keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Werken anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit wurde bisher weder gesamt noch in Teilen einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

3. September 2024

Dr. Thomas Schmitt