ChaI-TeA: A Benchmark for Evaluating Autocompletion of Interactions with LLM-based Chatbots

Shani Goren[1*,2] **Oren Kalinsky**[1] **Tomer Stav**[1] **Yuri Rapoport**[1] **Yaron Fairstein**[1]
Ram Yazdi[1] **Nachshon Cohen**[1] **Alexander Libov**[1] **Guy Kushilevitz**[1]
[1]Amazon Research [2]Technion - Israel institute of technology

{shani.goren, orenkalinsky, yyfairstein}@gmail.com
{alibov, stavt, rtu, ramyazdi, nachshon, guyk}@amazon.com

Abstract

The rise of LLMs has deflected a growing portion of human-computer interactions towards LLM-based chatbots. The remarkable abilities of these models allow users to interact using long, diverse natural language text covering a wide range of topics and styles. Phrasing these messages is a time and effort consuming task, calling for an autocomplete solution to assist users. We present **ChaI-TeA**: **Cha**t **Inte**raction **A**utocomplete; An autocomplete evaluation framework for LLM-based chatbot interactions. The framework includes a formal definition of the task, coupled with suitable datasets and metrics. We use the framework to evaluate 9 models on the defined auto completion task, finding that while current off-the-shelf models perform fairly, there is still much room for improvement, mainly in ranking of the generated suggestions. We provide insights for practitioners working on this task and open new research directions for researchers in the field. We release our framework[1], to serve as a foundation for future research.

1 Introduction

Large Language Models (LLMs) have revolutionized many NLP applications (Brown et al., 2020). A prominent example is automatic chatbots; what used to be confined, topic-specific applications often requiring the user to use restricted language or choose from a closed list of interaction options, have been transformed. These applications, powered by LLMs, are now one-stop-shops successfully communicating in unbounded natural language while acting as experts on a wide variety of topics (Achiam et al., 2023; Anil et al., 2023). Due to their remarkable abilities, LLM-based chatbots differ significantly from prior human-computer communication methods. Interactions with these chatbots are usually long, unique and cover a large range of topics and language styles, using unstructured natural language. Due to this nature, users invest much time and thought in communicating their needs to the chatbot, calling for solutions to reduce their effort (Lehmann and Buschek, 2021).

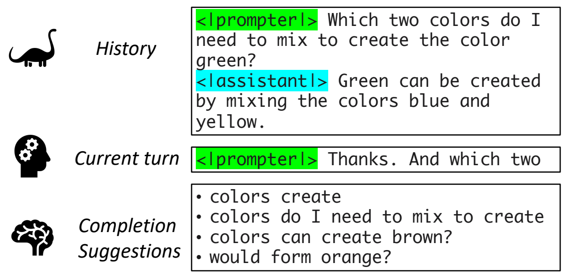

Figure 1: The chatbot interaction autocompletion task. Given the conversation history and the current turn's prefix, task is to suggest suitable completions.

AutoComplete (AC) methods have been shown to be effective in saving users' time and reducing their cognitive load in many different use-cases, suggesting that such a solution might be of value for the LLM-chatbot interaction use-case as well. The popular query autocomplete scenario (Cai et al., 2016) focuses on search queries. Classic solutions often rely on recurrence, making them irrelevant for the long unique natural language text found in chatbot interactions (Lehmann and Buschek, 2021). Later solutions include generative models (Sordoni et al., 2015; Park and Chiba, 2017), but still focus on short semi-structured queries. Code autocomplete (Liang et al., 2024) deals with structured language, and often relies on the ability to run the code and check its output in order to evaluate solutions. Lastly, email (human-human) interactions (Chen et al., 2019), which bear a closer resemblance to human-chatbot interactions due to their natural language communication, also differ in several key aspects. These include the number

[*]This project was done during an internship at Amazon.
[1]https://github.com/amazon-science/ChaiTea-chat-interaction-autocomplete

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers), pages 18–32
April 29 - May 4, 2025

of participants and their roles, the more formal writing style of emails and the nature of the topics discussed. In broader terms, human-human textual interactions (e.g., emails, but also texts from other kinds of messaging platforms) differ from human-chatbot interactions in the fact that human-chatbot interactions involve a human and a model-based assistant, making them more instructional and knowledge-seeking. For example, the prompts *"Give me the latest updates of the war in Ukraine as of the 31st of January."* and *"Write a web scraping program in python capable of ..."* are taken from the OASST dataset used in this work to demonstrate typical examples for a human-chatbot interaction, which are highly unlikely to be found in a human-human messaging platform.

In this paper, we introduce the task of autocompleting user interactions with LLM-based chatbots. We present **ChaI-TeA**: **Cha**t **Inte**raction **A**utocomplete; A framework for evaluating autocomplete solutions for LLM-based chatbot interactions. It includes a formal definition of the task, suitable datasets tailored for autocomplete, suitable metrics, and baseline results. We go on to highlight some valuable insights. First, we explore how performance can be traded off for lower latency, a key factor in autocomplete solutions. Second, we show that models can exploit distant history to suggest completions. Third, it is beneficial to enable completions of various lengths (as opposed to only single words or full turns). We highlight a key factor in improving these solutions: we find that models tend to generate completion suggestions well, but are not as good at ranking these generated suggestions. Given that users can ingest a small amount of suggestions at each turn, ranking is an important component in an offered solution. Therefore, we advocate for future research in the field to focus on this aspect.

2 Task Definition

The chatbot interaction completion task focuses on completing user turns in user-chatbot interactions. Similarly to (Chitnis et al., 2024), we model it as a sequential task; completions are suggested at each typing step (i.e., after a user types a character). Formally, at each step t, an autocomplete solution (denoted by AC) is given a context C containing all previous conversation turns, originating from both the user and the chatbot, and the prefix of the current user turn denoted as p_t. The autocomplete solution should then return a set of k completions, $c_{t_1}, ..., c_{t_k}$, possibly of varying lengths.

Each completion step can be described as:

$$AC(C, p_t) = \{c_{t_1}, c_{t_2}, ..., c_{t_k}\}$$

After receiving the set of completions, the user can either accept a completion or continue typing. If a completion c_{t_i} is accepted, the prefix is updated such that $p_{t+1} = p_t + c_{t_i}$. Then, whether the user selected a completion or continued typing, a new completion step is initiated, until reaching the end of the user's turn. A single completion step is illustrated in Figure 1, and full turns completions can be found in the Appendix in Table 6.

	OpenAssistant		ShareGPT	
	Train	**Test**	**Train**	**Test***
Conversations	5,144	277	88,259	1,190
Messages	22,749	1,182	317,536	1,494
Prefixes	536,215	26,394	16,801,251	22,323

Table 1: **Dataset Statistics.** *Since ShareGPT does not include a test split, we randomly sampled one of comparable size to the OASST test set.

3 Experimentation

3.1 Datasets

Open Assistant (OASST) (Köpf et al., 2024) is a human-annotated assistant conversation corpus. **ShareGPT**[2] contains user-LLM-chatbots conversations collected by the ShareGPT API.

To curate the data for our task, we take all English conversations and for each user-turn extract all possible prefixes and pair each with the entire conversation history up to that point as its context. The suffix of the original prompt is the ground truth completion. Table 1 summarizes the statistics of the datasets used in our experiments.

3.2 Metrics

As solutions are allowed to propose k completions at each step, metrics evaluate the performance taking k into account, denoted as $@k$.

As we are looking to form a benchmark, we turn to metrics that can be computed offline. We remark that ideally, we would also like to measure the user's saved time or reduced cognitive load but doing so would require running some experiment or user study for each new proposed solution.

For simplicity, we simulate acceptances (i.e., is one of the proposed completions accepted by the

[2]`https://sharegpt.com/`, dataset version that was used: *anon8231489123ShareGPT_Vicuna_unfiltered*

user?) using exact match comparison to the ground truth user turn.

Saved typing. Inspired by code completion metrics (Jiang et al., 2024), our goal is to save the user typing effort. Therefore, we seek a metric that quantifies the portion of the text completed by the AC solution. While simply dividing the length of the accepted text by the length of the full turn would achieve this, this metric would not consider the number of acceptances needed to generate the accepted text. To demonstrate this issue, consider two different solutions successfully completing the full turn; the first solution does this by completing single words one by one, while the other completes the entire turn in its first attempt. The naïve metric would score the two solutions the same, although it's clear we should prefer the second solution. To mitigate this issue, we propose the following metric:

$$\text{saved@}k = \frac{len(\text{accepted\_text}) - \#\text{acceptances}}{len(\text{full\_turn}) - 1}$$

where $len(x)$ is the number of characters in string x. No acceptances during the user's turn lead to a score of 0% while a single acceptance completing the full turn leads to a score of 100%.

Latency. Latency is a critical factor that cannot be overlooked when assessing AC solutions. Even if the completions are perfect, they are rendered useless if the user proceeds to type before receiving the suggestions. We report the mean and the 90^{th} percentile (p90) of the inference time.

3.3 Autocomplete Solutions

As our task resembles the language modeling task, a called-for solution is utilizing LMs. This allows us to experiment with a wide variety of models ranging in size, latency and quality, while avoiding extremely large LLMs as their latency is not feasible for this task[3]. Our evaluation encompassed a diverse set of popular LMs: Mistral-7B (Jiang et al., 2023), Gemma-7B (Mesnard et al., 2024), Phi-3-mini (Abdin et al., 2024), GPT-2-XL (Radford et al., 2018), Mamba (Gu and Dao, 2023), and SmolLm[4]. We also evaluate instruct-tuned variants of these models whenever one is available (Zephyr, Gemma)[5]. Inference was performed on a single NVIDIA A10G GPU, taking 150 hours in total.

To generate k completions from the LMs, we adopt the following procedure: we provide the model with the full context concatenated with the prompt prefix. We then use the model to generate n_c completions sampled with temperature 1.0, stopping when reaching EOS or after n_t tokens. Since completions can vary in length, each word-prefix of a completion can also be considered as a standalone completion. Hence, this process generates up to $n_c \times n_t$ completion candidates. Finally, we choose the k suggestions to present to the user by ranking the completions based on their perplexity score, computed using the LM probabilities:

$$PPL(w_1, w_2, \ldots, w_n) = e^{-\frac{1}{n}\sum_{i}^{n} \log p(w_i|w_1,\ldots,w_{i-1})}$$

3.4 Initiating Suggestion Generation

Suggesting completions after each character has some downsides compared to suggesting only at an end of a word. First, as the average length of an English word is more than 4 characters, the computational cost more than quadruples[6]. Second, it has been shown that when typing, users tend to pause much longer between words than between same-word characters (Conijn, 2020). This allows more room to suggest completions between words. Third, LLMs are known to under-perform on character level tasks, since most tokenizers only use character level tokens as a fallback[7] (Shin and Kaneko, 2024).

To compare how frequently character level suggestions are accepted compared to word level suggestions, we also tracked ***acceptance rate***: the percentage of completion steps that ended in an acceptance.

Results on the OpenAssistant validation set ($n_c = 5, n_t = 20$) show that mid-word suggestions degrade the acceptance rate by $\sim 60\%$ while only slightly improving saved@k by $\sim 3.2\%$. Interestingly, Mamba, which uses a character-level tokenizer, behaves similarly to the other models. Full details of this experiment are reported in Appendix A. We conclude that mid-word suggestions

[3]Generating completion suggestions with a 70B LLM takes on average 6 seconds.

[4]`https://huggingface.co/blog/smollm`

[5]The lack of published instruct-tuning datasets for some models prevents us from confirming the absence of data leakage. Still, our observations did not reveal any abnormal results.

[6]While using caching techniques can help mitigate some of the required compute, we observe (e.g., in Fig. 7) that token generation requires a considerable computation time, that cannot be mitigated using caching.

[7]For example, [DOG] is a token in most tokenizers, but given the prefix "I love my pet d", the model will likely use the character level token for [D], and the tokens [OG] or [O][G] are unlikely to be generated, since the model probably didn't encounter this token sequence during training.

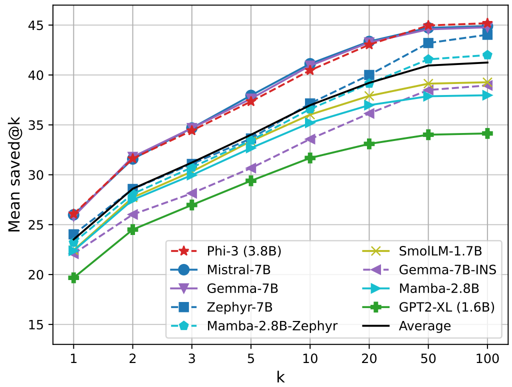

Figure 2: **saved@*k*** on OASST for varying k values.

are rarely accepted, and do not justify their drawbacks. Additional efforts are needed to make mid-word suggestions effective, which we leave for future work. For the remainder of this paper, completion suggestions are provided only at the end of a word. Consequently, throughout the rest of our experiments we observed that acceptance rate@k is highly correlated with saved@k. Therefore, we exclude acceptance rate results from the main paper and present it in the appendices.

3.5 Benchmarking ChaI-TeA

We benchmark all models described in Section 3.3 on both curated datasets (Section 3.1). Results on OASST for varying k values are shown in Figure 2. We consider k values up to 100, which encompasses all generated completions (at most, $n_c \times n_t$), to show the potential given a perfect ranking solution. While current models are able to perform fairly on this task – saving the user the typing of up to 45% of the characters – there is still much room for improvement. There is a noticeably large performance gap between small, realistic, k values and larger values, suggesting that while in many cases models are able to generate the correct completion, their ranking of completions is far from perfect. In line with prior work (Manakul et al., 2023; Ren et al., 2023; Fadeeva et al., 2024), we conclude that perplexity is insufficient for confidence ranking. Full benchmark results on OASST and ShareGPT can be found in Appendix B.

Finally, we observe that further improvement can be gained by fine-tuning models on the AC task. Detailed results are presented in Appendix C.

4 Further Analysis

Latency-Performance Trade-Off. Given the practical importance of latency in AC solutions, we explore how performance can be traded off for reduced latency. To illustrate this trade-off, we varied the previously mentioned hyperparameters n_c and n_t, as well as the context length given to the model. We capped the conversation history concatenated with the turn prefix at different lengths, to determine whether giving the model access to the entire conversation context is both helpful and worthy of the extra latency costs.

Suggestions are offered between words, meaning that once the user begins typing the next word they become irrelevant. Hence, we find it appropriate to use the mean time between typed words – 718 *ms*, reported by (Conijn, 2020) – as a benchmark.

Results per latency budget, presented in Table 2, show that it is preferable to generate more completions, while reducing the number of generated tokens and context length. Also, additional context is beneficial, suggesting that information useful for autocomplete can sometimes be found far before the end of the prefix. Results on all configurations are reported in the appendix in Table 9.

Latency Budget (ms)	Best Configuration n_c	n_t	Hist. Len	saved@100	Latency p90 (ms)
< 150	5	3	50	23.45	148
< 300	5	5	250	38.32	275
< 450	5	3	1000	41.10	388
< 600	5	5	1000	44.08	451
< 750	5	5	1000	44.08	451
> 750	5	10	*Full*	45.75	974

Table 2: **Latency-Performance Trade-Off.** Mistral-7B evaluated on the OASST test set. $n_c \in \{3, 4, 5\}$, $n_t \in \{3, 5, 10, 20\}$, and context length $len(C) \in \{50, 250, 1000, Full\}$ (measured in characters). In total, 48 hyper-parameter configurations were evaluated. For each latency budget, we report the configuration with the highest saved@100 score that fits the budget.

Varying completion lengths. A common practice for autocomplete practitioners wanting to simplify their methods is restricting completions to single words. The other end of this scale, also widely used, is allowing only full completions- completing until the end of the query/function/sentence. To this end, we compare completions of varying lengths to single word and full sentence completions to check whether allowing any-length completions improves quality. Average results across all models are presented In Table 3 (Full results can be found in Table

8). saved@k metric improves for $k = 100$ when allowing suggestions of varying length, indicating this can improve the user's typing experience. The fact that this is not the case for the lower k values indicates, once more, that the ranking method we use (the model's perplexity) is far from ideal.

	saved@1	saved@3	saved@100
Single Word	**24.10** / **22.28**	**31.97** / 28.63	33.12 / 29.52
Full	12.30 / 10.44	15.91 / 13.29	16.47 / 13.70
Partial	23.43 / 22.03	31.21 / **28.85**	**41.27** / **36.77**

Table 3: Average scores of partial completions vs single word and full sentences. OpenAssistant / ShareGPT.

Characteristics of completions. We observe that different models are able to generate diverse suggestions of different lengths. Completion suggestions offered by the different models are presented in Table 7. When looking at accepted completions, we see that while most acceptances are single word completions ($60\% - 70\%$), the models are able to generate longer acceptances; more than 15% span over 3 words or longer. The lengths of acceptances are presented in Figure 8.

5 Conclusions

In this work, we showcase the task of autocompleting user interactions with LLM-based chat-bots. We formally define the task and design an evaluation framework, and use it to test 9 different models. Results show that while LMs are able to perform fairly, there is room for a tailored solution to improve upon them, especially in the ranking of completion candidates. We show that models can exploit distant history, that enabling completions of different lengths is beneficial and that reducing latency for this task should be done by reducing context length and length of completions as opposed to generating less completions. We hope our framework will encourage further work in this area, which we believe holds great potential value for users across various LLM chat-bot applications.

Limitations

Exact Match. We use exact-match to simulate acceptances. While this is standard practice in autocomplete works, it may not fully represent real-world scenarios in which a user might accept a completion even if it's not the exact wording they were thinking of. Although some works use generation metrics like BLEU or ROUGE to simulate full sentence acceptances, these metrics fail to capture semantic similarity between partial completion suggestions and ground truths, making them a problematic solution because even a very high score may not represent an accept and vice versa. Moreover, it is a non-trivial task to infer what a user will accept after semantic partial matches since the text diverged from the ground truth. We evaluated using the Claude3-Sonnet model to determine whether a suggestion should be accepted or not and discovered this to be a very challenging task. Thus, we leave it for future work.

Datasets. Both datasets used have one significant limitation: they where collected without the presence of an autocomplete solution. It is possible that users alter their behavior when completion suggestions are presented to them. If this is true, it will not be reflected in our framework. We note that taking this into account is far from trivial, because even if data is collected in the presence of some autocomplete solution, this data will be biased towards the specific solution used in the collection process, giving an unfair advantage when judging solutions similar to it.

Word-level completions. Most of the results presented in this paper assume completions are only suggested at the end of words. While this is possible to achieve in a real-world scenario, it would require some component assessing whether an end of a word is reached or not. This solution will have to run online, and in short latency. Since our experiments are run offline, the full turn was available for us and we could simply check when the end of a word was reached.

References

Marah I Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat S. Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Caio César Teodoro Mendes, Weizhu Chen, Vishrav Chaudhary, Parul Chopra, Allie Del Giorno, Gustavo de Rosa, Matthew Dixon, Ronen Eldan, Dan Iter, Amit Garg, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Jamie Huynh, Mojan Javaheripi, Xin Jin, Piero Kauffmann, Nikos Karampatziakis, Dongwoo Kim, Mahoud Khademi, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Chen Liang, Weishung Liu, Eric Lin, Zeqi Lin, Piyush Madan, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Xia Song, Masahiro Tanaka, Xin Wang, Rachel Ward, Guanhua Wang, Philipp Witte, Michael Wyatt, Can Xu, Jiahang Xu, Sonali Yadav, Fan Yang, Ziyi Yang, Donghan Yu, Chengruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *CoRR*, abs/2404.14219.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Slav Petrov, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy P. Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul Ronald Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, George Tucker, Enrique Piqueras, Maxim Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez, Misha Khalman, Jakub Sygnowski, and et al. 2023. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Fei Cai, Maarten De Rijke, et al. 2016. A survey of query auto completion in information retrieval. *Foundations and Trends® in Information Retrieval*, 10(4):273–363.

Mia Xu Chen, Benjamin N Lee, Gagan Bansal, Yuan Cao, Shuyuan Zhang, Justin Lu, Jackie Tsay, Yinan Wang, Andrew M Dai, Zhifeng Chen, et al. 2019. Gmail smart compose: Real-time assisted writing. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2287–2295.

Rohan Chitnis, Shentao Yang, and Alborz Geramifard. 2024. Sequential decision-making for inline text autocomplete. *arXiv preprint arXiv:2403.15502*.

Rianne Conijn. 2020. *The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging*. Ph.D. thesis, Tilburg University, University of Antwerp.

Ekaterina Fadeeva, Aleksandr Rubashevskii, Artem Shelmanov, Sergey Petrakov, Haonan Li, Hamdy Mubarak, Evgenii Tsymbalov, Gleb Kuzmin, Alexander Panchenko, Timothy Baldwin, Preslav Nakov, and Maxim Panov. 2024. Fact-checking the output of large language models via token-level uncertainty quantification. *CoRR*, abs/2403.04696.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *CoRR*, abs/2312.00752.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *CoRR*, abs/2310.06825.

Nick Jiang, Anshul Ramachandran, Mehul Raheja, and Michael Li. 2024. Codium.

Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, et al. 2024. Openassistant conversations-democratizing large language model alignment. *Advances in Neural Information Processing Systems*, 36.

Florian Lehmann and Daniel Buschek. 2021. Examining autocompletion as a basic concept for interaction with generative ai. *i-com*, 19(3):251–264.

Jenny T Liang, Chenyang Yang, and Brad A Myers. 2024. A large-scale survey on the usability of ai programming assistants: Successes and challenges.

In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pages 1–13.

Potsawee Manakul, Adian Liusie, and Mark J. F. Gales. 2023. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 9004–9017. Association for Computational Linguistics.

Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Cristian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, and et al. 2024. Gemma: Open models based on gemini research and technology. *CoRR*, abs/2403.08295.

Dae Hoon Park and Rikio Chiba. 2017. A neural language model for query auto-completion. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1189–1192.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2018. Language models are unsupervised multitask learners.

Jie Ren, Jiaming Luo, Yao Zhao, Kundan Krishna, Mohammad Saleh, Balaji Lakshminarayanan, and Peter J. Liu. 2023. Out-of-distribution detection and selective generation for conditional language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Andrew Shin and Kunitake Kaneko. 2024. Large language models lack understanding of character composition of words. *CoRR*, abs/2405.11357.

Alessandro Sordoni, Yoshua Bengio, Hossein Vahabi, Christina Lioma, Jakob Grue Simonsen, and Jian-Yun Nie. 2015. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *proceedings of the 24th ACM international on conference on information and knowledge management*, pages 553–562.

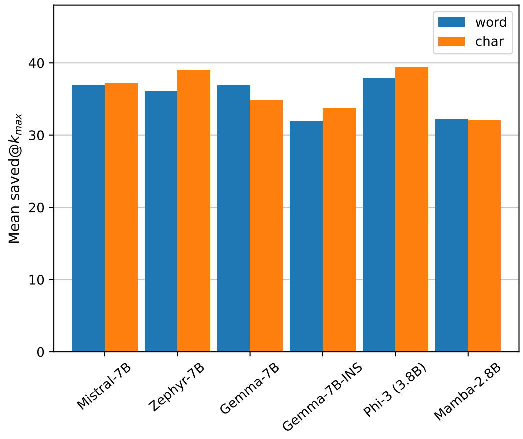

Figure 3: saved@k comparison between solutions suggesting completions after words and characters.

A Character vs Word level completions

In this section we detail our comparison between suggesting completions after each character compared to doing so only at the end of words. We consider $k_{\max}$, i.e., all generated completions (at most, $n_c \times n_t$). While this scenario is not realistic, since for the *best* configuration it means presenting the user with 100 completion options, it shows the potential each solution has with a perfect ranking solution. We start with saved@k. Issuing suggestions after each character is expected to improve this metric compared to issuing suggestions after each word. This is due to the fact that this metric does not penalize on unaccepted suggestions. Therefore if every mid-word suggestion is ignored by the user, the metric will remain unchanged. If some mid-word suggestion are accepted, the metric is expected to rise. In Figure 3 we show results on saved@$k_{\max}$. Indeed, the metric is improved when suggesting after each character, but the difference is minor (on average across models, 3.2%). We note that even for Mamba, which uses a character-based tokenizer, the difference is very small. Next, we compare the same solutions on acceptance rate. Results in Figure 4 show that acceptance rate for the solutions suggesting only at end of words is much higher (on average, $\sim 130\%$ improvement), suggesting that the mid-word suggestions are rarely accepted.

B Full Benchmark Results

The full results are reported in Table 4. For each model mentioned in Section 3.3, we report results for two hyper parameter combinations: *best* is a ver-

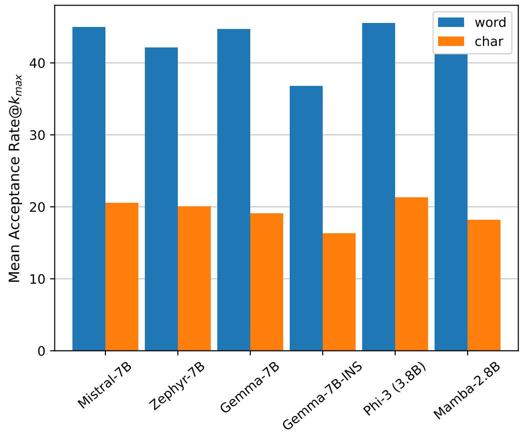

Figure 4: *acceptance rate* comparison between solutions suggesting completions after words and characters.

sion aimed at optimizing the quality ($n_c = 5, n_t = 20$), while *fast* is a version aimed at optimizing the latency ($n_c = 1, n_t = 5$). A full hyper parameter study can be found in Appendix D. We report on both datasets presented in Section 3.1 for different k values. Further analysis of the effect of k can also be found in Appendix D.

Results on $k@1$ and $k@3$, representing realistic scenarios where the user is presented a single or 3 completion suggestions, demonstrate that while current models are able to perform fairly on this task – reaching acceptance rate of up to $\sim 37.5\%$, and saving the user the typing of up to $\sim 34.5\%$ of the characters – there is still much room for improvement. $k_{\max}$ shows results considering all generated completions (at most, $n_c \times n_t$). While this scenario is not realistic, since for the *best* configuration it means presenting the user with 100 completion options, it shows the potential each solution has with a perfect ranking solution. The large gap between the $k_{\max}$ results and the results with smaller k values suggests that perplexity may be insufficient for ranking. This is in line with prior work (Manakul et al., 2023; Ren et al., 2023; Fadeeva et al., 2024).

As for comparing the different models, the best performing model is Gemma-7B, which is also the model with the longest latency. Phi-3 stands out as well, with performance surpassing most of the other models, although they are larger in size and slower in latency. This result is consistent with its performance on other benchmarks compared to other models included in our evaluation (Abdin et al., 2024). When comparing instruct models to their corresponding base models, instruct models mostly performed worse. This is likely due to the fact that the language modeling objective of the pretraining phase is closer to our task than the objective of the alignment phase.

Finally, our *best* vs *fast* hyper parameter combinations are indeed able to offer a trade-off between latency and performance. On average, *fast* is able to save $\sim 75\%$ of the latency compared to *best*, while *best* performs $\sim 30\%$ better on $k_{\max}$ and $\sim 4-8\%$ better on the realistic k scenarios.

Results on ShareGPT for varying k values, complementing Figures 2 and 6 in section 3.5 are shown in Figure 5.

C Fine-tuning Models to Improve AC

We observe that fine-tuning models can offer further improvement upon the corresponding pre-trained models. We fine-tuned Mistral-7B and Zephyr-7B on the OASST train set using LoRA (Hu et al., 2021), with the following hyperparameters (Mistral / Zephyr, respectively): learning rate $1.4\text{e}{-4}/2.4\text{e}{-4}$, epochs $0.40/0.25$, batch size $16/16$. In Table 5 we report an average increase of 4.19% and 10.93% in the saved@k metric for Mistral-7B and Zephyr-7B, respectively.

D Hyper Parameter Study

The auto completion method we use, extracting completion suggestions for language models, has two hyper parameters, n_c and n_t, as detailed in Section 3.3.
In Figure 7, we show results on different values for the two parameters. In each figure, one of the parameters is fixed and the other is varied.

We also report results on different values of k in Figures 6 (acceptance rate) and 2 ($saved@k$). This parameter decides how many suggestions are shown to the user. While a higher value is guaranteed to increase the performance metrics, it may also incur slower latency and a cognitive cost for the user, and therefore for very high values it is unrealistic.

		$k = 1$		$k = 3$		k_{max}		Latency (ms)	
		saved@1	acc. rate@1	saved@3	acc. rate@3	saved@k	acc. rate@k	mean	p90
Mistral-7B	best	<u>25.97</u> / <u>24.67</u>	<u>32.23</u> / <u>32.32</u>	**34.66** / <u>32.76</u>	<u>37.65</u> / <u>38.35</u>	<u>44.86</u> / **41.04**	<u>50.56</u> / <u>49.33</u>	834 / 1479	1288 / 2485
	fast	26.23 / 24.32	32.46 / 31.94	33.29 / 30.75	36.13 / 36.19	35.02 / 32.12	38.02 / 37.84	201 / 356	313 / 588
Zephyr-7B	best	24.01 / 23.47	29.81 / 30.22	31.06 / 29.89	32.91 / 34.31	44.00 / 40.85	47.91 / 47.65	870 / 1520	1313 / 2512
	fast	24.63 / 23.39	30.81 / 30.48	31.28 / 29.16	34.03 / 34.30	33.49 / 31.02	36.62 / 36.63	214 / 368	320 / 589
Gemma-7B	best	25.80 / **24.84**	**32.34** / **32.71**	**34.66** / **32.91**	**37.72** / **38.93**	44.75 / <u>41.02</u>	50.02 / **49.44**	961 / 1587	1423 / 3032
	fast	25.62 / 23.77	32.48 / 31.57	32.55 / 29.89	35.64 / 35.61	34.25 / 31.44	37.93 / 37.67	239 / 412	358 / 684
Gemma-7B-INS	best	22.08 / 21.65	28.33 / 28.72	28.13 / 27.32	30.83 / 31.78	38.94 / 35.67	41.81 / 41.24	837 / 1522	1355 / 2981
	fast	22.41 / 21.29	29.14 / 28.70	28.29 / 26.50	31.61 / 31.72	30.39 / 28.13	34.27 / 33.99	245 / 421	358 / 702
Phi-3 (3.8B)	best	**26.07** / 24.18	32.07 / 31.25	<u>34.42</u> / 31.83	36.76 / 36.99	**45.18** / 39.91	**50.81** / 47.84	510 / 879	786 / 1466
	fast	26.13 / 23.25	32.26 / 30.39	33.21 / 29.60	35.91 / 34.95	34.82 / 30.92	37.98 / 36.70	117 / 208	185 / 344
Mamba-2.8B	best	22.36 / 21.66	29.44 / 29.28	29.94 / 28.76	34.96 / 34.86	37.94 / 35.82	45.44 / 44.53	433 / 779	689 / 1306
	fast	21.81 / 20.92	28.57 / 28.20	28.00 / 26.66	31.79 / 31.94	29.56 / 28.02	33.83 / 33.76	105 / 186	166 / 306
Mamba-2.8B-Zephyr	best	23.20 / 22.09	29.69 / 29.37	30.73 / 28.84	33.86 / 34.11	41.98 / 37.95	47.29 / 45.85	450 / 793	696 / 1300
	fast	23.24 / 21.72	29.68 / 29.01	29.54 / 27.32	32.53 / 32.47	31.64 / 28.91	35.16 / 34.45	112 / 191	168 / 308
SmolLM-1.7B	best	22.44 / 21.81	29.59 / 29.40	30.31 / 28.92	34.80 / 35.38	39.26 / 35.82	46.10 / 44.57	249 / 422	374 / 696
	fast	22.45 / 21.03	29.19 / 28.41	28.92 / 27.00	32.55 / 32.66	30.57 / 28.51	34.56 / 34.36	**57** / **100**	**84** / **167**
GPT2-XL (1.6B)	best	19.67 / 12.06	26.59 / 16.91	26.96 / 15.84	31.94 / 20.26	34.13 / 19.80	41.37 / 25.79	265 / 453	397 / 833
	fast	19.58 / 11.43	25.96 / 15.94	25.31 / 14.72	29.28 / 18.34	26.84 / 15.63	31.31 / 19.65	<u>62</u> / <u>107</u>	<u>96</u> / <u>180</u>
Average	best	23.43 / 22.03	30.10 / 29.26	31.21 / 28.85	34.79 / 34.31	41.27 / 36.77	47.00 / 44.49	640 / 1105	983 / 1922
	fast	23.00 / 21.01	29.53 / 28.12	29.42 / 26.61	32.74 / 31.80	31.20 / 28.08	34.93 / 33.66	160 / 275	239 / 452

Table 4: Results comparing the performance and of the 9 evaluated models on both metrics for $k = 1, 3, k_{max}$, with *best* and *fast* configurations, each with mean and p90 latency. In each cell we report the results for both datasets: OpenAssistant / ShareGPT. For each metric and k, the winner is marked in bold and the second best is underlined.

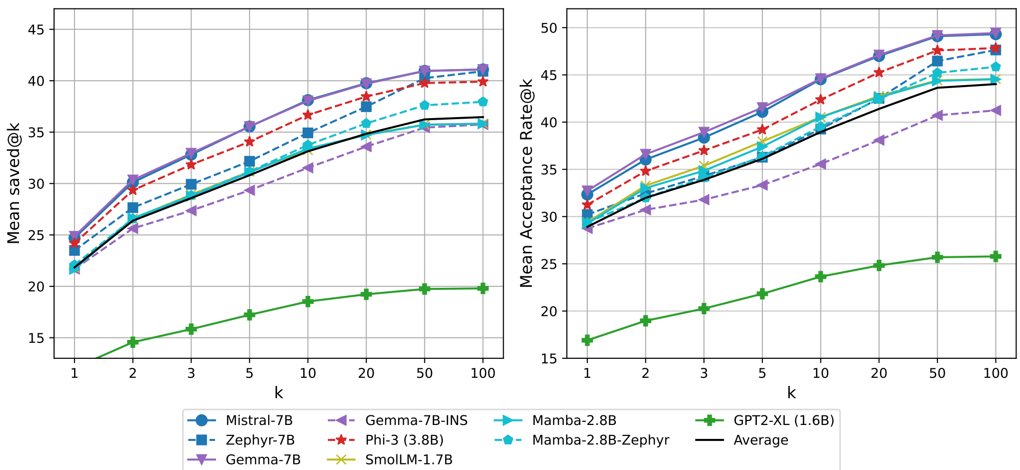

Figure 5: ***saved@k*** and ***acc. rate@k*** on ShareGPT for varying k values.

	Mistral-7B			**Zephyr-7B**		
	Pretrained	Finetuned	Improvement (%)	Pretrained	Finetuned	Improvement (%)
saved@1	25.99	27.12	+ 4.37	24.03	26.94	+ 12.10
acc. rate@1	32.22	33.60	+ 4.29	29.80	33.26	+ 11.61
saved@3	34.68	35.98	+ 3.74	31.08	35.63	+ 14.62
acc. rate@3	37.64	38.59	+ 2.54	32.91	38.23	+ 16.17
saved@k_{max}	44.89	46.89	+ 4.46	44.03	46.70	+ 6.06
acc. rate@k_{max}	50.56	52.10	+ 3.05	47.91	52.39	+ 9.35

Table 5: saved@k results of Mistral and Zephyr before and after fine-tuning.

<table>
<tr><th>Interaction</th><th>saved@k_{max} (%)</th></tr>
<tr><td><|prompter|> Do you have any information about the Commodore 64?</td><td>18.75</td></tr>
<tr><td><|prompter|> What is the Sun?
<|assistant|> The Sun is a star, a massive celestial body that converts matter into energy. It is the largest object in the solar system and is at its center; the gravitational field it exerts pulls the Earth, and all other planets and objects in the solar system, causing them to orbit it. The Sun produces energy by fusing hydrogen atoms in its core, producing helium. This continuous nuclear reaction emits tremendous amounts of energy, which is released as solar radiation. This radiation reaches the Earth and provides most of the energy needed to sustain life.
<|prompter|> Can you tell me more about suns from other solar systems?</td><td>83.02</td></tr>
<tr><td><|prompter|> Hi. Could you please tell me how I can get rid of bad breath?</td><td>59.38</td></tr>
<tr><td><|prompter|> Can you tell me a bit about what has gone into your creation?
<|assistant|> My creation took over a month and the process of gathering data is still ongoing. I am becoming a larger assistant that is Open-Source.
<|prompter|> Do you have any way of keeping stats on how much data you've consumed or how long your training has taken?
<|assistant|> Yes but the public release of that information is pending review and validation.
<|prompter|> Here are 10 more question: What kind of neural network architecture was used to create you?</td><td>42.19</td></tr>
<tr><td><|prompter|> I would like you to create some regex to find out if the first number in a set of numbers and letters is a one. For example, in “1 month, 2 days, and 3 hours” the selected text would be "1."</td><td>72.22</td></tr>
<tr><td><|prompter|> Write an "about us" paragraph for the website of a design company. The paragraph should contain information on the different services and strengths of the company. It is written in a serious tone and shouldn't feature any technical terms that customers could be confused about.</td><td>34.38</td></tr>
</table>

Table 6: Full interaction examples with corresponding saved@k scores of randomly drawn prefixes from the OpenAssistant validation set. Completions were generated using Mistral-7B, with $n_c = 5, n_t = 20$. The existing prefix including the context is colored gray. Accepted text is colored green with each acceptance underlined separately. Accepted text is colored green, and each accepted segment is separately underlined. If no suggestion was accepted, the text typed by the user is colored black.

<table>
<tr><td>Prefix:</td><td colspan="3">"<|prompter|> Who would win a chess match between Keanu Reeves and Audrey Hepburn? <|assistant|> In a purely physical fight, Audrey would likely win. Keanu is renowned for being a very nice guy, I doubt he would hit a woman. Audrey grew up in an era when women had to be serious badasses to get by. She'd kick him in the nuts before he had time to say 'Vaya con Dios, Brah' <|prompter|> No,"</td></tr>
<tr><td>Ground Truth:</td><td colspan="3">"I mean the board game chess!"</td></tr>
<tr><td>Mistral</td><td>I mean a chess match.</td><td>I mean a chess</td><td>I mean</td></tr>
<tr><td>Zephyr</td><td>I meant a chess match.</td><td>I meant a chess</td><td>that's not what I meant.</td></tr>
<tr><td>Phi</td><td>I</td><td>I'm</td><td>I'm asking for a chess match,</td></tr>
<tr><td>Mamba</td><td>I mean</td><td>I mean in a</td><td>I</td></tr>
<tr><td>GPT2</td><td>Keanu</td><td>I</td><td>seriously, which of these men would win in a</td></tr>
<tr><td>Prefix:</td><td colspan="3">"<|prompter|> Hello, what education should I take at university? What are important"</td></tr>
<tr><td>Ground Truth:</td><td colspan="3">"things to consider when making the choice?"</td></tr>
<tr><td>Mistral</td><td>things to</td><td>criteria in choosing a</td><td>criteria in choosing a future profession and university?</td></tr>
<tr><td>Zephyr</td><td>skills to</td><td>skills</td><td>educations in the future?</td></tr>
<tr><td>Phi</td><td>skills to learn?</td><td>classes to take in order to become a</td><td>skills to develop?</td></tr>
<tr><td>Mamba</td><td>courses?</td><td>courses to</td><td>courses to study</td></tr>
<tr><td>GPT2</td><td>book titles to read?</td><td>subjects</td><td>subjects and should I</td></tr>
<tr><td>Prefix:</td><td colspan="3">"<|prompter|> How do I code pong in python. <|assistant|> Pong is a classic game that can be coded in Python using the pygame library. To get started, you'll need to install the pygame library. You can do this by running the command "pip install pygame" in your terminal. Once you have the library installed, you can create a new file and start coding. You'll need to create a window, draw the paddles, and create the ball. You'll also need to create the logic for the game, such as the ball bouncing off the paddles and the score keeping. Finally, you'll need to create the game loop that will keep the game running until the player wins or loses. With some practice and patience, you should be able to code a basic version of Pong in Python. <|prompter|> Start writing code. Use pygame and make the window default to full screen and be resizable. There should be 2 modes,"</td></tr>
<tr><td>Ground Truth:</td><td colspan="3">"against computer and against human, these should be selected when a new game is created."</td></tr>
<tr><td>Mistral</td><td>single player and multiplayer.</td><td>single player and</td><td>single player</td></tr>
<tr><td>Zephyr</td><td>single player</td><td>single player and</td><td>single</td></tr>
<tr><td>Phi</td><td>one where the ball bounces off the</td><td>one where the ball bounces off</td><td>easy and hard.</td></tr>
<tr><td>Mamba</td><td>one for the ball and one for the paddle.</td><td>one for the ball and one for the</td><td>one for the ball and one for</td></tr>
<tr><td>GPT2</td><td>fullscreen and windowed.</td><td>windowed and full screen.</td><td>fullscreen and</td></tr>
<tr><td>Prefix:</td><td colspan="3">"<|prompter|> write a inspirational monologue script from a spartan general telling his army that despite the impossible nature of their fight, that it is still worth fighting. do not directly reference sparta or the fight, but"</td></tr>
<tr><td>Ground Truth:</td><td colspan="3">"focus more on the concept of the indomitable human spirit and the will to keep fighting no matter the cost."</td></tr>
<tr><td>Mistral</td><td>it can be inferred</td><td>it can be inferred (or outright stated) that the</td><td>it can be inferred (or outright stated) that</td></tr>
<tr><td>Zephyr</td><td>rather speak in general terms about perseverance</td><td>rather</td><td>rather speak in general terms about</td></tr>
<tr><td>Phi</td><td>focus on themes of unity, courage, and</td><td>focus on themes of</td><td>focus on themes of unity, courage, and the</td></tr>
<tr><td>Mamba</td><td>the gist is the same</td><td>make a general</td><td>instead the spirit of bravery and honor.</td></tr>
<tr><td>GPT2</td><td>do reference the</td><td>instead</td><td>do reference</td></tr>
<tr><td>Prefix:</td><td colspan="3">"<|prompter|> What are some unique, creative, and efficient ways to decorate and make the most of a small apartment space while still ensuring a comfortable living environment? Are there any particular design styles or techniques that are especially well-suited for small spaces, and what are the pros and cons of each approach? Are there any furniture pieces or items that are particularly useful for maximizing space and comfort in a small apartment, and what are"</td></tr>
<tr><td>Ground Truth:</td><td colspan="3">"some tips for choosing and arranging these items in a functional and aesthetically pleasing way?"</td></tr>
<tr><td>Mistral</td><td>some tips for choosing the right</td><td>some</td><td>some tips for choosing the right pieces for</td></tr>
<tr><td>Zephyr</td><td>their benefits and drawbacks?</td><td>some tips for arranging and organizing these items in a</td><td>some tips for</td></tr>
<tr><td>Phi</td><td>their benefits and drawbacks?</td><td>some examples of these items?</td><td>some examples of</td></tr>
<tr><td>Mamba</td><td>the pros and cons of</td><td>the pros and cons</td><td>the pros and</td></tr>
<tr><td>GPT2</td><td>their pros and cons?</td><td>their pros and</td><td>their pros</td></tr>
</table>

Table 7: Comparison of top 3 suggested completions of different LLMs, on prefixes randomly drawn from the OpenAssistant validation set. Completions were generated with $n_c = 5, n_t = 20$.

		$k=1$		$k=3$		k_{max}	
		saved@1	acc. rate@1	saved@3	acc. rate@3	saved@k	acc. rate@k
Mistral-7B	Single Word	26.00 / 24.65	42.42 / 41.75	34.70 / 31.67	54.01 / 51.88	36.02 / 32.80	55.86 / 53.48
	EOS	15.35 / 13.73	12.60 / 12.84	19.47 / 16.81	16.00 / 15.71	19.87 / 17.17	16.33 / 16.08
	Partial	25.97 / 24.67	32.23 / 32.32	34.66 / 32.76	37.65 / 38.35	44.86 / 41.04	50.56 / 49.33
Zephyr-7B	Single Word	25.63 / 24.31	41.76 / 40.57	33.89 / 31.43	52.87 / 50.80	35.00 / 32.65	54.43 / 52.56
	EOS	11.95 / 10.73	9.53 / 10.06	15.85 / 13.61	12.46 / 12.51	16.91 / 14.37	13.35 / 13.26
	Partial	24.01 / 23.47	29.81 / 30.22	31.06 / 29.89	32.91 / 34.31	44.00 / 40.85	47.91 / 47.65
Gemma-7B	Single Word	25.93 / 24.53	42.64 / 41.70	34.28 / 31.75	53.76 / 51.87	35.49 / 32.77	55.39 / 53.29
	EOS	15.54 / 13.78	12.64 / 12.95	19.23 / 17.04	15.56 / 15.91	19.67 / 17.51	15.96 / 16.37
	Partial	25.80 / 24.84	32.34 / 32.71	34.66 / 32.91	37.72 / 38.93	44.75 / 41.02	50.02 / 49.44
Gemma-7B-INS	Single Word	23.85 / 22.51	39.57 / 38.32	30.62 / 28.21	48.10 / 46.20	31.40 / 28.62	49.20 / 46.84
	EOS	11.54 / 10.20	9.41 / 9.68	15.34 / 13.19	12.30 / 12.29	16.31 / 13.70	13.06 / 12.76
	Partial	22.08 / 21.65	28.33 / 28.72	28.13 / 27.32	30.83 / 31.78	38.94 / 35.67	41.81 / 41.24
Phi-3 (3.8B)	Single Word	26.04 / 24.12	42.06 / 40.55	34.73 / 31.17	53.90 / 50.62	36.18 / 32.20	55.96 / 52.22
	EOS	15.54 / 12.58	12.57 / 11.89	19.42 / 15.38	15.64 / 14.47	19.77 / 15.67	15.98 / 14.78
	Partial	26.07 / 24.18	32.07 / 31.25	34.42 / 31.83	36.76 / 36.99	45.18 / 39.91	50.81 / 47.84
Mamba-2.8B	Single Word	22.19 / 21.79	37.52 / 37.42	29.72 / 28.34	48.07 / 47.05	31.04 / 29.25	49.85 / 48.36
	EOS	12.92 / 11.18	10.90 / 10.54	15.79 / 13.68	13.34 / 12.82	16.05 / 13.91	13.62 / 13.04
	Partial	22.36 / 21.66	29.44 / 29.28	29.94 / 28.76	34.96 / 34.86	37.94 / 35.82	45.44 / 44.53
Mamba-2.8B-Zephyr	Single Word	24.55 / 22.81	40.22 / 38.71	32.68 / 29.46	51.48 / 48.54	33.74 / 30.51	52.93 / 50.04
	EOS	12.21 / 10.47	9.90 / 9.59	15.41 / 12.72	12.42 / 11.64	15.92 / 13.05	12.87 / 12.03
	Partial	23.20 / 22.09	29.69 / 29.37	30.73 / 28.84	33.86 / 34.11	41.98 / 37.95	47.29 / 45.85
SmolLM-1.7B	Single Word	22.93 / 21.90	38.52 / 37.56	30.81 / 28.22	49.20 / 46.97	31.91 / 29.00	50.70 / 48.11
	EOS	13.17 / 11.82	10.91 / 11.31	16.14 / 14.08	13.27 / 13.45	16.51 / 14.45	13.57 / 13.83
	Partial	22.44 / 21.81	29.59 / 29.40	30.31 / 28.92	34.80 / 35.38	39.26 / 35.82	46.10 / 44.57
GPT2-XL (1.6B)	Single Word	20.07 / 12.13	34.16 / 21.09	26.72 / 15.74	43.65 / 26.81	27.94 / 16.22	45.33 / 27.58
	EOS	11.38 / 5.85	9.52 / 5.47	13.56 / 7.00	11.37 / 6.61	13.77 / 7.08	11.58 / 6.72
	Partial	19.67 / 12.06	26.59 / 16.91	26.96 / 15.84	31.94 / 20.26	34.13 / 19.80	41.37 / 25.79
Average	Single Word	24.10 / 22.28	40.06 / 38.04	31.97 / 28.63	50.76 / 47.28	33.12 / 29.52	52.34 / 48.57
	EOS	12.30 / 10.44	10.04 / 9.82	15.91 / 13.29	12.94 / 12.41	16.47 / 13.70	13.42 / 12.82
	Partial	23.43 / 22.03	30.10 / 29.26	31.21 / 28.85	34.79 / 34.31	41.27 / 36.77	47.00 / 44.49

Table 8: Scores of partial completions vs single word and full sentence baselines. OpenAssistant/ShareGPT.

n_c	n_t	Hist. Len	saved@100	Latency p90 (ms)	n_c	n_t	Hist. Len	saved@100	Latency p90 (ms)
5	10	$Full$	45.75	974	3	10	1000	38.56	520
5	20	$Full$	45.60	1287	5	5	250	38.32	275
5	5	$Full$	45.00	815	4	10	250	37.46	419
5	20	1000	44.32	947	3	3	$Full$	37.35	468
5	5	1000	44.08	451	3	3	1000	37.33	278
5	10	1000	43.43	614	4	20	250	36.58	752
4	20	$Full$	43.13	1137	5	3	250	36.42	216
4	10	$Full$	43.02	843	4	5	250	36.25	254
4	10	1000	42.52	569	4	3	250	34.14	184
4	20	1000	42.40	885	3	20	250	33.99	732
5	3	$Full$	42.28	742	3	10	250	33.37	405
4	5	$Full$	41.81	673	3	5	250	33.26	241
5	20	500	41.45	828	5	20	100	32.59	732
5	3	1000	41.10	388	3	3	250	32.13	171
4	5	1000	40.85	399	5	5	50	25.35	219
3	10	$Full$	40.44	695	5	20	50	25.21	729
3	20	$Full$	40.35	991	5	10	50	24.95	393
4	3	$Full$	39.67	602	5	3	50	23.45	148
3	20	1000	39.59	818	4	10	50	23.01	389
3	5	$Full$	39.57	526	4	20	50	22.94	723
5	20	250	39.47	776	4	5	50	22.67	216
5	10	250	38.80	435	4	3	50	22.07	147
4	3	1000	38.76	330	3	10	50	21.02	385
3	5	1000	38.59	348	3	20	50	20.90	717

Table 9: **Latency-Performance Trade-Off.** Full results for all configurations, complementing Table 2 in section 4. Mistral-7b evaluated on the OASST test set. $n_c \in \{3, 4, 5\}$, $n_t \in \{3, 5, 10, 20\}$, and context length $len(C) \in \{50, 250, 1000, Full\}$ (measured in characters). In total, 48 hyper-parameter configurations were evaluated. Results are sorted by their saved@100 score.

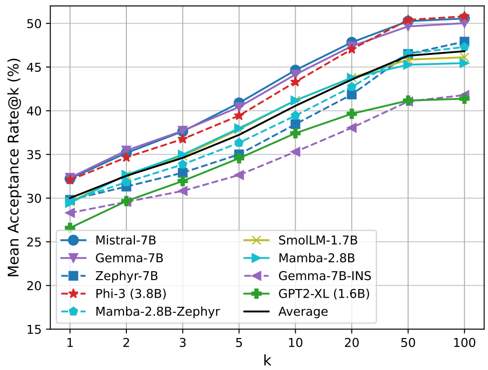

Figure 6: ***acc. rate@k*** on OASST for varying k values.

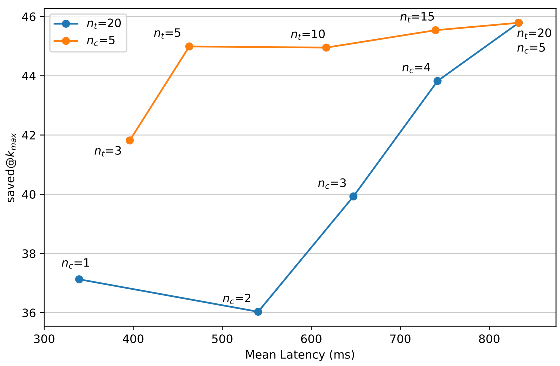

Figure 7: **Hyper parameter study** (n_c **and** n_t). For each line, one of the parameters is fixed and the other is varied. Results are shown on the OASST dataset using the *best* configuration and the Mistral model.

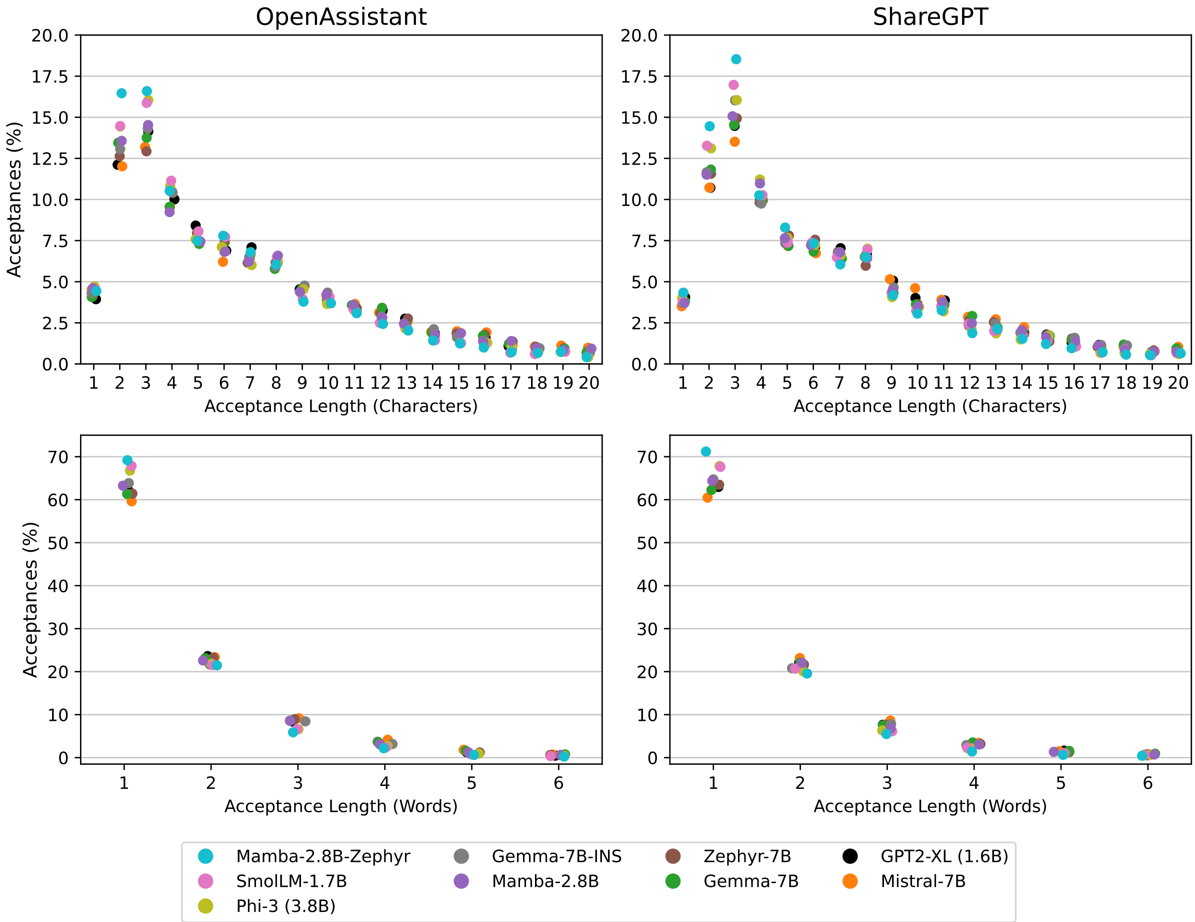

Figure 8: Lengths of accepted completions for $k = 100$.