MARC: A Multi-Agent Framework for Generating 3D CAD Models with Iterative Refinement

Stanford CS224N {Custom}

Sachal Malick
Department of Computer Science
Stanford University
sachalm@stanford.edu

Abstract

Computer Aided Design (CAD) software is a key tool that mechanical engineers use to construct 3D models of their designs which can then be used as a reference for construction, or sometimes printed directly using 3D printers. However, creating CAD models is a complex and tedious process. A recently published paper proposed a framework for generating 3D CAD models directly from text descriptions, with the help of a custom transformer that they pretrain to autoregressively generate CAD data. Empirical analysis shows this model has overfit its training set, and cannot generalize well enough to be practically useful. We propose an alternative framework which we call Multi-Agent Refined Construction (MARC) that requires no additional pretraining and leverages existing LLMs in an agentic fashion to generate CAD models directly. Early analysis shows this model generalizing far better than Text2CAD, and outperforming Text2CAD on an evaluation set gathered from data that it was trained upon. We demonstrate how to draw out latent capabilities of large models through familiar interfaces. Additionally, we inadvertently construct a new task for long context evaluation.

1 Key Information to include

- TA mentor: Bassem Akoush
- External collaborators: No:
- External mentor: No:
- Sharing project: No

2 Introduction

Research into 3D generative models dates back to the advent of neural networks, but progress in the field first started to accelerate after the introduction of Generative Adversial Networks [1]. Since then countless approaches have been pursued to accelerate the domain and in recent years, Diffusion Models have become the dominant architecture [2]. This has proven to be an effective solution for generating realistic 3D assets that responding well to conditioning, however these representations lack the geometric specification that are paramount to CAD models [3]. Engineers cannot import these artifacts and expand on them like they would with a CAD representation of a 3D structure.

Thus far there has been limited open research into controlling model generation through text conditioning, however a recently published a paper describes an architecture which the authors claim is the first neural network capable of generating CAD from text [4]. The authors of this paper chose to leverage the spatial reasoning capabilities of LLMs to guide a newly pretrained transformer that operates in a learned CAD embedding space and auto regressively decodes into CAD tokens.

We initially set out to contribute to improving the Text2CAD architecture based on promising data included with the paper, however after an initial evaluation revealed the limited range of the framework and strong signs that it had overfit its training set, we reverted to first principles thinking and questioned the value of the added complexity of the architecture.

Given that in the Text2CAD framework the LLM that is responsible for refining the prompt into detailed components is performing the challenging spatial reasoning work, our first goal was to identify if the model could generate CAD representations directly. We found that with the right formulation and iterative refinement, frontier LLMs are able to eventually produce valid CAD data that more consistently represents the structures described in the prompt. Next, we extended this iterative refinement process into a two agent framework (MARC) [5]that leverages a Large Multi Model (LMM) to provide feedback to the LLM CAD designer such that it can improve its generations. Our evaluations provided signal that this framework is significantly better than Text2CAD at handling complex object descriptions, and can handle a wider range of prompts. After evaluating MARC we also found that it is able to eventually able to produce generations preferred by GPT4o over Text2CAD when tested on the DeepCAD dataset that Text2CAD was generated on.

We experimented with both Supervised Finetuning, as well as PPO and Advantage Actor Critic reinforcement learning to expand the capabilities of smaller models. We found that supervised finetuning will improve the Design Agents ability to generate data in the expected format, but did not improve its spatial reasoning skills (as measured by its ability to create geometrically correct designs). We could not scale our PPO training enough to see a significant improvement on final generations.

3 Related Work

Earlier research into 3D CAD generation has ranged from generating CAD from high fidelity 3D representations such as 3D point clouds [**?**] to using generative models to generate novel CAD models through a random sampling proces [3]. Additionally, there has been significant research in Agentic and more specifically Mult-Agent LLM architectures [6].

DeepCAD [3] introduced a Autoencoder-style architecture with a Transformer backbone that can reconstruct CAD models from sketch-and-extrusion CAD representations. We will expand on this format in the next section as it is highly pertinent to our research. They open sourced a dataset of 178,238 CAD models and their corresponding sketch-an-extrusion representations. They found their model to be both adept at reconstruction as well as random structure generation. Text2CAD also trained their network on the DeepCAD sketch and extrusion dataset, however they also used a Large Multi Model (also called a Large Vision Model) to annotate the DeepCAD data by passing in rendered images and prompting the model to describe the structures.

Shortly after the release of Text2CAD, researchers in South Korea [7] put forth what they also claimed to be the first framework for generating CAD from text, and they also chose to call their framework Text2CAD. They also leverage a LLM for prompt refinement, however their pipeline then generates an isometric reference image which is passed as an input to a stable diffusion model [8] that they fine tuned to specifically generate orthographic images. They use deterministic third party software to convert their orthographic images into 3D mesh models. While they have not released any code or weights, in their pipeline the LLM is still responsible for performing the hard spatial reasoning task.

Finally, Wu et. al did significant work in evaluating the abilities of LVM-as a judge for spatial data, and found the preferences of frontier models like GPT4 to be mostly aligned with human experts [9]. This introduces a path past one of the key bottlenecks that stifles generative CAD research: a lack of scalable evaluation systems. Human expert evaluations are considered the gold standard of 3D CAD generation, and there are not proven quantitative metrics that consistently align with human preferences and capture all the data that human experts care about.

4 Approach

Both Text2CAD [4] and Text2CAD [] rely on premier Large Language Models to reason about the prompt and the related spatial data to create a thorough design plan which is then fed as an input to the Text to CAD transformer.

The core hypothesis beyond the approaches we have explored so far is that we can enable LLMs to create better CAD generations than Text2CAD without pretraining additional networks. LLMs are already performing the challenging spatial reasoning problems in the Text2CAD pipeline, and with improvements to the problem formulation we find that they can generate valid CAD data native.

There is validity to the hypothesis that connecting a language embedding space to a well-learned latent representation of CAD data can help language models efficiently connect language concepts to related CAD structures. Vision Transformers (ViT) have been used to create state of the art vision models that prove adept at media understanding [10]. However, successful implementations of this have worked by extending already extensively pretrained LLMs [11]. The pipeline proposed in Text2CAD creates a communication bottleneck by forcing the reasoner to communicate to the model through text tokens. The Text2CAD has no world-model to work off of, and therefore the LLM has to provide high fidelity instructions in either case, this is backed up by the data they present on the accuracy of their network when used with varying degrees of prompt detail.

Our proposal is that with a familiar interface to construct CAD models through, the LLM can perform the task without the help of the custom transformer. Therefore, we provide a way for a LLM to define a complete CAD model through python code, a programming language that frontier models have proven to be adept in using [12]

4.1 CAD through Code

DeepCAD [3] demonstrates that the format you provide CAD data in has a large impact on a LLMs ability to reason about the model. For example, Boundary Representation (brep) files which represent a solid by defining the limits of a volume, prove nearly impossible for a LLM to understand. The data is almost completely numerical, and does not contain abstract representations that the model can use to reason about the structures.

DeepCAD and Text2CAD use a JSON representation of sketch and extrusion commands which is significantly more abstract. However, we find that without any finetuning, SoTa models like GPT4o still are not able to reliably construct valid CAD representations in this format, even with the help of in-context learning examples.

Therefore we created a simple and intuitive python library for incrementally building a 3D CAD model with step and extrusion commands. The library wraps over code primarily written by DeepCAD and extended by Text2CAD that is used for transforming sketch and extrusion json into other CAD formats such as breps.

Claude 3.5 has proven adept at learning to use new libraries not seen at training time, and additionally it is able to reason about python execution errors well. Claude 3.5 shows a 100 percent success rate when it comes to constructing a valid CAD model using this library within 10 tries.

4.2 Inference-time Refinement

MARC consists of two LLMs working in an agentic fashion [6], a Design Agent and a Eval Agent which for most of our experimentation is GPT4o [13] . The Design Agent is responsible for constructing the CAD model through our python library. We enable the Design Agent to build the model incrementally by providing it with two commands it must include at the end of every response. <continue> indicating it would like to try running the code and evaluating the current state, and <done> indicating it is ready for a final evaluation. To get feedback on a submission, we define an orchestrator which executes the python code generated. If there are execution errors they are provided back to the Design Agent so it may correct them. If the execution succeeds and a model is constructed, we render it and take snapshots of it from four angles and feed that to the Eval Agent with the original prompt description.

The Eval Agent will describe the current state of the rendered model as the Design Agent does not have direct access to the images. It will also score the model on four criteria and then provide feedback for the Design Agent on how to improve the submission.

If the average of all four scores is greater than our target threshold, then the Design Agent does not get the opportunity to resubmit. Otherwise it will get up to 10 opportunities to refine its submission, after each attempt it will again solicit feedback from the Eval Agent.

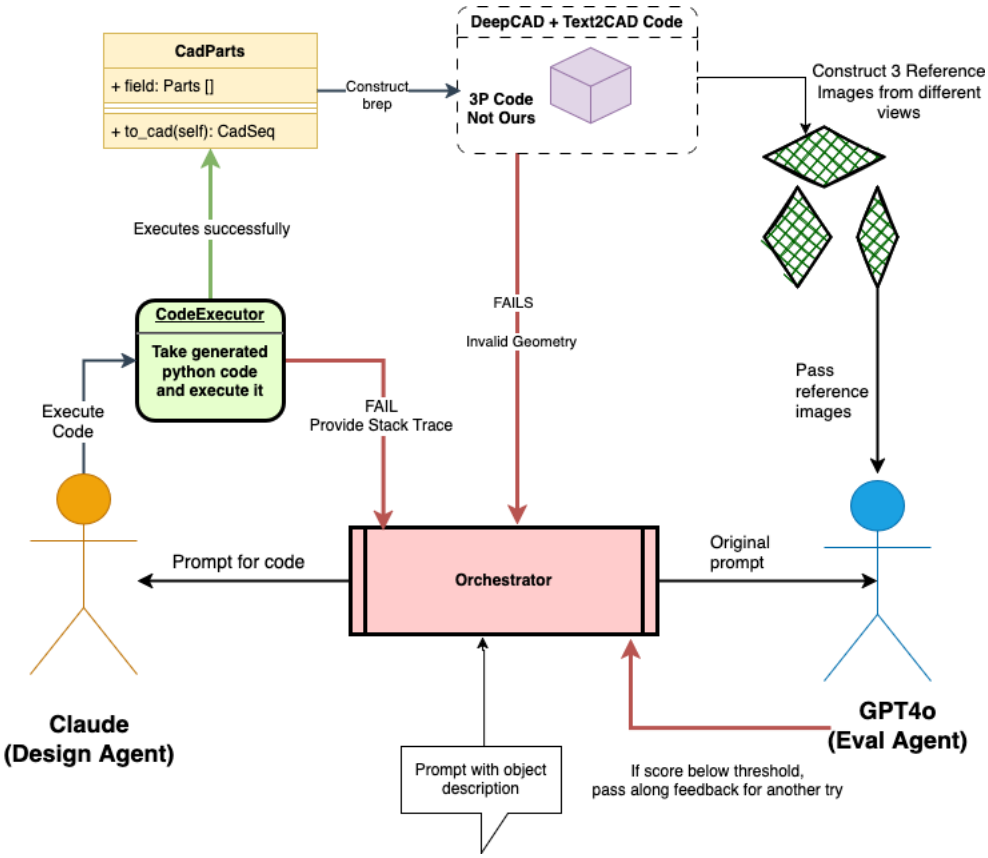

Figure 1: MARC architecture diagram

4.3 Supervised Finetuning on Task Examples

The main objective for our research was to improve the quality of the CAD generations using as many refinement steps as needed so long as the generations were getting continuously better. While there was an average of 3.2 failed generations per request, the eventual correctness was sufficient for this stage. After evaluating our agentic framework at inference, we wanted to see if we could leverage Deep Reinforcement Learning to to train the model to improve its generation abilities by using policy based gradient updates to increase the score assigned to it by GPT4o. The largest model that we could both inference and hope to train is a 70B so we chose to work with Llama 3.1 70B-Instruct. However, we found it failed completely at generating valid cad sequences with our library using the 10 steps that our compute budget allowed for. Therefore, in an attempt to improve performance for Llama 70B to support further experiments, we constructed a dataset of 5,000 examples of LLM-annotated sketch and extrusion models from DeepCAD that we converted to the code defintion that uses our library, where the LLM generated descriptions were the inputs and the labels were the tokens representing the valid code, the loss function was cross entropy for the tokens in the label position. Due to the lack of success we had in matching Claude 3.5's native performance, we will focus on our agentic framework and expand on this area in future work.

4.4 Latency Optimizations

Latency proved to be a key bottleneck for experimentation which constrained the evaluation size for our experiments. The observe latency between Claude and the GPT4o APIs were around 4 seconds on average for TTFT (Time To First Token). We were able to inference the Llama 70B model much more efficiently using VLLM which has native support for chunked prefill, kv cache, and efficient inter-rank communication for our tensor-parallel world. With VLLM we found that context encode (Time To First Token) improved dramatically and we were able to handle throughput of 585 tokens

per second, but decoding became a key bottleneck as were getting constrained at 20 OTPS (output tokens per second). We opted to use VLLM to host our eval agent as it had a simpler task with less rigid formatting requirements. This significantly expanded the extent to which we could scale our experiments, and perform thorough evaluations.

5 Experiments

5.1 Assessing the benefits of CAD through Code

As sketch and extrusion JSONs were the carefully selected representation used by DeepCAD and Text2CAD we will use this as our baseline.

We sampled 1000 sketch and extrusion files from the DeepCAD dataset with the LLM generated annotations for the models that were added in Text2CAD. For each annotation, which is a description of the model, we ask Claude 3.5 to construct a CAD model from this description using both the json format and the python format. For each prompt, we provide a single in-context example and a description of that format.

- **Syntax correctness**: Valid JSON: 96%, Valid Python 91%.
- **Valid model generated after 1 try**: JSON: 22 percent, Python: 47 percent. Python more than doubles the success rate of JSON on the first attempt.
- **Valid model generated within 10 tries**: Python: 100 percent, JSON: 29 percent. Due to the verbosity of python runtime errors, we were able to easily provide the model with more signal that it could use to correct itself.

5.2 Text2CAD vs our Agentic Framework

Our first goal was to evaluate our Agentic Framework relative to Text2CAD. One generation from our framework consists of at most 10 calls to the Anthropic API that serves Claude 3.5. For each valid CAD generation that the Design Agent produces, there can be at most 3 calls to the OpenAI api for evaluation loops. Therefore, in the worst case scenario, a single request can map to 30 API calls per request. Thus, given the need to iterate on our experimentation code and improve our agent codebase, we were only able to baseline with 2000 examples from the DeepCAD dataset.

Taking inspiration from Wu. et. al., for every valid generation, we ask our Eval Agent to provide four scores based on the original prompt and the images of the rendered model. Adherence: how well the generated model aligns with the description in the prompt. Detail: whether the model provided a model with sufficient detail. Consistency: how well the model shows consistency in geometry and topology. Correctness: How error-free the model is. An example of a error is an invalid hole, an unwanted gap, or other geometric issues. Each score is given as an integer between 1 and 10, where 1 represents poor performance and 10 represents excellent performance.

Below is the average for each of these metrics across the evaluation set, the metrics collected from MARC were taken using the highest scoring submission rather than the final submission. We consider this to be a valid experiment setup as this same strategy can be used at inference time in applications.

Framework	Adherance	Detail	Consistency	Correctness
Text2CAD	5.3	4.1	7.3	5.5
MARC	5	6	7	7

Note the difference in precision is due to an accidental flooring of the MARC scores.

5.3 Evaluating the Evaluation Agent

While validating the accuracy of the CAD generations is key to assessing the Design Agent, it is also important for us to understand how well the signal the Eval Agent is providing to the Design Agent is actually improving the generations.

We will leverage deterministic quantitative metric that was also used by Text2CAD to measure the reconstruction loss with the original CAD models provided by DeepCAD. The metric we chose is the Chamfer Distance (CD).

The Chamfer Distance (CD). Chamfer Distance measures the similarity between two sets of points, X and Y, by taking the sum of the distance between every point in X and its closest neighbor in Y and vice versa. We iterate through our successful generations, for each generation prompt we construct the point cloud from the step files that are a part of the DeepCAD and use this as our grand truth.

We will again rely on the Chamfer Distance. For all of our generations we iterate through all of the valid submissions, for each submission that succeeded as part of the Design Agents iterative refinement loop for that prompt, we calculate the CD. We take the average CD of all our examples at each index in the submission loop and use this this to assess how the average CD changes at each submission index.

The following graph shows this progress, as we can see there is continuous improvement up until submission 5. After which, there is a decline.

Now we will also measure alignment between the LLM Judge scores and the Chamfer Distance.

Figure 2: CD over time

5.4 Real World Examples

In order to assess how well each framework generalizes to real world task examples that do not fall under the category of the very specific geometric objects that are provided in the code base, we use Gemini to construct 50 real world examples and we construct CAD models from them using both frameworks. We used the full Text2CAD pipeline that provides the same prompt to the LLM to create instructions for the Text2CAD transformer. Because we don't have reference CADs to perform the Chamfer Distance on, we will again rely on LLM as a Judge metrics.

6 Analysis

Generations for Text2CAD collapse to a simple cube for anything that does look very similar to prompts in the DeepCAD codebase. The DeepCAD code base consists primary of very simplistic parts such as screws and simple hexagons, and the model does not show signs of being able to compose these parts using guidance from the LLM. Our Agentic Framework however proved successful at constructing real world items such as a desk. While this is far from the kind of complexity that you would expect from a expert CAD engineer, one of the main limitations appears to be deteriorating

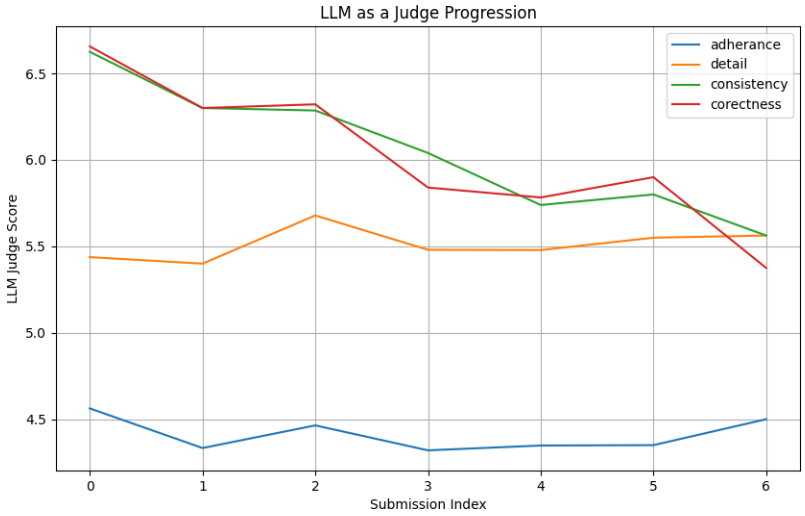

Figure 3: LLM Judge Scores over time

performance over longer context lengths. This is evidence by continuous average performance increase over the first 6 attempts and then performance deterioration after.

- Adapting to make the the python library more intuitive

7 Conclusions

Despite not having any pretraining we were able to attain better performance than Text2CAD, however our framework proves to be context and latency bound. Text2CAD however is more severely context bound as it only supports up to 500 text tokens. Without standardized benchmarks, we can't contend to have a superior pipeline, however we have shown how LLMs can adapt to tasks that may appear outside of their domain simply by adjusting the problem formulation. The fact that Claude was always able to eventually correct its programming errors and come up with a valid construction using the library that it only had seen one example of, is a testament to the benefits of test-time scaling. Going forward, significant work will need to be done to improve evaluation mechanisms for Generative CAD models.

References

[1] Jerry Liu, Fisher Yu, and Thomas A. Funkhouser. Interactive 3d modeling with a generative adversarial network. *CoRR*, abs/1706.05170, 2017.

[2] Zhen Wang, Dongyuan Li, and Renhe Jiang. Diffusion models in 3d vision: A survey, 2024.

[3] Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models, 2021.

[4] Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin Sheikh, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2cad: Generating sequential cad models from beginner-to-expert level text prompts, 2024.

[5] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges, 2024.

[6] Chris Sypherd and Vaishak Belle. Practical considerations for agentic llm systems, 2024.

[7] Mohsen Yavartanoo, Sangmin Hong, Reyhaneh Neshatavar, and Kyoung Mu Lee. Text2cad: Text to 3d cad generation via technical drawings, 2024.

[8] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

[9] Tong Wu, Guandao Yang, Zhibing Li, Kai Zhang, Ziwei Liu, Leonidas Guibas, Dahua Lin, and Gordon Wetzstein. Gpt-4v(ision) is a human-aligned evaluator for text-to-3d generation, 2024.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[11] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning, 2023.

[12] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024.

[13] OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, Alex Renzin, Alex Tachard Passos, Alexander Kirillov, Alexi Christakis, Alexis Conneau, Ali Kamali, Allan Jabri, Allison Moyer, Allison Tam, Amadou Crookes, Amin Tootoochian, Amin Tootoonchian, Ananya Kumar, Andrea Vallone, Andrej Karpathy, Andrew Braunstein, Andrew Cann, Andrew Codispoti, Andrew Galu, Andrew Kondrich, Andrew Tulloch, Andrey Mishchenko, Angela Baek, Angela Jiang, Antoine Pelisse, Antonia Woodford, Anuj Gosalia, Arka Dhar, Ashley Pantuliano, Avi Nayak, Avital Oliver, Barret Zoph, Behrooz Ghorbani, Ben Leimberger, Ben Rossen, Ben Sokolowsky, Ben Wang, Benjamin Zweig, Beth Hoover, Blake Samic, Bob McGrew, Bobby Spero, Bogo Giertler, Bowen Cheng, Brad Lightcap, Brandon Walkin, Brendan Quinn, Brian Guarraci, Brian Hsu, Bright Kellogg, Brydon Eastman, Camillo Lugaresi, Carroll Wainwright, Cary Bassin, Cary Hudson, Casey Chu, Chad Nelson, Chak Li, Chan Jun Shern, Channing Conger, Charlotte Barette, Chelsea Voss, Chen Ding, Cheng Lu, Chong Zhang, Chris Beaumont, Chris Hallacy, Chris Koch, Christian Gibson, Christina Kim, Christine Choi, Christine McLeavey, Christopher Hesse, Claudia Fischer, Clemens Winter, Coley Czarnecki, Colin Jarvis, Colin Wei, Constantin Koumouzelis, Dane Sherburn, Daniel Kappler, Daniel Levin, Daniel Levy, David Carr, David Farhi, David Mely, David Robinson, David Sasaki, Denny Jin, Dev Valladares, Dimitris Tsipras, Doug Li, Duc Phong Nguyen, Duncan Findlay, Edede Oiwoh, Edmund Wong, Ehsan Asdar, Elizabeth Proehl, Elizabeth Yang, Eric Antonow, Eric Kramer, Eric Peterson, Eric Sigler, Eric Wallace, Eugene Brevdo, Evan Mays, Farzad Khorasani, Felipe Petroski Such, Filippo Raso, Francis Zhang, Fred von Lohmann, Freddie Sulit, Gabriel Goh, Gene Oden, Geoff Salmon, Giulio Starace, Greg Brockman, Hadi Salman, Haiming Bao, Haitang Hu, Hannah Wong, Haoyu Wang, Heather Schmidt, Heather Whitney, Heewoo Jun, Hendrik Kirchner, Henrique Ponde de Oliveira Pinto, Hongyu Ren, Huiwen Chang, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian O'Connell, Ian Osband, Ian Silber, Ian Sohl, Ibrahim Okuyucu, Ikai Lan, Ilya Kostrikov, Ilya Sutskever, Ingmar Kanitscheider, Ishaan Gulrajani, Jacob Coxon, Jacob Menick, Jakub Pachocki, James Aung, James Betker, James Crooks, James Lennon, Jamie Kiros, Jan Leike, Jane Park, Jason Kwon, Jason Phang, Jason Teplitz, Jason Wei, Jason Wolfe, Jay Chen, Jeff Harris, Jenia Varavva, Jessica Gan Lee, Jessica Shieh, Ji Lin, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joanne Jang, Joaquin Quinonero Candela, Joe Beutler, Joe Landers, Joel Parish, Johannes Heidecke, John Schulman, Jonathan Lachman, Jonathan McKay, Jonathan Uesato, Jonathan Ward, Jong Wook Kim, Joost Huizinga, Jordan Sitkin, Jos Kraaijeveld, Josh Gross, Josh Kaplan, Josh Snyder, Joshua Achiam, Joy Jiao, Joyce Lee, Juntang Zhuang, Justyn Harriman, Kai Fricke, Kai Hayashi, Karan Singhal, Katy Shi, Kavin Karthik, Kayla Wood, Kendra Rimbach, Kenny Hsu, Kenny Nguyen, Keren Gu-Lemberg, Kevin Button, Kevin Liu, Kiel Howe, Krithika Muthukumar, Kyle Luther, Lama Ahmad, Larry Kai, Lauren Itow, Lauren Workman, Leher Pathak, Leo Chen, Li Jing, Lia Guy, Liam Fedus, Liang Zhou, Lien Mamitsuka, Lilian Weng, Lindsay McCallum, Lindsey Held, Long Ouyang, Louis Feuvrier, Lu Zhang, Lukas Kondraciuk, Lukasz Kaiser, Luke Hewitt, Luke Metz, Lyric Doshi, Mada Aflak, Maddie Simens, Madelaine Boyd, Madeleine Thompson, Marat Dukhan, Mark Chen, Mark Gray, Mark Hudnall, Marvin Zhang, Marwan Aljubeh, Mateusz Litwin,

Matthew Zeng, Max Johnson, Maya Shetty, Mayank Gupta, Meghan Shah, Mehmet Yatbaz, Meng Jia Yang, Mengchao Zhong, Mia Glaese, Mianna Chen, Michael Janner, Michael Lampe, Michael Petrov, Michael Wu, Michele Wang, Michelle Fradin, Michelle Pokrass, Miguel Castro, Miguel Oom Temudo de Castro, Mikhail Pavlov, Miles Brundage, Miles Wang, Minal Khan, Mira Murati, Mo Bavarian, Molly Lin, Murat Yesildal, Nacho Soto, Natalia Gimelshein, Natalie Cone, Natalie Staudacher, Natalie Summers, Natan LaFontaine, Neil Chowdhury, Nick Ryder, Nick Stathas, Nick Turley, Nik Tezak, Niko Felix, Nithanth Kudige, Nitish Keskar, Noah Deutsch, Noel Bundick, Nora Puckett, Ofir Nachum, Ola Okelola, Oleg Boiko, Oleg Murk, Oliver Jaffe, Olivia Watkins, Olivier Godement, Owen Campbell-Moore, Patrick Chao, Paul McMillan, Pavel Belov, Peng Su, Peter Bak, Peter Bakkum, Peter Deng, Peter Dolan, Peter Hoeschele, Peter Welinder, Phil Tillet, Philip Pronin, Philippe Tillet, Prafulla Dhariwal, Qiming Yuan, Rachel Dias, Rachel Lim, Rahul Arora, Rajan Troll, Randall Lin, Rapha Gontijo Lopes, Raul Puri, Reah Miyara, Reimar Leike, Renaud Gaubert, Reza Zamani, Ricky Wang, Rob Donnelly, Rob Honsby, Rocky Smith, Rohan Sahai, Rohit Ramchandani, Romain Huet, Rory Carmichael, Rowan Zellers, Roy Chen, Ruby Chen, Ruslan Nigmatullin, Ryan Cheu, Saachi Jain, Sam Altman, Sam Schoenholz, Sam Toizer, Samuel Miserendino, Sandhini Agarwal, Sara Culver, Scott Ethersmith, Scott Gray, Sean Grove, Sean Metzger, Shamez Hermani, Shantanu Jain, Shengjia Zhao, Sherwin Wu, Shino Jomoto, Shirong Wu, Shuaiqi, Xia, Sonia Phene, Spencer Papay, Srinivas Narayanan, Steve Coffey, Steve Lee, Stewart Hall, Suchir Balaji, Tal Broda, Tal Stramer, Tao Xu, Tarun Gogineni, Taya Christianson, Ted Sanders, Tejal Patwardhan, Thomas Cunninghman, Thomas Degry, Thomas Dimson, Thomas Raoux, Thomas Shadwell, Tianhao Zheng, Todd Underwood, Todor Markov, Toki Sherbakov, Tom Rubin, Tom Stasi, Tomer Kaftan, Tristan Heywood, Troy Peterson, Tyce Walters, Tyna Eloundou, Valerie Qi, Veit Moeller, Vinnie Monaco, Vishal Kuo, Vlad Fomenko, Wayne Chang, Weiyi Zheng, Wenda Zhou, Wesam Manassra, Will Sheu, Wojciech Zaremba, Yash Patil, Yilei Qian, Yongjik Kim, Youlong Cheng, Yu Zhang, Yuchen He, Yuchen Zhang, Yujia Jin, Yunxing Dai, and Yury Malkov. Gpt-4o system card, 2024.