TreeSearch at SemEval-2025 Task 8: Monte Carlo Tree Search for Question-Answering over Tabular Data

Aakarsh Nair and Huxin Yang
Seminar für Sprachwissenschaft
Eberhard Karls Universität Tübingen, Germany
{aakarsh.nair, huxin.yang}@student.uni-tuebingen.de

Abstract

Large Language Models (LLMs) can answer diverse questions but often generate factually incorrect responses. SemEval-2025 Task 8 focuses on table-based question-answering, providing 65 real-world tabular datasets and 1,300 questions that require precise filtering and summarization of underlying tables.

We approach this problem as a neuro-symbolic code generation task, translating natural language queries into executable Python code to ensure contextually relevant and factually accurate answers. We formulate LLM decoding as a Markov Decision Process, enabling Monte Carlo Tree Search (MCTS) as a lookahead-based planning algorithm while decoding from the underlying code-generating LLM, instead of standard beam search.

Execution success on synthetic tests and real datasets serves as a reward signal, allowing MCTS to explore multiple code-generation paths, validate outcomes, assign value to partial solutions, and refine code iteratively rather than merely maximizing sequence likelihood in a single step. Our approach improves accuracy by 2.38x compared to standard decoding.[1]

1 Introduction

Large Language Models (LLMs) based on the Transformer architecture (Vaswani et al., 2023) have demonstrated impressive performance in natural language question answering (QA) (OpenAI et al., 2024; Grattafiori et al., 2024). However, despite their fluency, these models often suffer from hallucinations—plausible but factually incorrect responses that deviate from source data (Ji et al., 2023). Detecting and mitigating these errors remains a major challenge (Farquhar et al., 2024), particularly in applications that require precise, structured answers.

Table 1: Sample questions from the Semeval Task-8 training set of along with their expected answer types.[2]

Type	Dataset ID	Example Question
Boolean	066_IBM_HR	Is our average employee older than 35?
Boolean	072_Admissions	Is there an applicant with a chance above 95 per cent of getting into the university they applied to?
Category	068_WorldBank_Awards	Which region has the most contracts?
Category	079_Coffee	What is the location of our most popular store?
Number	066_IBM_HR	What is the average age of our employees?
Number	075_Mortality	What is the total sum of all death rate values?
List[Category]	072_Admissions	List the university ratings associated with the top 5 CGPA scores.
List[Category]	080_Books	List the categories of the first five books.
List[Number]	077_Gestational	List the 3 heights of the 3 tallest women in the dataset (in cm).
List[Number]	078_Fires	What are the 3 hottest temperatures recorded?

One domain where factual consistency is critical is table-based QA, where answers must be derived from structured tabular data. SemEval-2025 Task 8 (Osés Grijalba et al., 2024) provides a benchmark for evaluating LLMs in this setting, requiring models to generate accurate answers from real-world datasets. Table 1 lists sample questions from the SemEval-2025 Task 8 training set of questions along with their expected answer types.

While databases have long relied on structured query languages (SQL) for precise data retrieval (Codd, 1970), translating natural language questions into executable queries remains a challenging task (Nan et al., 2022; Zhong et al., 2017).

In this task we aim to leverage code-generating Large Language Models (Rozière et al., 2024) to query tabular data. That is given a natural language query and the underlying table schema we attempt to generate a concise Pandas Python code (Wes McKinney, 2010) to answer questions regarding the given table data frame. This allows us to leverage existing code generation models like CodeLlama-7b-Python [3] model which are extensively trained on python code generation.

For reference, the organizers provide training data consisting of 1.3k data points each includes a

[1]Our code is available at: https://github.com/HuixinYang/SemEval25-Task8-Adrianna-Aakarsh-Yang

[2]Full training dataset is available at: https://huggingface.co/datasets/cardiffnlp/databench

[3]CodeLlama-7b-Python is openly available at: https://huggingface.co/codellama/CodeLlama-7b-Python-hf

natural language question, the target answer and its data type, relevant columns with their data types, and associated data frame (Grijalba et al., 2024). Queries are run in either *lite* mode using first 20 rows of the table or full mode using the entire table, with target answers for both modes provided. The organizers additionally provide a baseline decoder only code LLM, based on Stable Code 3b (Pinnaparaju et al., 2024), which tries to generate Python-Pandas code to be run against the underlying data-frame. Using a simple beam-search based decoder the LLM is able to achieve an accuracy of 27% in lite querying mode and 26% on full table querying mode on the task's test set.

While simple and effective, one of the draw backs of the simple beam-search based decoder which maximizes the likelihood of the generated sequence is that it fails to take into account of ground truth program quality information to effectively plan and guide the model during the program generation process. We thus propose the application of a Monte Carlo Tree Search (MCTS) planning based decoder (Zhang et al., 2023) as a wrapper around the base Code LLM (CodeLlama-7b-Python model) to facilitate table-question answering grounded by the runtime behavior of generated queries.

During the decoding process, the MCTS Planner is able to take advantage of a reward signal from terminal states to backpropogate and value intermediate decodings of the underlying Transformer model. The MCTS planner is able to balance exploration and exploitation in the token decoding process to generate a higher-quality set of possible Python-Pandas completions running them against the dataframe and automatically generated test-cases to weigh possible next tokens through a look-ahead process, while the transformer next token probabilities serve as a good heuristic to constraint the planners search space. Moreover the flexibility of the reward function allows us to to use a variety of much larger models to inform our judgments of the generated program solutions. For example we are able to use automatically generated test-cases from a `Qwen2.5-Coder-32B-Instruct` (Hui et al., 2024) model on synthetic data following the underlying table schema to validate and generate a reward signal for our decoding process.

2 Related Work

Reinforcement Learning (RL) has a rich literature of planning to take actions in an environment so as to maximize a reward signal (Sutton et al., 1998). Planning algorithms like Monte Carlo Tree Search (MCTS) has been effectively applied to games involving long term planning like Chess, Shogi and Go (Silver et al., 2016) where the reward signal is the sparse consisting of the outcome of the games. Application for reinforcement learning for natural language query translation to languages like Structured Query Languages (SQL) have been explored in (Zhong et al., 2017). Frameworks like (Anthony et al., 2017) define a general framework using search in planning algorithms and learn complicated behavior policies in simulation environments. LLMs themselves have been shown to have remarkable reasoning abilities in tackling complex mathematical, logical and programming tasks (OpenAI et al., 2024) even trained on apparently autoregressive next token prediction objectives. However there is a rich and growing literature on incorporating planning into LLM generation process. These vary from prompting strategies like chain of thought (Wei et al., 2023) which encourage LLM to act like step wise reasoners, to models explicitly model reasoning with *search*, *reward* and *reasoning* components (Hao et al., 2024).

3 Our Approach

Our approach applies planning based transformer decoding (Zhang et al., 2023) to table-question answering. Novel query program code synthesis is formulated as an Markov Decision Process (Sutton et al., 1998). A partial program along with its prompting description is considered to be a *state* s. The act of selecting a next-token from the underlying Transformer vocabulary is considered an *action* a. Thus *transition function* moves from one partial program to another by concatenating a selected token to one partial program to form another until a terminal token is appended. The *reward* for a program is the sum total of the validity of the program along with the number of synthetic test cases the generated program passes. The aim is to use the LLM to search for a path which maximizes expected future reward: $\sum_{t=0}^{n} r(s_t, a_t)$.

3.1 Monte-Carlo Tree Search Decoding

A *Monte-Carlo Tree Search (MCTS)*-based planner maximizes the accumulated reward of the gener-

ated program, replacing beam search, which prioritizes similarity to reference solutions but cannot explicitly optimize execution quality, making it sample-inefficient.

MCTS (Silver et al., 2016) treats planning as a look-ahead search through tree of *actions* to find a path to terminal nodes with highest-rewards. Search proceeds from the root node with child nodes representing next-token actions selected by the planning algorithm in phases meant to balance *exploration* and *exploitation*. The four phases are: *Selection*: A process of selecting which of the root's child node to examine, *Expansion*: A process of adding child nodes for the top-k most likely next tokens for given node (as guided by the Transformer model), *Evaluation*: Expanding greedily using beam search on Transformer model to the terminal node in order to estimate program reward and *Backpropogation*: Updating a node's ancestor's with visit counts and ground truth observed rewards. Throughout its execution for each node the planning algorithm maintains a node-visit count and $Q(s, a)$ which is the average reward for the algorithm taking action a when starting from state s. A *rollout* consists of a one full generation of a sample program and its final computed *reward*. All four phases are run as part of every full rollout. The algorithm maintains search tree of tokens till the required number of rollouts are processed.

3.1.1 Selection

This process starts at the root node and recursively select its child node until a leaf node is reached. The root node consists of the full prompt up until the slotted location for query generation. With its children the possible next tokens in the generation. The selection phase traverses will recursively select actions a down till it reaches an unexpanded node using a variant of the Predictive Upper Confidence Bound(P-UCB) (Silver et al., 2016; Zhang et al., 2023) action selection criterion, which balances *exploration* with *exploitation*, when selecting next token a to explore.

$$\text{argmax}_a \text{P-UCB}(s, a) = Q(s, a) + C \cdot P_{\text{LLM}}(a|s) \cdot \frac{\sqrt{\ln N(s)}}{1 + N(s')}$$

Where $Q(s, a)$ is the reward of the best program generated from this node, $P_{\text{LLM}}(a|s)$ is the transformers predicted probability that a is the next token and $N(s)$ and $N(s')$ are the prior visit counts of states s and the state s' achieved when we transition from state s to state s' by taking the token-concatenation action a, C is a ucb-constant which used to control the scaling of the *exploration* term.

3.1.2 Expansion

Once we reach an unexpanded/unexplored we discover its possible succeeding children by using the Code LLM *Top-k* next tokens. Thus the language model constraints the search-paths of next tokens reducing the probability that we will sample a syntactically invalid next token. The k represents the maximum number of child count of a node determines the fan-out size of our tree. Nodes for each of the sampled next tokens are created and added to the search tree.

3.1.3 Evaluation/Simulation

As the node added to the tree maybe a partial program, LLM *beam-search* is used generate a possible full completion of the program till a terminal node. The a full-suite test cases cases is run of on this greedily generated program if the program generated by this default policy is executable, and observed reward is recorded as a rollout. Criterion contributing the the reward include success in compiling and executing against the underlying Dataframe and the number of test-cases passed on a synthetic Dataframe following the same schema as original Dataframe.

3.1.4 Back Propagation

In the final phase the observed reward along with the visitation count are populated up the through the ancestors of the current node so as to contribute to future look-ahead searches and the ancestors P-UCB criterion leading to refining of the tree exploration policy over each subsequent rollout.

3.2 Answer Selection

The final results of running all roll-outs is a dictionary of all programs generated and their corresponding observed rewards from the evaluation phases. The chosen program is one which achieves the maximum rewards with the majority computed answer used to break tied rewards.

4 Experimental Setup

4.1 MCTS Decoder Configuration

We run the MCTS decoder (Zhang et al., 2023) for the base model `CodeLlama-7b-Python-hf` (Rozière et al., 2024), using a *horizon* of 32, which

controls the maximum number of steps taken or tokens produced. A total of 100 *rollouts* is performed, determining the number of programs generated for each question in the test set.

We use *P-UCB* as the *node selection algorithm* during the selection phase, with an expansion *width* of 5, which specifies the number of children each node can expand into. No *temporal discounting* is applied ($\gamma = 1$), meaning rewards are not penalized based on the number of steps taken to achieve them.

The base model is sampled using $\text{top}_k = 3$ and $\text{top}_p = 0.9$, with a *temperature* of 0.2 during the simulation/evaluation phase. Extensive hyperparameter tuning will be considered in future work on this task.

4.2 Enriched Prompting

We enrich the root prompt used by the base decoder to be to contain contextually relevant information which might be helpful for during the MCTS decoding. Thus the our root prompt contains: (1) Entire Dataframe *schema* for the table we are generating our prompt, including column names and types (2) A *predicted return type* depending on the question we are answering. We recognize 5 output categories for the task. *boolean*, *category*, *number*, *list[category]* , for list of categories and *list[number]* for a list of numbers. (3) *predicted columns used*, our prediction of the columns most relevant for answering the question.

We use a open source code LLM `Llama-3.1-Nemotron-70B-Instruct` (Wang et al., 2024), to predict the question category, and the `Qwen2.5-Coder-32B-Instruct` (Hui et al., 2024) to predict the columns to be used for answering user's natural language query.

4.3 Reward Function

The MCTS function is highly sensitive to the design of the reward function as the presence or absence of rewards guides the tree search procedure. As user questions are open-ended it can often be hard to verify whether the generated code and responses are indeed accurate.

We use a three pronged approach to guessing at users intent. (1) First we use a strictly larger model to predict from the users query the most probable output type. This probable output type is used to inform both the prompt used during the decoding process as well as *type-check* the generated responses. (2) Additionally the generated code is evaluated using Python interpreter with the dataframe in question in the context of the interpreter. The executable section of the code is *parsed* and extracted from the generated model response. (3) Thirdly the executable code fragment is run against a synthetic *test-suite* to verify that it passes multiple tests.

4.3.1 Sanity Type Checking and Malformed Code Detection

We extract the completion's Pandas query as a single line following the prompt's return statement, truncating tokens any additional extraneous tokens. While currently no explicit token penalty is applied, the short *horizon*, biases the model toward concise outputs. The extracted code is then executed against the relevant data-frame which is added to the interpreter context, and if a runtime error occurs *or* the resulting semantic data type differs from the expected output type, a single error penalty of -1 is imposed. This combination of execution-based feedback and type validation ensures that completions align with each query's requirements.

4.3.2 Test Set Generation

In the spirit of *test-driven development*, we leverage large open-source models (`Llama-3.1-Nemotron-70B-Instruct`, `Qwen2.5-Coder-32B-Instruct`) to generate test cases for a given query. Specifically, we prompt these models with the table schema and instruct them to generate randomized data, parameterized by a *random seed*, while ensuring compliance with the DataFrame schema. Since these models have access to the table schema, sample data, and user query, they can produce meaningful assertions for validating Python Pandas queries.

During the *evaluation phase*, we run the MCTS- decoder's generated Pandas on the dummy DataFrame and verify that all assertions hold. Using different random seeds allows us to execute the query multiple times with varying dummy data, increasing robustness. Additionally, leveraging a diverse ensemble of models for test-case generation enhances confidence in the correctness of a query that satisfies multiple test cases, rather than relying on any single test instance. A total of five tests are run for each query, with each passing test contributing 0.1 to the final reward value.[4]

[4]Set of generated tests for training set for competition are available at: `https:`

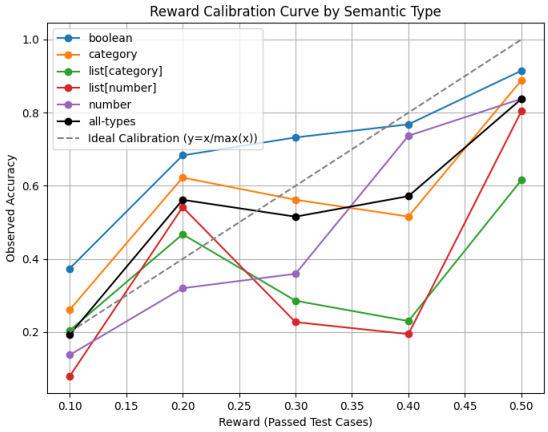

Figure 1: Plot shows the reward calibration for rollouts which passed at least one of the test cases. We note that boolean, category, and number show good calibration while list[category], list[number] show relatively poorer calibration indicating that further testing is required during rollout computation to ensure their accuracy. A maximum of 5 tests are run for each roll out with random seed. With a reward of 0.1 for each successfully passed test.

4.3.3 Reward Value

We use a mixture of penalties for faulty generation and rewards based on number of tests passed to evaluate MCTS rollouts.

$$r(\text{Query}) = -1 \cdot \mathbb{I}_{\text{error}} + 0.1 \cdot p \cdot (1 - \mathbb{I}_{\text{error}})$$

Where p is the number of test cases passed by the completion, and $\mathbb{I}_{\text{error}}$ is binary indicator function with values 0 or 1 where 1 indicates the program encouters and error during execution.

5 Results

Table 2 gives the result of running MCTS decoder with reward we compare the results by category of output types. We note that use of reward function leads to substantial improvement over the baseline. Figure 1 shows that for atomic return types such as *boolean*, *category* and *number*, rewards are well calibrated. That is passing higher number of tests in the synthetic test-suite corresponds to higher observed accuracy on the evaluation benchmark.

As MCTS decoder does not rely on in-context learning learning accuracy for both lite and and base strategies are roughly equivalent. We note that that the results requiring list outputs tend to have the worse performance. While boolean outputs have the highest accuracy level. Table 3 we note that MCTS decoding has a baseline accuracy of 61.68% on base tests and 64.36% on lite tests, compared to the baseline provided by the host on the test set which is 27% base and 26% on the lite dataset, corresponding to a relative improvement is 128.44% on the base test set and 147.54% on the lite test set compared to the baseline.

6 Conclusion

In this work, we apply a Monte Carlo Tree Search (MCTS) based decoder from (Zhang et al., 2023) to generate code for table-based question answering in SemEval-2025 Task 8.

Unlike conventional autoregressive methods that rely on greedy decoding or single-step chain-of-thought, our approach generates multiple candidate programs and refines them through look-ahead planning. Experimental results show that MCTS decoding achieves a substantial accuracy improvement (61.68% vs. 26% baseline decoding), demonstrating the effectiveness of search-based reasoning for code generation tasks.

Our techniques provide us a way to leverage multiple strong open source models to guide the decoding process of a much smaller local model, allowing us to generate multiple solutions to the same problem.

By using tree-search incorporating partial reward signals (e.g., number of passed test cases) and a semantic-type check, we further showcased how the model can refine solutions by balancing exploration and exploitation.

Our experiments also revealed that certain output types, such as lists, require more robust intermediate checks to ensure correctness then simpler atomic types for which the reward signal was well calibrated to the expected accuracy of the answer.

Overall, this work highlights the potential of tree-based planning in code generation and paves the way for more advanced reasoning-based techniques for table-centric QA. We hope this approach can generalize to other complex problem settings that involve which involves utilizing the strengths of multiple models to be used in concert for robust and factually accurate question answering across multiple data sources.

`//huggingface.co/datasets/aakarsh-nair/semeval-2025-task-8-test-cases-competition`

Output Type	MCTS Accuracy (%)	MCTS Accuracy (%) - Lite
Boolean	**76.74**	**74.41**
Category	64.86	67.57
Number	62.82	66.02
List [Category]	50.00	52.77
List [Number]	45.05	53.84
Overall	**61.68**	**64.36**

Table 2: Breakdown of MCTS-decoder accuracy by output type. The overall baseline accuracy is 26% and 27% for base and lite datasets, respectively. Accuracy for list-based outputs is lower than for single-entity outputs.

Model + Decoding	Base Accuracy (%)	Lite Accuracy (%)
Stable Code-3b-GGUF + Beam Search	26.00	27.00
CodeLlama-7b Python + MCTS Decoding	**61.68**	**64.36**

Table 3: Overall numerical accuracy comparison between the baseline (Stable Code-3b-GGUF with beam search) and our approach (CodeLlama-7b Python with MCTS decoding).

Output Type	Category Prediction (%)
Boolean	**100.00**
Category	**100.00**
Number	96.79
List [Category]	98.61
List [Number]	82.41
Overall	**95.78**

Table 4: Category prediction accuracy across different output types. Category prediction serves as one of the components in the reward signal. Programs which return output types different from predicted category types are penalized. We note that Boolean and Category return types were predicted with perfect accuracy on that set using sufficiently large models.

7 Limitation and Future Work

While our MCTS-based approach for table-centric QA significantly boosts accuracy, several limitations remain. First, we frequently observe *semantically identical but syntactically distinct* programs, resulting in unnecessary redundancy. This is compounded by occasional *extraneous tokens* at the end of generated completions, which we trim to prevent run-time errors but consequently reduce program diversity. Developing more robust code filters or pruning heuristics could improve the uniqueness and readability of generated solutions.

Second, although MCTS captures partial credit via our reward function, *reward design* remains imperfect. For example, incomplete or slightly incorrect solutions may pass partial tests without reflecting deeper logical errors. Future work could explore more fine-grained reward signals—such as dynamic coverage metrics or adversarial test generation—to improve the fidelity of feedback.

Third, we currently *do not fine-tune the model* on the MCTS rollouts or incorporate reward signals into a specialized training loop. Integrating *Expert Iteration* (Anthony et al., 2017) or iterative feedback mechanisms could further refine the policy beyond what standard prompting achieves. Additionally, we have not conducted extensive *hyperparameter searches* for aspects such as number of rollouts or maximum program length, potentially leaving performance gains on the table.

Fourth, our experiments focus on single-table question answering. Many real-world tasks involve *multiple tables* and heterogeneous data sources, requiring more advanced data integration strategies. Future iterations of our system could merge or join multiple data frames, broadening its applicability to multi-step queries.

Lastly, we anticipate that *adversarial testing*—specifically designed to confuse large language models—could uncover edge cases not seen in our current evaluation. Employing adversarially generated queries and integrating them into the MCTS rollouts would likely bolster the robustness

of the final solutions.

Overall, these considerations highlight avenues to enhance both the theoretical foundations and practical performance of our approach, from improved reward engineering to multi-table integration and expert iteration training cycles.

8 Acknowledgments

We would like to thank Dr. Çagrı Çöltekin, for his guidance, feedback and boundless optimism which helped in the completion of this project. The authors also acknowledge support by the state of Baden-Württemberg through bwHPC cluster on which most of the experiments were run.

References

Thomas Anthony, Zheng Tian, and David Barber. 2017. Thinking fast and slow with deep learning and tree search.

Edgar F Codd. 1970. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387.

Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. 2024. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630(8017):625–630.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Fe-

ichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024. The llama 3 herd of models.

Jorge Osés Grijalba, Luis Alfonso Ureña-López, Eugenio Martínez Cámara, and Jose Camacho-Collados. 2024. Question answering over tabular data with databench: A large-scale empirical evaluation of llms. In *Proceedings of LREC-COLING 2024*, Turin, Italy.

Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma, Adithya Samavedhi, Qiyue Gao, Zhen Wang, and Zhiting Hu. 2024. Llm reasoners: New evaluation, library, and analysis of step-by-step reasoning with large language models.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. 2024. Qwen2.5-coder technical report.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM computing surveys*, 55(12):1–38.

L. Nan et al. 2022. Fetaqa: Free-form table question answering. *Transactions of the Association for Computational Linguistics*, 10:35–49.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben

Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. Gpt-4 technical report.

Jorge Osés Grijalba, L. Alfonso Ureña-López, Eugenio Martínez Cámara, and Jose Camacho-Collados. 2024. Question answering over tabular data with DataBench: A large-scale empirical evaluation of LLMs. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 13471–13488, Torino, Italia. ELRA and ICCL.

Nikhil Pinnaparaju, Reshinth Adithyan, Duy Phung, Jonathan Tow, James Baicoianu, Ashish Datta, Maksym Zhuravinskyi, Dakota Mahan, Marco Bellagente, Carlos Riquelme, and Nathan Cooper. 2024. Stable code technical report.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Richard S Sutton, Andrew G Barto, et al. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need.

Zhilin Wang, Alexander Bukharin, Olivier Delalleau, Daniel Egert, Gerald Shen, Jiaqi Zeng, Oleksii Kuchaiev, and Yi Dong. 2024. Helpsteer2-preference: Complementing ratings with preferences.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, pages 56 – 61.

Shun Zhang, Zhenfang Chen, Yikang Shen, Mingyu Ding, Joshua B. Tenenbaum, and Chuang Gan. 2023. Planning with large language models for code generation.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning.