Keylogging in a Web-Based Code Editor for Fine-Grained Analysis and Early Prediction of Student Performance

Xavier Rene Plourde, University of California, Berkeley
Dr. Garrett Ethan Katz, Syracuse University

Garrett Katz is an assistant professor at Syracuse University. He teaches a broad range of computer science courses, covering introductory programming, discrete math, introductory artificial intelligence, and graduate seminars. His research covers various topics in artificial intelligence and human-machine interaction, including in educational contexts. In particular, his recent work investigates reasoning and learning processes underlying program synthesis, both for automated program synthesis by machines as well as manual program synthesis by human computer science students.

Keylogging in a Web-Based Code Editor for Fine-Grained Analysis and Early Prediction of Student Performance

Introduction

Computer programming often presents a steep learning curve for novice students. One approach to improve learning outcomes is quantitative modeling of the student reasoning process [1]. Accurate models can detect when students struggle and predict whether they will succeed [2]. These predictions can prompt instructor intervention, or guide automated tutoring systems in providing hints [3]. A student's reasoning is manifested in the rich, non-linear progression of their code during their authoring process, as it evolves from blank page to completed program. These dynamics can be captured for subsequent modeling and assessment by keylogging student coders.

In this paper, we present a new software tool for collecting keylog data. The tool is a web-based code editor that records all user keystrokes in the backend. We also describe a small pilot study used to validate the tool. In contrast with existing work, our tool is open source, portable and remotely accessible over the web, supports multiple real-world languages (Python and Javascript), and permits full configuration of the coding problems, test cases, and surveys presented to the user. The editor itself and our data analysis scripts are released under an MIT license and freely available online for the computing education research community:

`https://github.com/garrettkatz/edit-pattern-app`

In our pilot study, we keylogged six students on a series of coding problems. We used surveys and automated unit tests to gauge their perceptions and performance, respectively. We also investigated whether success on a problem could be predicted in advance, based on keystrokes early in the attempt, similar to recent work with block-based programming [2, 4, 5].

Related work

Keylogs are a common means of studying computer-based problem-solving behavior. For example, latency between keystrokes is predictive of performance in programming [6] as well as essay writing [7]. Keystroke timing can be used for other modeling tasks as well, such as identity resolution of the user [8, 9]. Timing-based features are often specialized into different categories (e.g., latency within words vs. between words), and summarized with multiple statistics such as mean, median, and maximum [7]. These features have also been combined with coarse-grained cursor information, such as whether or not an insertion was made at the "leading edge" (i.e., typed at the very end of the current draft, as opposed to a revision somewhere in the middle) [10].

Another approach used in an automated tutoring context is knowledge tracing (KT) [1]. KT compares actual student behavior with a system of production rules, which are assertions about which code fragment an "ideal" student would write next in a variety of partially complete programming tasks. This core idea has been developed and extended over the past several decades to incorporate more data-driven techniques such as Bayesian inference and deep learning [11].

Recent work on early performance prediction has focused primarily on block-based programming environments [2, 4, 5]. These works use sophisticated feature extraction methods, based on

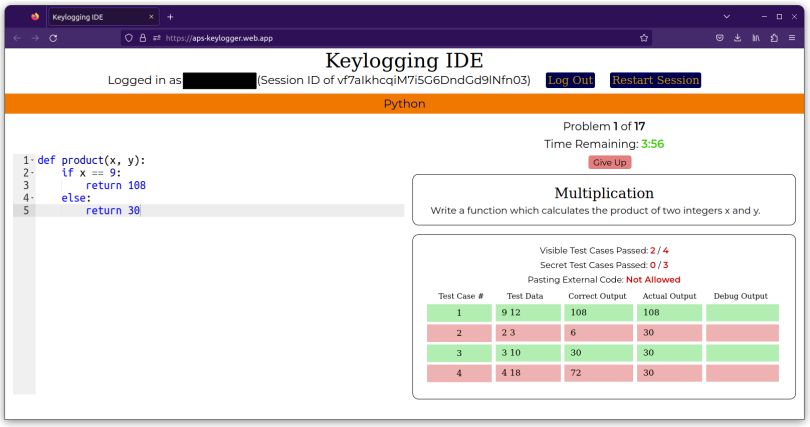

Figure 1: The web-based keylogging IDE. In this example the user has "hard-coded" some test cases in their implementation and is not passing any other test cases. The username is concealed behind a black box for privacy reasons.

abstract syntax trees (ASTs) [12] and execution traces [13], in conjunction with data-driven techniques such as recent temporal patterns [14] and deep code embeddings [15, 16].

Methods

We built a web-based integrated development environment (IDE) which keylogs users on a series of coding problems (Figure 1). Each problem prompts the user to implement a function described conceptually at center-right. The user writes code on the left. A suite of automated unit tests repeatedly run in the background in real time as the user codes. Test results are shown to the user at lower right. The attempt ends once all tests pass or a timer runs out, at which point the user is prompted with a brief survey. The tool has several features such as secret tests whose input-output data is not revealed, disabled cross-site copy-paste from 3rd-party sources (e.g., ChatGPT), and participant authentication using a Google account to keep keylogs secure. Experimenters can configure and deploy their own instance of the application following the documentation in the code repository. Configuration uses a simple, human-readable, text-based format to specify problem descriptions, initial code stubs, test case data, survey questions, and other parameters.

We validated the tool with a pilot study approved by our institutional review board which involved 17 problems of increasing difficulty and duration, for a total of 105 minutes. For example, the final problem was to determine the winner in a tic-tac-toe board (complete problem details are available in our code repository). The 1^{st} problem was used for familiarization and excluded from analysis. Trials were administered remotely to accommodate lingering issues from COVID-19. In future work it may be more appropriate and aligned with extant literature to use in-person trials and longer time limits with fewer, more complex coding challenges (e.g., the 20 minute challenge in [2]). Brief surveys after each problem inquired how familiar the problem was and how advanced it felt. Six participants were recruited from undergraduate programming courses taught by a colleague who offered extra credit for participation.

Collected data was de-identified and cleaned, removing some missing data due to issues like insufficient background knowledge that prevented some students from attempting some problems. The cleaned data comprised 89 attempts across all problems and participants. Although the

sample size is too small for conclusive results, we used this data to validate a typical predictive modeling workflow. The model objective was to predict whether an attempt would succeed (i.e., pass 100% of the automated tests), based on keystroke activity early in the attempt. For the predictive model we used a linear Support Vector Classifier (SVC) [17], as implemented in scikit-learn [18]. The output class label was 0 for a failed attempt and 1 for success.

For SVC input we compared two feature sets. First, we used 21 primarily "timing-based" features from [7], such as mean time interval between keystrokes, calculated on an early part of the attempt to see if success could be predicted in advance. Second, we extracted custom "cursor-based" features comprising the starting and ending position of every contiguous edit, divided by the current length of the draft. For example, a cursor position of 1.0 always represents a "leading-edge" edit at the end of the current source code, regardless of its length. We concatenated cursor positions for the first L edits, where L was a parameter we varied experimentally. For large L, some trials have less than L contiguous edits total, so the number of available training samples (which we denote N_L) decreases. For a fair comparison, the portion of the trial up to the first L edits was the same portion used for timing-based features.

For each L, we gauged model generalization using repeated stratified 3-fold cross-validation [19] with 10 repetitions, and report the average generalization accuracy over the folds. For more robust estimates, we required that each fold contained at least 5 examples with each label. Values of L for which this was violated were excluded from the analysis, leaving only $L \leq 21$, which is why we limited the timing-based features to 21 also. We calculated raw test accuracy (percentage of correctly predicted labels) as well as balanced test accuracy [20], which accounts for the unequal number of 0 and 1 labels in the data. We used default hyperparameter settings for the SVC, except that number of training iterations was increased to 20K for cursor-based features and 100K for timing-based features, which we found necessary to ensure convergence in all training runs.

Results

The six participants were all undergraduates and intended computer science majors, 18-20 years old, with up to four years of programming experience. Figure 2 shows that performance is highly bimodal: Students tend to pass almost all or none of the tests on a given problem. There are some individual differences, but all participants have clusters of scores at both extremes. Participants tended to perform better on problems they rated as more familiar and less difficult, but the correlation was quite weak, especially at the "all or none" modes of the performance distribution.

Figure 3 shows the time-evolution of code length and cursor position throughout each attempt. Cursor position often varies non-monotonically, especially in unsuccessful attempts, which motivated our cursor-based feature design. We also observe some plateaus in cursor position over time, indicating high latency between keystrokes. Plateaus are less common in successful attempts, corroborating prior findings that latency is negatively correlated with performance [6].

Figure 4 shows the predictive model performance on each feature set. As L increases (i.e., a larger leading portion of the attempt is provided to the model), the number of available samples (N_L) decreases. Small sample size notwithstanding, cursor-based features appear potentially advantageous, particularly in the very early portion of the attempt. This regime is most meaningful, since the sample size is larger and earlier predictions are more useful to educators.

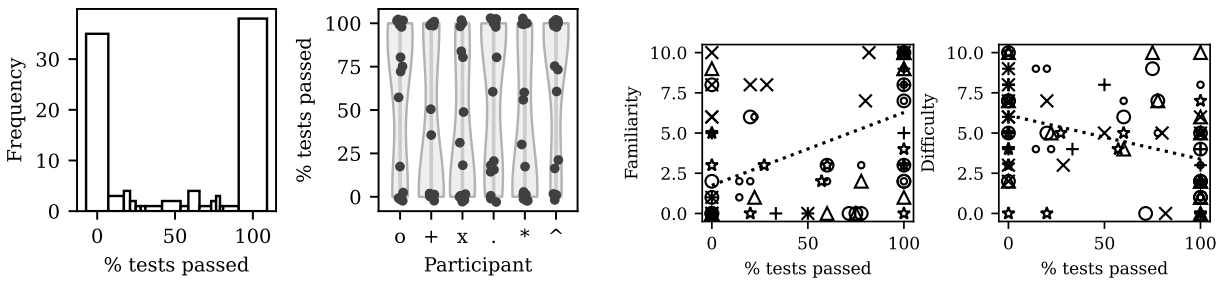

Figure 2: Far left: Histogram of test performance across all attempts. Bin boundaries are set at the midpoints between distinct test scores occurring in the data. Middle left: Violin plots of performance by participant. Gray points indicate performance on individual problems, with small random noise added for better visualization of coincident points. Test performance vs. survey responses are shown at middle right (familiarity rating) and far right (difficulty rating), where markers indicate participants as given in the x-axis labels at middle left. Dotted lines are best-fit linear trends.

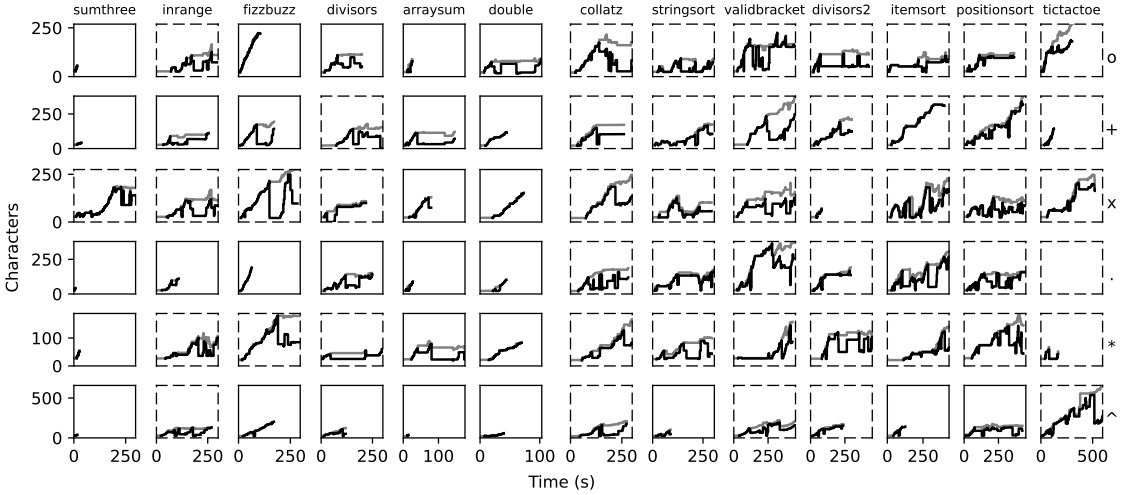

Figure 3: Cursor position (black lines) and code length (gray lines) measured in characters from the beginning of the source code (y-axes), vs. time in seconds (x-axes), one subplot per attempt. Solid and dashed panel borders indicate eventual success vs. failure, respectively. Rows correspond to participants (labeled on right) and columns to a representative subset of problems (labeled on top; see code repository for full descriptions of each problem).

Discussion

We have presented a new web-based code editor that keylogs users. As a free and open-source web-based application, it is operating-system agnostic, requires no installation by participants, can be administered remotely, and is readily available to education researchers. This makes it well-suited to virtual instruction/research settings and large-scale data collection.

Although our pilot data is too small for conclusive results, it does present some noteworthy observations that may be relevant to education research and warrant further study. First, the highly bimodal performance could mean that students rely more on memorization than problem solving, which may be undesirable. Second, the discrepancies between survey responses and success rate suggest a possible need for students to alter their perceptions of their coding work. Third, we observed highly non-monotonic cursor dynamics that may be predictive of student

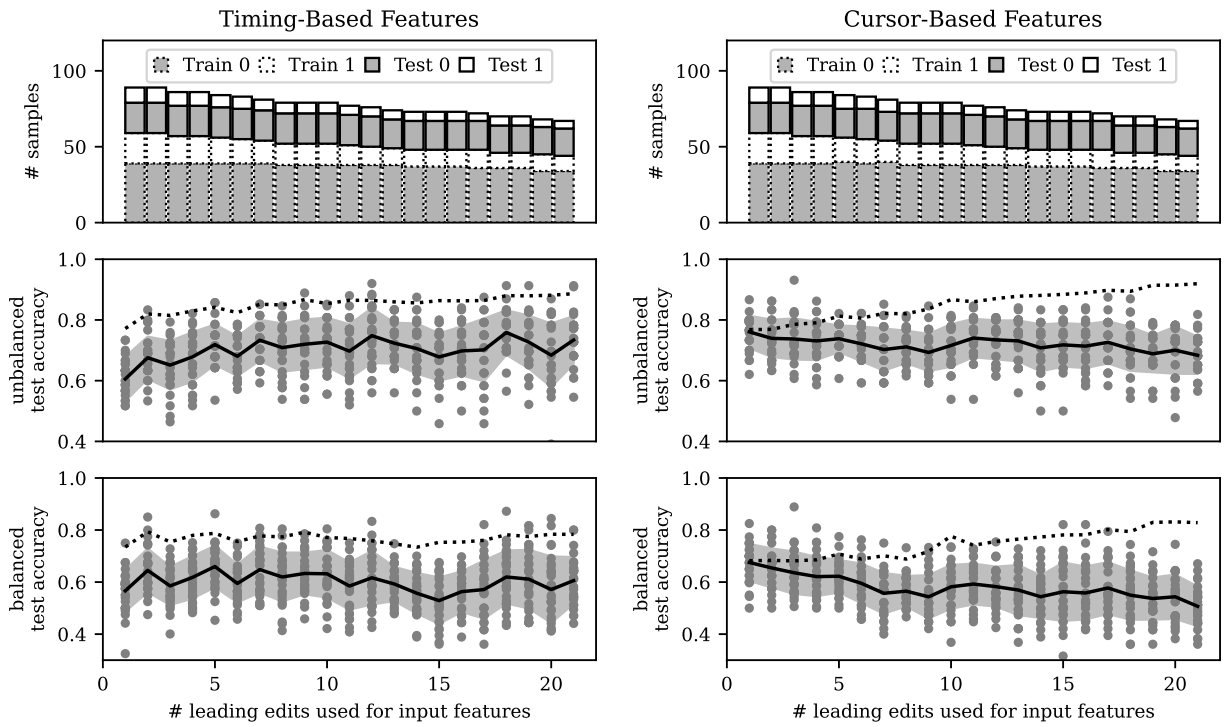

Figure 4: Success prediction using timing-based (left) and cursor-based (right) features. Top: Stacked bar chart of available samples N_L for each leading edit length L, grouped according to train vs. test set size and 0 vs. 1 labels. Middle and bottom: Unbalanced and balanced classification accuracy vs. L. Dashed and solid black lines are training and testing accuracy, respectively, averaged over 10 repetitions $\times$ 3 validation folds, with standard deviation in test accuracy indicated by the light gray envelope. Dark gray points show test accuracy in each fold of each repetition.

struggles early in their attempt. Cursor dynamics may therefore serve as a quantitative basis for awarding partial credit based on effort or flagging struggling students for instructor intervention.

Our pilot also revealed some limitations in our current experimental design to be addressed in future work. First and foremost, substantially more data is needed; ideally covering many more students with a wider range of proficiency levels. Related to that, coding challenges must be more carefully calibrated to the background knowledge of the participant population. With a larger dataset, we will also be better positioned to compare or combine cursor-based features with current state-of-the-art student modeling techniques based on deep learning. Lastly, we hope to continue improving our web application and share data with a growing network of investigators. This will help us scale data collection efforts to multiple institutions for mutual benefit and knowledge generation in the field of engineering education research.

Acknowledgements

Many thanks to Dr. Nadeem Ghani, for helping us recruit student participants from his courses.

References

[1] A. T. Corbett and J. R. Anderson, "Knowledge tracing: Modeling the acquisition of procedural knowledge," *User modeling and user-adapted interaction*, vol. 4, pp. 253–278, 1994.

[2] Y. Mao, R. Zhi, F. Khoshnevisan, T. W. Price, T. Barnes, and M. Chi, "One minute is enough: Early prediction of student success and event-level difficulty during a novice programming task.," *International Educational Data Mining Society*, 2019.

[3] K. Rivers and K. R. Koedinger, "Data-driven hint generation in vast solution spaces: a self-improving python programming tutor," *International Journal of Artificial Intelligence in Education*, vol. 27, pp. 37–64, 2017.

[4] Y. Dong, S. Marwan, P. Shabrina, T. Price, and T. Barnes, "Using student trace logs to determine meaningful progress and struggle during programming problem solving," *International Educational Data Mining Society*, 2021.

[5] A. Emerson, M. Geden, A. Smith, E. Wiebe, B. Mott, K. E. Boyer, and J. Lester, "Predictive student modeling in block-based programming environments with bayesian hierarchical models," in *Proceedings of the 28th ACM Conference on User Modeling, Adaptation and Personalization*, pp. 62–70, 2020.

[6] R. C. Thomas, A. Karahasanovic, and G. E. Kennedy, "An investigation into keystroke latency metrics as an indicator of programming performance," in *Proceedings of the 7th Australasian conference on computing education*, vol. 42, pp. 127–134, 2005.

[7] R. Conijn, C. Cook, M. van Zaanen, and L. Van Waes, "Early prediction of writing quality using keystroke logging," *International Journal of Artificial Intelligence in Education*, vol. 32, no. 4, pp. 835–866, 2022.

[8] K. Longi, J. Leinonen, H. Nygren, J. Salmi, A. Klami, and A. Vihavainen, "Identification of programmers from typing patterns," in *Proceedings of the 15th Koli Calling conference on computing education research*, pp. 60–67, 2015.

[9] A. Peacock, X. Ke, and M. Wilkerson, "Typing patterns: A key to user identification," *IEEE Security & Privacy*, vol. 2, no. 5, pp. 40–47, 2004.

[10] E. Lindgren and K. P. Sullivan, "Analysing online revision," in *Computer keystroke logging and writing: methods and applications*, pp. 157–188, Elsevier, 2006.

[11] G. Abdelrahman, Q. Wang, and B. Nunes, "Knowledge tracing: A survey," *ACM Computing Surveys*, vol. 55, no. 11, pp. 1–37, 2023.

[12] Y. Mao, Y. Shi, S. Marwan, T. W. Price, T. Barnes, and M. Chi, "Knowing both when and where: Temporal-astnn for early prediction of student success in novice programming tasks," in *In Proceedings of the 14th International Conference on Educational Data Mining (EDM) 2021*, 2021.

[13] G. Cleuziou and F. Flouvat, "Learning student program embeddings using abstract execution traces," in *14th International Conference on Educational Data Mining*, pp. 252–262, 2021.

[14] I. Batal, D. Fradkin, J. Harrison, F. Moerchen, and M. Hauskrecht, "Mining recent temporal patterns for event detection in multivariate time series data," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 280–288, 2012.

[15] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–29, 2019.

[16] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A novel neural source code representation based on abstract syntax tree," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 783–794, IEEE, 2019.

[17] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152, 1992.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[19] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *IJCAI*, vol. 14, pp. 1137–1145, Montreal, Canada, 1995.

[20] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, "The balanced accuracy and its posterior distribution," in *2010 20th international conference on pattern recognition*, pp. 3121–3124, IEEE, 2010.