Transformer Decoder Architektur

1st Stefan Maier
Function and Algorithm
ZF Lifetec
Alfdorf, Deutschland
Stefan.Maier1@zf-lifetec.com

***Abstract*—Große Sprache Modelle (engl. Large Language Models) wie Chat GPT basieren auf der Transformer Architektur neuronaler Netze. Die Transformer Architektur besteht dabei aus einem Encoder- sowie Decoder-Element. In der Wissenschaft und Forschung existieren auch Architekturen welche nur einen Teil eines Transformers zur Lösung eines Problems verwenden. In dieser Arbeit wird genauer auf Transformer Architekturen eingegange, welche lediglich aus dem Decoder Element bestehen. Eine weit Verbreitete Decoder Architektur ist die von Chat GPT verwendete GPT (kurz: Generative Pretrained Transformer) Architektur. Es wird auf die Architektur und die weiter Entwicklung der Architektur eingegangen.**

***Index Terms*—component, formatting, style, styling, insert.**

I. Einführung

Die Verarbeitung von natürlicher Sprache (engl. Natural Language Processing. kurz NLP) ist ein wichtiger und großer Bestandteil der künstlichen Intelligenz Forschung. Die Forschungsbereiche decken dabei ein breites Spektrum an Aufgaben ab, wie z.B. Beantwortung von Fragen, Semantische Ähnlichkeit, Text Generierung, Dokumenten Klassifikation o.ä. Die ersten Fortschritte wurde im NLP Bereich bereits durch statistische Hidden Markov Modelle erzielt. Die Leistung solcher statistischen Modelle ist jedoch begrenzt und konnte die Komplexität der natürlichen Sprache nicht geeignet abbilden. Mit den Entwicklungen im Bereich des Deep-Learnings gelangen entscheidende Schritte in der NLP Forschung. Die Transformer Architektur, und das damit verbundene vortrainieren großer Sprachmodelle wie BERT, GPT,T5 oder RoBERTa haben große Fortschritte gebracht, die sich nicht nur auf die NLP Forschung beschränken sondern in der allgemeinen Gesellschaft und Wirtschaft Einzug halten.

A. Ziel der Arbeit

Das Ziel der Arbeit ist es die Decoder Architektur welche Häufig von generativen Sprachmodellen verwendet wird genauer zu beleuchten. Dabei wird die Decoder-only Architektur anhand des Generative Pretrained Transformers erklärt. Neben dem Generative Pretrained Transformer existieren ebenfalls weitere Modelle die kurz erläutert werden. Weiterhin soll die Arbeit die Anwendungsgebiete sowie Potenziale und Grenzen der Decoder Architektur aufzeigen. Im Ausblick wird auf aktuelle Forschungsfelder im Bereich der Decoder Architekturen eingegangen.

B. Aufbau und Struktur der Arbeit

Die Arbeit ist wie folgt strukturiert: In Kapitel 2 wird zunächst auf die Grundlagen in Form der Decoder Architektur als Teil der Transformer Architektur eingegangen. Dazu wird die historische Entwicklung, der Aufbau und der Trainingsprozess der Decoder Architektur erläutert und diese vom Encoder Teil der Transformer Architektur abgegrenzt. In Kapitel 3 wird auf unterschiedliche mögliche Anwendungsgebiete aus der Forschung und Entwicklung eingegangen. Aus diesen Erkenntnissen werden in Kapitel 4 die Potenziale und damit einhergehenden Grenzen solcher großen Sprachmodelle eingegangen. Kapitel 5 fasst die Ergebnisse zusammen und gibt einen Ausblick auf die aktuellen Forschungsbereiche.

II. Decoder Architektur

Im folgenden Abschnitt wird die Decoder Architektur am Beispiel des Generative Pretrained Transformers (GPT) erläutert. Dabei wird auf die historische Entwicklung, den Aufbau, den Trainingsprozess und die Abgrenzung zur Encoder Architektur eingegangen.

A. Historische Entwicklung der GPT-Decoder Architektur

Transformer bilden das Grundgerüst moderne großer Sprachmodelle. Diese wurden 2017 in [1] beschrieben. 2018 wurde die erste Version von GPT veröffentlicht, welche auf dem Decoder Part der Transformer Architektur aufbaut. Auf Basis der GPT Modell entstanden weitere Decoder-only Modelle welche die folgende Figure 1 gut veranschaulicht. Die Figure 1 zeigt die Entwicklung unterschiedlicher Transformer Modell bis zum letzten Jahr. Im Rahmen dieser Seminararbeit zeigt Decoder-Only Zweig die Entwicklung von reinen Decoder Architekturen. Damit ist GPT1 eines der ersten Modelle welche auf eine reine Decoder Architekutur setzt. Daraus entsprange viele unterschiedliche Decoder-Only Modelle von Google, Meta oder anderen Forschungseinrichtungen. Die aktuellsten versionen von Llama, GPT-4 oder Bard stellen dabei bis heute die neusten Entwicklungen dar.

B. Aufbau der GPT-Decoder Architektur

Wie im vorherigen Abschnit beschrieben legt das GPT Modell den Grundstein für die Decoder-only Architektur Entwicklungen. Dies wurde durch das Paper [3] beschrieben

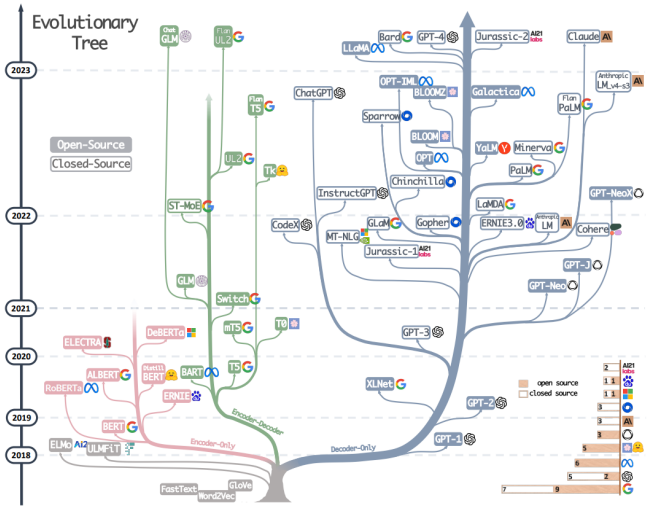

Fig. 1. Historische Entwicklung von Transformer Modellen aus [2]

und basiert dabei auf den Grundzügen der Transformer Architekturen welche bereits in Paper [1] beschrieben wurden. Die folgende Abbildung illustriert dabei den Aufbau einer Decoder Schicht des GPT Modells. Die Figure 2 zeigt

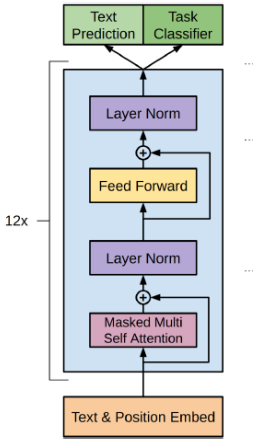

Fig. 2. Decoder Architektur nach [3]

dabei die Architektur Elemente einer Decoder Schicht. Diese Schichten werden in der ersten GPT Architektur 12 mal hintereinander geschaltet. Jedes dieser Decoder Schichten besteht dabei aus einem Masked Multi-Self Attention Layer, Feed Forward Netzwerken, sowie Add und Layer Normalisation Schicht um die Residual Verbindungen der vorherigen Schicht hinzuzufügen. Die Residual Verbindungen sollen vorallem das Problem der verschwindend Gradienten beheben. Der wichtigste Bestandteil der Decoder Schicht ist dabei das Masked Multi-Self Attention Layer. Das Masked Multi-Self Attention Layer ist dabei eine leichte Abwandlung des Mult Self Attention Layers das auch in Encoder Elementen verwendet wird. Nachffolgende Abbildung zeigt den Aufbau eines Multi-Self Attention Layers. Die Multiplen Heads in der Figure 3 repräsentieren dabei die Anzahl der Köpfe die in der Attention

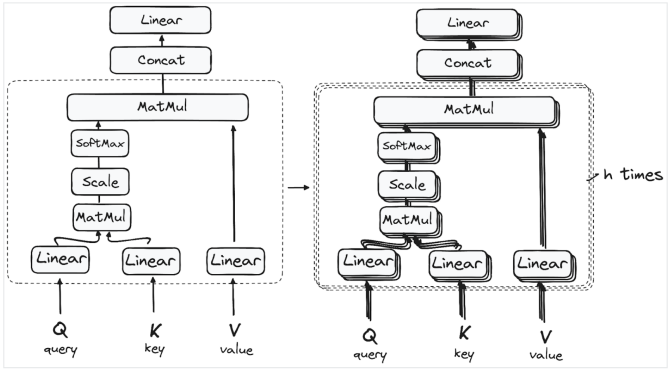

Fig. 3. Multi-Self Attention Layer nach [4]

Schicht verwendet werden welche durch die Anzahl an h Köpfe an parallelen Berechnung des Attention Layers bestimmt sind. zur Berechnung der Ausgabe des Attention Layers werden die Attention Weights berechnet welche bestimen wie wichtig ein Wort für ein anderes Wort ist, was durch nachfolgende Abbildung Visualisiert ist. Die Färbung in der

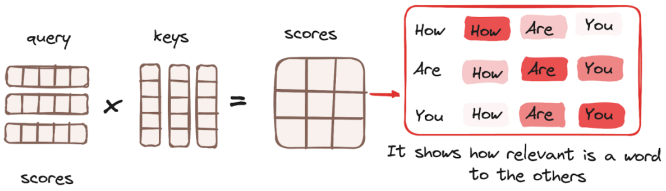

Fig. 4. Attention Mechanismus nach [4]

Figure 4 zeigt dabei die Wichtigkeit der Wörter zueinander. Die daraus berechnete Attention Weights werden mit dem Werten der Wörter multipliziert und über ein Lineares Layer verfeinert, dies ergibt die Ausgabe des Attention Layers. Der wesentliche Unterschied zwischen dem Multi-Self Attention Layer und dem Masked Multi-Self Attention Layer ist dabei die Maskierung der Attention Weights. Die Maskierung sorgt dafür, dass ein Wort nur von vorherigen Wörtern abhängt und damit die Attention einbezieht. Die nachfolgende Abbildung zeigt die Maskierung der Attention Weights mithilfe eine Maskierungsmatrix.

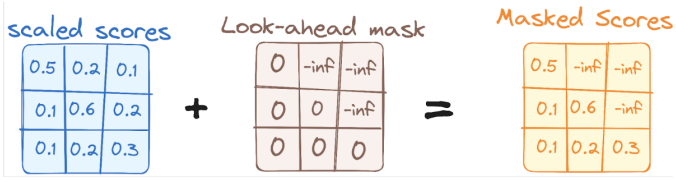

Fig. 5. Maskierung der Attention Weights nach [4]

Die genaue Funktionsweise der einzelnen Schichten aus Figure 2 wird in [1] beschrieben. Abschließend wird in der GPT Architektur ein Layer je nach Aufgabe angehängt, welche in Figure 2 durch Text Vorhersage oder Aufgaben Klassifizierung repräsentiert wird. Dies kann beispielsweise ein Lineares

Layer mit Softmax Aktivierungsfunktion sein, welches die Wahrscheinlichkeiten für das nächste Wort aus dem Vokabular berechnet [4].

C. Abgrenzung der Decoder Architektur zur Encoder Architektur

Die vorgeschlagende Decoder Architektur durch [3] unterscheidet sich dabei von der Encoder Architektur durch die Verwendung von Masked Multi-Self Attention Layern. Die Encoder Architektur verwendet hingegen Multi-Self Attention Layer ohne Maskierung. Entscheidend ist ebenfalls das die Architektur aus [3] keine einzeziehung der Encoder Schicht vorsieht und somit im Vergleich zur Decoder Architektur aus [1] keine weitere Multi-Self Attention Layer enthält. Die Decoder Architektur ist dabei auf die Generierung von Text ausgelegt und kann somit als autoregressives Modell betrachtet werden. Die Encoder Architektur hingegen ist auf die Verarbeitung von Texten ausgelegt und kann als diskriminatives Modell betrachtet werden.

D. Trainingsprozess der GPT-Decoder Architektur

Das entscheidende bei der in [3] beschrieben Decoder Architektur ist der Trainingsprozess der aus zwei Stufen besteht.

a) unüberwachtes Vortrainieren: Zum einen das unüberwachte Vortrainieren des om Abschnitt A beschriebenen Modells. Dabei ist die Zielfunktion folgende Likelyhoodfunktion:

$$\mathcal{L}_1(\mathcal{U}) = \sum_i logP(u_i|u_i - k...u_i - 1; \theta) \tag{1}$$

k ist dabei das Kontext Fenster und somit wird die Wahrscheinlichkeit des nächsten Wortes auf Basis der vorherigen k-Tokens bestimmt. Das erste GPT Modell wurde dabei auf Tokens aus dem BookCorpus dataset vortrainiert. Wie bereits in **??** beschrieben wurden 12 Decoder Schichten verwendet. Für die Attention Heads h wurden 12 Heads verwendet bei 768 Embedding Dimensionen. Für das Feed Forward Netzwerk wurde eine GELU (Gaussian Error Linear Unit) verwendet. Das Training wurde über 100 Epochen mit einer Batchgröße von 64 durchgeführt.

b) überwachtes Fine-Tuning: Der zweite Schritt ist das überwachte Fine-Tuning auf die gewünschte Zielaufgabe. Dabei handelt es sich um m-Eingabe Tokens, welche in eine Klasse aus $\mathcal{C}$ transformiert werden. Die Zielfunktion ist folgende Likelyhoodfunktion:

$$\mathcal{L}_2(\mathcal{C}) = \sum_{(x,y)} logP(y|x^1...x^m) \tag{2}$$

Dabei wird die Zielfunktion aus Equation 2 mit der Funktion aus Equation 1 zu Equation 3 kombiniert damit der Trainingsprozess besser konvergiert, sowie generalisiert [3].

$$\mathcal{L}_3(\mathcal{C}) = \mathcal{L}_2(\mathcal{C}) + \lambda\mathcal{L}_1(\mathcal{C}) \tag{3}$$

Da das vortrainierte Modell wie in Equation 1 beschrieben auf die vorhersage der nächsten Worte optimiert ist, müssen

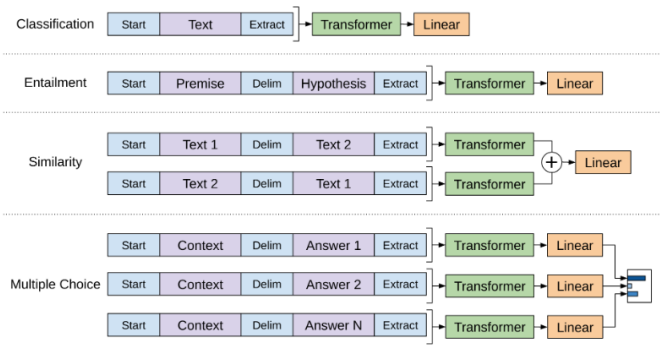

Fig. 6. Anpassung der Input Tokens nach [3]

für das Finetuning die Input Tokens entsprechend der nachfolgenden Figure 6 wie in [3] angepasst werden. Dabei werden strukturierte Daten in eine geordnete Reihenfolge gebracht, um die Eingabe für das Modell zu generieren. Figure 6 zeigt, dass für eine Multiple Choice Aufgabe zuerst ein zufällig ausgewähltes Start Token, der Kontext und anschließend die Antwort mit einem zufällig ausgewählten Wert für das Abschluss Token übergeben wird. Für diese spezifische Aufgabe werden anschließend drei Transformer parallel trainiert, um die korrekte Antwort zu vorherzusagen. Für das eigentliche Training wurden die meisten Hyperparameter aus dem Unsuperwised Training übernommen. Lediglich wurde ein Dropout vor der Klassifizierungsschicht hinzugefügt und die Lernrate wurde reduziert. Das Paper aus [3] zeigt dabei, dass 3 Epochen ausreichen um eine gewünschte Genauigkeit zu erzielen.

E. Weiterentwicklung der GPT-Decoder Architektur

Die erste GPT Version ist ein Meilenstein in der Entwicklung von Decoder-only Architekturen. Die Weiterentwicklung der GPT Architektur zeigt dabei die Entwicklung von GPT-2, GPT-3 und GPT-4. GPT2 verbesserte die Funktionsweise von großen Sprachmodellen dass diese auf mehrere Aufgaben trainiert werden. Dabei war das Ziel ein Modell zu entwerfen das eine Warscheinlichkeit für eine Output Token Sequenz in Abhängigkeit der Input Tokens sowie der Aufgaben Spezifikation zu erlernen.

$$P(s_{n-k}, ..., s_n|s_1, ..., s_{n-k-1}, task) \tag{4}$$

Für das Training das GPT2 Modell wird als Datensatz WebText verwendet welche aus Inhalten kuratierter Websiten besteht. Die Modell struktur ist ähnlich zur ursprünglichen GPT Architektur, jedoch wurde die Anzahl des Kontextfensters von 512 auf 1024 erhöht. Anstatt 12 Decoder-only Layers wurden bei GPT2 48 Layers verwenden mit einer Embedding Dimension von 1600 was zu einer Parametermenge von ca. 1,5 Milliarden Parameter führt. GPT2 zeigt zum ersten mal das ein großes Sprachmodell auf mehrere Aufgaben gleichzeitig trainiert werden kann, sofern der Trainingsdatensatz divers genug ist [5]. Während der Fokus bei GPT2 vorallem auf der Zero Shot Performance und das Unsupervised Mulittask Learning liegt, wurde das GPT3 Modell im Fokus des

Few Shot Learnings betrachtet. GPT3 erweitert auch hier die Funktionalität von GPT2 erneut und verwendet statt des Masked Multi-Self Attention Layer, Attention Konzepte aus [6] ähnlich der sogenannten Sparse Transformes. Diese ermöglichen es Längere Sequenzen zu verarbeiten und das Training tieferer Modelle zu ermöglichen. Das Paper aus [7] zeigt dass größere Modelle entscheidende Vorteile beim Meta Learning oder Few-Shot Learning haben. So besteht das größte GPT3 Modell aus 175 Milliarden Parametern. Das neueste Modell von OpenAI GPT4 zeigt die Weiterentwicklung von GPT3 und setzt zusätzlich zum Unsupervised Vortraining auf Finetuning über Menschen gesteuertes Feedback mittels Reinforcement Learning. Leider gibt es zum aktuellen Zeitpunkt keine weiteren Details zur Architektur von GPT4 wie in [8] beschrieben. Das Paper in [8] zeigt jedoch anhand der verwendeten Trainingsressourcen, dass das GPT Modell entsprechend groß ist. Zudem wird beschrieben, dass eine Infrastruktur im Rahmen der GPT4 Entwicklung entwickelt wurde um verlässliche die Entwicklung des Modells vorher zu sagen und die Trainingsressourcen zu optimieren.

III. Anwendungsbereiche

Die GPT Paper zeigen bereits bei der Model Evaluation auf unterschiedliche Datensätze, die Anwendungsbereiche großer Sprachmodelle. Die Anwendungsbereiche sind dabei vielfältig und reichen von Text Generierung, Text Klassifikation, Text Verständnis, Text Zusammenfassung, Text Übersetzung, Text Extraktion

IV. Potenzial und Grenzen von Decoder Architekturen

V. Fazit

A. Aktuelle Forschungbereiche

B. Ausblick

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," Aug. 2023.

[2] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, B. Yin, and X. Hu, "Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond."

[3] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving Language Understanding by Generative Pre-Training."

[4] "Wie Transformatoren funktionieren: Eine detaillierte Erkundung der Transformatorarchitektur," https://www.datacamp.com/tutorial/how-transformers-work.

[5] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners."

[6] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating Long Sequences with Sparse Transformers," Apr. 2019.

[7] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," Jul. 2020.

[8] OpenAI, L. Ahmad, F. L. Aleman, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, Ł. Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, Ł. Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O'Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. d. A. B. Peres, M. Petrov, H. P. d. O. Pinto, Michael, Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. J. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph, "GPT-4 Technical Report," Mar. 2024.