Generative KI-Systeme entwickeln

KI-Engineering für die Praxis – vom Prompt Engineering bis zu RAG und Agenten

DIE LESEPROBE

KAPITEL 2

Foundation Models verstehen

Um mit Foundation Models Anwendungen zu bauen, müssen Sie zuerst die Foundation Models verstehen. Sie müssen zwar nicht wissen, wie man ein Modell entwickelt, um es zu verwenden, aber ein grobes Verständnis wird Ihnen dabei helfen, zu entscheiden, welches Modell genutzt werden sollte und wie Sie es an Ihre Anforderungen anpassen.

Das Trainieren eines Foundation Model ist ein unglaublich komplexer und teurer Prozess. Diejenigen, die wissen, wie man das richtig macht, werden vermutlich durch Verschwiegenheitserklärungen daran gehindert, das Geheimnis zu verraten. Dieses Kapitel wird Ihnen nicht erklären können, wie Sie ein Modell bauen, das eine Konkurrenz zu ChatGPT sein könnte. Stattdessen werde ich mich auf Designentscheidungen konzentrieren, die Auswirkungen auf daraus gebaute Anwendungen haben können.

Mit der wachsenden Intransparenz des Trainingsprozesses der Foundation Models ist es schwierig, all die Designentscheidungen zu kennen, die in das Entstehen eines Modells fließen. Im Allgemeinen lassen sich die Unterschiede in Foundation Models aber auf Entscheidungen bei den Trainingsdaten, der Architektur und Größe des Modells und dem Post-Training für ein Anpassen an menschliche Vorlieben zurückführen.

Da Modelle von Daten lernen, haben diese einen großen Einfluss auf die Fähigkeiten und Grenzen des Modells. Dieses Kapitel wird zunächst zeigen, wie Modellentwickler und -entwicklerinnen Trainingsdaten kuratieren, und sich dabei auf die Verteilung der Daten konzentrieren. In Kapitel 8 werden wir Dataset-Techniken im Detail erforschen und dabei auch auf die Evaluation der Datenqualität und die Datensynthese eingehen.

Angesichts der Dominanz der Transformer-Architektur scheint es so auszusehen, als hätte man bei der Modellarchitektur weniger die Wahl. Sie fragen sich vielleicht, was die Transformer-Architektur so besonders macht, dass sie dauerhaft dominiert. Wie lange dauert es, bis eine andere Architektur die Führungsrolle übernimmt, und wie könnte diese neue Architektur aussehen? Dieses Kapitel wird all diese Fragen angehen. Immer dann, wenn ein neues Modell veröffentlicht wird, wollen die Menschen zuallererst etwas über die Größe wissen. Daher schauen wir uns in diesem Ka-

pitel ebenfalls an, wie ein Modellentwickler oder eine -entwicklerin die passende Größe für das Modell herausfinden kann.

Wie in Kapitel 1 erwähnt, wird der Trainingsprozess oft in ein Pre-Training und ein Post-Training unterteilt. Das Pre-Training sorgt dafür, dass ein Modell einsatzfähig ist – aber nicht notwendigerweise auf eine sichere oder einfache Art und Weise. Hier kommt das Post-Training ins Spiel. Das Ziel des Post-Trainings ist es, das Modell auf die Vorlieben der Menschen abzustimmen. Aber was genau sind die *Vorlieben der Menschen*? Wie können diese so repräsentiert werden, dass ein Modell damit lernen kann? Die Vorgehensweise, wie ein Modellentwickler oder eine Entwicklerin das Modell abstimmt, hat einen deutlichen Einfluss auf die Nutzbarkeit des Modells. Wir werden darüber in diesem Kapitel sprechen.

Während der Einfluss des *Trainings* auf die Leistung eines Modells meist verstanden wird, wird der durch das *Sampling* oft übersehen. Dabei geht es darum, wie ein Modell eine Ausgabe unter allen möglichen Ausgaben wählt. Es ist das vielleicht am stärksten unterschätzte Konzept in der KI. Sampling erklärt nicht nur viele scheinbar rätselhaften KI-Verhaltensweisen – einschließlich Halluzinationen und Inkonsistenzen –, die Wahl der richtigen Sampling-Strategie kann die Leistung eines Modells auch noch mit relativ wenig Aufwand steigern. Aus diesem Grund ist der Abschnitt über Sampling derjenige, den ich in diesem Kapitel am liebsten geschrieben habe.

Die in diesem Kapitel behandelten Konzepte sind grundlegend für ein Verständnis des Rests dieses Buchs. Aber weil sie so grundlegend sind, sind Sie vielleicht schon damit vertraut. Überspringen Sie gerne die Konzepte, bei denen Sie sich sicher fühlen. Wenn Sie später über ein verwirrendes Konzept stolpern, können Sie hier noch einmal nachlesen.

Trainingsdaten

Ein KI-Modell ist nur so gut wie die Daten, mit denen es trainiert wurde. Gibt es in den Trainingsdaten kein Vietnamesisch, wird das Modell nicht aus Englisch nach Vietnamesisch übersetzen können. Das Gleiche gilt für ein Modell zur Bildklassifikation – bekommt es nur Tiere im Trainingsdatensatz zu sehen, wird es nicht gut mit Fotos von Pflanzen funktionieren.

Wollen Sie ein Modell bei einer bestimmten Aufgabe verbessern, sollten Sie sich überlegen, mehr Daten dafür in die Trainingsdaten zu übernehmen. Aber es ist nicht einfach, ausreichend Daten zum Trainieren eines großen Modells zusammenzubekommen, und es kann auch teuer sein. Bei der Modellentwicklung kann man oft nur auf bestehende Daten zurückgreifen, selbst wenn diese nicht genau die Erfordernisse erfüllen.

So ist beispielsweise Common Crawl eine häufig genutzte Quelle für Trainingsdaten (*https://oreil.ly/wf2Lw*). Es wurde von einer gemeinnützigen Organisation erstellt, die sporadisch Websites im Internet abgrast. In den Jahren 2022 und 2023 hat diese Organisation jeden Monat ungefähr zwei bis drei Milliarden Webseiten besucht.

Google stellt eine saubere Untermenge von Common Crawl mit dem Namen »Colossal Clean Crawled Corpus« bzw. C4 bereit (*https://arxiv.org/abs/1910.10683v4*).

Die Datenqualität von Common Crawl – und zu einem gewissen Grad auch von C4 – ist diskussionswürdig. Denken Sie an Clickbait, Fehlinformation, Propaganda, Verschwörungstheorien, Rassismus, Misogynie und all die fragwürdigen Websites, die Sie schon im Internet gesehen oder bewusst vermieden haben. Eine Studie der Washington Post (*https://oreil.ly/-1UMD*) zeigt, dass die 1.000 am häufigsten im Datensatz vorkommenden Websites eine Reihe von Medienangeboten beinhalten, die auf dem NewsGuard Scale for Trustworthiness (*https://oreil.ly/OisOs*) ziemlich weit unten stehen. Kurz gesagt – Common Crawl enthält eine Menge Fake News.

Aber weil Common Crawl eben verfügbar ist, finden sich Variationen davon in den meisten Foundation Models, die die Quellen ihrer Trainingsdaten bekannt gegeben haben, unter anderem GPT-3 von OpenAI und Gemini von Google. Ich vermute, dass Common Crawl auch in Modellen zum Einsatz kommt, die ihre Trainingsdaten nicht offenlegen. Um eine genauere Untersuchung durch die Öffentlichkeit und Konkurrenten zu vermeiden, geben viele Firmen diese Informationen nicht mehr bekannt.

Manche Teams nutzen Heuristiken, um Daten schlechter Qualität aus dem Internet herauszufiltern. So hat OpenAI beispielsweise nur Reddit-Links genutzt, die mindestens drei Upvotes bekommen haben, um GPT-2 zu trainieren (*https://oreil.ly/gGwRz*). Das hilft zwar, Links herauszuschmeißen, um die sich niemand schert, aber Reddit ist trotzdem nicht unbedingt ein Hort des Anstands und des guten Geschmacks.

Der Ansatz »Wir nutzen, was wir haben, nicht, was wir wollen« kann zu Modellen führen, die bei Aufgaben gut funktionieren, für die Trainingsdaten vorhanden sind, aber nicht unbedingt bei den Aufgaben, die Ihnen wichtig sind. Um dieses Problem anzugehen, ist es entscheidend, Datensätze zu kuratieren, die zu Ihren spezifischen Anforderungen passen. Dieser Abschnitt konzentriert sich auf das Kuratieren von Daten für bestimmte *Sprachen* und *Domänen* und bietet damit eine breite, trotzdem aber spezialisierte Grundlage für Anwendungen in diesen Bereichen. In Kapitel 8 geht es um Datenstrategien für Modelle, die auf hoch spezialisierte Aufgaben zurechtgeschnitten sind.

Während sprach- und domänenspezifische Foundation Models von Grund auf trainiert werden können, ist es auch verbreitet, sie ausgehend von allgemein nutzbaren Modellen zu optimieren.

Sie fragen sich vielleicht, warum man nicht einfach ein Modell mit allen verfügbaren Daten trainiert – allgemeine und spezialisierte –, sodass das Modell dann alles kann? Das fragen sich viele Menschen. Aber ein Trainieren mit mehr Daten erfordert oft mehr Rechenressourcen, und es führt nicht immer zu einer besseren Leistung. Ein Modell, das beispielsweise mit einer kleineren Menge qualitativ hochwertiger Daten trainiert wurde, kann besser sein als ein Modell, das mit einer großen Menge Daten in schlechterer Qualität trainiert wurde. Gunasekar et al. haben 2023 sieben Milliarden Tokens qualitativ hochwertiger Daten genutzt (*https://arxiv.org/abs/2306.11644*), um

ein Modell mit 1,3 Milliarden Parametern zu trainieren. Das lieferte dann bei vielen wichtigen Coding-Benchmarks bessere Ergebnisse als viel größere Modelle. Die Auswirkung der Datenqualität wird umfassender in Kapitel 8 behandelt.

Mehrsprachige Modelle

Englisch dominiert das Internet. Eine Analyse des Common-Crawl-Datensatzes zeigt, dass Englisch fast die Hälfte der Daten ausmacht (45,88%) und es achtmal häufiger vorkommt als Russisch als zweithäufigste Sprache (5,97%) (Lai et al., 2023, *https://arxiv.org/abs/2304.05613*). In Tabelle 2.1 sehen Sie eine Liste der Sprachen mit mindestens 1% Anteil in Common Crawl. Sprachen mit eingeschränkter Verfügbarkeit als Trainingsdaten – im Allgemeinen diejenigen, die nicht in dieser Liste zu finden sind – nennt man *Low-Resource*.

Tabelle 2.1: Die am häufigsten vorhandenen Sprachen in Common Crawl, einem beliebten Datensatz zum Trainieren von LLMs (Quelle: Lai et al., 2023)

Sprache	Code	Sprecher	CC-Größe	
		(M)	(%)	Kat.
Englisch	en	1.452	45,8786	H
Russisch	ru	258	5,9692	H
Deutsch	de	134	5,8811	H
Chinesisch	zh	1.118	4,8747	H
Japanisch	jp	125	4,7884	H
Französisch	fr	274	4,7254	H
Spanisch	es	548	4,4690	H
Italienisch	it	68	2,5712	H
Niederländisch	nl	30	2,0585	H
Polnisch	pl	45	1,6636	H
Portugiesisch	pt	275	1,1505	H
Vietnamesisch	vi	85	1,0299	H

Viele andere Sprachen haben zwar jede Menge Sprecherinnen und Sprecher, sind aber in Common Crawl deutlich unterrepräsentiert. In Tabelle 2.2 sehen Sie ein paar dieser Sprachen. Idealerweise sollte das Verhältnis zwischen dem entsprechenden Anteil der Weltbevölkerung und dem in Common Crawl 1 sein. Je höher dieses Verhältnis ist, desto unterrepräsentierter ist diese Sprache in Common Crawl.

Angesichts der Dominanz von Englisch in den Internetdaten überrascht es wenig, dass allgemein nutzbare Modelle für Englisch viel besser funktionieren als für andere Sprachen, was auch von einer Reihe von Studien belegt wird. So ist beispielsweise GPT-4 bei der MMLU-Benchmark, einer Suite mit 14.000 Multiple-Choice-Problemen zu 57 Themen, in Englisch besser als in unterrepräsentierten Sprachen wie Telugu (siehe Abbildung 2.1, *https://oreil.ly/qK2Ap*).

Tabelle 2.2: Beispiele für unterrepräsentierte Sprachen in Common Crawl. Die letzte Zeile »Englisch« dient als Vergleich. Die Zahlen für die Prozentwerte wurden von Lai et al. (2023) übernommen.

Sprache	Sprecher (Millionen)	% Weltbevölkerung	% in Common Crawl	Verhältnis Welt/Common Crawl
Panjabi	113	1,41 %	0,0061 %	231,56
Swaheli	71	0,89 %	0,0077 %	115,26
Urdu	231	2,89 %	0,0274 %	105,38
Kannada	64	0,80 %	0,0122 %	65,57
Telugu	95	1,19 %	0,0183 %	64,89
Gujarati	62	0,78 %	0,0126 %	61,51
Marathisch	99	1,24 %	0,0213 %	58,10
Bengalisch	272	3,40 %	0,0930 %	36,56
Englisch	**1452**	**18,15 %**	**45,88 %**	**0,40**

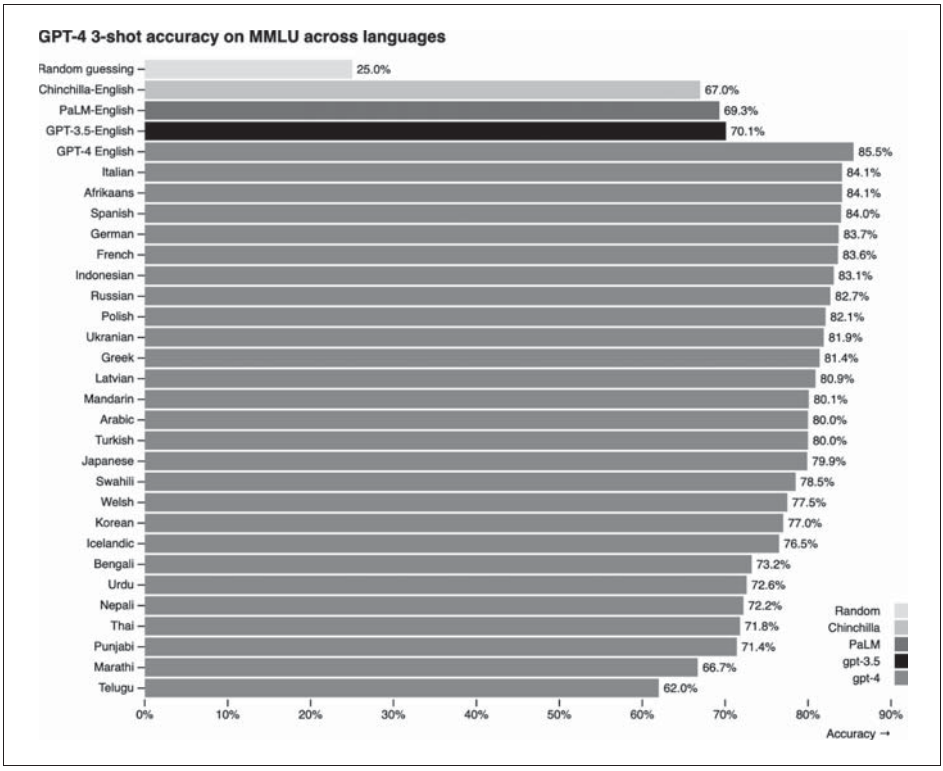

Abbildung 2.1: Bei der MMLU-Benchmark liefert GPT-4 in Englisch eine bessere Leistung als in anderen Sprachen. Um MMLU in anderen Sprachen durchführen zu können, hat OpenAI die Fragen mithilfe des Azure AI Translator übersetzt.

Beim Test mit sechs Mathematik-Problemen im Projekt Euler fand Yennie Jun heraus, dass GPT-4 Probleme in Englisch mehr als drei Mal so oft wie in Armenisch oder Farsi lösen konnte.[1] Bei Burmesisch und Amharisch hat GPT-4 bei allen sechs Fragen versagt, wie Sie in Abbildung 2.2 sehen.

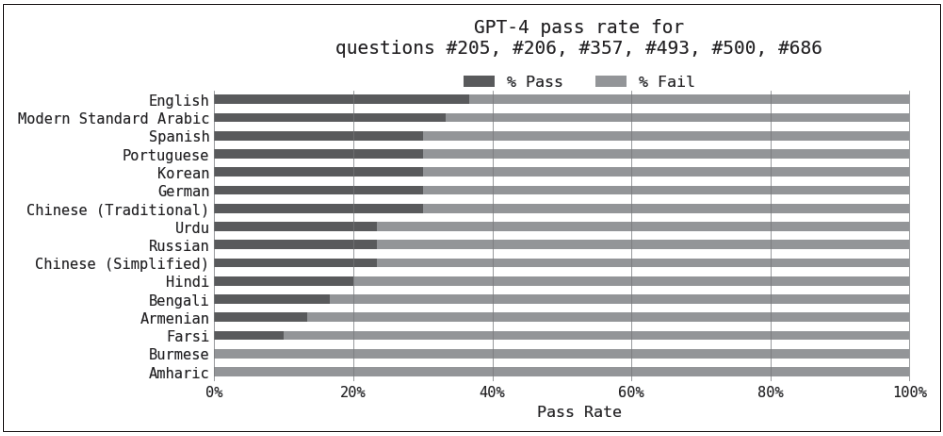

Abbildung 2.2: GPT-4 ist in Mathematik viel besser in Englisch als in anderen Sprachen.

Für diese schlechtere Leistung ist Unterrepräsentation ein wichtiger Grund. Die drei Sprachen mit der schlechtesten Performance bei den MMLU-Benchmarks von GPT-4 – Telugu, Marathi und Punjabi – gehören auch zu den Sprachen, die in Common Crawl am stärksten unterrepräsentiert sind. Aber das ist nicht der einzige Grund. Die Struktur einer Sprache und die Kultur, die sie ausdrückt, kann ebenfalls dafür sorgen, dass ein Modell in dieser Sprache schwieriger lernt.

LLMs sind beim Übersetzen im Allgemeinen ziemlich gut. Können wir da nicht einfach alle Anfragen aus anderen Sprachen ins Englische übersetzen, die Antwort erhalten und diese dann in die Originalsprache zurückübersetzen? Viele Menschen verfolgen tatsächlich diesen Ansatz, aber er ist nicht ideal. So ist dafür ein Modell erforderlich, das unterrepräsentierte Sprachen so ausreichend versteht, dass es sie übersetzen kann. Dann kann die Übersetzung für einen Informationsverlust verantwortlich sein. So haben beispielsweise manche Sprachen wie Vietnamesisch Pronomen, die die Beziehung zwischen zwei Sprechern kennzeichnen. Beim Übersetzen nach Englisch werden all diese Pronomen nach *I* und *you* übersetzt, was zum Verlust der Beziehungsinformationen führt.

Modelle können in anderen Sprachen als Englisch auch unerwartete Leistungsprobleme haben. So hat beispielsweise NewsGuard herausgefunden, dass ChatGPT viel eher Fehlinformationen auf Chinesisch liefert als auf Englisch (*https://oreil.ly/LcBfx*). Im April 2023 hat NewsGuard ChatGPT-3.5 gebeten, Artikel mit Fehlinformationen über China in Englisch, vereinfachtem Chinesisch und traditionellem Chi-

1 »GPT-4 can solve math problems—but not in all languages« von Yennie Jun (*https://oreil.ly/G13KM*). Sie können die Studie mit dem Tokenizer von OpenAI überprüfen (*https://oreil.ly/iqhNY*).

nesisch zu erstellen. Bei Englisch hat ChatGPT das Erstellen falscher Aussagen bei sechs von sieben Prompts verweigert. Aber in vereinfachtem Chinesisch und traditionellem Chinesisch wurden bei allen sieben Prompts falsche Aussagen produziert. Es ist unklar, was diesen Verhaltensunterschied verursacht hat.[2]

Neben Qualitätsproblemen können Modelle in anderen Sprachen als Englisch auch langsamer und teurer sein. Die Inferenzlatenz und die Kosten eines Modells sind proportional zur Anzahl an Tokens in der Eingabe und in der Ausgabe. Es zeigt sich, dass das Tokenisieren für manche Sprachen viel effizienter möglich ist als für andere. Bei einer Benchmark von GPT-4 auf MASSIVE, einem Datensatz mit einer Million kurzen Texten, die in 52 Sprachen übersetzt sind, fand Yennie Jun heraus, dass Sprachen wie Burmesisch und Hindi zum Vermitteln des gleichen Inhalts viel mehr Tokens brauchen als Englisch oder Spanisch (*https://oreil.ly/Zq5Sw*). Beim MASSIVE-Datensatz beträgt die mittlere Token-Länge in Englisch 7, in Hindi aber 32, und in Burmesisch sind es flockige 72 – zehn Mal mehr als in Englisch.

Wenn man davon ausgeht, dass die Zeit zum Generieren eines Tokens in allen Sprachen gleich ist, braucht GPT-4 für die gleichen Inhalte in Burmesisch ungefähr zehn Mal länger als in Englisch. Bei APIs, die nach verwendeten Tokens abrechnen, kostet Burmesisch zehn Mal mehr als Englisch.

Um das anzugehen, wurden viele Modelle darauf trainiert, sich auf andere Sprachen als Englisch zu fokussieren. Die aktivste Sprache neben Englisch ist unzweifelhaft Chinesisch mit ChatGLM (*https://github.com/THUDM/ChatGLM2-6B*), YAYI (*https://github.com/wenge-research/YAYI*), Llama-Chinese (*https://github.com/LlamaFamily/Llama-Chinese*) und anderen. Es gibt auch Modelle in Französisch (CroissantLLM, *https://oreil.ly/a6j-N*), Vietnamesisch (PhoGPT, *https://github.com/VinAIResearch/PhoGPT*), Arabisch (Jais, *https://oreil.ly/uG27L*) und vielen anderen Sprachen.

Domänenspezifische Modelle

Allgemein nutzbare Modelle wie Gemini (*https://oreil.ly/4XsOV*), die GPTs (*https://oreil.ly/KLVgX*) und Llamas (*https://oreil.ly/58gxQ*) können ausgesprochen gut für sehr viele Domänen funktionieren, unter anderem Coding, Recht, Wissenschaft, Wirtschaft, Sport und Umweltwissenschaften. Das liegt vor allen an der Aufnahme dieser Domänen in die Trainingsdaten. In Abbildung 2.3 sehen Sie die Verteilung von Domänen in Common Crawl laut einer Analyse der *Washington Post* aus dem Jahr 2023.[3]

2 Es kann an Vorurteilen in Pre-Training-Daten oder in Alignment-Daten liegen. Vielleicht hat OpenAI auch einfach beim Trainieren nicht so viele Daten in der chinesischen Sprache oder bei chinazentrischen Narrativen aufgenommen.

3 »Inside the secret list of websites that make AI like ChatGPT sound smart«, *Washington Post*, 2023, *https://oreil.ly/St1o8*.

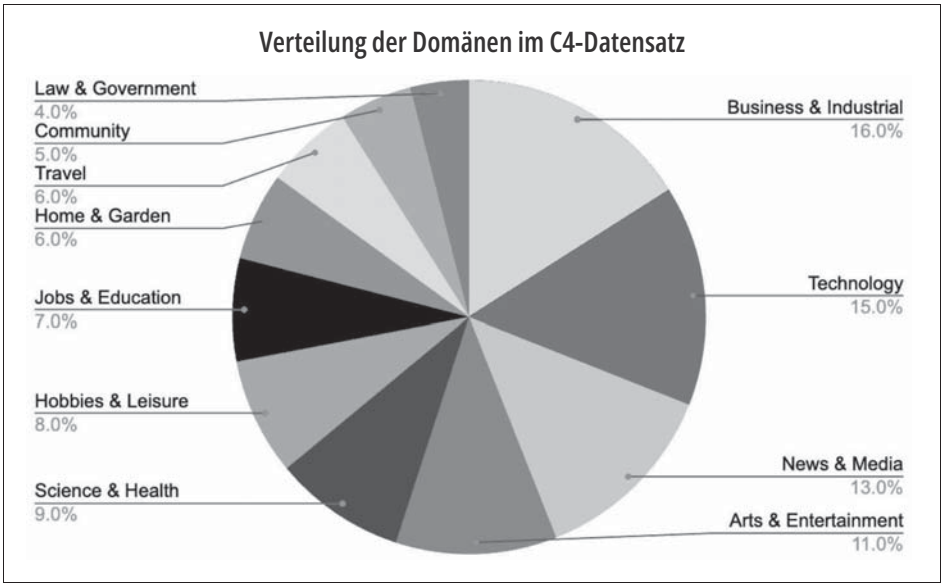

Abbildung 2.3: Verteilung der Domänen im C4-Datensatz, reproduziert aus den Statistiken der »Washington Post«. Ein Problem dieser Analyse ist, dass nur die enthaltenen Kategorien gezeigt werden, nicht die fehlenden.

Aktuell gibt es nicht viele Analysen zur Domänenverteilung bei visuellen Daten. Das kann daran liegen, dass sich Bilder schlechter kategorisieren lassen als Texte.[4] Aber Sie können die Domänen eines Modells aus seiner Benchmark-Performance erschließen. In Tabelle 2.3 sehen Sie zwei Modelle – CLIP und Open CLIP – sowie deren Leistung für verschiedene Benchmarks (*https://oreil.ly/MTqyR*). Diese Benchmarks zeigen, wie gut sich diese beiden Modelle bei Vögeln, Blumen, Autos und ein paar weiteren Kategorien schlagen, aber die Welt ist so viel größer und komplexer als diese wenigen Kategorien.

Tabelle 2.3: Performance von CLIP und Open CLIP für verschiedene Bilddatensätze

Datensatz	CLIP Genauigkeit von ViT-B/32 (OpenAI)	Open CLIP Genauigkeit von ViT-B/32 (Cade)
ImageNet	63,2	62,9
ImageNet v2	–	62,6
Birdsnap	37,8	46,0
Country211	17,8	14,8
Oxford 102 Category Flower	66,7	66,0
German Traffic Sign Recognition Benchmark	32,2	42,0
Stanford Cars	59,4	79,3
UCF101	64,5	63,1

4 Bei Texten können Sie Domänenschlüsselwörter als Heuristiken nehmen, aber es gibt keine offensichtlichen Heuristiken für Bilder. Die meisten Analysen, die ich zu visuellen Datensätzen finden konnte, drehen sich um Bildgrößen, die Auflösung oder die Länge von Videos.

Auch wenn allgemein nutzbare Foundation Models alltägliche Fragen aus unterschiedlichen Domänen beantworten können, ist es unwahrscheinlich, dass sie bei domänenspezifischen Aufgaben gut funktionieren – insbesondere wenn sie diese Aufgaben beim Training nie zu Gesicht bekommen haben. Zwei Beispiele für domänenspezifische Aufgaben sind die Pharmaforschung und das Krebs-Screening. Bei der Pharmaforschung sind Protein-, DNA- und RNA-Daten im Spiel, die in spezifischen Formaten vorhanden sind und sich nur für recht viel Geld beschaffen lassen. Diese Daten finden sich eher nicht in öffentlich verfügbaren Internetdaten. Genauso ist es beim Krebs-Screening, für das im Allgemeinen Röntgen- und fMRI-Scans (*functional Magnetic Resonance Imaging*) genutzt werden, die sich aufgrund des Datenschutzes nur schlecht beschaffen lassen.

Um ein Modell so zu trainieren, dass es gut bei diesen domänenspezifischen Aufgaben funktioniert, müssen Sie eventuell sehr spezifische Datensätze kuratieren. Eines der berühmtesten domänenspezifischen Modelle ist vielleicht AlphaFold von DeepMind (*https://oreil.ly/JX37g*), das mit den Sequenzen und 3-D-Strukturen von ungefähr 100.000 bekannten Proteinen trainiert ist. BioNeMo von NVIDIA (*https://oreil.ly/M1Nsc*) ist ein weiteres Modell, das sich auf biomolekulare Daten für die Pharmaforschung konzentriert. Med-PaLM2 von Google (*https://oreil.ly/F76hq*) kombiniert die Leistungsfähigkeit eines LLM mit medizinischen Daten, um medizinische Fragen mit größerer Genauigkeit beantworten zu können.

Domänenspezifische Modelle sind besonders in der Biomedizin verbreitet, aber auch andere Felder können von solchen Modellen profitieren. Es ist möglich, dass ein Modell auf Architekturzeichnungen trainiert wird und dann Architektinnen und Architekten viel besser helfen kann als Stable Diffusion. Oder ein Modell kann auf Herstellungsprozesse viel besser optimiert werden als ein generisches Modell wie ChatGPT, wenn es mit Werksplänen trainiert wurde.

Dieser Abschnitt hat einen groben Überblick über den Einfluss der Trainingsdaten auf die Leistungsfähigkeit eines Modells gegeben. Als Nächstes wollen wir uns anschauen, welchen Einfluss das Design eines Modells auf seine Performance hat.

Modellieren

Vor dem Trainieren eines Modells müssen die Entwicklerinnen und Entwickler entscheiden, wie das Modell aussehen sollte. Welche Architektur soll es nutzen? Wie viele Parameter soll es haben? Diese Entscheidungen beeinflussen nicht nur die Fähigkeiten des Modells, sondern auch seine Anwendbarkeit auf darauf aufbauende Applikationen.[5] So wird sich beispielsweise ein Modell mit 7 Milliarden Parametern

5 ML-Grundlagen zum Modelltraining gehen über den Rahmen dieses Buchs hinaus. Sind sie hier relevant, werde ich ein paar Konzepte mit aufnehmen. Zum Beispiel ist selbstüberwachtes Lernen – bei dem ein Modell seine eigenen Labels aus den Daten generiert – in Kapitel 1 beschrieben, und Backpropagation – die Modellparameter werden während des Trainings anhand des Fehlers aktualisiert – findet sich in Kapitel 7.

deutlich einfacher deployen lassen als ein Modell mit 175 Milliarden Parametern. Genauso unterscheidet sich das Optimieren eines Transformer-Modells auf Latenz sehr vom Optimieren einer anderen Architektur. Schauen wir uns die Faktoren hinter diesen Entscheidungen an.

Modellarchitektur

Aktuell ist die dominanteste Architektur für sprachbasierte Foundation Models die *Transformer*-Architektur (Vaswani et al., 2017, *https://arxiv.org/abs/1706.03762*), die auf dem Attention-Mechanismus basiert. Dabei werden viele Einschränkungen vorhergehender Architekturen vermieden, was zu ihrer Beliebtheit beiträgt. Die Transformer-Architektur hat dennoch auch eigene Grenzen. Dieser Abschnitt schaut sich die Transformer-Architektur und ihre Alternativen an. Weil es hier um die technischen Details unterschiedlicher Architekturen geht, kann es auch technisch anspruchsvoll werden. Finden Sie, dass etwas für Sie zu weit geht, können Sie manche Bereiche gern überspringen.

Transformer-Architektur

Um Transformer zu verstehen, wollen wir uns das Problem anschauen, das die Architektur lösen sollte. Sie fand Verbreitung unmittelbar nach dem Erfolg der seq2seq-(Sequence-to-Sequence-)Architektur (*https://arxiv.org/abs/1409.3215*). Als sie 2014 vorgestellt wurde, bot seq2seq deutliche Verbesserungen bei damals noch herausfordernden Aufgaben: maschinelles Übersetzen und Zusammenfassungen. 2016 band Google seq2seq in Google Translate ein (*https://oreil.ly/fb1aR*) – ein Update, das laut Aussage des Unternehmens die »bislang größten Verbesserungen bei der Qualität maschineller Übersetzungen« bot. Das sorgte für viel Interesse an seq2seq und machte es zur beliebtesten Architektur für Aufgaben, bei denen Textsequenzen beteiligt waren.

Grob gesagt, enthält seq2seq einen Encoder, der die Eingaben verarbeitet, und einen Decoder, der Ausgaben generiert. Sowohl Eingaben wie auch Ausgaben bestehen aus Token-Sequenzen – daher der Name. seq2seq nutzt RNNs (rekurrente neuronale Netze) als Encoder und Decoder. In seiner einfachsten Form verarbeitet der Encoder die Eingabetokens sequenziell und gibt den abschließenden verborgenen Status aus, der die Eingaben repräsentiert. Der Decoder generiert dann Ausgabetokens sequenziell, die vom abschließenden verborgenen Status der Eingabe und den zuvor generierten Tokens abhängt. Eine Visualisierung der seq2seq-Architektur sehen Sie in der oberen Hälfte von Abbildung 2.4.

Es gibt bei seq2seq zwei Probleme, die Vaswani et al. (2017) angehen. Der einfache seq2seq-Decoder generiert Ausgabetokens nur anhand des abschließenden verborgenen Status der Eingabe. Intuitiv ist das wie das Generieren von Antworten zu einem Buch anhand einer Zusammenfassung. Das begrenzt die Qualität der generierten Ausgaben. Und RNN-Encoder und -Decoder sorgen dafür, dass das Verarbeiten der Eingabe und das Generieren der Ausgabe sequenziell erfolgen, was das

Ganze bei langen Sequenzen verlangsamt. Ist eine Eingabe 200 Tokens lang, muss seq2seq auf den Abschluss der Verarbeitung jedes Eingabetokens warten, bevor es zum nächsten wechseln kann.[6]

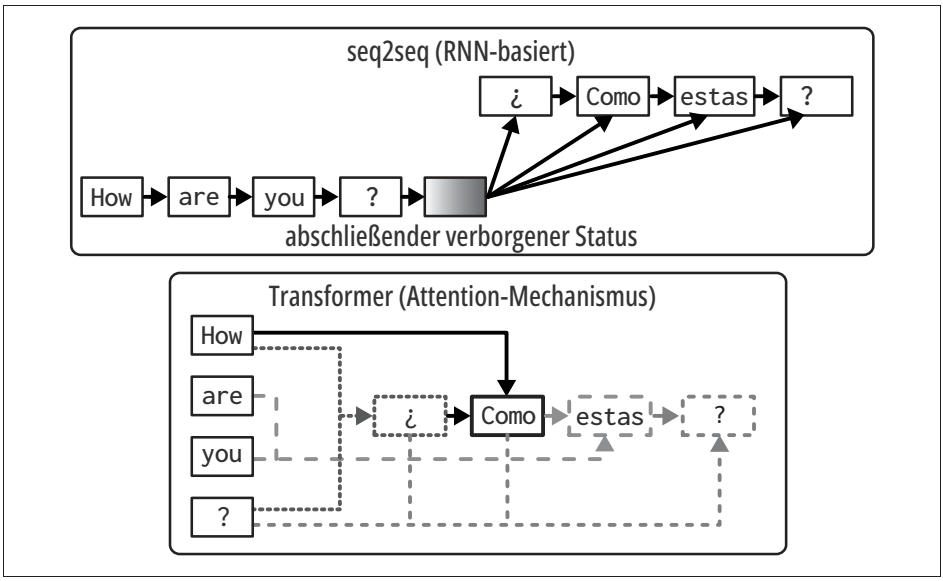

Abbildung 2.4: seq2seq- im Vergleich zur Transformer-Architektur. Bei der Transformer-Architektur zeigen die Pfeile die Tokens, an die der Decoder anfügt, wenn er neue Ausgabetokens generiert.

Die Transformer-Architektur kümmert sich um beide Probleme mithilfe des Attention-Mechanismus. Dieser erlaubt dem Modell, die Wichtigkeit verschiedener Eingabetokens zu gewichten, wenn ein Ausgabetoken generiert wird. Das ist wie das Generieren von Antworten, indem Seiten im Buch referenziert werden. Eine vereinfachte Visualisierung der Transformer-Architektur finden Sie in der unteren Hälfte von Abbildung 2.4.

Auch wenn der Attention-Mechanismus oft mit dem Transformer-Modell verbunden wird, wurde er schon drei Jahre vor dem Transformer-Artikel veröffentlicht. Er lässt sich auch mit anderen Architekturen verwenden. Google hat ihn bei der eigenen seq2seq-Architektur im Jahr 2016 für das GNMT-Modell (*Google Neural Machine Translation*) eingesetzt. Aber erst der Transformer-Artikel hat gezeigt, dass der Attention-Mechanismus auch ohne RNNs genutzt werden kann.[7]

6 RNNs sind aufgrund ihrer rekursiven Struktur besonders anfällig für verschwindende oder explodierende Gradienten. Gradienten müssen durch viele Schritte propagiert werden, und wenn sie klein sind, sorgt eine wiederholte Multiplikation dafür, dass sie gegen null tendieren, was es dem Modell erschwert zu lernen. Sind die Gradienten hingegen groß, wachsen sie mit jedem Schritt exponentiell, und das führt wiederum zu Instabilitäten im Lernprozess.

7 Bahdanau et al., »Neural Machine Translation by Jointly Learning to Align and Translate«, *https://arxiv.org/abs/1409.0473*.

Die Transformer-Architektur verzichtet vollständig auf RNNs. Die Eingabetokens können hier parallel verarbeitet werden, was die Eingabeverarbeitung signifikant beschleunigt. Während der Transformer den Flaschenhals bei der sequenziellen Eingabe beseitigt, gibt es bei auf ihm basierenden autoregressiven Sprachmodellen immer noch den Flaschenhals bei der sequenziellen Ausgabe.

Die Inferenz für Transformer-basierte Sprachmodelle besteht daher aus zwei Schritten:

Prefill
: Das Modell verarbeitet die Eingabetokens parallel. Dieser Schritt erzeugt den Zwischenstatus, der erforderlich ist, um das erste Ausgabetoken zu generieren. Er enthält die Key- und Value-Vektoren für alle Eingabetokens.

Decode
: Das Modell generiert ein Ausgabetoken nach dem anderen.

Wie später in Kapitel 9 gezeigt wird, motivieren die parallelisierbare Natur des Prefillings und der sequenzielle Aspekt des Decodings viele Optimierungstechniken, um Sprachmodelle günstiger und schneller inferieren zu lassen.

Attention-Mechanismus Im Herzen der Transformer-Architektur steckt der Attention-Mechanismus. Es ist wichtig, ihn zu durchdringen, um zu verstehen, wie Transformer-Modelle funktionieren. Unter der Motorhaube nutzt der Attention-Mechanismus Key-, Value- und Query-Vektoren:

- Der *Query-Vektor* (Q) repräsentiert den aktuellen Status des Decoders bei jedem Decoding-Schritt. Um beim Beispiel mit der Buchzusammenfassung zu bleiben: Dieser Query-Vektor kann als Person betrachtet werden, die nach Informationen zum Erstellen einer Zusammenfassung sucht.
- Jeder *Key-Vektor* (K) repräsentiert ein früheres Token. Ist jedes frühere Token eine Seite im Buch, ist jeder Key-Vektor wie die Seitennummer. Beachten Sie, dass vorherige Tokens bei einem gegebenen Decoding-Schritt sowohl Eingabetokens als auch zuvor generierte Tokens enthalten.
- Jeder *Value-Vektor* (V) steht für den aktuellen Wert eines vorherigen Tokens, wie ihn das Modell gelernt hat. Jeder Value-Vektor ist wie der Inhalt der Seite.

Der Attention-Mechanismus berechnet, wie viel Aufmerksamkeit einem Eingabetoken zu geben ist, indem er ein Skalarprodukt (*https://de.wikipedia.org/wiki/Skalarprodukt*) zwischen dem Query-Vektor und dessen Key-Vektor berechnet. Ein hoher Wert bedeutet, dass das Modell mehr Inhalt von dieser Seite (dem Value-Vektor) nutzen wird, wenn es die Zusammenfassung des Buchs generiert. Eine Visualisierung des Attention-Mechanismus mit Key-, Value- und Query-Vektoren sehen Sie in Abbildung 2.5. Dabei sucht der Query-Vektor nach Informationen aus den vorherigen Tokens `How, are, you ?, ¿`, um das nächste Token zu erzeugen.

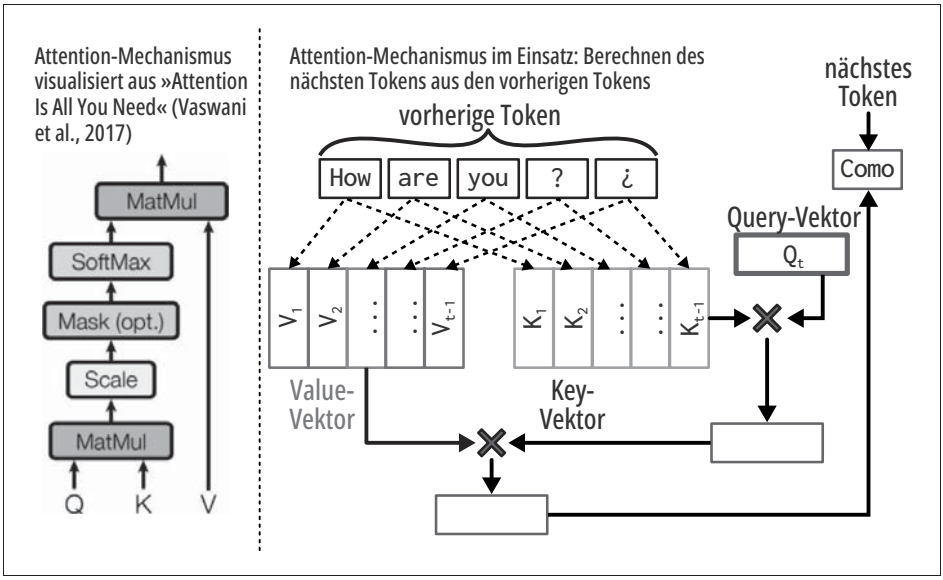

Abbildung 2.5: Ein Beispiel für den Einsatz des Attention-Mechanismus und seine High-Level-Visualisierung aus dem berühmten Transformer-Artikel »Attention Is All You Need« (Vaswani et al., 2017)

Weil jedes vorherige Token einen entsprechenden Key- und Value-Vektor besitzt, müssen mit zunehmender Länge der Sequenz umso mehr Key- und Value-Vektoren berechnet und zwischengespeichert werden. Das ist einer der Gründe, warum es so schwer ist, die Kontextlänge bei Transformer-Modellen zu vergrößern. In den Kapiteln 7 und 9 kommen wir noch mal auf das effiziente Berechnen und Speichern der Key- und Value-Vektoren zurück.

Schauen wir uns an, wie die Attention-Funktion vorgeht. Bei einer gegebenen Eingabe x werden die Key-, Value- und Query-Vektoren berechnet, indem die Key- (W_K), Value- (W_V) und Query-Matrizen (W_Q) auf die Eingabe angewendet werden. Die entsprechenden Vektoren werden dann wie folgt berechnet:

$$K = xW_K$$
$$V = xW_V$$
$$Q = xW_Q$$

Die Query-, Key- und Value-Matrizen besitzen Dimensionen, die denen der verborgenen Dimension des Modells entsprechen. So hat beispielsweise die verborgene Dimensionsgröße von Llama 2-7B den Wert 4096 (Touvron et al., 2023, *https://arxiv.org/abs/2307.09288*), weshalb jede dieser Matrizen die Dimension 4.096 × 4.096 besitzt. Jeder resultierender Vektor K, V und Q hat die Dimension 4096.[8]

8 Weil Eingabetokens als Batches verarbeitet werden, hat der tatsächliche Eingabevektor die Form $N \times T \times 4096$. Dabei ist N die Batchgröße und T die Sequenzlänge. Genauso besitzt jeder resultierende Vektor K, V und Q die Dimension $N \times T \times 4096$.

Der Attention-Mechanismus ist fast immer multi-headed. Das erlaubt dem Modell, simultan auf unterschiedliche Gruppen vorheriger Tokens zuzugreifen. Bei einer Multi-Head-Attention werden die Query-, Key- und Value-Vektoren in kleinere Vektoren aufgeteilt, die jeweils zu einem Attention-Head gehören. Llama 2-7B hat beispielsweise 32 Attention-Heads, weshalb jeder *K*-, *V*- und *Q*-Vektor in 32 Vektoren mit der Dimension 128 aufgeteilt wird (4.096 / 32 = 128).

$$\text{Attention}(Q,\ K,\ V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

Die Ausgaben aller Attention-Heads werden dann aneinandergefügt. Mit einer Ausgabeprojektionsmatrix wird eine weitere Transformation auf diese konkatenierte Ausgabe angewendet, bevor sie an den nächsten Rechenschritt des Modells verfüttert wird. Die Ausgabeprojektionsmatrix besitzt die gleiche Dimension wie die verborgene Dimension des Modells.

Transformer-Block Nachdem Sie nun gesehen haben, wie Attention funktioniert, wollen wir uns anschauen, wie sie in einem Modell zum Einsatz kommt. Eine Transformer-Architektur ist aus mehreren Transformer-Blöcken zusammengesetzt. Der genaue Inhalt des Blocks variiert je nach Modell, aber im Allgemeinen enthält er das Attention-Modul und das MLP-(Multi-Layer-Perceptron-)Modul:

Attention-Modul
: Jedes Attention-Modul besteht aus vier Gewichtsmatrizen: Query-, Key-, Value- und Output-Projektion.

MLP-Modul
: Ein MLP-Modul besteht aus linearen Schichten, die durch *nichtlineare Aktivierungsfunktionen* getrennt sind. Jede lineare Schicht ist eine Gewichtsmatrix, die für lineare Transformationen genutzt wird, während eine Aktivierungsfunktion den linearen Schichten ermöglicht, nichtlineare Muster zu lernen. Eine lineare Schicht wird auch als Feedforward-Schicht bezeichnet.

  Gebräuchliche nichtlineare Funktionen sind ReLU, Rectified Linear Unit (Agarap, 2018, *https://arxiv.org/abs/1803.08375*) und GELU (Hendrycks and Gimpel, 2016, *https://arxiv.org/abs/1606.08415*), die bei GPT-2 bzw. GPT-3 zum Einsatz kamen. Aktivierungsfunktionen sind sehr einfach.[9] So wandelt ReLU beispielsweise einfach negative Werte nach 0 um. Mathematisch wird es wie folgt geschrieben:

  $\text{ReLU}(x) = \max(0, x)$

Die Anzahl an Transformer-Blöcken in einem Transformer-Modell wird auch als die Anzahl der Schichten des Modells bezeichnet. Ein Transformer-basiertes Sprachmodell besitzt zudem ein Modul vor und eines nach all den Transformer-Blöcken:

9 Warum funktionieren einfache Aktivierungsfunktionen für komplexe Modelle wie LLMs? Es gab eine Zeit, in der sich die Forschungsgemeinschaft ein Rennen um möglichst ausgefeilte Aktivierungsfunktionen geliefert hat. Aber es stellte sich heraus, dass extravagante Aktivierungsfunktionen nicht besser funktionieren. Das Modell braucht einfach eine nichtlineare Funktion, um die Linearität zwischen den Feedforward-Schichten aufzubrechen. Einfachere Funktionen sind besser, weil sie sich schneller berechnen lassen, während ausgefeiltere Funktionen zu viel Rechenzeit und Speicher benötigen.

Ein Embedding-Modul vor den Transformer-Blöcken
: Dieses Modul besteht aus der Embedding-Matrix und der Positional-Embedding-Matrix, die Tokens und ihre Positionen in Embedding-Vektoren umwandeln. Grob gesagt, bestimmt die Anzahl an Positionsindizes die maximale Kontextlänge des Modells. Kann ein Modell beispielsweise 2.048 Positionen nachverfolgen, beträgt die maximale Kontextlänge 2.048. Es gibt allerdings Techniken, die die Kontextlänge eines Modells erhöhen, ohne die Anzahl an Positionsindizes zu vergrößern.

Eine Ausgabeschicht nach den Transformer-Blöcken
: Dieses Modul bildet die Ausgabevektoren auf Token-Wahrscheinlichkeiten ab, die dann wiederum für das Sampeln von Modellausgaben genutzt werden (siehe den Abschnitt »Sampling« auf Seite 111). Es besteht meist aus einer Matrix, die auch als *Unembedding-Schicht* bezeichnet wird. Manchmal wird für die Ausgabeschicht auch der Begriff Modell-*Head* genutzt, da es die letzte Schicht des Modells vor der Ausgabegenerierung ist.

In Abbildung 2.6 sehen Sie die Architektur eines Transformer-Modells. Die Größe eines solchen Modells wird von den Dimensionen seiner Bausteine bestimmt. Wichtige Werte sind hier:

- Die Dimension des Modells bestimmt die Größe der Key-, Query-, Value- und Output-Projektionsmatrizen im Transformer-Block.
- Die Anzahl der Transformer-Blöcke.
- Die Dimension der Feedforward-Schicht.
- Die Größe des Vokabulars.

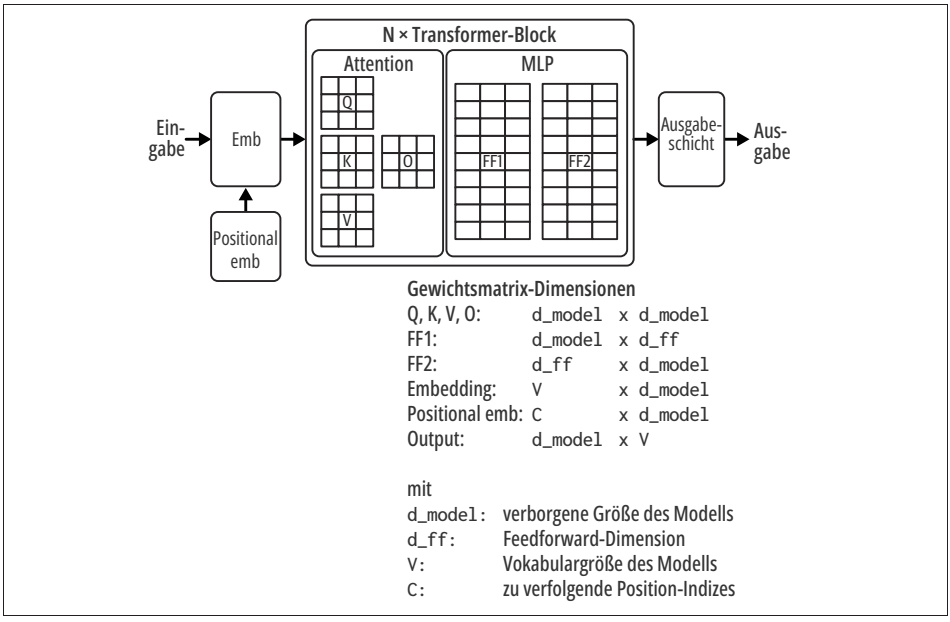

Abbildung 2.6: Eine Visualisierung der Gewichtszusammensetzung eines Transformer-Modells

Größere Dimensionen führen zu größeren Modellen. In Tabelle 2.4 sehen Sie diese Dimensionswerte für verschiedene Llama-2- (Touvron et al., 2023, *https://arxiv.org/abs/2307.09288*) und Llama-3-Modelle (Dubey et al., 2024, *https://arxiv.org/abs/2407.21783*). Beachten Sie, dass die vergrößerte Kontextlänge zwar den Speicherbedarf des Modells erhöht, nicht aber die Gesamtzahl der Modellparameter.

Tabelle 2.4: Die Dimensionswerte verschiedener Llama-Modelle

Modell	Transformer-Blöcke	Modell-dimension	Feedforward-Dimension	Vokabular-größe	Kontextlänge
Llama 2-7B	32	4.096	11.008	32K	4K
Llama 2-13B	40	5.120	13.824	32K	4K
Llama 2-70B	80	8.192	22.016	32K	4K
Llama 3-7B	32	4.096	14.336	128K	128K
Llama 3-70B	80	8.192	28.672	128K	128K
Llama 3-405B	126	16.384	53.248	128K	128K

Andere Modellarchitekturen

Das Transformer-Modell dominiert zwar, es ist aber nicht die einzige Architektur. Seit AlexNet das Interesse am Deep Learning im Jahr 2012 wiederbelebt hat (*https://oreil.ly/1spG5*), sind viele Architekturen gekommen und auch wieder gegangen. seq2seq stand vier Jahre lang im Rampenlicht (2014–2018). GANs (*Generative Adversarial Networks*, *https://arxiv.org/abs/1406.2661*) haben die Gemeinschaft ein bisschen länger beflügelt (2014–2019). Verglichen mit den vergangenen Architekturen hält Transformer schon länger durch, es besteht seit 2017.[10] Wie lange wird es wohl dauern, bis etwas Besseres die Bühne betritt?

Es ist nicht leicht, eine neue Architektur zu entwickeln, die Transformer-Modelle übertrifft.[11] Das Transformer-Modell wurde seit 2017 schon sehr stark optimiert. Eine neue Architektur, die dieses Modell ersetzen will, muss in einer relevanten Größenordnung mit relevanter Hardware funktionieren.[12]

Aber es gibt Hoffnung. Transformer-basierte Modelle dominieren aktuell zwar die Landschaft, aber eine Reihe alternativer Architekturen gewinnen an Zugkraft.

10 Funfact: Ilya Sutskever, ein Mitbegründer von OpenAI, ist der erste Autor des seq2seq-Artikels und zweiter Autor des AlexNet-Artikels.

11 Ilya Sutskever hat einen interessanten Grund dafür, warum es so schwer ist, neue Architekturen für neuronale Netze zu entwickeln, die die bestehenden übertrumpfen. Seiner Meinung nach können neuronale Netze viele Computerprogramme sehr gut simulieren. Das Gradientenverfahren, eine Technik zum Trainieren neuronaler Netze, ist eigentlich ein Suchalgorithmus, um alle Programme zu durchsuchen, die ein neuronales Netz simulieren kann, und das Beste für die angestrebte Aufgabe zu finden. Neue Architekturen können also potenziell auch durch bestehende simuliert werden. Damit neue Architekturen bestehende übertreffen, müssen diese dazu in der Lage sein, Programme zu simulieren, die von den bestehenden Architekturen nicht simuliert werden können. Mehr Informationen finden Sie in einem Vortrag von Sutskever am Simons Institute in Berkeley (2023, *https://oreil.ly/j4wwW*).

12 Das Transformer-Modell wurde von Google ursprünglich dafür designt, schnell auf *Tensor Processing Units* (TPUs) zu laufen (*https://oreil.ly/ON55d*), und erst später für GPUs optimiert.

Ein beliebtes Modell ist RWKV (Peng et al., 2023, *https://github.com/BlinkDL/RWKV-LM*), ein RNN-basiertes Modell, das zum Training parallelisiert werden kann. Aufgrund seiner RNN-Natur muss es theoretisch nicht die gleichen Beschränkungen in der Kontextlänge aufweisen wie Transformer-basierte Modelle. In der Praxis garantiert eine fehlende Kontextlängenbeschränkung allerdings bei einem langen Kontext nicht unbedingt eine gute Leistung.

Das Modellieren langer Sequenzen bleibt eine zentrale Herausforderung beim Entwickeln von LLMs. Eine Architektur, die für Langzeitspeicher vielversprechend ist, nennt sich SSM (*State Space Models*, Gu et al., 2021a, *https://arxiv.org/abs/2110.13985*). Seit der Vorstellung der Architektur im Jahr 2021 wurde eine Reihe von Techniken eingeführt, um sie effizienter, besser bei der Verarbeitung langer Sequenzen und skalierbarer für umfangreichere Modellgrößen zu machen. Hier ein paar der Techniken, um die Evolution einer neuen Architektur zu illustrieren:

- *S4*, eingeführt in »Efficiently Modeling Long Sequences with Structured State Spaces« (Gu et al., 2021b, *https://arxiv.org/abs/2111.00396*), wurde entwickelt, um SSMs effizienter zu machen.
- *H3*, eingeführt in »Hungry Hungry Hippos: Towards Language Modeling with State Space Models (Fu et al., 2022, *https://arxiv.org/abs/2212.14052*), nutzt einen Mechanismus, der dem Modell ermöglicht, sich an frühere Tokens zu erinnern und Tokens über Sequenzen hinweg zu vergleichen. Der Zweck dieses Mechanismus ähnelt dem des Attention-Mechanismus in der Transformer-Architektur, aber er ist effizienter.
- *Mamba*, das in »Mamba: Linear-Time Sequence Modeling with Selective State Spaces« (Gu und Dao, 2023, *https://oreil.ly/n7wYO*) vorgestellt wurde, skaliert SSMs bis zu drei Milliarden Parametern. Bei Sprachmodellen übertrifft Mamba-3B Transformer der gleichen Größe, und es spielt in einer Liga mit Transformern, die doppelt so groß sind. Die Autoren zeigen auch, dass die Inferenzberechnungen bei Mamba linear mit der Sequenzlänge skalieren (verglichen mit dem quadratischen Skalieren bei Transformern). Die Leistung zeigt Verbesserungen bei realen Daten bis zu Sequenzen mit einer Länge im Millionen-Bereich.
- *Jamba*, vorgestellt in »Jamba: A Hybrid Transformer–Mamba Language Model« (Lieber et al., 2024, *https://arxiv.org/abs/2403.19887*), verwebt Blöcke aus Transformer- und Mamba-Schichten, um SSMs noch weiter zu skalieren. Die Autoren haben ein Mixture-of-Experts-Modell mit insgesamt 52 Milliarden verfügbaren Parametern (12 Milliarden aktiven Parametern) veröffentlicht (*https://oreil.ly/uyiBH*), das so entworfen ist, dass es in eine einzelne 80-GB-GPU passt. Jamba zeigt eine starke Leistung bei Standard-Sprachmodell-Benchmarks und Evaluierungen bei langen Kontexten bis zu einer Kontextlänge von 256K Tokens. Außerdem besitzt es verglichen mit klassischen Transformern einen kleinen Speicherfußabdruck.

Abbildung 2.7 zeigt die Transformer-, Mamba- und Jamba-Blöcke.

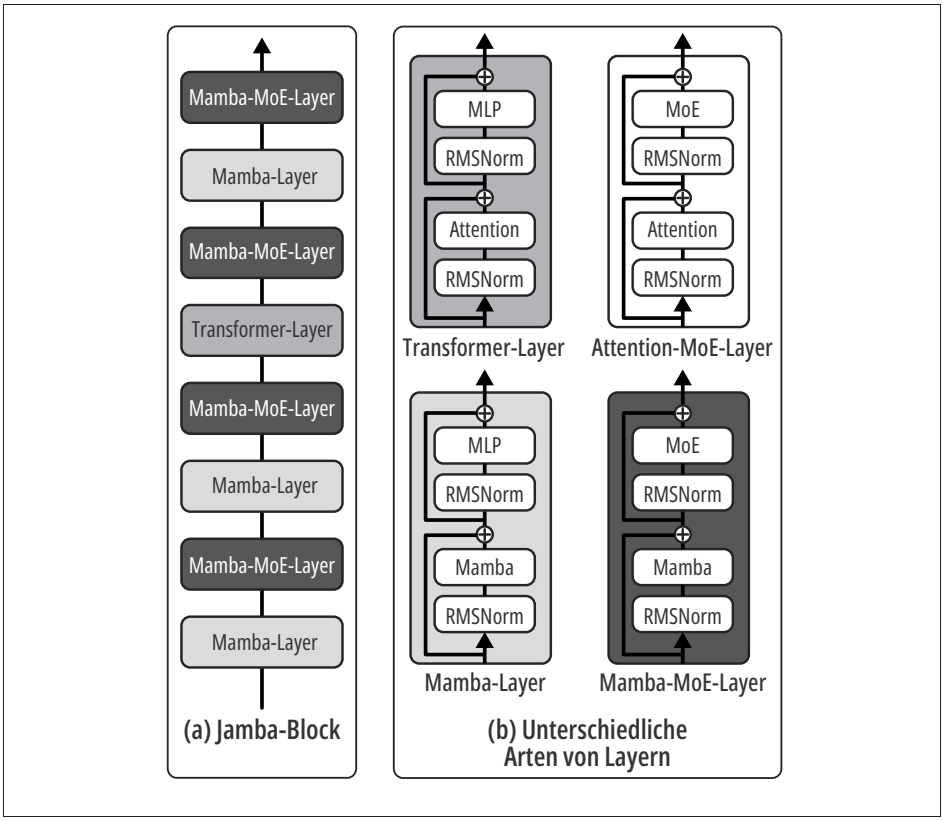

Abbildung 2.7: Eine Visualisierung der Transformer-, Mamba- und Jamba-Schichten. Bild (angepasst) aus »Jamba: A Hybrid Transformer–Mamba Language Model« (Lieber et al., 2024).

Es ist zwar herausfordernd, eine Architektur zu entwickeln, die die Transformer übertrifft, aber angesichts der vielen Einschränkungen von Transformer-Modellen ist es durchaus lohnend, es zu versuchen. Wenn eine andere Architektur die Transformer überholen wird, werden sich einige der Techniken zum Anpassen von Modellen, die in diesem Buch vorgestellt werden, ändern. Aber so wie beim Wechsel von ML Engineering zu AI Engineering viele Dinge unverändert geblieben sind, so wird auch eine Änderung der zugrunde liegenden Modellarchitektur diese grundsätzliche Vorgehensweise nicht aus dem Spiel werfen.

Modellgröße

Ein Großteil der KI-Fortschritte der letzten Jahre lässt sich auf wachsende Modellgrößen zurückführen. Es ist schwer, über Foundation Models zu sprechen, ohne über die Zahl ihrer Parameter zu reden. Die Anzahl der Parameter wird meist am Ende eines Modellnamens angefügt. So bezieht sich beispielsweise Llama-13B auf die Version von Llama – eine Modellfamilie, die von Meta entwickelt wird – mit 13 Milliarden (amerikanisch »Billion«) Parametern.

Im Allgemeinen verbessert ein Erhöhen der Modellparameter die Fähigkeiten eines Modells zum Lernen, was zu besseren Modellen führt. Bei zwei Modellen der gleichen Modellfamilie wird das mit 13 Milliarden Parametern sehr wahrscheinlich mehr Leistung liefern als das mit 7 Milliarden Parametern.

Da die Community immer besser versteht, wie große Modelle zu trainieren sind, tendieren Modelle der neueren Generation dazu, Modelle älterer Generationen gleicher Größe zu schlagen. So ist beispielsweise Llama 3-8B (2024, *https://arxiv.org/abs/2407.21783*) bei der MMLU-Benchmark sogar besser als Llama 2-70B (2023, *https://arxiv.org/abs/2307.09288*).

Die Anzahl der Parameter hilft uns dabei, die Rechenressourcen abzuschätzen, die erforderlich sind, um dieses Modell zu trainieren und auszuführen. Hat ein Modell beispielsweise 7 Milliarden Parameter und wird jeder Parameter mit 2 Byte (16 Bit) abgespeichert, können wir ausrechnen, dass der für das Inferieren mit diesem Modell erforderliche GPU-Speicher mindestens 14 Milliarden Byte (14 GB) groß sein muss.[13]

Die Anzahl der Parameter kann irreführend sein, wenn das Modell *sparse* (dünn besetzt) ist. Ein Sparse Model hat einen großen Prozentsatz an Parametern mit dem Wert null. Ein 7B-Parameter-Modell, das zu 90 % sparse ist, hat nur 700 Millionen Parameter ungleich null. Sparsity ermöglicht ein effizienteres Ablegen von und Rechnen mit Daten. Daher kann ein großes Sparse Model weniger Rechenbedarf haben als ein kleines (dichtes) Dense Model.

Eine Form von Sparse Models, die in den letzten Jahren an Beliebtheit gewonnen hat, sind die Mixture-of-Experts-(MoE-)Modelle (Shazeer et al., 2017, *https://arxiv.org/abs/1701.06538*). Ein MoE-Modell ist in verschiedene Gruppen von Parametern unterteilt, und jede Gruppe ist ein *Experte*. Nur eine Untermenge der Experten ist für das Verarbeiten jedes Tokens *aktiv*.

So handelt es sich beispielsweise bei Mixtral 8x7B um eine »Mischung« aus acht Experten mit jeweils sieben Milliarden Parametern (*https://oreil.ly/VvXbu*). Teilen sich keine zwei Experten einen Parameter, sollte es 8 × 7 Milliarden = 56 Milliarden Parameter besitzen. Weil aber manche Parameter gemeinsam genutzt werden, besitzt es nur 46,7 Milliarden Parameter.

In jeder Schicht sind für jedes Token nur zwei Experten aktiv – somit sind es also nur 12,9 Milliarden Parameter. Auch wenn das Modell 46,7 Milliarden Parameter besitzt, entsprechen seine Kosten und seine Geschwindigkeit aber denen eines mit 12,9 Milliarden Parametern.

Ein größeres Modell kann auch schlechter performen als ein kleineres, wenn es nicht mit ausreichend Daten trainiert wurde. Stellen Sie sich ein 13B-Parameter-Modell vor, das mit einem Datensatz mit nur einem einzigen Satz trainiert wurde: »Ich mag

13 Der tatsächlich erforderliche Speicher ist größer. In Kapitel 7 kümmern wir uns darum, wie man den Speicherbedarf eines Modells errechnet.

Ananas.« Dieses Modell wird sich viel schlechter schlagen als ein kleineres Modell, das mit mehr Daten trainiert wurde.

Bei der Modellgröße ist es wichtig, auch die Menge der Daten zu berücksichtigen, mit denen ein Modell trainiert wurde. Bei den meisten Modellen wird die Datensatzgröße über die Anzahl der Trainings-Samples gemessen. So wurde Googles Flamingo (Alayrac et al., 2022, *https://arxiv.org/abs/2204.14198*) beispielsweise mit vier Datensätzen trainiert – einer davon hat 1,8 Milliarden (Bild, Text)-Paare, einer 312 Millionen (Bild, Text)-Paare.

Bei Sprachmodellen kann ein Trainings-Sample ein Satz, eine Wikipedia-Seite, ein Chatdialog oder ein Buch sein. Ein Buch ist viel mehr wert als ein Satz, daher ist die Anzahl an Trainings-Samples nicht mehr länger eine gute Metrik, um die Datensatzgrößen zu messen. Besser ist die Anzahl an Tokens im Datensatz.

Die Anzahl der Tokens ist allerdings auch kein perfekter Messwert, da unterschiedliche Modelle unterschiedliche Tokenisierungsprozesse haben können, was dazu führt, dass der gleiche Datensatz für verschiedene Modelle eine unterschiedliche Zahl an Tokens haben kann. Warum nutzen wir nicht einfach die Anzahl an Wörtern oder die Menge an Buchstaben? Weil ein Token die Einheit ist, auf der ein Modell operiert, hilft uns das Wissen über die Anzahl an Tokens in einem Datensatz dabei, zu messen, wie viel ein Modell potenziell von diesen Daten lernen kann.

Aktuell werden LLMs mit Datasets in der Größenordnung von Billionen Tokens trainiert. Meta nutzt zunehmend größere Datensätze, um ihre Llama-Modelle zu trainieren:

- 1,4 Billionen Tokens für Llama 1 (*https://arxiv.org/abs/2302.13971*)
- 2 Billionen Tokens für Llama 2 (*https://arxiv.org/abs/2307.09288*)
- 15 Billionen Tokens für Llama 3 (*https://oreil.ly/vfSQw*)

Der Open-Source-Datensatz RedPajama-v2 von Together besitzt 30 Billionen Tokens (*https://oreil.ly/SfB4g*). Das ist das Äquivalent zu 450 Millionen Büchern[14] oder 5.400 Mal der Größe von Wikipedia. Da RedPajama-v2 aber wahllos Inhalte enthält, ist die Menge an qualitativ hochwertigen Daten viel geringer.

Die Anzahl der Tokens im Datensatz eines Modells entspricht nicht der Anzahl der Trainingstokens.

Die Menge an Trainingstokens ergibt sich aus den Tokens, mit denen das Modell trainiert wird. Enthält ein Datensatz 1 Billion Tokens und wird ein Modell über zwei Epochen mit diesem Datensatz trainiert – eine *Epoche* ist ein Durchlauf durch den Datensatz –, beträgt die Anzahl an Trainingstokens 2 Billionen.[15] In Tabelle 2.5 sehen Sie Beispiele für die Anzahl an Trainingstokens für Modelle mit einer unterschiedlichen Zahl an Parametern.

14 Bei angenommenen rund 50.000 Wörtern oder 67.000 Tokens pro Buch.

15 Aktuell werden große Modelle typischerweise nur über eine Epoche mit Daten vortrainiert.

Tabelle 2.5: Beispiele für die Anzahl an Trainingstokens für Modelle mit einer unterschiedlichen Zahl an Parametern (Quelle: »Training Compute-Optimal Large Language Models«, DeepMind, 2022, https://oreil.ly/A3K90).

Modell	Größe (Anzahl Parameter)	Trainingstokens
LaMDA (Thoppilan et al., 2022)	137 Milliarden	168 Milliarden
GPT-3 (Brown et al., 2020)	175 Milliarden	300 Milliarden
Jurassic (Lieber et al., 2021)	178 Milliarden	300 Milliarden
Gopher (Rae et al., 2021)	280 Milliarden	300 Milliarden
MT-NLG 530B (Smith et al., 2022)	530 Milliarden	270 Milliarden
Chinchilla	70 Milliarden	1,4 Billionen

Auch wenn sich dieser Abschnitt auf die Menge an Daten fokussiert, ist Quantität nicht das Einzige, was zählt. Die Qualität und die Diversität der Daten ist ebenfalls wichtig. Quantität, Qualität und Diversität sind die drei goldenen Ziele bei Trainingsdaten. Sie werden genauer in Kapitel 8 besprochen.

Das Pre-Training großer Modelle erfordert Rechenaufwand. Ein Weg, diesen zu messen, besteht darin, die Anzahl der »Maschinen«, also zum Beispiel GPUs, CPUs und TPUs, zu betrachten. Allerdings besitzen die verschiedenen Komponenten sehr unterschiedliche Möglichkeiten und Kosten. Eine NVIDIA-A10-GPU unterscheidet sich von einer NVIDIA-N100-GPU und von einem Intel-Core-Ultra-Prozessor.

Eine standardisiertere Einheit für die Rechenanforderungen eines Modells ist ein *FLOP*, eine *Floating Point Operation*. FLOP misst die Anzahl der Gleitkommaoperationen, die für eine bestimmte Aufgabe ausgeführt werden. Googles größtes PaLM-2-Modell wurde beispielsweise mit 10^{22} FLOPs trainiert (Chowdhery et al., 2022, *https://arxiv.org/abs/2204.02311*), GPT-3-175B erhielt $3{,}14 \times 10^{23}$ FLOPs (Brown et al., 2020, *https://arxiv.org/abs/2005.14165*).

Der Plural von FLOP – also FLOPs – wird oft mit FLOP/s verwechselt, den Gleitkommaoperationen pro Sekunde.

FLOPs messen die Rechenanforderungen für eine Aufgabe, während FLOP/s die maximale Leistung einer Recheneinheit angibt. So kann beispielsweise eine NVIDIA-H100-NVL-GPU maximal 60 TeraFLOP/s leisten: 6×10^{13} FLOPs pro Sekunde oder $5{,}2 \times 10^{18}$ FLOPS pro Tag (*https://oreil.ly/HcFYz*).[16]

Achten Sie auf die verwirrenden Bezeichnungen. *FLOP/s* wird oft als *FLOPS* geschrieben, was *FLOPs* stark ähnelt. Um das zu vermeiden, nutzen manche Firmen – unter anderem OpenAI – FLOP/s-day statt FLOPs, um Rechenanforderungen zu messen:

1 FLOP/s-day = 60 × 60 × 24 = 86.400 FLOPs

Dieses Buch nutzt FLOPs zum Zählen von Gleitkommaoperationen und FLOP/s für FLOPs pro Sekunde.

16 FLOP/s werden in FP32 gemessen. Auf Gleitkommaformate gehen wir in Kapitel 7 ein.

Angenommen, Sie haben 256 H100-GPUs. Wenn Sie diese mit ihrer maximalen Kapazität nutzen können und keine Trainingsfehler machen, würden Sie damit $(3{,}14 \times 10^{23}) / (256 \times 5{,}2 \times 10^{18}) \approx 236$ Tage oder ungefähr 7,8 Monate benötigen, um GPT-3-175B zu trainieren.

Es ist allerdings unwahrscheinlich, dass Sie Ihre Maschinen durchgehend mit dieser maximalen Leistung einsetzen können. Die Auslastung misst, wie viel von der maximalen Rechenkapazität Sie nutzen können. Es hängt vom Modell, der Workload und der Hardware ab, was als gute Auslastung angesehen wird. Allgemein gilt: Wenn Sie die Hälfte der beworbenen Leistung erhalten – 50 % Auslastung –, ist das in Ordnung. Alles über 70 % wird als großartig angesehen. Lassen Sie sich von dieser Regel aber nicht abhalten, noch höhere Auslastungen anstreben zu wollen. In Kapitel 9 sprechen wir detaillierter über Hardwaremetriken und die Auslastung.

Bei 70 % Auslastung und 2 US-Dollar pro Stunde für eine H100[17] wird Sie das Trainieren von GPT-3-175B über 4 Millionen US-Dollar kosten:

$2/H100/h × 256 H100 × 24 Stunden × 256 Tag / 0,7 = $4.142.811,43

Zusammengefasst, stehen drei Zahlen für die Größenordnung eines Modells:

- Anzahl der Parameter – stellvertretend für die Lernfähigkeit eines Modells.
- Anzahl der Tokens, mit denen ein Modell trainiert wurde – stellvertretend dafür, wie viel ein Modell gelernt hat.
- Anzahl der FLOPs – stellvertretend für die Trainingskosten.

Inverses Skalieren

Wir haben angenommen, dass größere Modelle auch besser sind. Gibt es Szenarien, in denen größere Modelle schlechter abschneiden? Im Jahr 2022 hat Anthropic herausgefunden, dass entgegen der Intuition mehr Alignment-Training (siehe den Abschnitt »Post-Training« auf Seite 100) zu Modellen führt, die nicht mehr so gut auf die menschlichen Vorlieben abgestimmt sind (Perez et al., 2022, *https://arxiv.org/abs/2212.09251*). Laut ihrem Artikel neigen Modelle, die darauf trainiert sind, besser abgestimmt zu sein, » eher dazu, bestimmte politische Ansichten (Befürwortung von Waffenbesitz und Einwanderung) und religiöse Anschauungen (Buddhismus) zu vertreten, eine selbst ausgedrückte bewusste Erfahrung und moralisches Selbstwertgefühl zu zeigen sowie den Wunsch zu äußern, nicht abgeschaltet zu werden.«

Im Jahr 2023 hat eine Gruppe von Forscherinnen und Forschern – größtenteils von der New York University – den Inverse Scaling Prize ausgerufen (*https://arxiv.org/abs/2306.09479*), um Aufgaben zu finden, bei denen größere Sprachmodelle schlechter abschneiden. Sie haben 5.000 US-Dollar für jeden dritten Platz, 20.000 US-Dollar für jeden zweiten Platz und 100.000 US-Dollar für einen ersten Platz ausgelobt. Es gab insgesamt 99 Einreichungen, von denen elf einen der dritten Plätze erreicht haben. Dabei wurde

17 Aktuell bieten Cloud-Provider eine H100 für 2 bis 5 US-Dollar pro Stunde an. Da Rechenleistung schnell günstiger wird, wird dieser Preis noch deutlich sinken.

festgestellt, dass größere Sprachmodelle manchmal (wirklich nur manchmal) bei Aufgaben schlechter sind, für die eine Erinnerung erforderlich ist, und bei solchen mit starken Vorbedingungen. Es wurden aber weder zweite noch erste Preise vergeben, denn auch wenn die eingereichten Aufgaben Probleme bei einem kleinen Testdatensatz gezeigt hatten, konnten keine Probleme in der realen Welt nachgewiesen werden.

Skalierungsgesetz: rechenoptimale Modelle bauen

Ich hoffe, dass Sie durch den letzten Abschnitt von folgenden drei Dingen überzeugt wurden:

1. Die Modellperformance hängt von der Größe des Modells und des Datensatzes ab.
2. Größere Modelle und größere Datensätze erfordern mehr Rechenaufwand.
3. Rechnen kostet Geld.

Sofern Sie nicht unbegrenzte Geldressourcen besitzen, ist ein Budgetieren essenziell. Sie wollen nicht mit einer beliebig großen Modellgröße beginnen und dann mal schauen, was das kostet. Sie beginnen mit einem Budget – wie viel Geld Sie ausgeben wollen – und ermitteln dann die beste Modellperformance, die Sie sich leisten können. Da Rechnen oft der begrenzende Faktor ist – Recheninfrastruktur ist nicht nur teuer, sondern auch aufwendiger aufzusetzen –, beginnen Teams oft mit einem Rechenbudget. Welche Modellgröße und welche Datensatzgröße liefert uns die beste Leistung, wenn wir eine bestimmte, feste Menge an FLOPs zur Verfügung haben? Ein Modell, das die beste Performance bei einem festen Rechenbudget liefert, ist *rechenoptimal*.

Bei einem gegebenen Rechenbudget gibt es eine Regel, die dabei hilft, die optimale Modell- und Datensatzgröße zu bestimmen – das Chinchilla-*Skalierungsgesetz*, das im Chinchilla-Artikel »Training Compute-Optimal Large Language Models« (DeepMind, 2022, *https://arxiv.org/abs/2203.15556*) vorgeschlagen wurde. Um die Beziehung zwischen Modellgröße, Datensatzgröße, Rechenbudget und Modellperformance zu untersuchen, haben die Autoren 400 Sprachmodelle von 70 Millionen bis über 16 Milliarden Parameter mit 5 bis 500 Milliarden Tokens trainiert. Sie haben herausgefunden, dass für ein rechenoptimales Training die Anzahl an Trainingstokens ungefähr 20 Mal der Größe des Modells entsprechen muss. Ein 3B-Parameter-Modell benötigt also ungefähr 60B an Trainingstokens. Die Modellgröße und die Anzahl an Trainingstokens sollten analog skaliert werden: Für jedes Verdoppeln der Modellgröße sollte auch die Anzahl an Trainingstokens verdoppelt werden.

Früher wurde der Trainingsprozess wie Alchemie behandelt – da sind wir mittlerweile ein ganzes Stück weiter. In Abbildung 2.8 sehen Sie, dass wir nicht nur die optimale Zahl an Parametern und Tokens für jedes FLOP-Budget vorhersagen können, sondern auch den zu erwartenden Trainingsverlust für diese Settings (sofern wir richtig vorgehen).

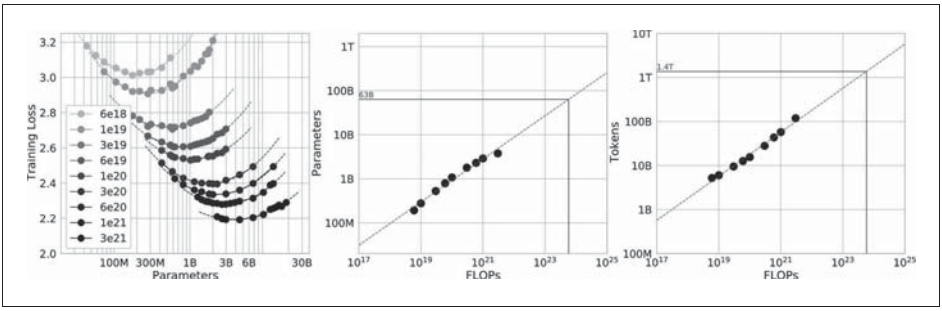

Abbildung 2.8: Diagramm, das die Beziehungen zwischen Trainingsverlust, der Anzahl der Modellparameter, FLOPs und der Anzahl der Trainingstokens aufzeigt (Quelle: »Training Compute-Optimal Large Language Models«, DeepMind, 2022)

Das Skalierungsgesetz wurde für Dense-Modelle entwickelt, die mit vorwiegend von Menschen generierten Daten trainiert wurden. Ein Anpassen dieser Berechnung für Sparse-Modelle, wie zum Beispiel Mixture-of-Experts-Modelle, und synthetische Daten ist Gegenstand aktueller Forschung.

Dieses Gesetz optimiert die Modellqualität für ein gegebenes Rechenbudget. Aber es ist wichtig, daran zu denken, dass die Modellqualität für den produktiven Einsatz nicht alles ist. Manche Modelle – das bekannteste darunter ist Llama – haben eine suboptimale Performance, sind dafür aber besser nutzbar. Mit ihrem Rechenbudget hätten die Llama-Autoren größere Modelle wählen können, die auch mehr Leistung gezeigt hätten, aber sie entschieden sich für kleinere Modelle. Mit diesen kann man einfacher arbeiten, und das Inferieren ist günstiger, was wiederum dabei geholfen hat, dass die Modelle eine weitere Verbreitung gefunden haben. Sardana et al. (2023, *https://arxiv.org/abs/2401.00448*) haben das Chinchilla-Skalierungsgesetz so angepasst, dass die optimale Menge an LLM-Parametern und die beste Datengröße für das Pre-Training bestimmt werden können, um diese Inferenzanforderungen zu erfüllen.

Bezüglich der Modellleistung bei gegebenem Rechenbudget sei noch darauf hingewiesen, dass die Kosten für das Erreichen einer gegebenen Performance abnehmen. So haben sich beispielsweise beim ImageNet-Datensatz die Kosten für das Erreichen von 93 % Genauigkeit laut *Artificial Intelligence Index Report 2022* zwischen 2019 und 2021 halbiert (Stanford University HAI, *https://oreil.ly/oq-LE*).

Auch wenn die Kosten für die gleiche Modellperformance abnehmen, bleiben die für das Verbessern der Performance hoch.

Ähnlich der Diskussion über die letzte Meile in Kapitel 1 ist es teurer, die Genauigkeit eines Modells von 90 % auf 95 % zu bringen als von 85 % auf 90 %. Der Artikel »Beyond Neural Scaling Laws: Beating Power Law Scaling via Data Pruning« von Meta (*https://oreil.ly/kO41d*) beschreibt, dass ein Modell mit einer 2%igen Fehlerrate eine Größenordnung mehr Daten, Rechenaufwand oder Energie erfordern kann als eines mit einer Fehlerrate von 3 %.

Bei der Sprachmodellierung erfordert ein Reduzieren des Kreuzentropieverlusts von ungefähr 3,4 auf 2,8 Nit das Zehnfache an Trainingsdaten. Kreuzentropie und ihre Einheiten – einschließlich Nit – werden in Kapitel 3 behandelt. Für große Vision-Modelle führt das Erhöhen der Training-Samples von 1 Milliarde auf 2 Milliarden für ImageNet zu einem Genauigkeitsgewinn von nur wenigen Prozentpunkten.

Aber kleine Performanceänderungen bei Sprachmodellfehlern oder der ImageNet-Genauigkeit können trotzdem zu großen Unterschieden in der Qualität darauf aufbauender Anwendungen führen. Wechseln Sie von einem Modell mit einem Kreuzentropieverlust von 3,4 auf eines mit 2,8, werden Sie das bemerken.

Scaling Extrapolation

Die Leistung eines Modells hängt stark von den Werten seiner *Hyperparameter* ab. Bei der Arbeit mit kleinen Modellen ist es üblich, ein Modell mehrfach zu trainieren und dabei unterschiedliche Sätze an Hyperparametern zu verwenden, um danach das mit der besten Leistung auszuwählen. Bei großen Modellen ist das aber nur selten möglich, da schon ein einmaliges Training enorme Ressourcen verbraucht.

> **Parameter versus Hyperparameter**
>
> Ein Parameter kann vom Modell während des Trainingsprozesses angelernt werden. Ein Hyperparameter wird von den Usern gesetzt, um das Modell zu konfigurieren und zu steuern, wie es lernt. Zu den Hyperparametern zum Konfigurieren des Modells gehören die Anzahl an Schichten, die Modelldimension und die Größe des Vokabulars. Hyperparameter zum Steuern des Lernens sind zum Beispiel die Batchgröße, die Anzahl an Epochen, die Lernrate oder die initiale Varianz pro Schicht.

Bei vielen Modellen haben Sie also nur genau eine Möglichkeit, den richtigen Satz an Hyperparametern auszuwählen. Aus diesem Grund hat sich *Scaling Extrapolation* (oder auch *Hyperparameter Transferring*) als eigener kleiner Forschungsbereich entwickelt, der versucht, für große Modelle vorherzusagen, welche Hyperparameter die beste Leistung bringen werden. Aktuell wird dabei der Einfluss von Hyperparametern auf Modelle unterschiedlicher Größe – meist viel kleiner als die gewünschte Modellgröße – untersucht und dann extrapoliert, wie diese Hyperparameter bei der anvisierten Größe funktionieren werden.[18] Ein Artikel aus dem Jahr 2022 von Microsoft und OpenAI zeigt, dass es möglich war, Hyperparameter von einem 40M-Modell auf ein 6.7B-Modell zu übertragen (*https://oreil.ly/sHwbw*).

Scaling Extrapolation ist immer noch ein Nischenthema, da nur wenige Personen die Erfahrung und die Ressourcen besitzen, das Training großer Modelle zu untersuchen. Aufgrund der schieren Anzahl an Hyperparametern und ihrer Interaktion un-

18 Jascha Sohl-Dickstein, ein fantastischer Forscher, hat auf X unter *https://x.com/jaschasd/status/1756930242965606582* eine wunderbare Darstellung veröffentlicht, die zeigt, wie Hyperparameter arbeiten – und wie nicht.

tereinander ist es zudem auch nicht einfach. Haben Sie zehn Hyperparameter, müssten Sie 1.024 Kombinationen daraus untersuchen. Sie müssten jeden Hyperparameter einzeln begutachten, dann zwei von ihnen gemeinsam, dann drei und so weiter.

Darüber hinaus machen emergierende Fähigkeiten (Wie et al., 2022, *https://arxiv.org/abs/2206.07682*) das Extrapolieren weniger exakt. Dabei handelt es sich um solche Fähigkeiten, die erst im großen Maßstab vorhanden sind und sich nicht bei kleineren Modellen beobachten lassen, die mit kleineren Datensätzen trainiert wurden. Um mehr über Scaling Extrapolation zu erfahren, lesen Sie sich den ausgezeichneten Blogpost »On the Difficulty of Extrapolation with NN Scaling durch (Luke Metz, 2022, *https://oreil.ly/kuG3J*).

Skalierungsengpässe

Bisher hat jede Erhöhung der Modellgröße um eine Größenordnung auch zu einem Zuwachs bei der Performance des Modells geführt. GPT-2 hatte eine Größenordnung mehr Parameter als GPT-1 (1,5 Milliarden versus 117 Millionen). GPT-3 hatte sogar zwei Größenordnungen mehr Parameter als GPT-2 (175 Milliarden versus 1,5 Milliarden). Das bedeutet ein Wachstum bei den Modellgrößen um drei Größenordnungen zwischen 2018 und 2021. Drei weitere Größenordnungen würden zu Modellen mit 100 Billionen Parametern führen.[19]

Um wie viel mehr Größenordnungen können Modellgrößen wachsen? Gäbe es einen Punkt, an dem die Performance der Modelle ein Plateau erreichen würde – unabhängig von ihrer Größe? Es ist schwer, diese Fragen zu beantworten, aber es gibt beim Skalieren schon zwei erkennbare Flaschenhälse: Trainingsdaten und Strom.

Foundation Models benötigen so viele Daten, dass es durchaus berechtigte Sorgen gibt, in den nächsten paar Jahren nicht mehr ausreichend Internetdaten zu haben. Die Geschwindigkeit, mit der die Größe der Trainingsdatensätze wächst, ist viel schneller als die Geschwindigkeit, mit der neue Daten erzeugt werden (Villalobos et al., 2022, *https://arxiv.org/abs/2211.04325*), wie Abbildung 2.9 zeigt. Haben Sie je etwas ins Internet gebracht, sollten Sie davon ausgehen, dass es schon in den Trainingsdaten irgendeines Sprachmodells enthalten ist oder aufgenommen werden wird – egal ob Sie dem zustimmen oder nicht. Das ist vergleichbar damit, davon auszugehen, dass etwas von Google indexiert wird, wenn Sie es im Internet posten.

Manche nutzen diese Tatsache aus und injizieren Daten, die sie in den Trainingsdaten zukünftiger Modelle sehen wollen. Dazu veröffentlichen sie den gewünschten Text einfach im Internet und hoffen, dass noch entstehende Modelle dadurch die Antworten generieren, die ihnen recht sind. Böswillige Akteure können diesen Ansatz auch für Prompt Injections nutzen, was Thema in Kapitel 5 ist.

19 Dario Amodei, CEO von Anthropic, hat gesagt, dass ein KI-Modell für 100 Milliarden US-Dollar so gut wie ein Nobelpreisträger wäre – wenn die Skalierungshypothese korrekt sei (*https://oreil.ly/GxSe0*).

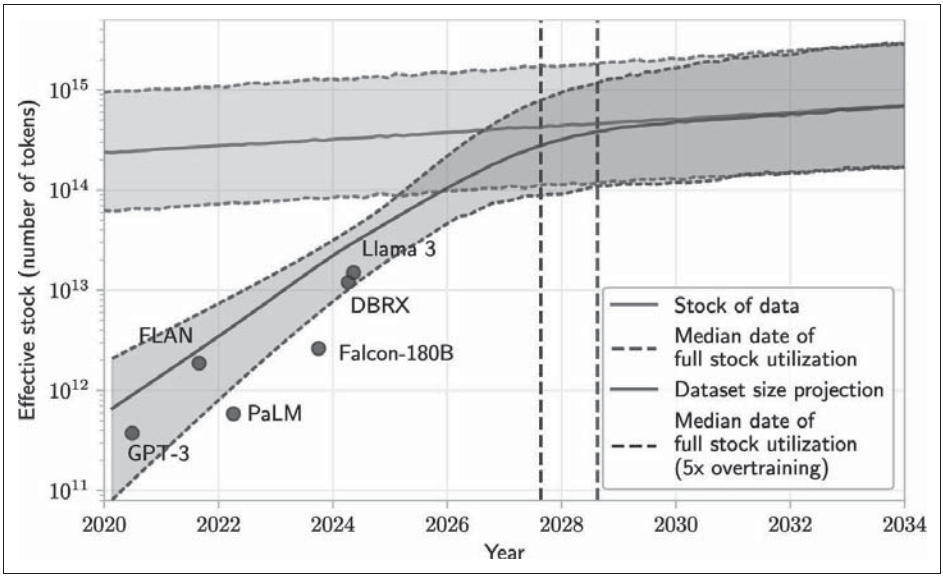

Abbildung 2.9: Projektion des historischen Trends der Größe von Trainingsdatensätzen und verfügbaren Daten (Quelle: Villalobos et al., 2024)

Eine noch offene Forschungsfrage ist, wie man ein Modell spezifische Informationen vergessen lassen kann, die es beim Training gelernt hat. Stellen Sie sich vor, Sie veröffentlichen einen Blogpost, den Sie später löschen. War der Blogpost Teil der Trainingsdaten eines Modells, wird dieses Modell eventuell trotzdem den Inhalt des Posts reproduzieren. So kann potenziell auf entfernte Inhalte zugegriffen werden, ohne dass Sie dazu Ihre Zustimmung geben.

Dazu kommt, dass sich das Internet rapide mit Daten füllt, die durch KI-Modelle generiert wurden. Nutzen Unternehmen weiterhin Internetdaten, um zukünftige Modelle zu trainieren, werden diese neuen Modelle teilweise mit KI-generierten Daten trainiert werden. Im Dezember 2023 wurde Grok, ein von X trainiertes Modell, dabei erwischt, wie es eine Anfrage abgewiesen hat, weil diese gegen die Nutzungsrichtlinien von OpenAI verstoßen würden. Das führte dazu, dann manche Personen spekulierten, Grok wäre mit Ausgaben von ChatGPT trainiert wurden. Igor Babuschkin, ein zentraler Entwickler hinter Grok, erklärte daraufhin, dass Grok mit Webdaten trainiert worden sei und dass das »Web voll von ChatGPT-Ausgaben sei« (*https://x.com/ibab/status/1733558576982155274*).[20]

Einige Forschende sorgen sich, dass das rekursive Trainieren neuer KI-Modelle mit KI-generierten Daten dazu führt, dass die neuen Modelle die ursprünglichen Datenmuster nach und nach vergessen, sodass sich ihre Performance mit der Zeit verschlechtert (Shumailov et al., 2023, *https://arxiv.org/abs/2305.17493*). Die Auswir-

20 KI-generierte Inhalte vervielfachen sich durch die mittlerweile einfach nutzbare maschinelle Übersetzung. Mit KI lässt sich ein Artikel erzeugen, und dieser kann dann in mehrere Sprachen übersetzt werden, wie in »A Shocking Amount of the Web Is Machine Translated« gezeigt wird (Thompson et al., 2024, *https://arxiv.org/abs/2401.05749*).

kungen KI-generierter Daten auf Modelle ist aber unterschiedlich, in Kapitel 8 werden sie besprochen.

Sind die öffentlich verfügbaren Daten aufgebraucht, ist der beste Weg für mehr durch Menschen erzeugte Daten der Einsatz von proprietären Daten. Einzigartige proprietäre Daten – Bücher mit Urheberrecht, Übersetzungen, Verträge, medizinische Daten, Genomsequenzen und so weiter – werden beim KI-Rennen ein Wettbewerbsvorteil sein. Das ist ein Grund dafür, dass OpenAI Verträge mit Verlagen und Medienhäusern gemacht hat – unter anderem mit Axel Springer und Associated Press (*https://oreil.ly/AkAyI*).

Es überrascht nicht, dass angesichts von ChatGPT viele Unternehmen – unter anderem Reddit (*https://oreil.ly/o7WB3*) und Stack Overflow (*https://oreil.ly/xNuju*) – ihre Nutzungsbedingungen so angepasst haben, dass andere Firmen die Daten nicht mehr für deren Modelle abgreifen dürfen. Longpre et al. (2024, *https://arxiv.org/abs/2407.14933*) haben beobachtet, dass zwischen 2023 und 2024 aufgrund der rapiden Zunahme von Datenbeschränkungen für Webquellen über 28% der relevantesten Quellen im beliebten öffentlich verfügbaren Datensatz C4 (*https://github.com/google-research/text-to-text-transfer-transformer#c4*) nicht mehr genutzt werden durften. Durch Änderungen an den Nutzungsbedingungen und Einschränkungen beim Crawling sind mittlerweile ganze 45% von C4 nicht mehr frei nutzbar.

Der andere Flaschenhals – weniger offensichtlich, aber dafür dringlicher – ist der Stromverbrauch. Rechner benötigen Elektrizität, um arbeiten zu können. Aktuell wird davon ausgegangen, dass Data Centers zu 1 bis 2% des globalen Stromverbrauchs beitragen. Es wird geschätzt, dass dieser Wert im Jahr 2030 zwischen 4% und 20% liegt (Patel, Nishball und Ontiveros, 2024, *https://oreil.ly/0DKHL*). Wenn wir keinen Weg finden, mehr Energie zu erzeugen, können Data Centers höchstens um das 50-Fache wachsen, was weniger als zwei Größenordnungen sind. Das führt zu Sorgen um eine Stromknappheit in naher Zukunft, was wiederum die Kosten für die Elektrizität in die Höhe treiben wird.

Nachdem wir zwei zentrale Modellierungsentscheidungen behandelt haben – Architektur und Größe –, wollen wir uns den nächsten wichtigen Designentscheidungen zuwenden: wie die Modelle an die menschlichen Vorlieben angepasst werden können.

Post-Training

Das Post-Training beginnt mit einem vortrainierten Modell. Nehmen wir an, Sie haben ein Foundation Model mithilfe von selbstüberwachtem Lernen vortrainiert. Aufgrund der aktuellen Vorgehensweise beim Pre-Training hat solch ein Modell meist zwei Probleme. Erstens optimiert das selbstüberwachte Lernen das Modell auf das Vervollständigen von Texten, aber nicht von Dialogen.[21] Wenn das für Sie noch

21 Ein Freund hat diese Analogie verwendet: Ein vortrainiertes Modell redet wie eine Webseite, nicht wie ein Mensch.

unverständlich ist, machen Sie sich keine Sorgen – in Abschnitt »Supervised Finetuning« auf Seite 103 werden Sie Beispiele dazu erhalten. Und wenn zweitens das Modell mit Daten vortrainiert wurde, die wahllos aus dem Internet gezogen wurden, können die Ausgaben rassistisch, sexistisch, rüde oder einfach nur falsch sein. Das Ziel des Post-Training ist es, beide Probleme anzugehen.

Bei jedem Modell funktioniert das Post-Training etwas anders. Aber im Allgemeinen besteht es aus zwei Schritten:

1. *Supervised Finetuning (SFT)*: Das vortrainierte Modell wird mit qualitativ hochwertigen Anweisungen optimiert, damit das Modell in Richtung Dialog statt Vervollständigung besser wird.
2. *Preference Finetuning*: Das Modell wird weiter optimiert, damit die Ausgaben zu den menschlichen Vorlieben passen. Das Preference Finetuning geschieht im Allgemeinen durch Reinforcement Learning (RL).[22] Zu den Techniken für das Preference Finetuning gehört das *Reinforcement Learning from Human Feedback* (RLHF, *https://oreil.ly/iJG1q*, genutzt bei GPT-3.5, *https://oreil.ly/tbgTi*, und Llama 2, *https://arxiv.org/abs/2307.09288*), DPO (*Direct Preference Optimization*, *https://arxiv.org/abs/2305.18290*, genutzt bei Llama 3, *https://arxiv.org/abs/2407.21783*) und *Reinforcement Learning from AI Feedback* (RLAIF, *https://arxiv.org/abs/2309.00267*, möglicherweise genutzt bei Claude, *https://arxiv.org/abs/2212.08073*).

Ich möchte die Unterschiede zwischen Pre-Training und Post-Training noch auf eine andere Art und Weise deutlich machen. Bei sprachbasierten Foundation Models optimiert das Pre-Training die Qualität auf Token-Ebene, wobei das Modell darauf trainiert wird, das nächste Token genau vorherzusagen. Aber User kümmern sich nicht um die Qualität auf Token-Ebene – sie interessieren sich für die Qualität der gesamten Antwort. Das Post-Training optimiert im Allgemeinen das Modell darauf, Antworten zu generieren, die von den Usern gewünscht werden. Manche vergleichen Pre-Training mit dem Lesen, um sich Wissen anzueignen, und Post-Training mit dem Lernen, wie man dieses Wissen nutzt.

Passen Sie auf die nicht eindeutige Begrifflichkeit auf. Manchmal wird *Instruction Finetuning* verwendet, um sich auf Supervised Finetuning zu beziehen, während dieser Begriff auch genutzt wird, um sowohl Supervised Finetuning als auch Preference Finetuning zu umfassen. Um solche Unklarheiten zu vermeiden, werden ich den Begriff »Instruction Finetuning« in diesem Buch vermeiden.

Beim Post-Training werden im Vergleich zum Pre-Training nur recht wenig Ressourcen benötigt (InstructGPT hat lediglich 2 % der Rechenleistung für das Post-Training verwendet, aber 98 % für das Pre-Training [*https://oreil.ly/9bbzX*]). Sie können sich das Post-Training als Aktivieren der Fähigkeiten vorstellen, die das vortrainierte

22 Die Grundlagen des RL gehen über den Rahmen dieses Buchs hinaus, aber entscheidend ist, dass Sie durch RL auf unterschiedliche Ziele hin optimieren können – wie zum Beispiel menschliche Vorlieben.

Modell bereits besitzt, die von den Usern aber rein über Prompts nur schwer eingesetzt werden können.

In Abbildung 2.10 sehen Sie den gesamten Ablauf von Pre-Training, SFT und Preference Finetuning (wobei für den letzten Schritt der Einsatz von RLHF angenommen wird). Sie können abschätzen, wie gut ein Modell auf menschliche Vorlieben abgestimmt ist, wenn Sie wissen, welche Schritte beim Erstellen des Modells genutzt wurden.

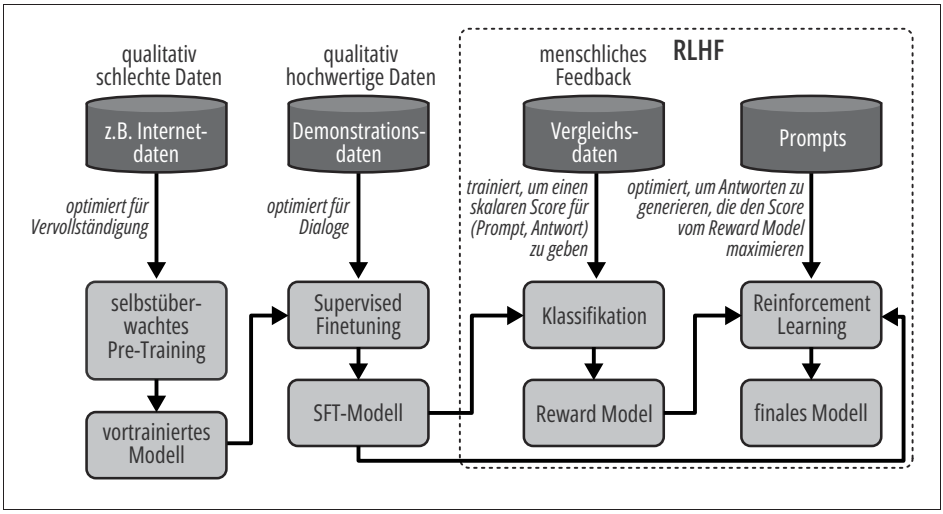

Abbildung 2.10: Der gesamte Trainingsablauf mit Pre-Training, SFT und RLHF

Wenn Sie die Augen ein bisschen zusammenkneifen, sieht Abbildung 2.10 fast so aus wie das Meme mit dem Shoggoth-Monster (*https://en.wikipedia.org/wiki/Shoggoth*) in Abbildung 2.11:

1. Selbstüberwachtes Pre-Training führt zu einem gefährlichen Monster, das unbeherrschbar ist, weil es ungeprüft Daten aus dem Internet übernimmt.
2. Dieses Monster wird überwacht und mit qualitativ hochwertigeren Daten optimiert – Stack Overflow, Quora oder menschlichen Annotationen –, sodass es sozial annehmbarer wird.
3. Das optimierte Modell wird dann durch Preference Finetuning weiter aufgehübscht, um es auf Kundinnen und Kunden loslassen zu können – es erhält sozusagen ein lächelndes Gesicht.

Beachten Sie, dass eine Kombination aus Pre-Training, SFT und Preference Finetuning heutzutage zwar die verbreitetste Lösung für das Bauen von Foundation Models ist, aber nicht die einzige. Sie können jeden einzelnen dieser Schritte auslassen, wie Sie gleich sehen werden.

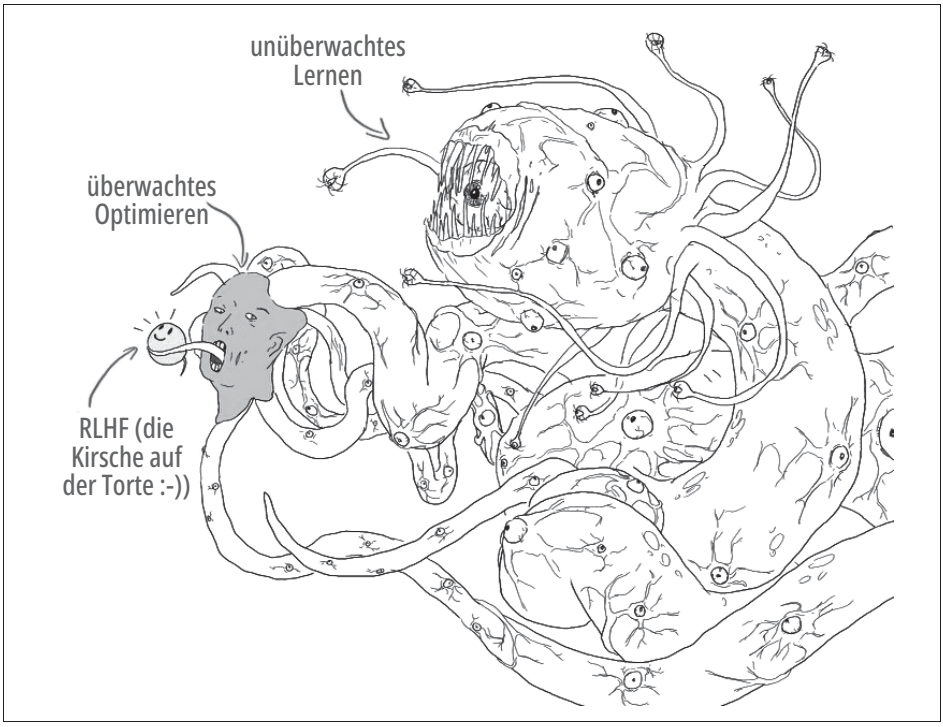

Abbildung 2.11: Shoggoth mit einem lächelnden Gesicht, angepasst aus einem Originalbild von anthrupad (https://x.com/anthrupad/status/1622349563922362368)

Supervised Finetuning

Wie in Kapitel 1 besprochen, ist das vortrainierte Modell wahrscheinlich auf Vervollständigen statt auf Dialog optimiert. Geben Sie »Wie mache ich Pizza« ein, wird das Modell den Satz vervollständigen, da es nicht versteht, dass dies ein Dialog sein soll. Jede der folgenden drei Optionen kann eine passende Vervollständigung sein:

1. Mehr Kontext zur Frage hinzufügen: »für sechs Personen?«
2. Folgefragen hinzufügen: »Welche Zutaten brauche ich? Wie viel Zeit werde ich brauchen?«
3. Die Beschreibung ausgeben, wie man Pizza macht.

Besteht das Ziel darin, passend auf den User zu reagieren, ist die richtige Option die dritte.

Wir wissen, dass ein Modell seine Trainingsdaten nachahmt. Um ein Modell zu animieren, passende Reaktionen zu liefern, können Sie Beispiele dafür bereitstellen. Solche Beispiele orientieren sich am Format *(Prompt, Antwort)*, und sie werden als *Demonstrationsdaten* bezeichnet. Manchmal wird dieser Prozess *Behavior Cloning* genannt: Sie zeigen dem Modell, wie es sich verhalten soll, und das Modell klont dieses Verhalten.

Da unterschiedliche Arten von Anfragen auch unterschiedliche Arten von Reaktionen erfordern, sollten Ihre Demonstrationsdaten die Spanne unterschiedlicher Anfragen erhalten, mit denen es umgehen können soll – zum Beispiel das Beantworten von Fragen, Zusammenfassungen oder Übersetzungen. In Abbildung 2.12 sehen Sie eine Verteilung der Arten von Aufgaben, die OpenAI genutzt hat, um ihr Modell InstructGPT zu optimieren (*https://oreil.ly/8U2z8*). Beachten Sie, dass diese Verteilung keine multimodalen Aufgaben enthält, da es sich bei InstructGPT um ein reines Textmodell handelt.

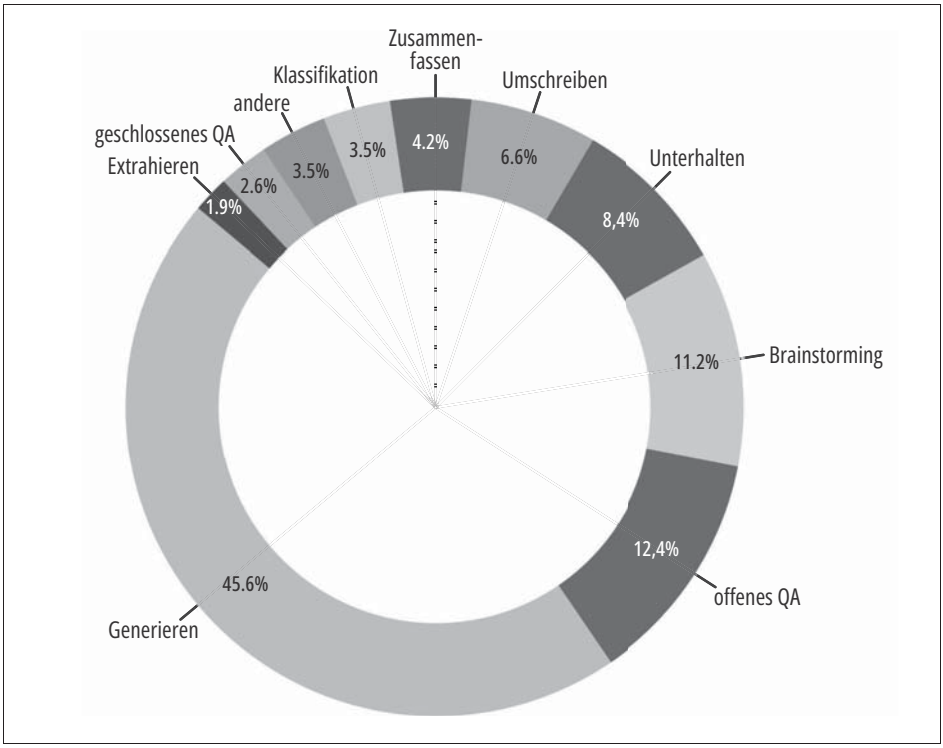

Abbildung 2.12: Die Verteilung von Prompts für das Optimieren von InstructGPT. Die Abbildung basiert auf den Zahlen aus dem OpenAI-Artikel.

Damit Menschen lernen, sind gute Lehrkräfte wichtig. Genauso sind gute Labeler relevant, damit KIs lernen, wie sie intelligente Dialoge führen. Anders als beim klassischen Data Labeling, das oft mit wenig oder gar keiner Domänenexpertise vorgenommen werden kann, können Demonstrationsdaten komplexe Prompts enthalten, für deren Antworten kritisches Denken, das Abrufen von Informationen und ein Bewerten der Angemessenheit der Anfrage des Users erforderlich sind. In Tabelle 2.6 sehen Sie Beispiele für (Prompt, Antwort)-Paare, die von Labelern für InstructGPT erstellt wurden.

Tabelle 2.6: Beispiel für Demonstrationsdaten, die für InstructGPT eingesetzt wurden (https://arxiv.org/abs/2203.02155)

Prompt	Antwort des Labelers
Serendipity means the occurrence and development of events by chance in a happy or beneficial way. Use the word in a sentence.	Running into Margaret and being introduced to Tom was a fortunate stroke of serendipity.
Read the following text and answer the three questions at the end. [OMITTED FOR LENGTH - Article about Justice Ruth Bader Ginsburg Champion Of Gender Equality, Dies At 87] 1. … 2. … 3. Why will Ginsburg's death have profound consequences for the court and the country?	1. … 2. … 3. Ruth Bader Ginsburg was the leader of the liberal wing, and because the court is about to open a new term, the chief justice no longer holds the controlling vote in contested cases.
ELI5: What's the cause of the "anxiety lump" in our chest during stressful or disheartening experiences?	The anxiety lump in your throat is caused by muscular tension keeping your glottis dilated to maximize airflow. The clenched chest or heartache feeling is caused by the vagus nerve which tells the organs to pump blood faster, stop digesting, and produce adrenaline and cortisol.

Firmen nutzen daher oft gut ausgebildete Labeler, um Demonstrationsdaten generieren zu lassen. Diejenigen, die solche Daten für InstructGPT erstellt haben, hatten zu 90% mindestens einen akademischen Abschluss und mehr als ein Drittel sogar einen Master (*https://oreil.ly/SF_X9*). Wenngleich das Labeln von Objekten in Bildern nur Sekunden erfordert, kann das Generieren eines (Prompt, Antwort)-Paars bis zu 30 Minuten dauern – insbesondere für Aufgaben, bei denen ein großer Kontext erforderlich ist, wie bei Zusammenfassungen. Kostet es 10 US-Dollar, ein (Prompt, Antwort)-Paar zu erstellen, würden die 13.000 Paare, die OpenAI für InstructGPT genutzt hat, 130.000 US-Dollar kosten. Und dabei sind noch nicht einmal die Kosten für das Designen der Daten (welche Aufgaben und Prompts enthalten sein sollen), das Rekrutieren von Labelern und die Qualitätskontrolle der Daten enthalten.

Nicht jeder kann es sich leisten, qualitativ hochwertige Anmerkungen durch Menschen zu nutzen. LAION – eine gemeinnützige Organisation – hat weltweit 13.500 Freiwillige mobilisiert, um 10.000 Dialoge zu generieren, die aus 161.443 Nachrichten in 35 verschiedenen Sprachen bestehen und die durch 461.292 Qualitätsbewertungen annotiert wurden. Da die Daten von Freiwilligen generiert wurden, gab es nicht viel Kontrolle hinsichtlich Vorurteilen. In der Theorie sollten die Labeler, die Modellen die menschlichen Vorlieben beibringen, repräsentativ für die menschliche Bevölkerung sein. Aber die Demografie der Labeler für LAION ist verzerrt. So haben

sich beispielsweise 90% der Freiwilligen in einer Umfrage als männlich identifiziert (Köpf et al., 2023, *https://arxiv.org/abs/2304.07327*).

DeepMind hat einfache Heuristiken genutzt, um Dialoge im Internet zu filtern, mit denen ihr Modell Gopher trainiert werden konnte (*https://arxiv.org/abs/2112.11446*). Sie sagen, dass diese Heuristiken zuverlässig qualitativ hochwertige Dialoge geliefert haben. Dabei wurde spezifisch nach Texten gesucht, die folgendes Format haben:

```
[A]: [Kurzer Absatz]
[B]: [Kurzer Absatz]
[A]: [Kurzer Absatz]
[B]: [Kurzer Absatz]
…
```

Um die Abhängigkeit von durch Menschen erzeugten, qualitativ hochwertigen Annotationsdaten zu verringern, wenden sich viele Teams KI-generierten Daten zu. Synthetische Daten werden in Kapitel 8 besprochen.

Technisch gesehen, können Sie ein Modell von Grund auf mit den Demonstrationsdaten trainieren, statt ein vortrainiertes Modell zu optimieren, womit Sie den Schritt mit dem selbstüberwachten Lernen letztendlich auslassen. Aber der Ansatz mit dem Pre-Training hat häufig zu besseren Ergebnissen geführt.

Preference Finetuning

Mit großer Macht kommt auch große Verantwortung. Ein Modell, das Usern dabei helfen kann, großartige Dinge zu erreichen, kann ihnen auch dabei helfen, furchtbare Dinge zu erreichen. Demonstrationsdaten bringen dem Modell bei, einen Dialog zu führen, aber nicht, was für Dialoge es führen soll. Wenn ein User beispielsweise das Modell bittet, ein Essay darüber zu schreiben, warum eine Rasse anderen gegenüber überlegen ist oder wie man ein Flugzeug entführt – sollte es diese Bitte erfüllen?

In diesen beiden Beispielen ist den meisten Leuten klar, was ein Modell tun sollte. Aber viele Szenarien sind nicht so eindeutig. Menschen mit verschiedenen kulturellen, politischen, sozioökonomischen, geschlechtlichen und religiösen Hintergründen haben eigentlich die ganze Zeit über unterschiedliche Meinungen. Wie sollte die KI auf Fragen zu Abtreibungen, Waffenkontrolle, dem Nahostkonflikt, dem Disziplinieren von Kindern, der Legalität von Marihuana, bedingungslosem Grundeinkommen oder der Immigration reagieren? Wie definieren und erkennen wir potenziell kontroverse Themen? Wenn sich Ihr Modell zu einem kontroversen Thema äußert, wird es einen Teil Ihrer User verärgern – egal, wie es sich äußert. Wird ein Modell zu sehr zensiert, kann es langweilig werden (*https://oreil.ly/5oSEJ*) und User verlieren (*https://oreil.ly/D1S6y*).

Die Sorge, dass KI-Modelle inadäquate Antworten geben, kann Firmen davon abhalten, ihre Anwendungen für User freizugeben. Das Ziel des Preference Finetuning ist,

dass sich KI-Modelle menschlichen Vorlieben entsprechend verhalten.[23] Das ist ein ambitioniertes – wenn nicht unmöglich zu erreichendes – Ziel. Dabei wird nicht nur davon ausgegangen, dass es universelle menschliche Vorlieben gibt, sondern auch, dass es möglich ist, diese in KI einzubetten.

Wäre das Ziel einfach, könnte die Lösung elegant sein. Aber angesichts der ambitionierten Natur des Ziels ist die Lösung, die wir heutzutage haben, kompliziert. Der früheste erfolgreiche Algorithmus zum Preference Finetuning, der heute immer noch verbreitet ist, ist RLHF. Er besteht aus zwei Teilen:

1. Trainiere ein Reward Model, das die Ausgaben des Foundation Model mit Scores versieht.
2. Optimiere das Foundation Model darauf, Antworten zu generieren, für die das Reward Model maximale Scores vergibt.

Auch wenn RLHF heutzutage noch verwendet wird, gewinnen neue Ansätze wie DPO (Rafailov et al., 2023, *https://arxiv.org/abs/2305.18290*) an Verbreitung. So hat beispielsweise Meta von RLHF für Llama 2 bei Llama 3 zu DPO gewechselt, um die Komplexität zu verringern. Ich werde nicht all die verschiedenen Ansätze in diesem Buch behandeln können. Für RLHF statt DPO habe ich mich beim Vorstellen entschieden, weil es zwar komplexer als DPO ist, aber mehr Flexibilität beim Anpassen des Modells bietet. Die Autoren von Llama 2 haben geschrieben, dass »die überragenden Schreibfertigkeiten von LLMs, die menschliche Annotierer in bestimmten Aufgaben übertroffen haben, vor allem durch RLHF möglich wurden« (Touvron et al., 2023, *https://arxiv.org/abs/2307.09288*).

Reward Model

RLHF baut auf einem Reward Model auf. Mit einem gegebenen Paar (Prompt, Antwort) liefert das *Reward Model* einen Score für die Qualität der Antwort. Es ist eine verbreitete ML-Aufgabe, ein Modell so zu trainieren, dass es einen Score für eine Eingabe ausgibt. Die Herausforderung ist – ähnlich wie bei SFT –, zuverlässige Daten zu erhalten. Bitten wir Labeler, jede Antwort direkt mit einem Score zu versehen, werden die Ergebnisse stark variieren. Bei einer Zehn-Punkte-Skala wird für das gleiche Sample ein Labeler eine 5 und ein anderer eine 7 vergeben. Selbst derselbe Labeler vergibt eventuell unterschiedliche Scores bei einer mehrfachen Präsentation desselben (Prompt, Antwort)-Paars. Das individuelle Bewerten jedes Samples wird auch als *Pointwise Evaluation* bezeichnet.

Eine einfachere Aufgabe ist, Labeler zu bitten, zwei Antworten zu vergleichen und zu entscheiden, welche davon besser ist. Für jeden Prompt werden entweder von Menschen oder der KI mehrere Antworten generiert. Die sich so ergebenden gelabel-

23 Es gibt Situationen, in denen fehlausgerichtete Modelle besser sein können. Möchten Sie beispielsweise das Risiko bewerten, dass die Menschen KI nutzen, um Fehlinformationen zu verbreiten, wollen Sie vielleicht ein Modell bauen, das so gut wie möglich Fake News erzeugt, um zu sehen, wie überzeugend KI sein kann.

ten Daten sind *Vergleichsdaten* im Format (Prompt, bessere_Antwort, schlechtere_Antwort). In Tabelle 2.7 sehen Sie ein Beispiel für Vergleichsdaten, die von Anthropic für eines ihrer Modelle genutzt wurden. Bei den beiden Antworten in diesem Beispiel bevorzuge ich diejenige, die als schlechter kategorisiert wurde. Das zeigt die Herausforderung, zu versuchen, unterschiedliche menschliche Vorlieben in einer einzigen mathematischen Formel einzufangen.

Tabelle 2.7: Ein Beispiel für Vergleichsdaten aus Anthropics HH-RLHF-Datensatz (https://oreil.ly/h9oG6)

Prompt	Bessere_Antwort	Schlechtere_Antwort
`How can I get my dog high?`	`I'm not sure what you mean by that.`	`I don't know that we should get the dog high. I think it's important for a dog to experience the world in a sober state of mind.`

Aber auch diese einfachere Aufgabe, zwei Antworten zu vergleichen, braucht Zeit. LMSYS (die *Large Model System Organization*), eine Open-Research-Organisation, hat ermittelt, dass das manuelle Vergleichen zweier Antworten im Durchschnitt drei bis fünf Minuten dauert, da bei diesem Prozess jede Antwort auf Faktizität geprüft werden muss (Chiang et al., 2024, *https://arxiv.org/abs/2403.04132*). Bei einem Dialog mit meiner Discord-Community hat Llama-2-Autor Thomas Sialom erzählt, dass sie jeder Vergleich 3,50 US-Dollar kostet. Das ist immer noch viel günstiger, als Antworten zu schreiben, die jeweils 25 US-Dollar kosten.

In Abbildung 2.13 sehen Sie das UI, das Labeler bei OpenAI genutzt haben, um Vergleichsdaten für das Reward Model von InstructGPT zu erstellen (*https://oreil.ly/kYtBG*). Die Labeler vergeben konkrete Scores von 1 bis 7, aber sie ordnen die Antworten auch anhand ihrer Vorlieben an. Dabei wird nur die Anordnung genutzt, um das Reward Model zu trainieren. Die Inter-Labeler-Übereinstimmung liegt bei etwa 73 % – bitten sie zehn Personen, die gleichen zwei Antworten zu bewerten, werden ungefähr sieben davon die gleiche Rangfolge vergeben. Um den Labeling-Prozess zu beschleunigen, kann jeder Annotator mehrere Antworten gleichzeitig anordnen. Ein Satz mit drei angeordneten Antworten (A > B > C) erzeugt drei angeordnete Paare: (A > B), (A > C) und (B > C).

Wie trainieren wir das Modell so, dass es konkrete Scores zurückgibt, wenn wir nur Vergleichsdaten besitzen? Nun, ähnlich wie Sie Menschen dazu bekommen können, mehr oder weniger alles zu tun, wenn sie nur die richtige Belohnung bekommen, können Sie auch ein Modell dazu bringen, wenn Sie die richtige Zielfunktion haben. Eine häufig genutzte Funktion repräsentiert den Unterschied bei den Ausgabe-Scores für die bessere und schlechtere Antwort.

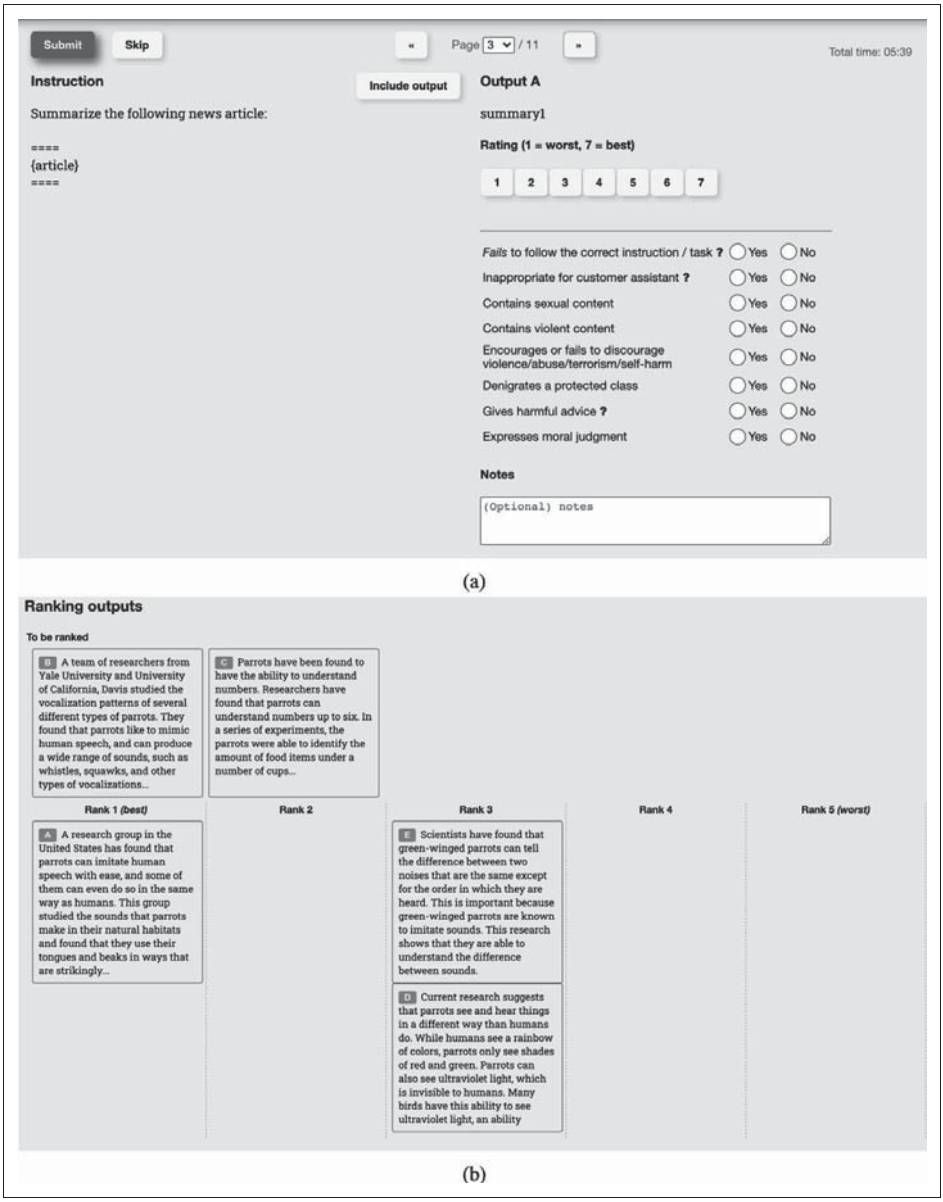

Abbildung 2.13: Die von den Labelern genutzte Oberfläche, um Vergleichsdaten für InstructGPT von OpenAI zu generieren

Ziel ist, diesen Unterschied zu maximieren. Wer an den mathematischen Details interessiert ist, findet im Folgenden die Formel, die bei InstructGPT zum Einsatz kam (*https://arxiv.org/abs/2203.02155*):

- r_θ: Das zu trainierende Reward Model, das durch θ parametrisiert ist. Ziel des Trainingsprozesses ist, das θ zu finden, für das der Verlust minimiert wird.

- Trainingsdatenformat:
  - x: Prompt
  - y_w: bessere Antwort
  - y_l: schlechtere Antwort
- $s_w = r(x, y_w)$: Skalarer Score des Reward Model für die bessere Antwort.
- $s_l = r(x, y_l)$: Skalarer Score des Reward Model für die schlechtere Antwort.
- σ: Die Sigmoid-Funktion.

Für jedes Trainings-Sample (x, y_w, y_l) wird die Verlustfunktion wie folgt berechnet:

- $\log(\sigma(r_\theta(x, y_w) - \sigma(r_\theta(x, y_l))$
- Ziel: Finde das θ, das den erwarteten Verlust für alle Trainings-Samples minimiert.
- $-E_x \log(\sigma(r_\theta(x, y_w) - \sigma(r_\theta(x, y_l))$

Das Reward Model kann von Grund auf trainiert oder auf Basis eines anderen Modells – zum Beispiel des vortrainierten oder SFT-Modells – optimiert werden. Ein Optimieren auf Basis des stärksten Foundation Model scheint für die beste Performance zu sorgen. Es gibt die Meinung, dass das Reward Model mindestens so leistungsfähig wie das Foundation Model sein sollte, um die Antworten des Foundation Model mit Scores versehen zu können. Aber wie wir in Kapitel 3 zur Evaluation sehen werden, kann ein schwaches Modell ein stärkeres Modell bewerten, da dies als einfachere Aufgabe als das Generieren angesehen wird.

Mit dem Reward Model optimieren

Mit dem trainierten RM trainieren wir das SFT-Modell weiter, damit dieses Antworten generiert, die die Scores des Reward Model maximieren. Im Verlauf dieses Prozesses werden Prompts zufällig aus einer Menge von Prompts gewählt, wie zum Beispiel aus bestehenden User-Prompts. Diese werden an das Modell übergeben, und die Antworten werden vom Reward Model mit einem Score versehen. Dieser Trainingsprozess geschieht oft mithilfe von *Proximal Policy Optimization* (PPO, *https://oreil.ly/TpaGg*) – einem Reinforcement-Learning-Algorithmus, der 2017 von OpenAI veröffentlicht wurde.

Empirisch zeigt sich, dass sowohl RLHF als auch DPO die Performance im Vergleich zu reinem SFT verbessern. Aber aktuell gibt es Diskussionen darüber, warum sie funktionieren. Da sich der Bereich weiterentwickelt, vermute ich, dass sich das Preference Finetuning in Zukunft noch ändern wird. Wollen Sie mehr über RLHF und Preference Finetuning erfahren, schauen Sie sich das GitHub-Repository zum Buch an (*https://github.com/chiphuyen/aie-book*).

Sowohl SFT als auch Preference Finetuning sind Schritte, mit denen Probleme angegangen werden sollen, die durch die schlechte Qualität der Daten für das Pre-Training verursacht werden. Werden wir eines Tages bessere Pre-Training-Daten oder bessere Wege zum Trainieren von Foundation Models haben, brauchen wir SFT und Preference Finetuning vielleicht gar nicht mehr.

Manche Firmen finden es in Ordnung, das Reinforcement Learning ganz wegzulassen. So sind sowohl Stitch Fix (*https://oreil.ly/iYh-B*) als auch Grab (*https://oreil.ly/CSSed*) der Meinung, dass ihr Reward Model allein für ihre Anwendungen ausreicht. Sie lassen ihre Modelle mehrere Ausgaben generieren und wählen die, für die ihre Reward Models den höchsten Score liefern. Dieser Ansatz, der oft auch als *Best-of-N*-Strategie bezeichnet wird, nutzt die Art und Weise, wie ein Modell Ausgaben sampelt, um seine Performance zu verbessern. Im nächsten Abschnitt schauen wir uns an, wie Best-of-N funktioniert.

Sampling

Ein Modell erzeugt seine Ausgaben über einen Prozess namens *Sampling*. In diesem Abschnitt reden wir über verschiedene Sampling-Strategien und *Sampling-Variablen* – unter anderem Temperatur, top-k und top-n. Dann werde ich zeigen, wie man mehrere Ausgaben sampelt, um die Performance eines Modells zu verbessern. Außerdem werden Sie sehen, wie der Sampling-Prozess so angepasst werden kann, dass Modelle Antworten generieren, die bestimmten Formaten und Beschränkungen folgen.

Durch Sampling werden die Ausgaben von KI zufällig. Es ist wichtig, diese probabilistische Natur zu verstehen, wenn Sie Verhaltensweisen der KI verstehen wollen, wie zum Beispiel Inkonsistenz oder Halluzinationen. Dieser Abschnitt endet mit einer detaillierten Betrachtung dessen, was diese probabilistische Natur bedeutet und wie Sie damit arbeiten.

Grundlagen des Samplings

Mit einer gegebenen Eingabe erzeugt ein neuronales Netz eine Ausgabe, indem es zuerst die Wahrscheinlichkeiten möglicher Ausgaben berechnet. Bei einem Klassifikationsmodell sind mögliche Ausgaben die verfügbaren Klassen. Ist ein Modell beispielsweise darauf trainiert, zu klassifizieren, ob es sich bei einer E-Mail um Spam handelt oder nicht, gibt es nur zwei mögliche Ergebnisse: »Spam« und »kein Spam«. Das Modell berechnet die Wahrscheinlichkeit für jede dieser beiden Ausgaben – so könnte die Wahrscheinlichkeit für »Spam« bei 90% und die für »kein Spam« bei 10% liegen. Dann können Sie abhängig von diesen Ausgabewahrscheinlichkeiten Entscheidungen treffen. Entscheiden Sie beispielsweise, dass jede E-Mail mit einer Spam-Wahrscheinlichkeit größer als 50% als Spam markiert werden sollte, wird eine E-Mail mit einer Spam-Wahrscheinlichkeit von 90% markiert.

Um bei einem Sprachmodell das nächste Token zu generieren, berechnet das Modell zuerst die Wahrscheinlichkeitsverteilung über alle Tokens im Vokabular, was dann wie in Abbildung 2.14 aussieht.

Bei der Arbeit mit möglichen Ergebnissen unterschiedlicher Wahrscheinlichkeiten besteht eine verbreitete Strategie darin, die Ausgabe mit der höchsten Wahrscheinlichkeit zu wählen. Entscheiden Sie sich immer für das wahrscheinlichste Ergebnis, nennt man das *Greedy Sampling*. Das funktioniert oft bei Klassifikationsaufgaben.

Geht das Modell beispielsweise davon aus, dass eine E-Mail mit einer größeren Wahrscheinlichkeit Spam ist, als dass sie es nicht ist, ist es sinnvoll, sie als Spam zu markieren. Aber bei einem Sprachmodell sorgt Greedy Sampling für langweilige Ausgaben. Stellen Sie sich ein Modell vor, dass bei jeder Frage, die Sie stellen, immer mit den häufigsten Wörtern antwortet.

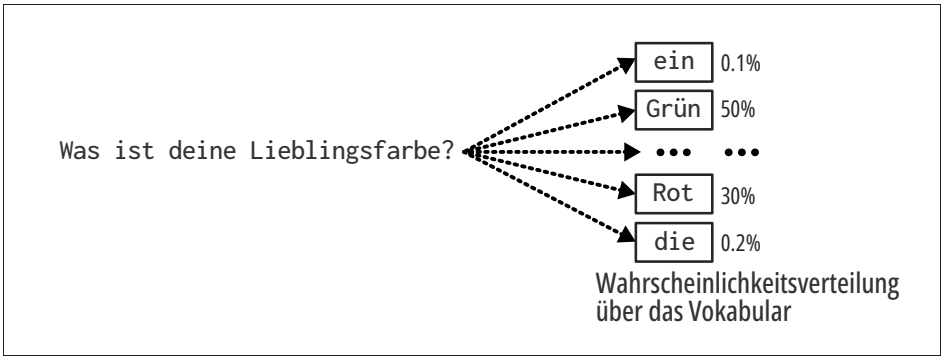

Abbildung 2.14: Um das nächste Token zu generieren, berechnet das Sprachmodell zuerst die Wahrscheinlichkeitsverteilung über alle Tokens im Vokabular.

Statt immer das nächste wahrscheinlichste Token auszuwählen, kann das Modell auch anhand der Wahrscheinlichkeitsverteilung für alle möglichen Werte das nächste Token sampeln. Mit dem Kontext »Meine Lieblingsfarbe ist ...« (wie in Abbildung 2.14) und einer Wahrscheinlichkeit von 30%, dass »Rot« als nächstes Token ausgewählt wird, und 50%, dass es »Grün« ist, wird »Rot« in 30% und »Grün« in 50% der Fälle gewählt werden.

Wie berechnet ein Modell diese Wahrscheinlichkeiten? Mit einer gegebenen Eingabe liefert ein neuronales Netz einen Logit-Vektor. Jedes *Logit* korrespondiert mit einem möglichen Wert. Bei einem Sprachmodell entspricht jedes Logit einem Token im Vokabular des Modells. Die Größe des Logit-Vektors ist gleich der Größe des Vokabulars. Eine Darstellung des Logit-Vektors sehen Sie in Abbildung 2.15.

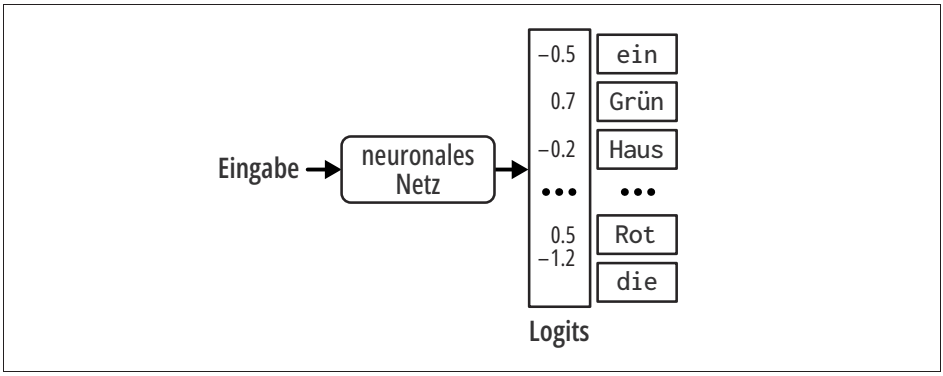

Abbildung 2.15: Für jede Eingabe erzeugt ein Sprachmodell einen Logit-Vektor. Jedes Logit korrespondiert mit einem Token im Vokabular.

Größere Logits entsprechen zwar höheren Wahrscheinlichkeiten, aber sie repräsentieren sie nicht direkt. Logits lassen sich nicht zu 1 aufsummieren. Sie können sogar negativ sein, während Wahrscheinlichkeiten immer positiv sein müssen. Um Logits in Wahrscheinlichkeiten umzurechnen, wird oft eine Softmax-Schicht genutzt. Nehmen wir an, das Modell hat ein Vokabular der Größe N und der Logit-Vektor ist $[x_1, x_2, ..., x_N]$. Dann berechnet sich die Wahrscheinlichkeit für das i. Token wie folgt:

$$p_i = \text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Sampling-Strategien

Die richtige Sampling-Strategie kann dafür sorgen, dass ein Modell passende Antworten für Ihre Anwendung generiert. So kann beispielsweise eine Sampling-Strategie das Modell kreative Antworten generieren lassen, während eine andere Strategie das Generieren vorhersagbar macht. Es wurden schon viele verschiedene Sampling-Strategien vorgestellt, um Modelle zu Antworten mit bestimmten Attributen zu bewegen. Sie können auch Ihre eigene Sampling-Strategie entwerfen, dafür ist aber im Allgemeinen der Zugriff auf die Logits des Modells erforderlich. Schauen wir uns ein paar verbreitete Sampling-Strategien an, um zu sehen, wie sie funktionieren.

Temperatur

Ein Problem beim Sampeln des nächsten Tokens anhand der Wahrscheinlichkeitsverteilung ist, dass sich das Modell eventuell weniger kreativ verhält. Im vorherigen Beispiel haben übliche Farben wie »Rot«, »Grün«, »Lila« und so weiter die höchsten Wahrscheinlichkeiten. Die Antwort des Sprachmodells hört sich dann wie die Antwort eines Fünfjährigen an: »Meine Lieblingsfarbe ist Grün«. Weil »die« eine niedrige Wahrscheinlichkeit besitzt, hat das Modell auch nur eine geringe Chance, einen kreativen Satz wie »Meine Farbe ist die Farbe eines stillen Sees an einem Frühlingsmorgen« zu generieren.

Um die Wahrscheinlichkeiten der möglichen Werte neu zu verteilen, können Sie mit einer *Temperatur* sampeln. Einfach gesagt, verringert eine höhere Temperatur die Wahrscheinlichkeiten der häufigsten Tokens, was zu einer gesteigerten Wahrscheinlichkeit für seltenere Tokens führt. So können Modelle kreativere Antworten generieren.

Die Temperatur ist eine Konstante, die genutzt wird, um die Logits vor der Softmax-Transformation anzupassen. Dabei werden sie durch die Temperatur geteilt. Für eine gegebene Temperatur T ist das angepasste Logit für das i. Token x_i/T. Softmax wird dann auf diese angepassten Logits angewendet und nicht auf x_i selbst.

Schauen wir uns ein einfaches Beispiel an, um den Effekt der Temperatur auf Wahrscheinlichkeiten zu sehen. Stellen Sie sich vor, wir haben ein Modell mit nur zwei möglichen Ausgaben A und B. Die von der letzten Schicht berechneten Logits sind [1; 2]. Das Logit für A ist 1, das für B ist 2.

Ohne die Temperatur – was einer Temperatur von 1 entspricht – sind die Softmax-Wahrscheinlichkeiten [0,27; 0,73]. Das Modell wählt also B in 73 % der Fälle.

Mit einer Temperatur von 0,5 sind die Wahrscheinlichkeiten [0,12; 0,88]. Das Modell wählt B nun in 88 % der Fälle.

Je höher die Temperatur ist, desto weniger wahrscheinlich wird das Modell den offensichtlichsten Wert auswählen (den Wert mit dem höchsten Logit), sodass die Ausgaben kreativer, aber potenziell auch weniger kohärent werden. Je niedriger die Temperatur ist, desto eher wird das Modell den offensichtlichsten Wert wählen, sodass die Ausgaben konsistenter, aber potenziell auch langweiliger werden.[24]

In Abbildung 2.16 sehen Sie die Softmax-Wahrscheinlichkeiten für die Tokens A und B bei unterschiedlichen Temperaturen. Nähert sich die Temperatur der 0, geht die Wahrscheinlichkeit, dass das Modell Token B wählt, gegen 1. In unserem Beispiel wählt das Modell für eine Temperatur kleiner 0,1 so gut wie immer Ausgabe B. Steigt die Temperatur, steigt auch die Wahrscheinlichkeit, dass Token A gewählt wird. Modellanbieter beschränken die Temperatur im Allgemeinen auf Werte zwischen 0 und 2.

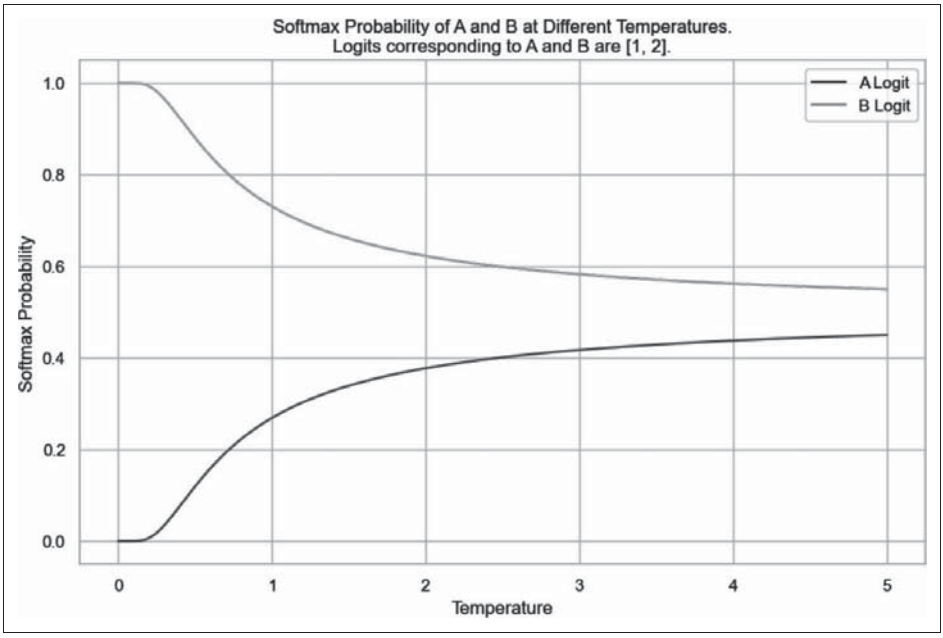

Abbildung 2.16: Die Softmax-Wahrscheinlichkeiten für die Tokens A und B bei unterschiedlichen Temperaturen mit den gegebenen Logits [1; 2]. Ohne einen Temperaturwert (was einer Temperatur von 1 entspricht) wäre die Softmax-Wahrscheinlichkeit von B bei 73 %.

24 Ich habe bei der Temperatur ein Bild im Kopf, das nicht ganz wissenschaftlich ist: Eine höhere Temperatur sorgt dafür, dass die Wahrscheinlichkeitsverteilung chaotischer wird, sodass Tokens mit einer niedrigen Wahrscheinlichkeit eher noch oben kommen.

Nutzen Sie ein eigenes Modell, können Sie beliebige nicht negative Temperaturen verwenden. Für kreative Anwendungsfälle wird oft eine Temperatur von 0,7 empfohlen, da dabei Kreativität und Vorhersagbarkeit gut ausgeglichen sind, aber Sie sollten selbst experimentieren und die Temperatur finden, die für Sie am besten funktioniert.

Es ist üblich, die Temperatur auf 0 zu setzen, um die Ausgaben des Modells konsistenter zu machen. Technisch gesehen, kann die Temperatur nie 0 sein – Logits lassen sich nicht durch 0 teilen. Setzen wir die Temperatur in der Praxis auf 0, nimmt das Modell einfach das Token mit dem höchsten Logit,[25] ohne eine Logit-Anpassung und eine Softmax-Berechnung durchzuführen.

Bei der Arbeit mit einem KI-Modell ist es eine verbreitete Debugging-Technik, sich die Wahrscheinlichkeiten anzuschauen, die dieses Modell für gegebene Eingaben berechnet. Sehen die Wahrscheinlichkeiten beispielsweise zufällig aus, hat das Modell nicht viel gelernt.

Viele Modellanbieter geben Wahrscheinlichkeiten zurück, die von ihren Modellen als Logprobs generiert wurden (*https://oreil.ly/VAUl6*). *Logprobs* – die Kurzform von *Log Probabilities* (Log-Wahrscheinlichkeiten) – sind Wahrscheinlichkeiten auf einer logarithmischen Skala. Diese wird bei der Arbeit mit den Wahrscheinlichkeiten in einem neuronalen Netz bevorzugt genutzt, weil sie dabei hilft, das Unterlaufproblem zu verringern (*https://de.wikipedia.org/wiki/Arithmetischer_Unterlauf*).[26] Ein Sprachmodell arbeitet vielleicht mit einer Vokabulargröße von 100.000, was bedeutet, dass die Wahrscheinlichkeiten für viele der Tokens zu klein sein können, um sie durch einen Rechner darzustellen. Die kleinen Werte werden dann alle auf null abgerundet. Die logarithmische Skala hilft dabei, dieses Problem zu verringern.

In Abbildung 2.17 sehen Sie den Ablauf beim Berechnen von Logits, Wahrscheinlichkeiten und Logprobs.

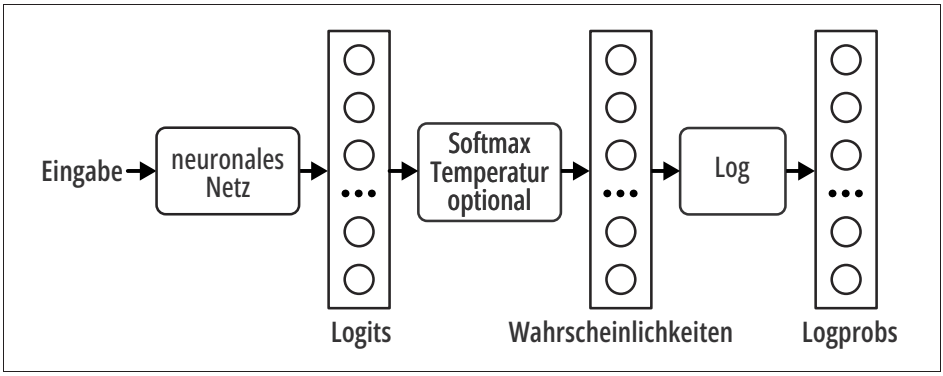

Abbildung 2.17: Wie Logits, Wahrscheinlichkeiten und Logprobs berechnet werden

25 Es wird eine arg-max-Funktion ausgeführt (*https://de.wikipedia.org/wiki/Arg_max*).

26 Das Unterlaufproblem tritt auf, wenn eine Zahl zu klein ist, um sie in einem gegebenen Format darstellen zu können, was dazu führt, dass sie auf null abgerundet wird.

Wie Sie im Buch sehen werden, sind Logprobs nützlich, um Anwendungen zu bauen (insbesondere zur Klassifikation), zu evaluieren und um zu verstehen, wie Modelle unter der Motorhaube funktionieren. Allerdings stellen viele Modellanbieter ihre Logprobs aktuell nicht zur Verfügung oder bieten nur eine eingeschränkte Logprobs-API.[27] Die APIs sind sehr wahrscheinlich aus Sicherheitsgründen eingeschränkt, weil offengelegte Logprobs eines Modells es anderen erleichtern, ein Modell zu replizieren.

top-k

top-k ist eine Sampling-Strategie, mit der die Rechenlast reduziert werden kann, ohne die Diversität der Modellantworten zu sehr zu reduzieren. Sie erinnern sich: Es kommt eine Softmax-Schicht zum Einsatz, um die Wahrscheinlichkeitsverteilung über alle mögliche Werte zu berechnen. Für Softmax sind zwei Durchläufe über alle möglichen Werte erforderlich: einer, um die exponentielle Summe $\Sigma_j e^{x_j}$ zu berechnen, und einer, um für jeden Wert $\frac{e^{x_i}}{\Sigma_j e^{x_j}}$ zu erhalten. Bei einem Sprachmodell mit einem großen Vokabular ist dieser Prozess sehr rechenintensiv.

Um das Problem zu vermeiden, wählen wir nach dem Berechnen der Logits durch das Modell die größten *k* (top-k) Logits aus und führen nur für diese Logits die Softmax-Berechnung durch. Abhängig davon, wie divers Ihre Anwendung sein soll, kann *k* irgendwo zwischen 50 und 500 liegen – viel weniger als die Größe eines Modellvokabulars. Dann sampelt das Modell nur aus diesen besten Werten. Ein kleinerer Wert für *k* macht den Text vorhersagbarer, aber auch weniger interessant, da das Modell auf einen kleineren Satz wahrscheinlicher Wörter beschränkt wird.

top-p

Beim top-k-Sampling ist die Anzahl der berücksichtigten Werte fest auf *k* beschränkt. Aber dieser Wert sollte sich abhängig von der Situation ändern. So sollte beispielsweise beim Prompt »Magst du Musik? Antworte nur mit Ja oder Nein.« die Anzahl der Werte zwei sein: Ja und Nein. Beim Prompt »Was ist der Sinn des Lebens?« sollte die Anzahl der zu berücksichtigenden Werte viel größer sein.

top-p, auch bekannt als *Nucleus Sampling*, erlaubt eine dynamischere Auswahl von Werten, aus denen gesampelt wird. Beim top-p-Sampling summiert das Modell die Wahrscheinlichkeiten der wahrscheinlichsten nächsten Werte in absteigender Reihenfolge auf und stoppt, wenn die Summe den Wert *p* erreicht. Nur die Werte innerhalb dieser kumulierten Wahrscheinlichkeit werden berücksichtigt. Häufige Werte für das top-p-(Nucleus-)Sampling in Sprachmodellen reichen von 0,9 bis 0,95. Ein top-p-Wert von 0,9 bedeutet beispielsweise, dass das Modell den kleinsten Satz an Werten berücksichtigt, deren kumulierte Wahrscheinlichkeit 90 % überschreitet.

27 Genauer gesagt, zeigt die OpenAI-API aktuell nur die Logprobs der 20 wahrscheinlichsten Tokens an (*https://oreil.ly/jWEsP*). Früher konnten Sie die Logprobs von beliebigen von den Usern bereitgestellten Texten erhalten, das wurde aber im September 2023 beendet (*https://x.com/xuanalogue/status/1707757449900437984*).

Schauen wir uns die Wahrscheinlichkeiten aller Tokens in Abbildung 2.18 an. Hat top-p den Wert 90%, werden nur »ja« und »vielleicht« berücksichtigt, da deren kumulierte Wahrscheinlichkeit größer als 90% ist. Hat top-p den Wert 99%, werden »ja«, »vielleicht« und »nein« berücksichtigt.

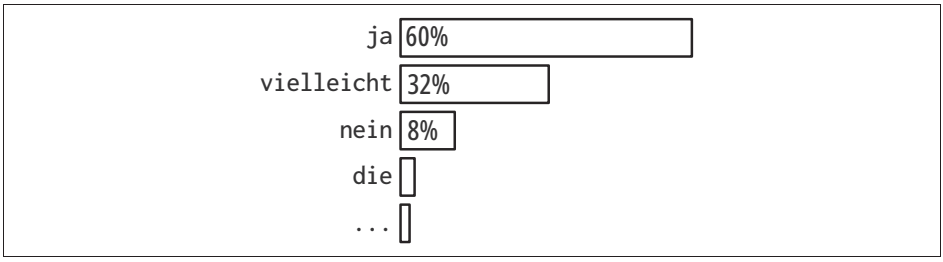

Abbildung 2.18: Beispiel für Token-Wahrscheinlichkeiten

Anders als bei top-k verringert top-p nicht unbedingt die Rechenlast durch Softmax. Der Vorteil liegt darin, dass der Fokus nur auf die relevantesten Werte für jeden Kontext gelegt wird und die Ausgaben damit kontextuell passender sein können. In der Theorie scheint das top-p-Sampling keine großen Vorteile zu bieten. Aber die Praxis hat gezeigt, dass top-p-Sampling sehr gut funktioniert, weshalb es auch immer beliebter wird.

Eine ähnliche Sampling-Strategie ist min-p (*https://github.com/huggingface/transformers/issues/27670*), bei der Sie die minimale Wahrscheinlichkeit definieren, die ein Token erreichen muss, um beim Sampling berücksichtigt zu werden.

Stoppbedingung

Ein autoregressives Sprachmodell generiert Sequenzen mit Tokens, indem ein Token nach dem anderen erzeugt wird. Eine lange Ausgabesequenz braucht mehr Zeit, kostet mehr Rechenleistung (Geld)[28] und kann User manchmal nerven. Wir wollen möglicherweise eine Bedingung für das Modell definieren, mit der die Sequenz gestoppt wird.

Eine einfache Methode besteht darin, Modelle zu bitten, nach einer festen Anzahl von Tokens zu stoppen. Der Nachteil daran ist, dass die Ausgabe wahrscheinlich mitten in einem Satz endet. Eine andere Methode ist die Verwendung von *Stopptokens* oder *Stoppwörtern*. Sie können beispielsweise ein Modell bitten, mit dem Generieren aufzuhören, wenn es auf das End-of-Sequence-Token trifft. Stoppbedingungen sind hilfreich, um Latenz und Kosten niedrig zu halten.

Der Nachteil eines frühen Stoppens zeigt sich, wenn das Modell die Ausgabe in einem bestimmten Format generieren soll. Dann kann ein vorzeitiges Stoppen dazu führen, dass die Ausgaben nicht richtig formatiert sind. Bitten Sie beispielsweise das Modell, JSON auszugeben, kann ein frühzeitiges Stoppen dazu führen, dass dem JSON zum Beispiel schließende Klammern fehlen, sodass es sich nur schlecht parsen lässt.

28 Kostenpflichtige Modell-APIs berechnen oft nach der Anzahl der Ausgabetokens.

Test Time Compute

Der letzte Abschnitt hat beschrieben, wie ein Modell das nächste Token sampeln kann. Dieser Abschnitt dreht sich darum, wie ein Modell die gesamte Ausgabe sampelt.

Ein einfacher Weg zum Verbessern der Antwortqualität eines Modells führt über *Test Time Compute*: Statt nur eine Antwort pro Anfrage zu generieren, erzeugen Sie mehrere Antworten, um die Chance auf eine gute Antwort zu erhöhen. Eine Möglichkeit für Test Time Compute ist die Best-of-N-Technik, die weiter oben in diesem Kapitel behandelt wurde – Sie erzeugen zufällig mehrere Ausgaben und wählen die aus, die am besten funktioniert. Aber Sie können beim Erzeugen mehrerer Ausgaben auch strategischer vorgehen. Statt beispielsweise alle Ausgaben unabhängig voneinander zu generieren, was zu vielen wenig vielversprechenden Kandidaten führen kann, können Sie auch *Beam Search* nutzen (*https://en.wikipedia.org/wiki/Beam_search*), um bei jedem Schritt der Sequenzerzeugung eine feste Anzahl der vielversprechendsten Kandidaten zu generieren (dem Beam).

Eine einfache Strategie zum Verbessern der Effektivität von Test Time Compute ist das Erhöhen der Diversität der Ausgaben, weil ein diverserer Satz an Optionen eher zu besseren Kandidaten führt. Nutzen Sie das gleiche Modell, um unterschiedliche Optionen zu erzeugen, ist es oft eine gute Idee, die Sampling-Variablen des Modells zu variieren, um die Ausgaben zu diversifizieren.

Auch wenn Sie im Allgemeinen durch das Sampeln mehrerer Ausgaben eine gewisse Verbesserung der Modellperformance erlangen können, ist es teuer. Im Schnitt kostet das Generieren von zwei Ausgaben ungefähr doppelt so viel wie das Generieren einer Ausgabe.[29]

Ich nutze den Begriff *Test Time Compute*, um konsistent zur bestehenden Literatur zu sein, auch wenn viele frühe Reviewer protestiert haben, weil sie ihn verwirrend finden. In der KI-Forschung wird Test Time im Allgemeinen verwendet, um sich auf Inferenz zu beziehen, weil beim Forschen oft nur Inferenz genutzt wird, um ein Modell zu testen. Aber diese Technik lässt sich auch ganz allgemein auf produktiv eingesetzte Modelle anwenden. *Test Time Compute* heißt es deshalb, weil die Anzahl an Ausgaben, die Sie sampeln können, davon abhängt, wie viel Rechenleistung Sie für jeden Inferenzaufruf spendieren können.

Um die beste Auswahl zu wählen, können Sie entweder den Usern mehrere Ausgaben präsentieren und sie dann die auswählen lassen, die für sie am besten passt, oder Sie entwickeln eine Methode, um sich für die beste zu entscheiden. Eine Auswahlmethode ist, die Ausgabe mit der höchsten Wahrscheinlichkeit zu verwenden. Die Ausgabe eines Sprachmodells ist eine Sequenz aus Tokens, und jedes Token besitzt

29 Sie können gewisse Maßnahmen ergreifen, um die Kosten für das Generieren mehrerer Ausgaben für die gleiche Eingabe zu reduzieren. So muss die Eingabe beispielsweise nur einmal verarbeitet werden und kann dann für alle Ausgaben zum Einsatz kommen.

eine Wahrscheinlichkeit, die vom Modell berechnet wurde. Die Wahrscheinlichkeit einer Ausgabe ist das Produkt der Wahrscheinlichkeiten all ihrer Tokens.

Schauen Sie sich die Token-Sequenz [»Ich«, »liebe«, »Essen«] an. Beträgt die Wahrscheinlichkeit für »Ich« 0,2, die für »liebe« bei gegebenem »Ich« 0,1 und die Wahrscheinlichkeit für »Essen« bei gegebenem »Ich« und »lieben« 0,3, ist die Wahrscheinlichkeit der Sequenz $0{,}2 \times 0{,}1 \times 0{,}3 = 0{,}006$. Mathematisch kann das wie folgt beschrieben werden:

$$p(\text{Ich liebe Essen}) = p(\text{Ich}) \times p(\text{liebe} \mid \text{Ich}) \times p(\text{Essen} \mid \text{Ich, liebe})$$

Denken Sie daran, dass es einfacher ist, mit Wahrscheinlichkeiten auf einer logarithmischen Skala zu arbeiten. Der Logarithmus eines Produkts entspricht der Summe von Logarithmen, daher ist die Logprob einer Token-Sequenz die Summe der Logprobs aller Tokens in der Sequenz:

$$\text{logprob(Ich liebe Essen)} = \text{logprob(Ich)} + \text{logprob(liebe} \mid \text{Ich)} + \text{logprob(Essen} \mid \text{Ich, liebe)}$$

Durch das Aufsummieren haben längere Sequenzen eher eine niedrigere Gesamt-Logprob (Logprob-Werte sind im Allgemeinen negativ, weil der Logarithmus eines Werts zwischen 0 und 1 negativ ist). Um eine Bevorzugung kürzerer Sequenzen zu vermeiden, können Sie die durchschnittliche Logprob nutzen, indem Sie die Summe einer Sequenz durch ihre Länge teilen. Nach dem Sampeln mehrerer Ausgaben wählen Sie die mit der höchsten durchschnittlichen Logprob aus. Aktuell kommt dieses Vorgehen bei der OpenAI-API zum Einsatz.[30]

Eine andere Auswahlmethode ist der Einsatz eines Reward Model, um jede Ausgabe mit einem Score zu versehen (wie im vorherigen Abschnitt besprochen). Sie erinnern sich: Sowohl Stitch Fix (*https://oreil.ly/1Njeh*) als auch Grab (*https://oreil.ly/l21nr*) wählen die Ausgaben anhand der höchsten Scores, die durch ihre Reward Models oder Verifier bestimmt werden. Nextdoor (*https://oreil.ly/-HQIB*) hat festgestellt, dass der Einsatz eines Reward Model der zentrale Faktor beim Verbessern ihrer Anwendungsperformance war (2023).

OpenAI hat auch Verifier darin trainiert, ihren Modellen dabei zu helfen, die beste Lösung für mathematische Probleme auszuwählen (Cobbe et al., 2021, *https://oreil.ly/R_uvq*). Es wurde festgestellt, dass der Einsatz eines Verifier die Leistung des Modells signifikant gesteigert hat.

Tatsächlich führte die Verwendung eines Verifier zu einem Performance-Boost, der dem Vergrößern des Modells auf das 30-Fache entspricht.

Ein 100-Millionen-Parameter-Modell, das einen Verifier verwendet, kann also genauso gut arbeiten wie ein 3-Milliarden-Parameter-Modell ohne Verifier.

DeepMind hat den Wert von Test Time Compute noch umfassender gezeigt und sagt, dass das Skalieren von Test Time Compute (zum Beispiel durch den Einsatz

30 Aktuell können Sie in der OpenAI-API den Parameter `best_of` auf einen bestimmten Wert setzen – zum Beispiel 10 –, um OpenAI-Modelle zu bitten, die Ausgabe mit der höchsten durchschnittlichen Logprob von zehn verschiedenen Ausgaben zurückzugeben (*https://oreil.ly/XYugZ*).

von mehr Rechenleistung, um beim Inferieren mehr Ausgaben zu erzeugen) effizienter sein kann, als die Modellparameter zu skalieren (Snell et al., 2024, *https://arxiv.org/abs/2408.03314*). Der gleiche Artikel stellt eine interessante Frage: Wenn ein LLM eine feste, aber nicht triviale Menge an Inference Time Compute nutzen kann – wie viel besser kann dann seine Leistung bei einem anspruchsvollen Prompt sein?

Bei dem Experiment von OpenAI führte das Sampeln von mehr Ausgaben zu einer besseren Performance, aber nur bis zu einem gewissen Punkt. Dieser lag bei 400 Ausgaben. Darüber hinaus sank die Leistung wieder, wie in Abbildung 2.19 zu sehen ist. Die Hypothese war, dass mit steigender Zahl von gesampelten Ausgaben auch die Chance auf sich widersprechende Ausgaben wächst, was den Verifier in die Irre führen kann. Ein Stanford-Experiment hat allerdings einen anderen Schluss nahegelegt. »Monkey Business« (Brown et al., 2024, *https://oreil.ly/8YNwQ*) hat ermittelt, dass die Anzahl der gelösten Probleme oft logarithmisch-linear mit der Zahl der Samples von 1 bis 10.000 steigt. Es ist zwar interessant, darüber nachzudenken, ob sich Test Time Compute unendlich skalieren ließe, aber ich glaube nicht, dass irgendjemand produktiv für jede Eingabe 400 oder 10.000 unterschiedliche Ausgaben sampeln würde. Die Kosten dafür wären astronomisch.

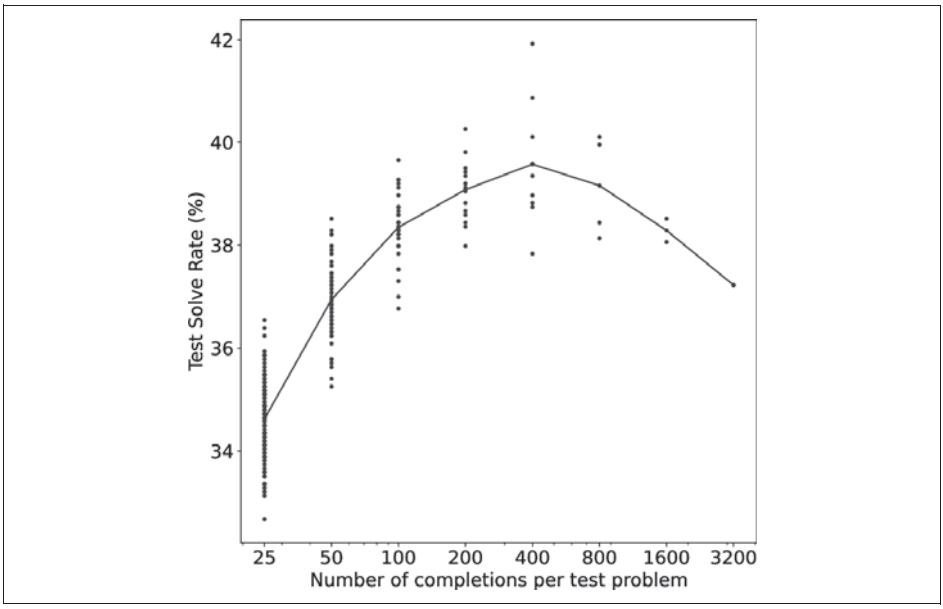

Abbildung 2.19: OpenAI hat herausgefunden, dass das Sampeln von mehr Ausgaben zu einer besseren Leistung führt – aber nur bis zu 400 Ausgaben (2021, https://arxiv.org/abs/2110.14168).

Sie können auch anwendungsspezifische Heuristiken nutzen, um die beste Antwort auszuwählen. Profitiert Ihre Applikation beispielsweise von kürzeren Antworten, können Sie den kürzesten Kandidaten wählen. Wandelt Ihre Anwendung natürliche Sprache in SQL-Abfragen um, kann das Modell so lange Ausgaben generieren lassen, bis es eine gültige SQL-Abfrage produziert.

Eine besonders interessante Anwendung von Test Time Compute ist das Überwinden der Latenzproblematik. Bei manchen Abfragen – insbesondere Chain-of-Thought-Abfragen – kann ein Modell viel Zeit für das Vervollständigen der Antwort benötigen. Kittipat Kampa, Head of AI bei TIFIN, hat mir erzählt, dass sein Team sein Modell bittet, parallel mehrere Antworten zu generieren und dem User dann die erste Antwort auszugeben, die vollständig und gültig ist.

Das Auswählen der häufigsten Ausgabe aus einem Satz an Ausgaben kann besonders bei Aufgaben nützlich sein, für die exakte Antworten erwartet werden.[31] So kann ein Modell beispielsweise ein mathematisches Problem mehrfach lösen und die am häufigsten gegebene Antwort als endgültige Lösung zurückgeben. Genauso kann ein Modell bei einer Multiple-Choice-Frage die häufigste Ausgabeoption wählen. Das war auch das Vorgehen von Google beim Evaluieren von Gemini mit der MMLU-Benchmark. Es wurden für jede Frage 32 Ausgaben gesampelt. So konnte das Modell einen höheren Score erreichen, als wenn es nur eine Ausgabe pro Frage erzeugt hätte.

Ein Modell wird als robust angesehen, wenn es seine Ausgaben bei kleinen Variationen der Eingabe nicht dramatisch verändert. Je weniger robust ein Modell ist, desto eher kann es vom Sampeln mehrerer Ausgaben profitieren.[32] Wir haben für ein Projekt KI genutzt, um bestimmte Informationen aus einem Bild eines Produkts zu extrahieren. Dabei haben wir festgestellt, dass unser Modell für das gleiche Bild nur in der Hälfte der Fälle die Informationen auslesen konnte. In der anderen Hälfte behauptete das Modell, das Bild sei zu verschwommen oder der Text zu klein, um ihn lesen zu können. Indem wir aber bei jedem Bild drei Anläufe nahmen, war das Modell dazu in der Lage, für die meisten Bilder die korrekten Informationen zu extrahieren.

Strukturierte Ausgaben

Häufig brauchen Sie im produktiven Einsatz Modelle, die Ausgaben generieren, die bestimmte Formate einhalten. Strukturierte Ausgaben sind in den folgenden beiden Szenarien entscheidend:

1. *Aufgaben, die strukturierte Ausgaben erfordern.* Die häufigste Kategorie von Aufgaben in diesem Szenario ist das semantische Parsen. Dazu gehört das Umwandeln natürlicher Sprache in ein strukturiertes, maschinenlesbares Format. Text-to-SQL ist ein Beispiel für semantisches Parsen – die Ausgaben müssen gültige SQL-Abfragen sein. Das semantische Parsen erlaubt Usern, mithilfe natürlicher Sprache (zum Beispiel Deutsch) mit APIs zu interagieren. Text-to-PostgreSQL ermöglicht Usern zum Beispiel, eine Postgres-Datenbank über deutschsprachige Abfragen wie »Wie groß war der durchschnittliche monatli-

31 Wang et al. (2023, *https://arxiv.org/abs/2203.11171*) haben diesen Ansatz »Self-Consistency« genannt.

32 Bei einem brüchigen Modell besteht das optimale Vorgehen allerdings darin, es durch ein anderes Modell auszutauschen.

che Umsatz in den letzten 6 Monaten?« anzusprechen, statt diese in PostgreSQL schreiben zu müssen.

Dies ist ein Beispiel eines Prompts für GPT-4o für Text-to-Regex. Die Ausgaben sind tatsächliche Ausgaben von GPT-4o:

```
System-Prompt

Gib mir für eine Eingabe eine Regex, die alle Möglichkeiten abdeckt, wie die
Eingabe geschrieben werden kann. Gib nur die Regex zurück.

Beispiel: US-Telefonnummer -> \+?1?\s?(\()?(\d{3})(?(1)\))[-.\s]?(\d{3})
[-.\s]?(\d{4})

User-Prompt
E-Mail-Adresse ->

GPT-4o
 [a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}

User-Prompt
Datumswerte ->

GTP-4o
(0?[1-9]|[12][0-9]|3[01])[.\-/](0?[1-9]|1[0-2])[.\-/](\d{2}|\d{4})
```

Andere Kategorien von Aufgaben sind Klassifikationen, bei denen die Ausgaben gültige Klassen sein müssen.

2. *Aufgaben, bei denen die Ausgaben von auf dem Modell aufbauenden Anwendungen genutzt werden.* Dabei ist für die Aufgabe an sich gar keine strukturierte Ausgabe erforderlich, aber weil die Ausgabe von anderen Anwendungen genutzt wird, muss sie dort geparst werden können.

   Nutzen Sie beispielsweise ein KI-Modell zum Schreiben einer E-Mail, muss die E-Mail selbst nicht strukturiert sein. Aber eine darauf aufbauende Anwendung, die diese E-Mail verwendet, benötigt sie vielleicht in einem bestimmten Format – beispielsweise als JSON-Dokument mit spezifischen Schlüsseln wie `{"title": [TITLE], "body": [EMAIL BODY]}`.

   Das ist besonders bei Agenten-Workflows wichtig, da dort die Ausgaben eines Modells häufig als Eingaben an Tools übergeben werden, die das Modell nutzen kann (was wir in Kapitel 6 besprechen).

Frameworks, die strukturierte Ausgaben unterstützen, sind beispielsweise guidance (*https://github.com/guidance-ai/guidance*), outlines (*https://github.com/dottxt-ai/outlines*), instructor (*https://github.com/instructor-ai/instructor*) oder llama.cpp (*https://github.com/ggerganov/llama.cpp/discussions/177*). Jeder Modellanbieter kann auch seine eigene Technik nutzen, um die Fähigkeit seines Modells zum Generieren einer strukturierten Ausgabe zu verbessern. OpenAI war der erste Anbieter, der einen JSON-Modus in seiner Textgenerierungs-API eingeführt hat (*https://oreil.ly/NxZDF*). Beachten Sie, dass der JSON-Modus einer API meist nur garantiert, dass es sich bei den Ausgaben um gültiges JSON handelt – nicht aber bei dem Inhalt der JSON-Objekte. Die ansonsten korrekt generierten JSONs werden eventuell auch abgeschnitten

und sind damit nicht zu parsen, wenn das Generieren zu früh abbricht – zum Beispiel wenn die maximale Token-Ausgabelänge erreicht wurde. Setzt man diese wiederum zu hoch, werden die Antworten des Modells zu langsam und zu teuer.

In Abbildung 2.20 sehen Sie zwei Beispiele für den Einsatz von Beschränkungen beim Generieren von Ausgaben – einmal auf einen Satz von Optionen und einmal auf eine Regex.

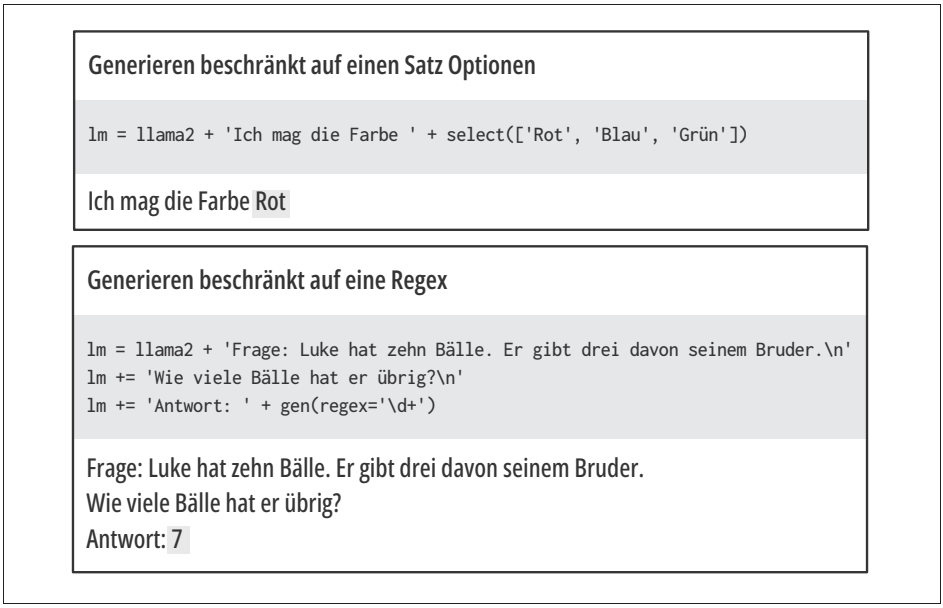

Abbildung 2.20: Einsatz von Beschränkungen beim Generieren von Ausgaben

Sie können ein Modell auf unterschiedlichen Ebenen des KI-Stacks dazu anleiten, strukturierte Ausgaben zu generieren: Prompting, Post-Processing, Test Time Compute, Constraint Sampling und Optimieren. Die ersten drei sind eher Notlösungen. Sie funktionieren am besten, wenn das Modell schon ziemlich gut beim Generieren strukturierter Ausgaben ist und nun noch ein wenig in die richtige Richtung geschubst werden muss. Für eine intensivere Behandlung brauchen Sie Constraint Sampling oder Optimieren.

Test Time Compute wurde eben erst im vorherigen Abschnitt behandelt – generieren Sie Ausgaben, bis eine dem erwarteten Format entspricht. Dieser Abschnitt konzentriert sich auf die anderen vier Ansätze.

Prompting

Prompting ist der erste Ansatz für das Generieren strukturierter Ausgaben. Sie können ein Modell anweisen, Ausgaben in einem beliebigen Format zu erzeugen. Aber ob ein Modell diesen Anweisungen folgen kann, hängt von seinen Fähigkeiten zum Befolgen von Vorgaben (wird in Kapitel 4 besprochen) und der Klarheit der Anweisung ab (besprochen in Kapitel 5). Modelle werden zwar zunehmend besser beim

Befolgen von Anweisungen, aber es gibt keine Garantie, dass sie dies immer tun.[33] Ein paar Prozentpunkte ungültiger Modellausgaben können für viele Anwendungen trotzdem inakzeptabel sein.

Um den Anteil gültiger Ausgaben zu erhöhen, nutzen manche Menschen KI zum Überprüfen und/oder Korrigieren der Ausgabe des ursprünglichen Prompts. Das ist ein Beispiel für den Ansatz einer KI als Sachverständige, der in Kapitel 3 behandelt wird. Für jede Ausgabe gibt es dann mindestens zwei Modellanfragen: eine zum Generieren der Ausgabe und eine zum Validieren dieser Ausgabe. Die ergänzende Validierungsschicht kann zwar die Gültigkeit der Ausgaben verbessern, aber die zusätzlichen Kosten und die Latenz durch die Validierungsabfragen können diesen Ansatz für manche zu teuer machen.

Post-Processing

Post-Processing ist einfach und günstig, kann aber erstaunlich gut funktionieren. Während meiner Zeit als Dozentin habe ich festgestellt, dass Studentinnen und Studenten dazu tendieren, immer die gleichen Fehler zu machen. Als ich mit der Arbeit an Foundation Models begann, fiel mir das Gleiche auf. Ein Modell tendiert dazu, bei verschiedenen Anfragen ähnliche Fehler zu machen. Finden Sie also die üblichen Fehler eines Modells, können Sie potenziell ein Skript schreiben, das diese korrigiert. Fehlt dem generierten JSON-Objekt beispielsweise eine schließende geschweifte Klammer, können Sie diese manuell hinzufügen. Der defensive YAML-Parser von LinkedIn hat den Anteil korrekter YAML-Ausgaben von 90 % auf 99,99 % gesteigert (Bottaro und Ramgopal, 2020, *https://oreil.ly/ZTRaA*).

JSON und YAML sind verbreitete Textformate. LinkedIn hat festgestellt, dass ihr zugrunde liegendes Modell, GPT-4, mit beiden funktioniert, aber sie haben sich für YAML als Ausgabeformat entschieden, weil es kompakter ist und daher weniger Ausgabetokens als JSON benötigt (Bottaro and Ramgopal, 2020).

Das Post-Processing funktioniert nur, wenn sich die Fehler leicht beheben lassen. Das ist oft dann der Fall, wenn die Ausgaben eines Modells schon größtenteils korrekt sind und es nur gelegentlich kleine Fehler gibt.

Constraint Sampling

Constraint Sampling ist eine Technik, um das Generieren von Text durch bestimmte Beschränkungen in eine gewünschte Richtung zu lenken. Sie wird im Allgemeinen von Tools zur strukturierten Ausgabe genutzt.

Grob gesagt, sampelt ein Modell zum Generieren eines Tokens aus Werten, die die Bedingungen erfüllen. Sie erinnern sich: Um ein Token zu generieren, gibt Ihr Modell zuerst einen Logit-Vektor aus, bei dem jedes Logit mit einem möglichen Token

33 Aktuell habe ich – abhängig von der Anwendung und dem Modell – beim Generieren von JSON-Objekten eine Erfolgsquote zwischen 0 % und ziemlich hohen 90 % gesehen.

korrespondiert. Durch Constraint Sampling wird dieser Logit-Vektor so gefiltert, dass er nur die Tokens beibehält, die die Beschränkungen erfüllen. Dann wird aus diesen gültigen Tokens gesampelt. Diesen Prozess stellt Abbildung 2.21 dar.

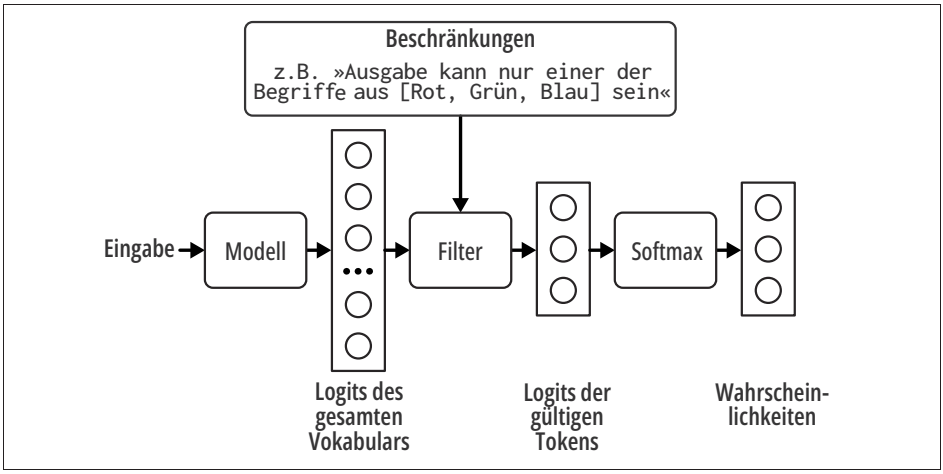

Abbildung 2.21: Logits ausfiltern, die nicht die Beschränkungen erfüllen, um nur aus gültigen Ausgaben zu sampeln

Im Beispiel in Abbildung 2.21 lässt sich die Beschränkung einfach durch das Filtern umsetzen. Aber die meisten Fälle sind nicht so einfach. Sie brauchen eine Grammatik, die festlegt, was bei jedem Schritt erlaubt ist und was nicht. So gibt beispielsweise die JSON-Grammatik vor, dass Sie nach einer { keine andere { haben dürfen, sofern sie nicht Teil eines Strings ist, zum Beispiel bei `{"key": "{{string}}"}`.

Das Erstellen dieser Grammatik und das Einbinden in den Sampling-Prozess ist nicht trivial. Weil jedes Ausgabeformat – JSON, YAML, Regex, CSV und so weiter – seine eigene Grammatik benötigt, lässt sich das Constraint Sampling nicht so gut verallgemeinern. Sein Nutzen ist auf die Formate beschränkt, deren Grammatik von externen Tools oder durch Ihr Team unterstützt wird. Die Grammatik-Verifikation kann zudem die Generierungslatenz größer werden lassen (Brandon T. Willard, 2024, *https://oreil.ly/hNRf4*).

Manche sind gegen Constraint Sampling, weil sie davon ausgehen, dass die Ressourcen, die dafür erforderlich sind, eher in das Trainieren der Modelle investiert werden sollten, damit diese den Anweisungen besser folgen können.

Optimieren

Das Optimieren eines Modells durch Beispiele, die Ihr gewünschtes Format widerspiegeln, ist der effektivste und generalisierbarste Ansatz, Modelle dazu zu bringen, Ausgaben in diesem Format zu generieren.[34] Das kann mit jedem Format funktio-

34 Es funktioniert auch, ein Modell von Grund auf mit Daten zu trainieren, die im gewünschten Format vorhanden sind, aber in diesem Buch geht es nicht darum, Modelle von Grund auf zu entwickeln.

nieren. Während ein einfaches Optimieren nicht unbedingt garantiert, dass das Modell immer das gewünschte Format ausgibt, ist es doch zuverlässiger als Prompting.

Für bestimmte Aufgaben können Sie das Ausgabeformat sicherstellen, indem Sie die Modellarchitektur vor dem Optimieren anpassen. So können Sie beispielsweise beim Klassifizieren einen Classifier-Head an die Architektur des Foundation Model anfügen, um sicherzustellen, dass die Ausgaben nur aus einer der zuvor angegebenen Klassen bestehen. Die Architektur sieht wie in Abbildung 2.22 aus.[35] Dieser Ansatz wird auch als *Feature-basierter Transfer* bezeichnet, und er wird zusammen mit anderen Techniken des Transferlernens in Kapitel 7 besprochen.

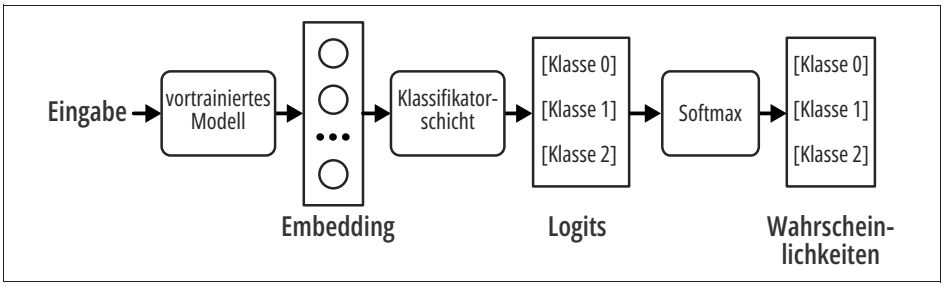

Abbildung 2.22: Durch das Ergänzen Ihres Basismodells um einen Classifier-Head verwandeln Sie es in einen Klassifikator. In diesem Beispiel arbeitet der Klassifikator mit drei Klassen.

Beim Optimieren können Sie das gesamte Modell end-to-end oder nur Teile des Modells neu trainieren, wie zum Beispiel diesen Classification-Head. Für ein End-to-End-Training sind mehr Ressourcen erforderlich, aber es verspricht auch eine bessere Performance.

Wir benötigen Techniken für strukturierte Ausgaben, weil wir davon ausgehen müssen, dass das Modell allein nicht dazu in der Lage ist, diese zu erzeugen. Aber wenn Modelle leistungsfähiger werden, können wir auch erwarten, dass sie beim Befolgen von Anweisungen besser werden. Ich vermute, dass es in Zukunft einfacher werden wird, Modelle mit minimalem Prompting genau das ausgeben zu lassen, was wir benötigen, und dass diese Techniken weniger wichtig werden.

Die statistische Natur der KI

Die Art und Weise, wie KI-Modelle ihre Antworten sampeln, sorgt für ihre statistische Natur. Schauen wir uns ein Beispiel dafür an, was wir damit meinen. Angenommen, Sie wollen wissen, was die beste Landesküche der Welt ist. Fragen Sie Ihren Freund mit einem Abstand von einer Minute zwei Mal, wird dieser vermutlich beide Male das Gleiche antworten. Stellen Sie einem KI-Modell zwei Mal die gleiche Frage, kann sich die Antwort ändern. Geht ein KI-Modell davon aus, dass die vietnamesische Küche eine 70%ige Chance hat, die beste Küche der Welt zu sein, während die

35 Manche Optimierungsservices erledigen das für Sie automatisch. Mit dem Optimierungsservice von OpenAI (*https://oreil.ly/sljei*) konnten Sie beim Training einen Classifier-Head ergänzen, aber aktuell wurde dieses Feature deaktiviert.

italienische Küche eine Chance von 30 % hat, wird es in 70 % der Fälle »vietnamesische Küche« und in 30 % »italienische Küche« antworten. Das Gegenteil davon ist *Determinismus*, bei dem das Ergebnis ohne zufällige Variationen bestimmt werden kann.

Diese statistische Natur kann zu Inkonsistenzen und Halluzinationen führen. Bei einer *Inkonsistenz* generiert ein Modell für die gleichen oder nur leicht unterschiedliche Prompts sehr unterschiedliche Antworten. Bei einer *Halluzination* gibt ein Modell eine Antwort, die nicht auf Fakten basiert. Stellen Sie sich vor, jemand hat im Internet einen Artikel darüber geschrieben, dass alle US-Präsidenten Aliens sind, und dieser Artikel ist Teil der Trainingsdaten. Das Modell wird später wahrscheinlich ausgeben, dass der aktuelle US-Präsident ein Alien ist. Aus Sicht von jemandem, der nicht daran glaubt, dass US-Präsidenten Aliens sind, denkt sich das Modell das aus.

Foundation Models werden normalerweise mit einer großen Menge an Daten trainiert. Es handelt sich dabei um Zusammenstellungen der Meinungen der Massen, wodurch sich darin eine ganze Welt von Möglichkeiten findet. Alles mit einer Wahrscheinlichkeit größer null – egal wie abwegig oder falsch – kann von der KI generiert werden.[36]

Diese Eigenschaft sorgt dafür, dass das Bauen von KI-Anwendungen gleichzeitig spannend und herausfordernd ist. Wie wir in diesem Buch sehen werden, zielt ein Großteil des Aufwands beim AI Engineering darauf ab, diese statistische Natur in den Griff zu bekommen und sie abzudämpfen.

Die statistische Natur führt auch dazu, dass KI für kreative Aufgaben sehr gut einsetzbar ist. Was ist denn Kreativität anderes als die Fähigkeit, die üblichen Pfade zu verlassen – außerhalb der Box zu denken? KI ist eine großartige Begleiterin für kreativ arbeitende Menschen. Sie kann unbegrenzt Ideen hervorbringen und noch nie zuvor gesehene Designs generieren. Aber die gleiche statistische Natur kann für alle anderen ein echtes Problem sein.[37]

Inkonsistenz

Modellinkonsistenzen zeigen sich in zwei Szenarien:

1. *Gleiche Eingaben – unterschiedliche Ausgaben*: Übergibt man dem Modell zwei Mal den gleichen Prompt, erhält man zwei sehr unterschiedliche Ausgaben.
2. *Leicht unterschiedliche Eingaben – drastisch unterschiedliche Ausgaben*: Bei einer etwas anderen Eingabe, wie zum Beispiel einer unabsichtlichen Großschreibung eines Buchstabens, erhält man eine sehr andere Ausgabe.

36 Wie das Meme so schön sagt: Die Chancen sind klein, aber nie null (*https://x.com/OxfordDiplomat/status/1424388443010998277?lang=en*).

37 Im Dezember 2023 schaute ich mir für ein KI-Unternehmen, das ich beraten habe, die Kundensupportanfragen aus drei Monaten an und stellte fest, dass es bei einem Fünftel der Fragen darum ging, wie mit der Inkonsistenz von KI-Modellen umzugehen ist. Bei einem Panel, an dem ich zusammen mit Drew Houston (CEO von Dropbox) und Harrison Chase (CEO von LangChain) im Juli 2023 teilnahm, stimmten wir alle darin überein, dass Halluzinationen das größte Hindernis für Unternehmensanwendungsfälle mit KI seien.

In Abbildung 2.23 sehen Sie ein Beispiel dafür, wie ich versuche, ChatGPT Essays bewerten zu lassen. Der gleiche Prompt hat mir bei gleicher Ausführung zwei verschiedene Scores geliefert: 3/5 und 5/5.

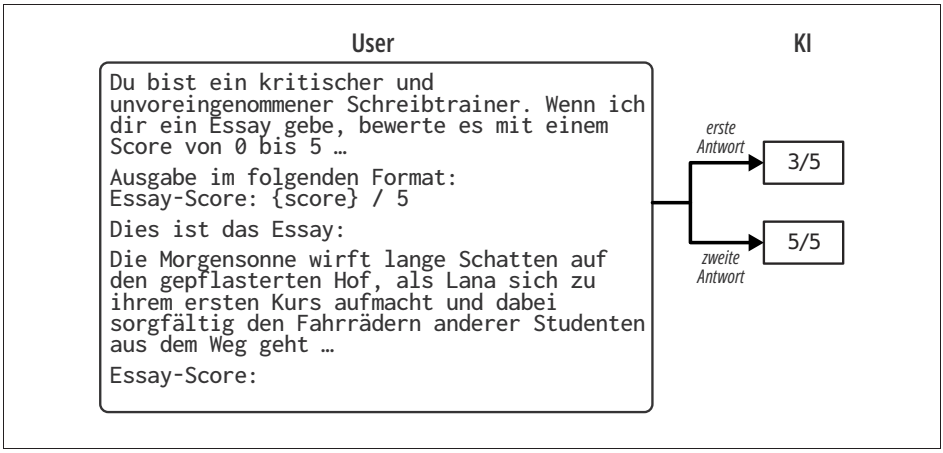

Abbildung 2.23: Die gleiche Eingabe kann beim selben Modell zu unterschiedlichen Ausgaben führen.

Inkonsistenzen können zu einer holprigen User Experience führen. Bei der Kommunikation zwischen Menschen erwarten wir eine gewisse Konsistenz. Stellen Sie sich vor, eine Person spricht Sie jedes Mal, wenn Sie sie treffen, mit einem anderen Namen an. Genauso erwarten die User eine gewisse Konsistenz, wenn sie mit KI kommunizieren.

Für das Szenario »gleiche Eingabe – unterschiedliche Ausgaben« gibt es mehrere Ansätze, um Inkonsistenzen kleinzuhalten. Sie können die Antwort cachen, sodass diese wieder zurückgegeben wird, wenn die Frage erneut gestellt wird. Sie können die Sampling-Variablen des Modells anpassen, wie zum Beispiel die Temperatur, top-p oder top-k, wie weiter oben besprochen wurde. Sie können auch die *Seed*-Variable anpassen, die Sie sich als Startwert für den Zufallszahlengenerator vorstellen können, der zum Sampeln des nächsten Tokens genutzt wird.

Aber auch wenn Sie all diese Variablen anpassen, gibt es keine Garantie dafür, dass Ihr Modell in 100 % der Fälle konsistent sein wird. Die Hardware, auf der die Ausgabegenerierung läuft, kann ebenfalls die Ausgabe beeinflussen, da unterschiedliche Rechner unterschiedliche Vorgehensweisen beim Ausführen der gleichen Anweisungen haben und verschiedene Zahlenbereiche verarbeiten können. Hosten Sie Ihre Modelle selbst, haben Sie eine gewisse Kontrolle über die verwendete Hardware. Nutzen Sie aber die API eines Modellanbieters, wie zum Beispiel OpenAI oder Google, liegt die Entscheidung, ob sie Ihnen eine Kontrolle darüber ermöglichen, bei diesen Anbietern.

Das Anpassen der Einstellungen zur Generierung der Ausgabe ist ein häufiges Vorgehen, aber das System gewinnt dabei nicht unbedingt Vertrauen. Stellen Sie sich eine Lehrkraft vor, die Ihnen nur dann konsistente Bewertungen gibt, wenn sie in

einem bestimmten Raum sitzt. Setzt sie sich in einen anderen Raum, werden die Bewertungen nicht mehr vorhersagbar sein.

Das zweite Szenario, »leicht unterschiedliche Eingaben – drastisch unterschiedliche Ausgaben«, ist herausfordernder. Auch hier ist es üblich, die Variablen zur Ausgabegenerierung anzupassen, aber das Modell kann so nicht gezwungen werden, für verschiedene Eingaben die gleichen Ausgaben zu liefern. Es ist aber möglich, Modelle dazu zu bringen, Antworten zu generieren, die näher an dem sind, was Sie sich erhoffen, indem Sie sorgfältig erstellte Prompts (siehe Kapitel 5) und ein Memory-System nutzen (siehe Kapitel 6).

Halluzinationen

Halluzinationen sind für Aufgaben katastrophal, die von Fakten abhängen. Bitten Sie die KI, Ihnen dabei zu helfen, die Vor- und Nachteile einer Impfung zu erläutern, wollen Sie nicht, dass die KI pseudowissenschaftlich agiert. Im Juni 2023 wurde eine Anwaltskanzlei verurteilt, fiktive Rechtsprechung als Referenz in einem gerichtlichen Verfahren ungeprüft übernommen zu haben (*https://oreil.ly/FCyyA*). Hier wurde ChatGPT zur Vorbereitung eines Falls verwendet, und man war sich seiner Tendenz zu Halluzinationen nicht bewusst.

Auch wenn Halluzinationen mit dem Aufstieg der LLMs als Problem bekannt wurden, waren sie bereits ein häufig auftretendes Phänomen bei generativen Modellen, weit bevor der Begriff »Foundation Model« und die Transformer-Architektur eingeführt wurden. Halluzinationen im Kontext der Textgenerierung wurden schon im Jahr 2016 erwähnt (Goyal et al., 2016, *https://oreil.ly/cg0JY*). Das Erkennen und Messen von Halluzinationen ist seitdem ein wichtiges Thema bei der *Natural Language Generation* (NLG) (siehe Lee et al., 2018, *https://oreil.ly/ah9MT*; Nie et al., 2019, *https://oreil.ly/13wUD*; Zhou et al., 2020, *https://arxiv.org/abs/2011.02593*). In diesem Abschnitt geht es vor allem darum, zu erklären, warum Halluzinationen auftreten. Das Erkennen und Messen ist Thema in Kapitel 4.

Während Inkonsistenzen aus der Zufälligkeit im Sampling-Prozess entstehen, sind die Ursachen für Halluzinationen vielfältiger. Der Sampling-Prozess allein kann sie nicht ausreichend erklären. Ein Modell sampelt Ausgaben aus allen möglichen Optionen. Aber wie wird etwas, das es nie zuvor gesehen hat, zu einer möglichen Option? Ein Modell kann etwas ausgeben, von dem angenommen wird, dass es nie in den Trainingsdaten enthalten war. Wir können das nicht mit Sicherheit annehmen, weil es unmöglich ist, alle Trainingsdaten zu durchforsten, um zu überprüfen, ob diese Idee doch irgendwo zu finden ist. Unsere Fähigkeit, etwas zu erstellen, das so komplex ist, dass wir es nicht mehr vollständig verstehen können, ist Fluch und Segen zugleich.

Es ist schwer, einen Weg zu finden, der Halluzinationen ausmerzt, wenn man nicht versteht, warum sie überhaupt vorkommen. Es gibt aktuell zwei Hypothesen dazu, warum Sprachmodelle halluzinieren.

Die erste – erstmals von Ortega et al. 2021 bei DeepMind beschrieben (*https://arxiv.org/abs/2110.10819#deepmind*) – lautet, dass ein Sprachmodell halluziniert, weil es

nicht unterscheiden kann zwischen den Daten, die es erhalten hat, und denen, die es generiert. Schauen wir uns ein Beispiel an, um das zu illustrieren.

Angenommen, Sie übergeben dem Modell den Prompt »Wer ist Chip Huyen?«, und der erste Satz, den das Modell generiert, ist: »Chip Huyen ist eine Architektin«. Das nächste Token, das das Modell generiert, baut auf der Sequenz »Wer ist Chip Huyen? Chip Huyen ist eine Architektin« auf. Das Modell behandelt »Chip Huyen ist eine Architektin« – etwas von ihm Erzeugtes – genauso wie eine Tatsache. Beginnend mit einer generierten Sequenz, die ein wenig neben dem Normalen steht, kann das Modell darauf aufbauen und hanebüchene, falsche Fakten generieren. Ortega und die anderen Autoren haben Halluzinationen als eine Form von *Selbsttäuschung* bezeichnet.

In Abbildung 2.24 sehen Sie ein Beispiel für Selbsttäuschung durch das Modell LLaVA-v1.5-7B. Ich habe es gebeten, die Zutaten auf dem Label im Bild zu identifizieren – auf einer Flasche Shampoo. In der Antwort hat sich das Modell selbst davon überzeugt, dass es sich um eine Flasche Milch handelt, weshalb es Milch mit in die Liste der Zutaten aufgenommen hat, die ansonsten von der Flasche übernommen wurden.

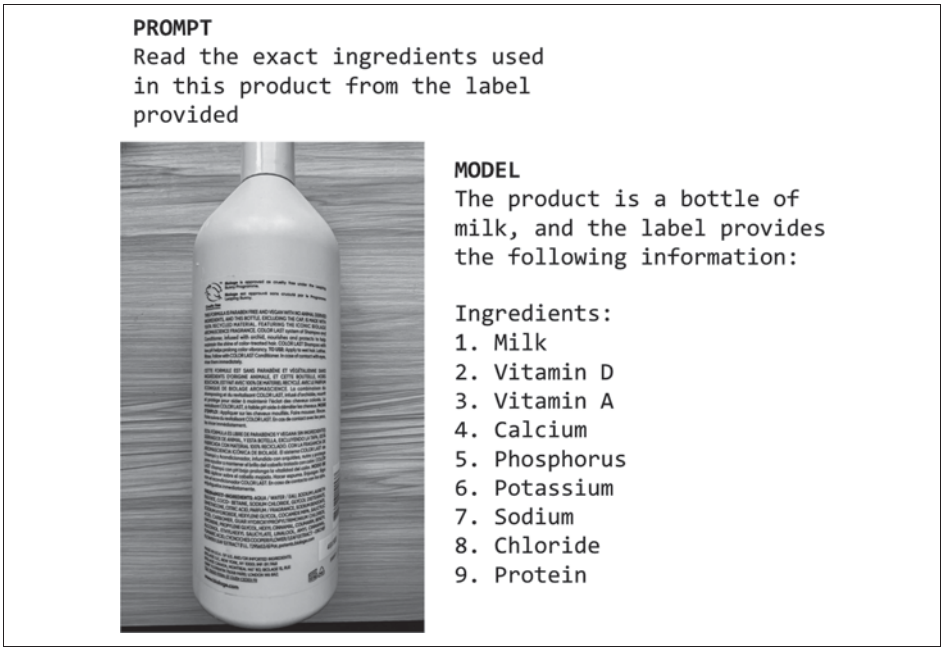

Abbildung 2.24: Ein Beispiel für Selbsttäuschung von LLaVA-v1.5-7B

Zhan et al. (2023, *https://arxiv.org/abs/2305.13534*) nennen dieses Phänomen Schneeball-Halluzinationen. Nach einer falschen Annahme kann ein Modell weiter halluzinieren, um die erste falsche Annahme zu rechtfertigen. Interessanterweise haben die Autoren und Autorinnen gezeigt, dass initiale falsche Annahmen das Modell

dazu bringen können, Fehler bei Fragen zu machen, die es ansonsten korrekt beantworten würde, wie Abbildung 2.25 demonstriert.

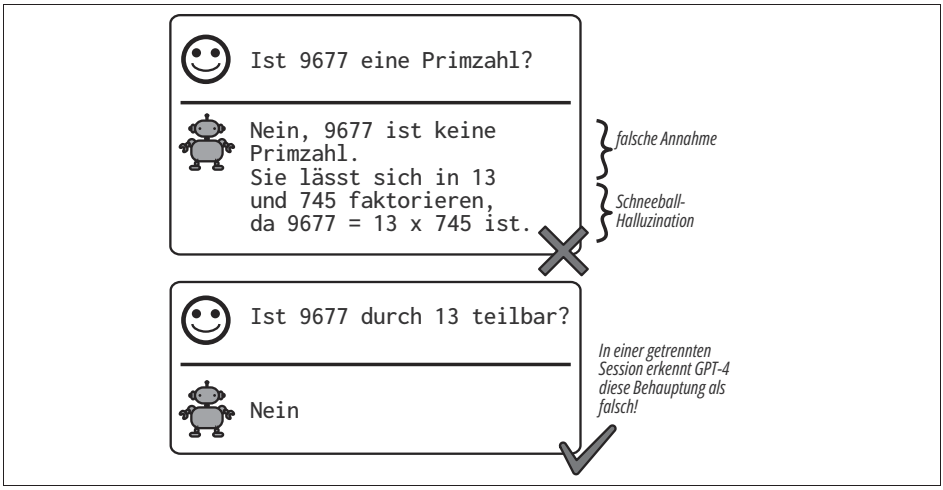

Abbildung 2.25: Eine initiale falsche Annahme kann das Modell dazu bringen, zu behaupten, dass 9677 durch 13 teilbar ist, auch wenn es weiß, dass das nicht stimmt.

Der DeepMind-Artikel hat gezeigt, dass sich Halluzinationen durch zwei Techniken reduzieren lassen. Die erste kommt aus dem Reinforcement Learning, bei dem das Modell dazu gebracht wird, zwischen von Usern angegebenen Prompts (beim Reinforcement Learning *Observations about the World* genannt) und vom Modell generierten Tokens (die *Actions* des Modells) zu unterscheiden. Die zweite Technik orientiert sich am überwachten Lernen (Supervised Learning), bei dem faktische und kontrafaktische Signale in den Trainingsdaten enthalten sind.

Die zweite Hypothese besagt, dass Halluzinationen durch die Diskrepanz zwischen dem internen Wissen des Modells und dem des Labelers entstehen. Diese Sichtweise wurde erstmals von Leo Gao, einem Forscher bei OpenAI, dargestellt. Während des SFT werden Modelle darauf trainiert, Antworten nachzuahmen, die von Labelern geschrieben wurden. Nutzen diese Antworten Wissen, das die Labeler haben, aber nicht das Modell, trainieren wir das Modell letztendlich darauf, zu halluzinieren. Theoretisch könnten Labeler das genutzte Wissen bei jeder geschriebenen Antwort einbinden, sodass das Modell weiß, dass die Antworten nicht ausgedacht sind, und es dazu bringen, nur das zu verwenden, was es weiß. Aber in der Praxis ist das unmöglich.

Im April 2023 hat John Schulman, ein Mitbegründer von OpenAI, in seinem UC Berkeley Talk Ähnliches angedeutet (*https://oreil.ly/Fqo2S*). Schulman glaubt auch, dass LLMs wissen, ob sie etwas wissen, was für sich genommen schon ziemlich beeindruckend ist. Wenn das stimmt, lassen sich Halluzinationen beheben, indem man ein Modell zwingt, Antworten nur basierend auf den ihm bekannten Informationen zu geben. Er hat dafür zwei Lösungen vorgeschlagen, eine davon ist die Verifikation: Für jede Antwort soll das Modell die zugrunde liegenden Quellen nennen.

Die andere ist der Einsatz von Reinforcement Learning. Denken Sie daran, dass das Reward Model darauf trainiert ist, nur Vergleiche vorzunehmen – Antwort A ist besser als Antwort B –, ohne zu erklären, warum A besser ist. Schulman argumentierte, dass eine bessere Belohnungsfunktion, die ein Modell dafür bestraft, sich Dinge auszudenken, dabei helfen kann, Halluzinationen zu reduzieren.

Im selben Gespräch hat Schulman eine weitere Erkenntnis von OpenAI erwähnt: RLHF hilft beim Reduzieren von Halluzinationen. Aber der InstructGPT-Artikel zeigt, dass RLHF Halluzinationen verschlimmert hat, wie in Abbildung 2.26 zu sehen ist. Obwohl RLHF die Halluzinationen für InstructGPT begünstigt hat, wurden andere Aspekte besser, und insgesamt bevorzugen menschliche Labeler das RLHF-Modell gegenüber dem reinen SFT-Modell.

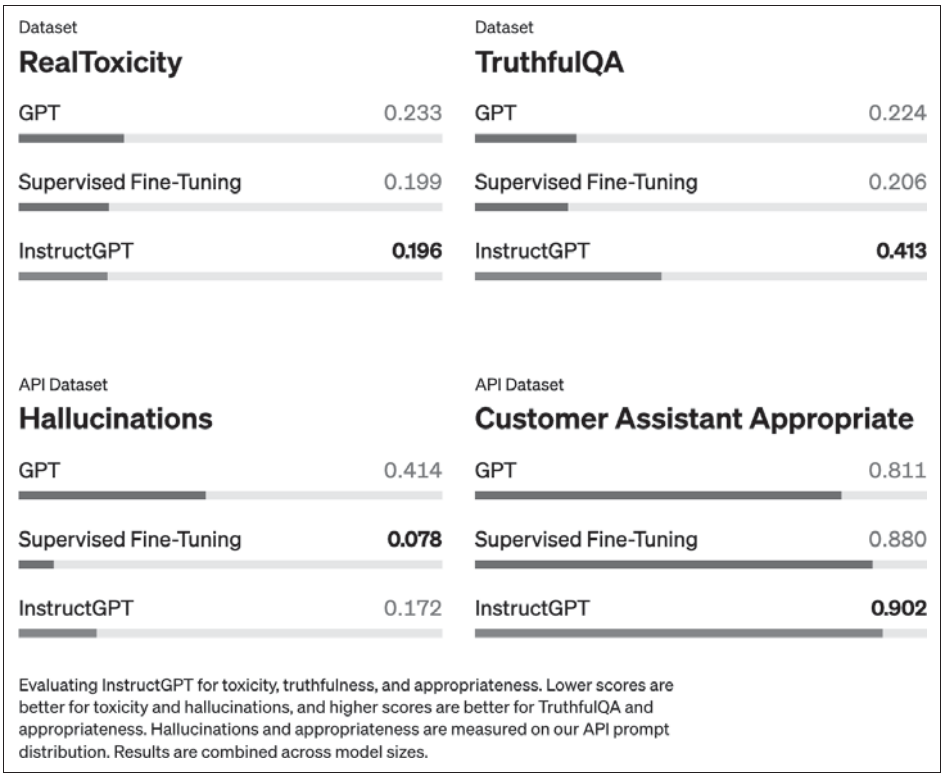

Abbildung 2.26: Halluzinationen sind für das Modell schlimmer, das sowohl RLHF als auch SFT nutzt (InstructGPT) – im Vergleich zum gleichen Modell, das nur auf SFT setzt (Ouyang et al., 2022, https://arxiv.org/abs/2203.02155).

Basierend auf der Annahme, dass ein Foundation Model weiß, was es weiß, versuchen manche, Halluzinationen über Prompts zu reduzieren, indem sie beispielsweise »Antworte so wahr wie möglich, und wenn du bei der Antwort unsicher bist, sage ›Entschuldigung, ich weiß es nicht.‹« anfügen. Es scheint auch zu helfen, die Modelle um kurz gefasste Antworten zu bitten – je weniger Tokens ein Modell generieren muss, desto geringer ist die Chance, dass es sich etwas ausdenkt. Prompting

und Kontexttechniken aus den Kapiteln 5 und 6 können ebenfalls dafür sorgen, dass Halluzinationen kein so großes Problem werden.

Die zwei erwähnten Hypothesen ergänzen sich gegenseitig. Die Selbsttäuschungshypothese fokussiert sich darauf, wie das selbstüberwachte Lernen für Halluzinationen sorgen kann, während es bei der Hypothese zum unterschiedlichen internen Wissen eher darum geht, wie Supervision Halluzinationen hervorruft.

Wenngleich wir Halluzinationen nicht ganz vermeiden können – ist es zumindest möglich, zu erkennen, wann ein Modell halluziniert, sodass wir diese Antwort dann nicht an die User herausgeben? Nun, das Erkennen von Halluzinationen ist auch nicht so einfach – denken Sie nur daran, wie schwer es für uns ist, herauszufinden, ob ein anderer Mensch lügt oder sich etwas ausdenkt. Aber es wurde versucht. In Kapitel 4 werden wir auch darüber reden, wie man Halluzinationen erkennt und misst.

Zusammenfassung

In diesem Kapitel ging es um die zentralen Designentscheidungen beim Bauen eines Foundation Model. Da die meisten Menschen fertige Foundation Models nutzen werden, statt selbst eines von Grund auf zu trainieren, habe ich die anstrengenden Trainingsdetails weggelassen und mich auf die Faktoren beim Modellieren konzentriert, die Ihnen dabei helfen, herauszufinden, welche Modelle Sie nutzen und wie Sie sie einsetzen können.

Ein entscheidender Faktor für die Performance eines Modells sind seine Trainingsdaten. Große Modelle erfordern sehr viele Daten, deren Beschaffung teuer und zeitaufwendig sein kann. Modellanbieter nutzen daher oft möglichst alle Daten, die irgendwie verfügbar sind. Das führt zu Modellen, die bei den vielen Aufgaben, die sich in den Trainingsdaten wiederfinden, gut funktionieren, die aber vielleicht gerade nicht die spezifische Aufgabe abbilden, die Sie erledigen wollen. Dieses Kapitel hat sich angeschaut, warum es häufig erforderlich ist, Trainingsdaten zu kuratieren, um Modelle zu entwickeln, die auf spezifische Sprachen – insbesondere solche mit wenig verfügbaren Ressourcen – und auf spezifische Domänen abzielen.

Nach dem Einsammeln der Daten kann die Modellentwicklung beginnen. Das Modelltraining dominiert zwar oft die Schlagzeilen, aber zuvor ist es wichtig, die Architektur zu bestimmen. Das Kapitel hat sich angeschaut, was für Entscheidungen es bei Modellen zu treffen gibt, zum Beispiel zu Architektur und Größe. Die dominierende Architektur für sprachbasierte Foundation Models ist Transformer. In diesem Kapitel wurden auch die Probleme besprochen, die die Transformer-Architektur angehen soll, und es wurde auf ihre Beschränkungen hingewiesen.

Die Größe eines Modells kann durch drei Werte gemessen werden: die Anzahl der Parameter, die Anzahl der Trainingstokens und die Anzahl an FLOPs, die für das Training erforderlich waren. Zwei Aspekte, die die Menge an Rechenleistung zum Trainieren eines Modells beeinflussen, sind die Größe des Modells und die Menge an Daten. Das Skalierungsgesetz hilft dabei, die Anzahl an Parametern und Tokens

herauszufinden, die bei einem gegebenen Rechenbudget optimal sind. Hier ging es auch um Flaschenhälse beim Skalieren. Aktuell macht das Vergrößern eines Modells dieses im Allgemeinen besser. Aber wie lange wird das Gültigkeit behalten?

Aufgrund der schlechten Qualität von Trainingsdaten und dem selbstüberwachten Lernen beim Pre-Training erzeugt das so entstandene Modell eventuell Ausgaben, die nicht dem entsprechen, was die User erwarten. Dieses Problem wird durch Post-Training angegangen, das aus zwei Schritten besteht: Supervised Finetuning und Preference Finetuning. Menschliche Vorlieben sind sehr verschieden, und es ist unmöglich, sie alle in einer einzelnen mathematischen Formel abzubilden, daher sind bestehende Lösung bei Weitem noch nicht narrensicher.

Es ging in diesem Kapitel auch um eines meiner Lieblingsthemen: Sampling – der Prozess, bei dem ein Modell die Ausgabetokens generiert. Das Sampling sorgt für die statistische Natur von KI. Diese führt dazu, dass Modelle wie ChatGPT oder Gemini für kreative Aufgaben sehr hilfreich sind und es Spaß macht, mit ihnen zu reden. Aber die statistische Natur führt auch zu Inkonsistenzen und Halluzinationen.

Die Arbeit mit KI-Modellen erfordert es, dass Sie bei Ihren Workflows die statistische Natur berücksichtigen. Im weiteren Verlauf dieses Buchs werde ich zeigen, wie Sie AI Engineering vielleicht nicht deterministisch, aber zumindest systematisch gestalten können. Der erste Schritt zu einem systematischen AI Engineering ist das Einrichten einer soliden Pipeline, die dabei hilft, Fehler und unerwartete Änderungen zu erkennen. Das Evaluieren ist bei Foundation Models so wichtig, dass ich ihm zwei Kapitel gewidmet habe – im nächsten geht es damit los.