GGML, der KI-Computer

von Franz Fiala | Aug 9, 2023 | Grundlagen | 0 Kommentare

GGML, der KI-Computer

Inhaltsverzeichnis

Eine etwas andere Einführung in die Technik aktueller Künstlicher Intelligenz
Für mich ist das Wesentliche an der neuen bzw. aktuellen Künstlichen Intelligenz (KI), dass mit ihr die Computer zum ersten Mal uns Menschen gut verstehen – die Bedeutung verstehen, was wir schreiben, wir sagen und sogar, was wir zeichnen. Und damit können sie uns nun auch gut Antworten – mit Text, Sprache, Bildern, etc., aber auch unmittelbar mit Programmfunktionen. In diesem Artikel beschränke ich mich nur auf die sog. generative KI, d.h. eine KI, die etwas (Texte, Sprache, Bilder, ...) erzeugt.
Die KI ist im Verhalten anders, als wir es bisher von Computern und auch Menschen kennen. Diese, auf englisch AI, bedeutet nicht nur Artificial Intelligence, sondern auch Alien Intelligence, weil sie sehr anders zu uns und unseren bisherigen Erfahrungen mit unseren Computern ist.
Ich versuche hier das funktionale, technische „Warum" dafür grob verständlich zu machen, damit Ihr mit den aktuellen und absehbar kommenden KI-Anwendungen besser umgehen könnt. Auf die philosophischen, kommerziellen, politischen, etc. Aspekte gehe ich nicht ein.
In diesem Artikel versuche ich, Euch die Funktionsweise der KI über eine Art von KI-Computer zu erklären, nennen wir ihn GGML (die Bedeutung des Namens erklär ich später). Ich beschreibe diesen Schritt für Schritt anhand von Elementen, aus denen er besteht. Und vor allem darüber, wie sich diese von uns und denen unserer klassischen Computer unterscheiden – selbst, wenn sie eigentlich auf klassischen Computern arbeitet. Am Ende gibt's dann auch weitere Details für die technisch tiefer Interessierten.
Hergestellt mit Adobe Firefly https://firefly.adobe.com

Die Elemente des KI-Computers

Euer PC, Handy, … hat eine CPU und Arbeitsspeicher, um Applikationen auszuführen. Die CPU arbeitet dabei Schritt für Schritt die Anweisungen der Apps ab. Der Arbeitsspeicher hilft dabei, dass sich die CPU kurzfristig Dinge merken kann – fürs Längerfristige gibt's einen Massenspeicher, z.B. SSD, Harddisk, etc. Der klassische Computer merkt sich nichts, wenn es nicht in den Arbeitsspeicher geschrieben wird. Mit dem Ende der Applikation wird der zugehörige Arbeitsspeicher gelöscht. Es gibt auch eine Ein-/Ausgabe – Displays, Lautsprecher, Tastatur, Mikrofon, Kamera, … Der Computer verarbeitet intern ja nur Binärdaten.
Wie sieht das nun beim KI-Computer aus?

Ein-/Ausgabe

Unser KI-Computer versteht und verarbeitet nur Zahlen – keine Texte, Tonsignale, Bilder wie unser klassischer Computer, dieser muss sie dem KI-Computer übersetzen. Für die Ein-/Ausgabe wird die reale Welt in Zahlen, sog. *Token*s, umgewandelt bzw. von Tokens aus zurückgewandelt. Token sind z.B. Wortteile, Frequenzspektrum Elemente, … z.B. auf https://platform.openai.com/tokenizer könnt Ihr mit einem solchen Tokenizer zur Umwandlung von Texten auf Tokens experimentieren. Die Token-Umwandlung und -Bedeutung ist spezifisch für die KI-Anwendung. Sie ist meist statistisch ermittelt (d.h. keine Silben, …).
Warum ist „Token" wichtig? Die KI kennt nur Tokens, und z.B. die Anzahl der verarbeitbaren Tokens ist ein wesentliches Limit bei der KI-Verarbeitung. Ich versuche die nötigen KI-Fachbegriffe im Text *hervorgehoben* darzustellen

CPU und Arbeitsspeicher

Die CPU mit dem Arbeitsspeicher unseres KI-Computers verarbeitet dann diese Tokens, um aus einer Eingabe aus Tokens ein Ergebnis aus Tokens zu erzeugen – z.B. Fragetext zu Antworttext, oder Sprache zu Text. Das Programm ist dabei unserem Gehirn nachempfunden. Es ist ein *künstliches Neuronales Netzwerk* – eine Anordnung von nebeneinander und hintereinander dicht verbundenen *künstlichen Neuronen* **(siehe Abb. 1).**

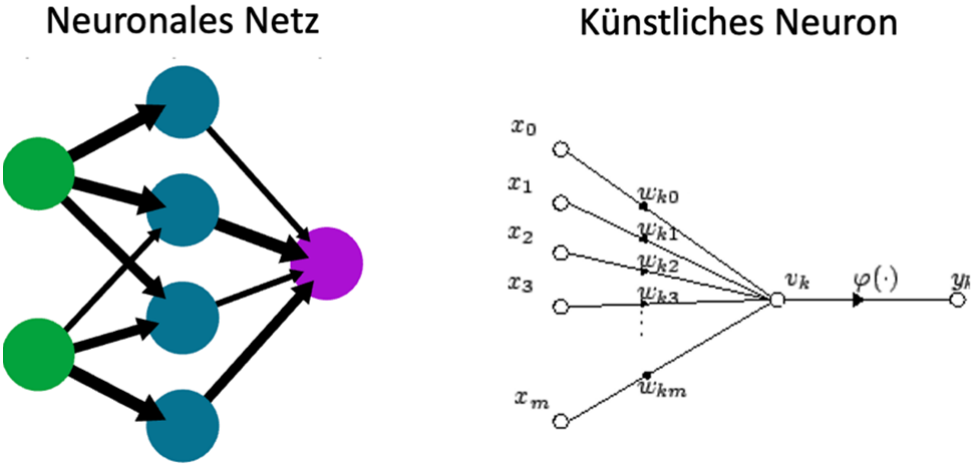

Abbildung 1 – künstliche Neuronale Netze (Quelle: Wikipedia)

Die einzelnen Neuronen verarbeiten dabei die Zahlen an ihren Eingängen mit mathematischen Funktionen zu einer Zahl an ihrem Ausgang. Dieser Ausgang bildet wieder die Eingänge für andere Neuronen im Netzwerk. Die mathematischen Funktionen im Neuron sind verschiedene, z.B. Addition, Multiplikation aber auch z.B. tanh. Diese Funktionen haben intern sog. *Parameter* (Zahlen) um die Funktion zu steuern. Die Werte dieser Parameter

bestimmen eigentlich die „Intelligenz“ des Neurons bzw. des Neuronalen Netzwerkes. Und die Parameter werden nicht von Menschen programmiert, sondern vom KI-Computer gelernt (über *Machine Learning*, mehr dazu später), die Art und Anordnung der Neuronen im Netzwerk wird hingegen programmiert.
Alle diese Zahlen sind aber nicht nur einfache Zahlen, sondern auch Vektoren und mehrdimensionale Matrizen von Gleitkommazahlen – allgemein sog. *Tensor*(s). D.h. die CPU (inkl. Hauptspeicher) unseres KI Computers verarbeitet eigentlich Tensors – unser eigentliches KI-Programm ist damit eine Folge von Tensor-Verarbeitungsinstruktionen (technisch ein Graph).
Die Tensor-Verarbeitungsinstruktionen bilden gemeinsam mit den Parametern und den Token-Bedeutungen das „Programm“ des KI-Computers – ein sog. *KI-Modell* (engl. AI-Model, wie z.B. GPT-4 dem Model hinter chatGPT, oder LLaMa, …). Diese Modelle sind große Dateien, welche vom KI-Programm des klassischen Computers von der SSD, Harddisk, … geladen werden. Die Ausführung dieser Modelle erfolgt im Arbeitsspeicher. Das Modell kann bei der normalen Ausführung nichts speichern, es muss dafür ein eigener Lernmodus gestartet werden, dieser kann aber dann nur lernen.

Transformer

Aktuell sind die üblichen KI Netzwerkstrukturen sog. *Transformer* (von Google, 2017), und wir beschäftigen uns hier nur mit diesen. Transformer verhalfen der KI zum Durchbruch im letzten Jahr nach vielen Jahren. chatGPT z.B. ist eine Abkürzung für chat mit einem Generative Pretrained Transformer. Das faszinierende an Transformern ist, dass sie vergleichsweise einfach zu verarbeiten sind, und auch deutlich einfacher lernen als ihre Vorläufer. Sie liefern gute Ergebnisse für Text, Bilder, Video, … und konnten damit die dafür vorher komplett verschiedenen Forschungsrichtungen der KI vereinen.

Was ist das Wichtige an Transformern?

- Transformer basierte Netzwerke verarbeiten die Eingabe zum Ergebnis als erstes Token der Ausgabe. Dieses Ausgabetoken wird der Eingabe angefügt und fürs nächste Token wieder durchs Netzwerk geschickt. Das wird so lange wiederholt, bis das Ergebnis komplett ist.
- Dies wird innerhalb eines Dialoges wiederholt, in dem die nächste Frage allem bisherigen im Dialog angefügt und damit in den Transformer geschickt wird. Damit gibt es eine Art Kurzzeitgedächtnis innerhalb des Dialoges. Denn das Abarbeiten des Netzwerkes verändert ja keine Parameter, es gibt kein „Merken“.
- Dieses Kurzzeitgedächtnis hat aber ein Limit, z.B. 4096 Token. *Ein Transformer berücksichtigt den Dialog für Antworten nur innerhalb des Token-Limits!*

Mehr Details zu Transformern gibt es z.B. in meinem ClubComputer Vortrag am 19.9.23.

KI Verarbeitung ist technisch aufwändig

KI-Modelle sind sehr groß, z.B. gute Textgenerierungsmodelle (Large Language Models *LLM*s) haben 10^{10}, (LLaMa) bis 10^{12} (GPT-4) Parameter. Diese sind eigentlich Gleitkommazahlen (4 Byte bzw. mit reduzierter, aber ausreichender Genauigkeit 2 Byte groß) und haben damit 20GB bis einige TB-Speicherbedarf. Ein Beispiel für eine Tensor-Verarbeitungsinstruktion ist eine Matrixmultiplikation von einem 4096×1 Tensor mit einem 4096×4096 Tensor und dem Resultat eines 4096×1 Tensors. Für einen Taschenrechner wären das 16‘777‘261 Multiplikationen und dann beinahe ebenso viele Additionen. Dies ist sehr aufwendig, aber

eignet sich ideal zum Ausführen auf GPUs (Nvidia, AMD, …). Diese GPUs können oft tausende Operationen gleichzeitig ausführen. Damit brauchen wir für die KI aber idealerweise GPUs mit sehr viel schnellem VRAM – 20GB für kleine, gute Modelle und bis zu einigen TB für chatGPT-ähnliche.
Die CPU und Arbeitsspeicher des physischen Computers für unseren KI-Computer ist damit üblicherweise ein guter PC mit einer modernen Grafikkarte (idealerweise mindestens 24GB VRAM). Mit einem KI-Programm, welches die Token-Umwandlung, Steuerung und die Tensor-Instruktionen für die GPU ausführt. Wenn der KI-Computer nichts lernen muss (das warum dazu folgt später) bzw. die Verarbeitung länger dauern darf, dann reicht auch ein kleiner PC. Moderne Handys, neue Macs, … haben zur Beschleunigung kleinerer KI-Modelle eigene Hardware (TPU, NPU, …) eingebaut, aber nur für die einfacheren Abfragen eines Modells, z.B. für KI-Bildverbesserung, Spracherkennung.

Der Speicher des KI Computers – eigentlich das Gedächtnis

Euer Handy, PC, … speichert Bilder, Texte, usw. im Speicher oder Massenspeicher (SSD, Harddisk, …), damit Ihr sie z.B. anschauen und lesen könnt – speichern und auslesen ist einfach (sofern man etwas findet).
Der Speicher unseres KI-Computers ist anders – die Parameter des KI-Modells sind hier unser Speicher. Dieser Speicher ist aber ähnlich unserem Gedächtnis – *er kann nichts Neues speichern, sondern er lernt*. Auch die Abfrage des Speichers ist anders, er wird *über ähnliche Inhalte ausgelesen*, z.B. als Antwort auf eine Frage.
Dieses Lernen ist, so wie für uns Menschen, mühsam und zeitraubend. Aber anders als wir vergisst der Speicher nicht, und einmal angelernt, kann er sehr schnell auf neue KI-Computer kopiert werden (das geht bei uns Menschen nicht so einfach). Die Abfrage des Speichers ist im Vergleich zum Anlernen viel schneller – z.B. eine Sekunde zum Abfragen eines Antwortsatzes, vs. Stunden, um einen ähnlich langen Satz auf einem gleichartigen Computer anzulernen. Ich kann mich Gott sei Dank nicht mehr erinnern, wie lange ich zum Auswendiglernen der Bürgschaft von Schiller gebraucht habe, aber ich hoffe, das Verhältnis von Lernen zum aufsagen Können war bei mir weit besser!
Dieser angelernte Speicher hat noch eine Eigenheit – er speichert völlig anders als ein normaler Computerspeicher – nicht direkt, sondern über eine *selbst erkannte Bedeutung und Zusammenhänge*. Die Bedeutung und Zusammenhänge der Parameter sind für uns Menschen nicht erkennbar, sie sind eine „Black Box“, selbst wenn wir die eigentlichen Parameterzahlen im Speicher sehen können. Das Gelernte wird bei einer Abfrage dann nicht direkt ausgelesen, sondern wieder zusammengesetzt.
Aber das *Abfrageergebnis ist nicht direkt identisch zum Gelernten*, sondern beim Abfragen liefert das Neuronale Netzwerk eigentlich mehrere statistisch wahrscheinliche Ergebnisse. Im Detail werden bei der Abfrage die Ergebnisse Token um Token, und mit für das jeweilig nächste Token-Ergebnis verschiedenen Werten samt Trefferwahrscheinlichkeiten zusammengesetzt. Manchmal wird „gewürfelt“, wenn die Wahrscheinlichkeiten für das nächstfolgende Token ähnlich sind. Für Techies ist diese Zufallsauswahlschwelle einstellbar, d.h. ab welchem Wahrscheinlichkeitsunterschied gewürfelt wird. Sie heißt *temperature* – engl. für Fieber, bei hohem gibt's ja oft Halluzinationen – eine KI mit Fieber ist ein sehr kreativer „G'schichtlerfinder“.
Wie funktioniert das bei unserem GGML KI Computer in einem Beispiel:

- Unser KI Computer lernte für seinen Speicher *„The Second Law of Robotics: A robot must obey the orders given to it by human beings …"* – etwas für Asimov Fans und passend zur KI.
- Abgefragt wird der Speicher dann über *„What is the second law of Robotics?"* oder so ähnlich – die Abfrage muss ja nicht genau sein.

· Die Antwort aus dem KI Speicher beginnt als 1. Token mit zwei Alternativen:
1. *„A"* für *„A robot must obey the orders …"* mit ca. 50% Wahrscheinlichkeit,
2. *„Robots"* für *"Robots must obey orders …",* mit fast 50% Wahrscheinlichkeit,
alles andere ist unwahrscheinlich

- Es wird hier damit meist „gewürfelt", was als Ergebnis genommen wird (meistens ist temperature so eingestellt, dass immer nur das Wahrscheinlichste genommen wird)
- Wenn einmal ein Token gewählt wurde, so geht es dann mit diesem auch fix weiter. Unser Speicher liefert plausiblen, dazu passenden Rest, wieder Token um Token statistisch ausgewählt als nächste Schritte. In der 2. Variante fehlt dann auch ein „the" vs. der 1.

Das heißt, die Antwort ist hier inhaltlich korrekt, aber ev. nicht ganz genauso wie angelernt. Und nicht immer gleich.
Das ist eben sehr anders als ein normaler Computerspeicher, der uns (hoffentlich) immer genau das Gespeicherte liefert. Aber die Abfrage des Speichers ist auch Teil einer KI, d.h. es muss nicht exakt so wie angelernt abgefragt werden (siehe unser Beispiel), der Speicher ist über die Bedeutung abfragbar!

Künstliche Intelligenz

Wie siehts aus, wenn eine Frage so eigentlich nicht im Speicher ist? Es werden in den Parametern des Neuronalen Netzes ja keine Texte, Bilder, … gespeichert, sondern nur selbst gelernte Zusammenhänge und Bedeutungen. Und aus diesen ergeben sich dann unerwartete Fähigkeiten, speziell bei großen KI-Modellen mit vielen Parametern:

- KI-Modelle verstehen den Inhalt unsere Kommunikation (Text, Sprache, Bilder)
- Sie bilden daraus interne Muster und können damit Neues generieren – z.B. übersetzen, manche Mathematik, neu formulieren, Fragen beantworten, …
- Große Modelle wie GPT-4 entwickeln deutlich mehr unerwartete Fähigkeiten, die ähnlich logischem Denken scheinen

Zur Wiederholung, intern liefert die KI mit den Parametern aus dem Speicher ja immer nur Schätzwerte fürs nächste Token, und diese passend zu den Tokens der Frage und den bisherigen Tokens der Antwort. Die KI kann nicht vorausplanen, sie erkennt aber unglaublich gut die gelernten und vielfach abstrahierten Muster und wählt darüber aus.
Der aktuelle Stand der Forschung ist, dass die KI-Forscher nicht verstehen, warum diese riesengroßen Neuronalen Netze aus Transformer Technologie solche unerwarteten Fähigkeiten entwickeln. Und die Fähigkeiten von neuen, größeren Modellen sind aktuell nicht vorhersehbar und planbar. Das ist einer der Gefahrenpotenziale der KI.

Künstliche Intelligenz & Halluzination

Ein weiterer großer Unterschied des KI-Speichers ist aber auch leider, dass es auf eine Anfrage kein „ich weiß nicht“ gibt, sie wird immer beantwortet. *Unser KI Computer Speicher weiß nicht, wenn er nichts weiß* – Sokrates hätte keine Freude damit!
Wir bekommen aus unserem KI Computer IMMER eine Antwort, diese ist aber ev. völlig falsch, klingt aber wahrscheinlich grammatikalisch richtig und durchaus glaubwürdig – unsere KI hat ja unsere Sprache und große Teile des Internets perfekt gelernt. Das bezeichnet man dann als *Halluzination* der KI. Dieses massive Problem ist aktuell einer der wichtigsten Forschungsbereiche der KI.
Ein Beispiel: Bei mir ist die Antwort auf die Frage „Wie lautet die erste Strophe der Bürgschaft?“ sofort „Ich weiß sie nicht mehr ganz, nur den Anfang“. Unser KI Computer Speicher liefert immer ein Ergebnis. Hier z.B. chatGPT:

Zu Dionys, dem Tyrannen, schlich
Damon, den Dolch im Gewande,
Ihn schlugen die Häscher in Bande,
‘Grausamer’, rief er, ‘dich
Führt keine Gewalt des Gesetzes
Vor’s Tribunal der Gerechtigkeit.

Nach dem anscheinend Gelernten der ersten drei Zeilen geht es nahtlos in die Halluzination basierend auf den gelernten Mustern!
Aber wie lernt unser KI Computer Speicher? Die Parameter bestimmen die Intelligenz des Neuronalen Netzwerkes und werden angelernt.

Machine Learning

Das Anlernen der Parameter erfolgt darüber, dass dem Neuronalen Netzwerk Beispiel um Beispiel vorgesetzt wird, das es auswendig lernen soll. Es fragt dazu den bestehenden Speicher mit dem ersten Beispiel ohne dessen letztes Token als „Soll“ ab, und vergleicht das Ergebnis-Token, d.h. „Ist“ mit diesem „Soll“. Wenn „Ist“ nicht mit „Soll“ übereinstimmt, berechnet es die Abweichung und versucht es mit daraus geänderten Parametern des Neuronalen Netzes nochmals. Solange, bis das Ergebnis übereinstimmt. Dann wird das nächste Beispiel verwendet.
Ein Anlernbeispiel wäre wieder mal Asimov:
„The Second Law of Robotics“…
Dies wird in folgende Unterbeispiele zum Anlernen unterteilt:
1. *„The“* als Eingabe soll *„ Second*“ liefern,
2. *„The Second*“ als Eingabe soll *„ Law*“ liefern,
3. *„The Second Law*“ als Eingabe soll *„ of*“ liefern,
4. *„The Second Law of*‘ als Eingabe soll *„ Robotics*“ liefern
5. …

Jedes der Unterbeispiele wird durchs Neuronale Netz mit den aktuellen Parametern geschickt, und die Abweichung vom Sollwert wird berechnet. Daraus werden die Parameteränderungen ermittelt und wiederholt, bis das Ergebnis mit dem Soll übereinstimmt.
Noch zur Komplexität des Parameter-Änderungen-Ermittelns: Das mag einfach klingen, aber bei den z.B. 10^{10}-10^{12} Parametern der üblichen Modelle ist es sehr schwierig herauszufinden, welche Parameter um wieviel geändert werden sollen. Technisch funktioniert das meist darüber, dass die mathematischen Funktionen des Neuronalen Netzes differenziert werden und dann das Neuronale Netz von Richtung Ergebnis her mit der Größe der Soll-Ist Abweichung (*loss*) und diesen differenzierten Funktionen durchgerechnet wird (*back-propagation*). Damit werden dann die Parameter gering verändert. Die gängigen Tensor-Verarbeitungsbibliotheken können das automatisiert.

Dies veranschaulicht hoffentlich, warum das Lernen so viel länger dauert als das Abfragen.
Beispiel: Ein typisches LLM wie z.B. LLaMa von Meta (dem facebook Mutterkonzern) lernte ca. einen Monat auf einem schätzungsweise 50 Mio. EUR teuren Superrechner und einem Stromverbrauch von 1'500 Haushalten! Und es lernte dabei aus fast einer Billion (10^{12}) Wörtern des Internets – gesamtes Wikipedia in verschiedenen Sprachen, Projekt Gutenberg … (öffentlich verfügbare Bücher), ArXiv.org (Wissenschaftliche Arbeiten), GitHub und Stack Exchange (Programmierung/Technik), sowie aus dem allgemeinen Internet (mit einer eigenen KI vorsortiert). LLaMa-2 hat auch angelernte Sicherheitsmechanismen.
Aber vom Lernmaterial sind nur ~0,2% deutschsprachige Inhalte. Es spricht damit relativ gut Deutsch, aber hat fast keine deutschsprachige Kultur gelernt. Das ist typisch, und momentan sind damit fast alle KI-Modelle meist sehr in Richtung US-Anglikanischem Sprachraum vor-eingenommen (*bias*). Siehe unser Beispiel mit chatGPT und der Bürgschaft.
Durch das Anlernen des Internets kommen aber auch ggf. illegale Inhalte in das Modell. Alle notwendigen sicherheitsrelevanten Dinge werden dem LLM auch mit angelernt. z.B. GPT-4 lernte viele Monate vor Veröffentlichung und auch noch laufend Sicherheit.
Wichtig ist auch noch, dass, während auf dem KI-Computer intensiv gelernt wird, nichts abgefragt werden kann. *Ein KI-Computer ist entweder im langsamen Lernmodus oder im Abfragemodus*. Und im Lernmodus wird meistens ein weit leistungsfähigerer Computer benötigt als im Abfragemodus. Dies gilt für die aktuellen KI-Computer, und kann sich in Zukunft ändern.

Ein Typischer KI Computer in der Anwendung

Praktisch startet unser typischer KI Computer meist damit, dass er ein KI Modell lädt, welches von einem sehr, sehr großen KI Computer angelernt wurde. Dieser hat dafür wahrscheinlich Monate lang gelernt, so dass das Modell dann als KI brauchbar ist.
Dann arbeitet unser KI Computer brav mit diesem Speicher, erzeugt damit z.B. Texte als Antworten auf Abfragen. Eigentlich einfach!
Er lernt aber beim Arbeiten nichts dazu. Alles, was er verarbeitet, wird wieder vergessen. Außer jemand speichert das parallel im konventionellen Computer und gibt das ev. später einem andern, größeren KI Computer zum Anlernen. So ist z.B. chatGPT im allgemeinen Internetwissen von 2021 stecken geblieben, hat nur eher Kleinigkeiten später dazugelernt!
Damit KI-Modelle auf kleineren Rechnern ausgeführt werden können, wurde ein Trick erfunden. Die meisten Parameter eines KI-Modells brauchen nicht sehr genau zu sein – 4 Bit Informationen als Abweichung zu einem Hauptparameter reichen meistens. Über diese *Quantisierung* kann der Speicherbedarf dramatisch (z.B. ~75%) gesenkt werden.
Ein anderer Trick hilft dabei, ein Modell einfacher mit neuem Wissen zu verfeinern. Bei einer fertig angelernten KI ändern sich auch mit vielen neuen Daten nur vergleichsweise wenige Parameter. Diese werden separat gespeichert (*LoRA*, low-*rank adaption*). Das vermindert den Speicherplatz und Aufwand, vor allem, wenn es auch noch mit Quantisierung verbunden wird – als QLoRA.

Der echte GGML KI Computer

Bisher war es ja eher ein Gedankenexperiment, aber unseren GGML KI Computer gibt es wirklich, auch wenn meine bisherige Beschreibung eine Vereinfachung ist. Er wurde initial von Georgi Gerganov aus Bulgarien in einem Nachmittag gebaut (GGML sind seine Initialen und „Model Language"). Er ist inzwischen aber dramatisch gewachsen und eines der erfolgreichsten Open Source Community KI Projekte. Georgi wollte KI auf seinem M1 MacBook Air (einem sehr kleinen und leichten, aber sehr leistungsfähigen Laptop) ausführen können. Und zwar für zwei Anwendungen:

1. *llama.cpp* (https://github.com/ggerganov/llama.cpp) – eine chatGPT-ähnliche Kommandozeilen/Terminal App für das LLaMa KI-Model und andere, sowie
2. *whisper.cpp* https://github.com/ggerganov/whisper.cpp – ähnlich dazu eine KI Spracherkennungslösung mit KI-Modell von openAI.

Die zwei Programme laufen auf Windows, Mac und Linux, idealerweise auf einem modernen Rechner (whisper.cpp aber auch am Raspberry Pi).

llama.cpp

Der Einfachheit halber sehen wir uns llama.cpp an. Wenn Ihr es ausprobieren wollt, einfach von der obigen GitHub Adresse herunterladen und lt. README.md kompilieren.

llama.cpp ist ein einfaches Programm rund um den GGML code (welcher grob unserem KI-Computer entspricht). Das „main“ Programm von llama.cpp wird mit einer KI-Model Datei gestartet. Diese enthält, neben allgemeinen Einstellungen, die KI-Model Parameter (unseren KI Computer Speicherinhalt) sowie die Tensor-Verarbeitungsinstruktionen für das Modell.

Die Modelle für llama.cpp findet Ihr z.B. auf huggingface.co. Mein Favorit ist dabei https://huggingface.co/TheBloke/koala-7B-GGML und dort das koala-7B.ggmlv3.q4_0.bin

Modell. Es kommt bei Modellen immer auf Euren Rechner an, Koala 7B in 4-Bit Quantisierung ist ein guter Minimalkompromiss und braucht nur 6,3 GB RAM.

Hier ein Beispiel, wie llama.cpp das Modell intern auf meinem M2 Mac sieht, welche Tensor Größen (in 3 Tensor-Dimensionen) und Operationen beteiligt sind, sowie die Ausführungszeiten der Operationen:

```
=== GRAPH ===
n_nodes = 1188                                                  ms
  -   0: [  4096,     1,     1]         GET_ROWS   (  1) cpu =   0,006
  -   1: [  4096,     1,     1]         RMS_NORM   (  1) cpu =   0,011
  -   2: [  4096,     1,     1]              MUL   (  1) cpu =   0,036
  -   3: [  4096,     1,     1]          MUL_MAT   (  1) cpu =   3,667
  -   4: [   128,    32,     1]          RESHAPE   (  1) cpu =   0,000
  -   5: [   128,    32,     1]             ROPE   (  1) cpu =   0,004
  -   6: [  4096,     1,     1]             VIEW   (  1) cpu =   0,001
  -   7: [  4096,     1,     1]             CPY    (  1) cpu =   0,001
  -   8: [  4096,     1,     1]          MUL_MAT   (  1) cpu =   5,017
  -   9: [  4096,     1,     1]          RESHAPE   (  1) cpu =   0,001
  -  10: [     1,  4096,     1]        TRANSPOSE   (  1) cpu =   0,000
  -  11: [     1,  4096,     1]             VIEW   (  1) cpu =   0,000
  -  12: [     1,  4096,     1]             CPY    (  1) cpu =   1,341
  -  13: [     1,   128,    32]             VIEW   (  1) cpu =   0,000
  -  14: [  4096,     1,     1]             VIEW   (  1) cpu =   0,002
  -  15: [   128,    32,     1]          RESHAPE   (  1) cpu =   0,000
  -  16: [   128,     1,    32]          PERMUTE   (  1) cpu =   0,000
  -  17: [  4096,     1,     1]          MUL_MAT   (  1) cpu =   2,574
  -  18: [   128,    32,     1]          RESHAPE   (  1) cpu =   0,001
  -  19: [   128,    32,     1]             ROPE   (  1) cpu =   0,002
  -  20: [   128,     1,    32]          PERMUTE   (  1) cpu =   0,000
  -  21: [     1,     1,    32]          MUL_MAT   (  1) cpu =   0,054
  -  22: [     1,     1,    32]            SCALE   (  1) cpu =   0,002
  -  23: [     1,     1,    32]     DIAG_MASK_INF   (  1) cpu =   0,001
  -  24: [     1,     1,    32]          SOFT_MAX   (  1) cpu =   0,001
  -  25: [   128,     1,    32]          MUL_MAT   (  1) cpu =   0,037
  -  26: [   128,    32,     1]          PERMUTE   (  1) cpu =   0,000
  -  27: [  4096,     1,     1]             CPY    (  1) cpu =   0,007
  -  28: [  4096,     1,     1]          MUL_MAT   (  1) cpu =   3,009
  -  29: [  4096,     1,     1]              ADD   (  1) cpu =   0,008
  -  30: [  4096,     1,     1]         RMS_NORM   (  1) cpu =   0,010
  -  31: [  4096,     1,     1]              MUL   (  1) cpu =   0,008
  -  32: [ 11008,     1,     1]          MUL_MAT   (  1) cpu =   6,596
  -  33: [ 11008,     1,     1]             SILU   (  1) cpu =   0,015
  -  34: [ 11008,     1,     1]          MUL_MAT   (  1) cpu =   7,957
  -  35: [ 11008,     1,     1]              MUL   (  1) cpu =   0,009
  -  36: [  4096,     1,     1]          MUL_MAT   (  1) cpu =   7,210
  -  37: [  4096,     1,     1]              ADD   (  1) cpu =   0,006
  -  38: [  4096,     1,     1]         RMS_NORM   (  1) cpu =   0,008
  -  39: [  4096,     1,     1]              MUL   (  1) cpu =   0,009
  -  40: [  4096,     1,     1]          MUL_MAT   (  1) cpu =   4,150
  -  41: [   128,    32,     1]          RESHAPE   (  1) cpu =   0,000
  -  42: [   128,    32,     1]             ROPE   (  1) cpu =   0,003
  -  43: [  4096,     1,     1]             VIEW   (  1) cpu =   0,000
  -  44: [  4096,     1,     1]             CPY    (  1) cpu =   0,410
  -  45: [  4096,     1,     1]          MUL_MAT   (  1) cpu =   3,897
  -  46: [  4096,     1,     1]          RESHAPE   (  1) cpu =   0,000
  -  47: [     1,  4096,     1]        TRANSPOSE   (  1) cpu =   0,000
...

n_leafs = 582
  -   0: [  4096,  4096]     NONE
  -   1: [  4096, 32000]     NONE
  -   2: [     1,     1]     NONE
  -   3: [  4096,     1]     NONE
  -   4: [     3,     1]     NONE
  -   5: [ 67108864,     1]     NONE
  -   6: [     2,     1]     NONE
  -   7: [  4096,  4096]     NONE
  -   8: [ 67108864,     1]     NONE
  -   9: [     2,     1]     NONE
  -  10: [  4096,  4096]     NONE
  -  11: [ 11008,  4096]     NONE
  -  12: [  4096, 11008]     NONE
  -  13: [  4096,  4096]     NONE
  -  14: [     2,     1]     NONE
  -  15: [     2,     1]     NONE
  -  16: [  4096,  4096]     NONE
  -  17: [     3,     1]     NONE
  -  18: [     1,     1]     NONE
  -  19: [     2,     1]     NONE
  -  20: [  4096,     1]     NONE
  -  21: [  4096,     1]     NONE
  -  22: [  4096, 11008]     NONE
  -  23: [  4096,     1]     NONE
  -  24: [     3,     1]     NONE
  -  25: [     2,     1]     NONE
  -  26: [  4096,  4096]     NONE
  -  27: [     2,     1]     NONE
  -  28: [  4096,  4096]     NONE
  -  29: [ 11008,  4096]     NONE
  -  30: [  4096, 11008]     NONE
  -  31: [  4096,  4096]     NONE
  -  32: [     2,     1]     NONE
  -  33: [     2,     1]     NONE
```

Abbildung 2 llama.cpp Tensorverarbeitung und Parameterstruktur

Mit llama.cpp und z.B. Koala7B (das steht für 7 Milliarden Parameter) könnt ihr komplett lokal auf Eurem Rechner experimentieren. Gutes Gelingen und viel Spaß damit!