

Personalization of Large Language Models for Predictive Typing

Soufia Bahmani & Keith Vertanen

Department of Computer Science
Michigan Technological University
Houghton, MI 49931, USA
{sbahmani, vertanen}@mtu.edu

Abstract

We study improving predictive typing via per-user personalization of a large language model (LLM). We evaluate personalization in a time-ordered setting, using a user’s earlier messages to adapt the LLM to improve predictions on later messages. We compare a static n-gram and a static LLM against an LLM adapted via full-parameter fine-tuning, one-time low-rank adaptation (LoRA), and multiple progressive LoRA updates. In a simulated predictive keyboard, the static LLM increased keystroke savings by 13% compared to the n-gram model. Adapting the parameters of the LLM only offered a small additional increase of 0.5%. We found this was due to the strong personalization provided by conditioning the LLM on the 512 previous characters of a user’s sent emails. Without this conditioning, perplexity increased sharply by 20% relative.

1 Introduction and related work

Typing can be slow and effortful for some people, including those with motor impairments (Cai et al., 2024), those facing situational constraints (e.g. typing on a phone while walking), or those writing in a language they are learning (Sarsenbayeva et al., 2017; Al-Fattah, 2018). Within this space, Augmentative and Alternative Communication (AAC) provides a high-impact motivating case. AAC users often write letter-by-letter using input methods such as eye tracking or single-switch (Beukelman & Light, 2020). Typical AAC text entry rates are 5–10 words per minute, much slower than spoken speech (Trnka & McCoy, 2008).

Text entry interfaces often try to reduce this burden by using a language model to autocorrect noisy input and by offering text predictions. However, these models are typically trained on generic corpora and deployed unchanged for every user, despite users’ need to write specific text (e.g. names, contacts, or habitual phrases) and changes in their language use over time. Language model personalization aims to address this mismatch.

Early adaptive language modeling approaches included cache models (Kuhn & De Mori, 1990; Jelinek et al., 1991) and Prediction by Partial Matching (PPM) (Cleary & Witten, 1984; Ward et al., 2000). Later neural approaches used recurrent neural network language models (RNNLMs) (Mikolov et al., 2010). Using time-ordered Enron email, Fowler et al. (2015) studied cache-based personalization for touchscreen typing, while Adhikary & Vertanen (2023) evaluated personalized PPM and RNNLM approaches. Recent AAC work has explored keyword-conditioned utterance generation, phrase suggestions, and abbreviation expansion (Shen et al., 2022; Valencia et al., 2023; Cai et al., 2024). Closest to our setting, Gaines & Vertanen (2025) evaluated LLMs as character predictors for AAC and reported both character-level perplexity and keystroke savings in a predictive keyboard simulator. While previous work shows the value of personalization and LLM-based assistance for text entry, they do not evaluate progressive per-user LLM personalization.

We study progressive personalization of LLMs on real email data from the Enron corpus (Klimt & Yang, 2004). We ask how much text prediction improves when we adapt an LLM

to an employee’s past messages. To measure prediction accuracy and typing effort, we use per-character perplexity and keystroke savings (Trnka & McCoy, 2008).

We consider two personalization settings. In *one-time personalization*, we first adapt the model on the first half of the sent emails of a user (an Enron employee). We leave the model frozen while evaluating on the user’s second half of sent emails. In *progressive personalization*, we periodically update the model as we pass through the second half of a user’s emails. Progressive personalization aims to adapt to ongoing changes in a user’s topics or style.

We make the following contributions:

1. We find conditioning on long left context offers strong LLM personalization ability.
2. We show progressive fine-tuning provides a small, but consistent gain across all users.
3. We show parameter efficient fine-tuning performs similar to full parameter fine-tuning.

2 Dataset, procedure, and language models

We used the Enron Personalization Validation Set.¹ The dataset contains sent emails from 45 development and 44 test users. It was assembled by Fowler et al. (2015) for touch-screen typing and later normalized by Adhikary & Vertanen (2023) for PPM and RNNLM personalization; we use the latter version.²

We split each user’s time-ordered sent emails into two equal-sized sets denoted *seed* (the first 50%) and *evaluation* (the second 50%). The seed set was the only user-specific text available before evaluation. All results were on the evaluation set. This approach follows Adhikary & Vertanen (2023) who evaluated with and without adapting on earlier messages (“primed” versus “flat start”). In the progressive setting, we processed the evaluation set in time-ordered chunks, scoring a chunk before using it to adapt the model.

We used 10 of the 45 development users, denoted DEV, for choosing the LLM size and left-context length and for tuning hyperparameters. We selected users representative of the distribution of available email data (Figure 1). After fixing all choices, we used 44 test users (denoted TEST) for our final evaluation.

We normalized all text to lowercase and stripped characters aside letters, space, and apostrophe. We excluded phrases containing digits. Character perplexity was over this vocabulary of 28 characters. We used the algorithm from Gaines & Vertanen (2025) for calculating character perplexity under a subword tokenized LLM.

We computed keystroke savings as the percentage of keystrokes a user making no typing mistakes would save by making optimal use of offered word predictions. We simulated a keyboard with three prediction slots. One slot was reserved for the literal characters typed. Thus, our simulated user could always obtain their desired text. The other two slots extended the currently typed text with the most likely next words under the language model. Our simulator had no explicit word list, rather, we conducted a beam search looking for character sequences ending in a space, comma, exclamation mark, question mark, or period. We assume selecting a prediction required one keystroke and produced any following space. Higher keystroke savings is better.

We tested LLMs from the OPT family (Zhang et al., 2022) as Gaines & Vertanen (2025) found it performed well compared to other model families. As a non-LLM baseline, we used a 12-gram model with 408 M parameters trained on 21 B characters of text.³

We tuned our LLM’s fine-tuning hyperparameters on DEV. For all approaches, we tuned the dropout rate, learning rate, weight decay, and epochs. For LoRA (Hu et al., 2022) approaches, we also tuned adapter rank and alpha. For progressive personalization, we also tuned the number of phrases between adaptation. See Section A.3 for details.

¹<https://github.com/google-research-datasets/EnronPersonalizationValidation>

²<https://osf.io/45p3j/>

³We used the Dec 2019 huge model from <https://osf.io/ajm7t/>

Language model	Perplexity ↓	Seconds per character
OPT-125M	2.73	0.068
OPT-350M	2.66	0.083
OPT-1.3B	2.57	0.082
OPT-2.7B	2.54	0.092
OPT-6.7B	2.51	0.096
OPT-13B	2.50	0.115

Table 1: Per-character perplexity and inference time on DEV of different size LLMs.

Left context character limit	Perplexity ↓	Seconds per character
0	2.50	0.115
64	2.33	0.119
128	2.20	0.125
256	2.11	0.144
512	2.05	0.185
1024	2.02	0.275

Table 2: Per-character perplexity and inference time on DEV with increasing left context.

3 Experiments

LLM model size. We tested OPT LLMs of increasing size. We computed the perplexity of the evaluation split set for each user in the DEV set. The perplexity of each phrase was computed in isolation (i.e. we did not condition on previous phrases in a user’s data). As expected, LLMs with more parameters provided better predictions (Table 1). Based on this, we used the largest 13 billion parameter model for our remaining experiments. We have released our dataset to facilitate future research.⁴

Amount of left context. On DEV, we explored the impact of how much context to use from a user’s previous emails. We added words until we reached a specific *left context character limit*. As shown in Table 2, the LLM was able to leverage the left context to make substantially better predictions. For the remainder of our results, we use a left context limit of 512 as it obtained the majority of the perplexity gains without unduly increasing inference latency.

Static models. Our main evaluation results on TEST are shown in Table 3. We tested a static n-gram language model and a pretrained LLM (OPT-13B). As expected, the LLM performed substantially better, reducing perplexity from 3.00 to 2.05 and increasing keystroke savings from 54.8% to 67.5%. For comparison, the perplexity on TEST using the LLM using no left context was 2.46. This shows the LLM could leverage large amounts of left context to provide improved user-specific predictions even without adaptation of model parameters.

Adaptive models. We first tried one-time full parameter fine-tuning of the LLM on each user’s seed set. This reduced the perplexity slightly from 2.05 to 2.03 and increased the keystroke savings by 0.5%. We found a LoRA adapter performed similar to full fine-tuning with the same perplexity of 2.03 with a slight decrease in keystroke savings from 68.0% to 67.9%. However, LoRA reduced the mean fine-tuning time per user from 66.9 to 30.8 minutes on four NVIDIA A100 GPUs. We attempted to further improve personalization by progressively fine-tuning the adapter every 128 evaluation phrases. While this slightly reduced perplexity from 2.03 to 2.02, it produced no improvement in keystroke savings.

We also inspected per-user changes in perplexity relative to the baseline LLM. Full fine-tuning and LoRA improved perplexity for most users, while progressive LoRA improved perplexity for every user (see Section A.2 for details). Thus the average gains were driven by improvement across most users and not by improvements on a small number of users.

⁴<https://osf.io/f8x6q/>

Language	Adaptation			Perplexity ↓	Keystroke savings ↑
model	method	on seed?	on eval?		
12-gram	none	no	no	3.00 ± 0.047	$54.8 \pm 0.5\%$
OPT-13B	none	no	no	2.05 ± 0.038	$67.5 \pm 0.6\%$
OPT-13B	one-time full fine-tuning	yes	no	2.03 ± 0.039	$68.0 \pm 0.6\%$
OPT-13B	one-time LoRA fine-tuning	yes	no	2.03 ± 0.038	$67.9 \pm 0.6\%$
OPT-13B	progressive LoRA fine-tuning	yes	yes	2.02 ± 0.039	$67.9 \pm 0.6\%$

Table 3: Perplexity and keystroke savings of different language models and adaptation methods on TEST. $\pm$ values are nonparametric user-level bootstrap 95% confidence intervals.

4 Discussion and conclusion

Our results show OPT-13B substantially outperformed the static 12-gram baseline at the task of predictive typing. We found using 512 characters of left context captured most of the personalization benefit. This suggests recent text from a user’s own email stream is already a powerful form of personalization, even when the LLM’s parameters are not updated.

Fine-tuning provided smaller but useful additional gains. Full-parameter fine-tuning and LoRA achieved similar performance, while LoRA was faster. Progressive LoRA further improved perplexity but not keystroke savings. A likely explanation is that perplexity responds to improvements in the probability assigned to the correct next character, whereas keystroke savings improve only when the target word enters one of the prediction slots early. Thus, a small perplexity improvement does not necessarily cause a reduction in keystrokes.

Our per-user results showed the vast majority of users benefited from some form of model adaptation (Section A.2). Progressive LoRA was the most robust and improved perplexity for all users. This suggests that periodic updates may be advisable to help the language model track ongoing changes in a user’s topics or writing style.

Context-only and parameter-based personalization have different deployment trade-offs. Context-only personalization requires no user-specific adaptation or storage. However, processing longer context increases inference latency. We found increasing from 0 to 512 characters of context increased latency from 0.115 to 0.185 seconds. While this might be reduced via methods such as key-value (KV) caching, such methods can add complexity and require more memory. Parameter-based personalization can avoid increased latency, but requires occasional compute-intensive periods for adaptation and model storage. Both approaches raise privacy concerns: context-only personalization requires retaining a user’s recent text, whereas parameter adaptation creates a user model that must be protected. Given the small gains of parameter adaptation, context-only personalization appears preferable.

There are several limitations. While AAC was a motivating application for this work, our experiments used Enron emails, which may differ from the style of text AAC users wish to generate. We used a predictive-typing simulator that assumed a user making no mistakes. Gains in a real-world keyboard would likely be lower. We used an LLM with 13 billion parameters that required multiple high-end GPUs to perform adaptation. Thus our results should be viewed as an exploration of possible future gains rather than a currently deployable system. We used a single set of optimized hyperparameters. This was despite users having very different amounts of available adaptation text. Users with short, medium, and long histories may benefit from different hyperparameters. Additionally, a user might benefit from updating their initial hyperparameters as more text becomes available.

In conclusion, we conducted experiments on the sent emails of a large set of 44 users. We explored different ways to leverage each user’s past emails to predict their future messages. While we initially thought fine-tuning an LLM’s parameters would be key, we found simply conditioning on a large amount of previous text provided the majority of the gains.

Acknowledgments

This material is based upon work supported by the NSF under Grant No. IIS-2402876.

References

- Jiban Adhikary and Keith Vertanen. Language model personalization for improved touch-screen typing. In *Interspeech 2023*, pp. 1344–1348, 2023. doi: 10.21437/Interspeech.2023-276.
- Nawal Al-Fattah. Cognitive load theory in the context of second language academic writing. *Higher Education Pedagogies*, 3(1):385–402, 2018. doi: 10.1080/23752696.2018.1513812.
- David R. Beukelman and Janice C. Light. *Augmentative and Alternative Communication: Supporting Children and Adults with Complex Communication Needs*. Paul H. Brookes Publishing, 5th edition, 2020.
- Shanqing Cai, Subhashini Venugopalan, Katie Seaver, Xiang Xiao, Katrin Tomanek, Sri Jalasutram, Meredith Ringel Morris, Shaun K. Kane, Ajit Narayanan, Robert L. MacDonald, Emily Kornman, Daniel Vance, Blair Casey, Steve M. Gleason, Philip Q. Nelson, and Michael P. Brenner. Using large language models to accelerate communication for eye gaze typing users with ALS. *Nature Communications*, 15(1):9449, 2024. doi: 10.1038/s41467-024-53873-3.
- John G. Cleary and Ian H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984. doi: 10.1109/TCOM.1984.1096090.
- Andrew Fowler, Kurt Partridge, Ciprian Chelba, Xiaojun Bi, Tom Ouyang, and Shumin Zhai. Effects of language modeling and its personalization on touchscreen typing performance. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI ’15*, pp. 649–658. ACM, 2015. doi: 10.1145/2702123.2702503.
- Dylan Gaines and Keith Vertanen. Adapting large language models for character-based augmentative and alternative communication. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 15273–15291, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-emnlp.826. URL <https://aclanthology.org/2025.findings-emnlp.826/>.
- Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=nZeVKeeFYf9>.
- Frederick Jelinek, Bernard Meriello, Salim Roukos, and Martin Strauss. A dynamic language model for speech recognition. In *Speech and Natural Language: Proceedings of a Workshop Held at Pacific Grove, California, February 19–22, 1991*, pp. 293–295, 1991. doi: 10.3115/112405.112464. URL <https://aclanthology.org/H91-1057/>.
- Bryan Klimt and Yiming Yang. The Enron corpus: A new dataset for email classification research. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi (eds.), *Machine Learning: ECML 2004*, pp. 217–226, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-30115-8.
- Roland Kuhn and Renato De Mori. A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):570–583, 1990. doi: 10.1109/34.56193.
- Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech 2010*, pp. 1045–1048, 2010. doi: 10.21437/Interspeech.2010-343.
- Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. *arXiv preprint arXiv:1910.02054*, 2020. URL <https://arxiv.org/abs/1910.02054>.

- Zhanna Sarsenbayeva, Niels van Berkel, Chu Luo, Vassilis Kostakos, and Jorge Goncalves. Challenges of situational impairments during interaction with mobile devices. In *Proceedings of the 29th Australian Conference on Computer-Human Interaction, OzCHI '17*, pp. 477–481. Association for Computing Machinery, 2017. doi: 10.1145/3152771.3156161.
- Junxiao Shen, Boyin Yang, John J. Dudley, and Per Ola Kristensson. Kwickchat: A multi-turn dialogue system for AAC using context-aware sentence generation by bag-of-keywords. In *Proceedings of the 27th International Conference on Intelligent User Interfaces, IUI '22*, pp. 853–867. Association for Computing Machinery, 2022. doi: 10.1145/3490099.3511145.
- Keith Trnka and Kathleen McCoy. Evaluating word prediction: Framing keystroke savings. In *Proceedings of ACL-08: HLT, Short Papers*, pp. 261–264, Columbus, Ohio, 2008. Association for Computational Linguistics. URL <https://aclanthology.org/P08-2066/>.
- Stephanie Valencia, Richard Cave, Krystal Kallarackal, Katie Seaver, Michael Terry, and Shaun K. Kane. “The less I type, the better”: How AI language models can enhance or impede communication for AAC users. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, CHI '23*, pp. 1–14. Association for Computing Machinery, 2023. doi: 10.1145/3544548.3581560.
- David J. Ward, Alan F. Blackwell, and David J. C. MacKay. Dasher—a data entry interface using continuous gestures and language models. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology, UIST '00*, pp. 129–137. ACM Press, 2000. doi: 10.1145/354401.354427.
- Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Table 4: Dataset statistics after preprocessing.

Partition	Total			Words per user		
	Users	Phrases	Words	Median	Min.	Max.
DEV	45	109,696	1,069,532	13,585	2,581	177,216
TEST	44	92,784	968,531	11,651	2,196	107,709

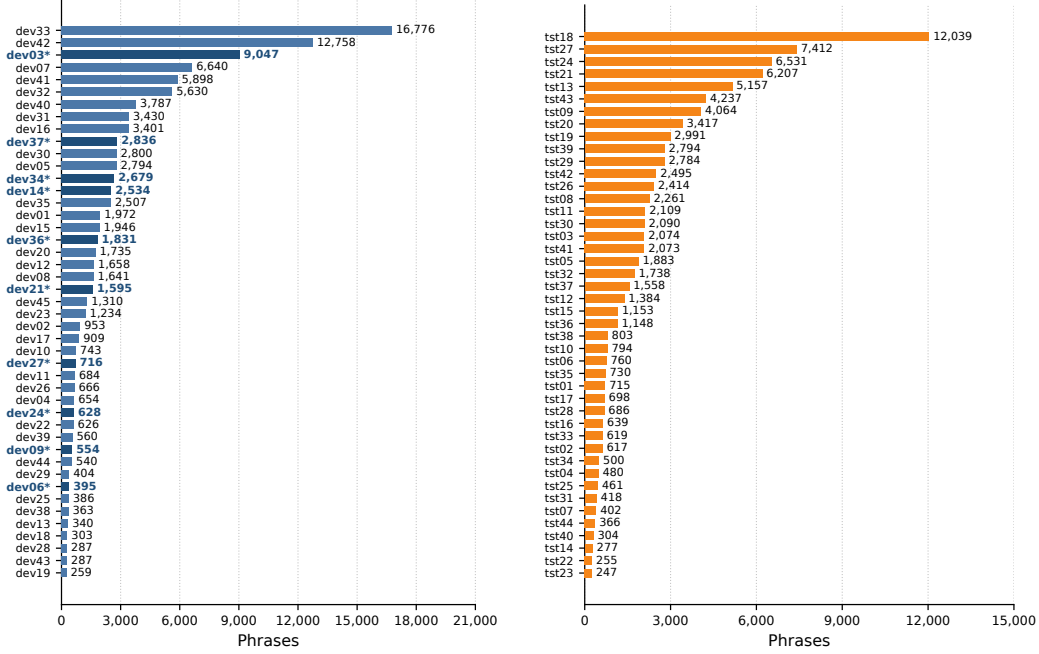


Figure 1: Number of phrases per user in the full 45-user development partition (left) and the 44-user TEST set (right), sorted from most to fewest phrases. Asterisks and darker labels indicate the fixed 10-user DEV subset used for model selection and hyperparameter tuning.

A Appendix

A.1 Enron dataset processing

We started with the modified version of the Enron Personalization Validation Set provided by [Adhikary & Vertanen \(2023\)](#). For each Enron employee, the dataset has a time-ordered list of phrases from the employee’s sent emails. A phrase is normally a full sentence. The employees were split into a development partition containing 45 users and a test partition containing 44 users.

We further processed the dataset by removing phrases containing digits. For the remaining phrases, prediction targets were converted to lowercase and restricted to letters, spaces, and apostrophes by stripping other characters. We split each user’s chronologically ordered stream at its midpoint, using the earlier half as seed data and the later half for evaluation. Table 4 summarizes the full 45-user development partition and the 44-user TEST partition after preprocessing.

Figure 1 shows the number of phrases available for each user in the full development and test partitions after our preprocessing. To reduce computation for our development experiments, we constructed DEV from a representative subset of the full 45-user development partition. In particular, we used dev03, dev06, dev09, dev14, dev21, dev24, dev27, dev34, dev36, and dev37. These users were used for model selection and hyperparameter tuning. We constructed our TEST set from all 44 users in the test partition.



Figure 2: Per-user changes in per-character perplexity relative to the non-personalized OPT-13B model. The top panel shows full fine-tuning, the middle panel shows LoRA fine-tuning, and the bottom panel shows progressive LoRA fine-tuning. Negative values indicate lower perplexity than the base model. Users are sorted from most to least improved in each graph.

A.2 Per-user perplexity changes

We compared three different methods for personalizing an LLM based on each user’s data:

- Full parameter fine-tuning on the user’s seed set.
- LoRA fine-tuning on the user’s seed set.
- LoRA fine-tuning on the user’s seed set, followed by progressive LoRA fine-tuning every 128 phrases.

For each user and method, we computed the change in per-character perplexity between the adaptation method (PPL_{method}) and the non-personalized OPT-13B baseline (PPL_{base}):

$$\Delta PPL = PPL_{\text{method}} - PPL_{\text{base}}. \quad (1)$$

Negative values indicate personalization improved prediction for that user, while positive values indicate personalization degraded prediction. As shown in Figure 2, the vast majority of users saw improved predictions. In the case of progressive fine-tuning, all users saw an improvement. Full-parameter fine-tuning improved PPL for 38 of 44 users, LoRA improved PPL for 41 of 44 users, and progressive LoRA improved PPL for all 44 users. The mean ΔPPL values for the methods were -0.018 , -0.016 , and -0.024 , respectively.

A.3 Hyperparameter tuning

For our three LLM personalization methods, we searched for the best hyperparameters. We selected the hyperparameters that minimized the mean per-character perplexity across

Setting	Chunk size	LoRA rank	LoRA alpha	Dropout rate	Warmup	Stable	Learning rate	Weight decay	Epochs
Full fine-tuning	-	-	-	0.2	2	0.15	1×10^{-5}	0.05	2
LoRA	-	8	16	0.1	2	0.15	3×10^{-4}	0.00	2
Progressive LoRA	128	8	16	0.1	2	0.15	3×10^{-4}	0.00	1

Table 5: Fine-tuned hyperparameters for our three OPT-13B personalization methods.

the on our fixed 10-user DEV subset, selected from the full partition of 45 development users. Each optimization trial was conducted on four NVIDIA A100 GPUs, each with 40 GB of memory. To reduce per-device memory requirements, we employed DeepSpeed ZeRO stage 3 (Rajbhandari et al., 2020). All trials used OPT-13B with a left context of up to 512 characters. We optimized each method for approximately two GPU days. The reported LLM inference latencies were measured using FP16 on a single NVIDIA A100 GPU per evaluation process.

We tuned the dropout rate, learning rate, weight decay, and epochs. We selected the learning rate from the candidate set [5e-6, 7e-6, 8e-6, 1e-5, 1.3e-5, 1e-4, 3e-4, 3e-3]. The learning-rate schedule had two additional tunable parameters: the number of warmup steps, selected from 0 to 10, and the stable fraction, selected from 0.00 to 0.80. The stable fraction controlled the fraction of training steps for which the learning rate was held fixed before decay. For the LoRA methods, we additionally tuned the adapter’s rank and alpha.

The progressive LoRA method had a parameter *chunk size* controlling how often we re-fine-tuned the adapter. We started each of the 10 users with a model fine-tuned on their seed set using the best hyperparameters found for the non-progressive LoRA method. We then tested each user at chunk sizes of 64, 128, 256, 512, and 1024. Table 5 shows our tuned hyperparameters.