CBIE 2023
CONGRESSO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO
Uma escola para o futuro: Tecnologia e conectividade a serviço da educação

Dealing with a large number of students and inequality when teaching programming in higher education

Gabriel Xará[1], Laura O. Moraes[1], Carla A. D. M. Delgado[2], João Pedro Freire[3], Claudio Miceli de Farias[4]

[1]Programa de Pós-Graduação em Informática (PPGI)
Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

[2]Instituto de Computação (IC)
Universidade Federal do Rio de Janeiro (UFRJ)

[3]Instituto de Matemática (IM)
Universidade Federal do Rio de Janeiro (UFRJ)

[4]Programa de Engenharia de Sistemas e Computação (PESC/COPPE)
Universidade Federal do Rio de Janeiro (UFRJ)

gabriel.xara@edu.unirio.br, laura@uniriotec.br, carla@ic.ufrj.br,

joao.freire@poli.ufrj.br, cmicelifarias@cos.ufrj.br

Abstract. *Considering the characteristics found in the post-pandemic scenario of public higher education in Brazil, we must address issues related to equity in access to educational resources. We present the new features of Machine Teaching, a web system used in introductory programming classes to support students and instructors. The innovations aim to adapt the system to post-pandemic conditions and the diverse public resulting from policies to democratize access to Brazilian universities. We present functionalities to mitigate problems related to many students per instructor, such as student dashboards and alerts indicating disengaged students who are likely to drop out. We also address computer and internet access difficulties by transforming the architecture from client-based to remote-based.*

1. Introduction

The 21st century's first two decades were a significant encouragement period from the Brazilian Federal Government to expand access to higher education. During these years, the Federal Government offered more than 9 million higher education student vacancies, directly or indirectly, in all of Brazil's federate units[1]. The attendees' scenario in Brazilian universities changed due to policies to democratize access to higher education, which include opening new institutions in all regions of the country, reserving places for students from public schools (quotas policy [Lima et al. 2014]), and offering students financial support. Consequently, the educational work of higher education also changes.

A critical point is the increase in the number of students per class. Although instructors and students may prefer classes with a small number of students

[1]Data obtained from the 2017 report "Censo da Educação Superior: Sinopse Estatística – 2017" produced by "Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (INEP)". Available at https://www.gov.br/inep/pt-br/acesso-a-informacao/dados-abertos/sinopses-estatisticas

[Paul J. Baker and Tolone 1974], this option has lost economic viability compared to other priorities for resource investments. To deal with a large number of students per instructor, we must address issues related to equity in access to educational resources. In a country like Brazil, a significant portion of university students must be assisted by financial support policies to make their studies viable. Their home access conditions to resources such as the internet and computers are precarious. In regions of social vulnerability, such as *favelas*, broadband internet access services are rarely available [Cunha et al. 2023]. Conditions worsened during the COVID-19 pandemic when remote teaching became the only possible practice [Castioni et al. 2021]. Also, several students' families lost their main sources of income. In 2023, as the WHO declared the end of the pandemic state, the Brazilian universities fully returned to face-to-face activities, with a deficient budget and highly deteriorated infrastructure and computational equipment [Hipólito et al. 2022].

In the quest to maintain introductory programming courses running at *Universidade Federal do Rio de Janeiro (UFRJ)*, the main pedagogical support tool used, Machine Teaching [Moraes et al. 2021], was modified in favor of students' equal access. Machine Teaching is an online learning environment to support introductory programming classes. Its features include an in-browser integrated development environment (IDE) for students to code and submit their assignments; dashboards where instructors can analyze students' difficulties; and alerts for instructors signaling students who are at risk of low performance or dropout. Machine Teaching's main functionalities and achievements were already described in previous papers [Moraes et al. 2021, Moraes et al. 2022]. The system is useful for students as it provides didactic support for programming practice through automatic testing and feedback mechanisms, as well as operational support through the student activities' organization and dashboards with information on expected resolution time and the activities' difficulty. For instructors, the tool is useful for dealing with a large number of students by supporting programming tasks feedback, providing class difficulties consolidated views and students' performance over time, and alerting about students at risk of dropping out.

In this article, we address issues related to the adaptation of Machine Teaching to the scenario of inequalities [2], which we perceive to have worsened in the post-pandemic. The updated architecture allows access by smartphones and client machines with low computational power and deals with weak and intermittent connections. The tool supports instructors in managing a great number of students.

Section 2 presents works related to ours – other online environments that support programming teaching and learning tasks. Section 3 briefly describes the system Machine Teaching, highlighting the functionalities to support instructors in dealing with a massive number of students. Section 3.2 details how Machine Teaching's architecture was modified to support students accessing from any device with internet connectivity. Our conclusions and future work are stated in section 4.

[2]These are challenges related to United Nations (UN) sustainable global goals number 4, "quality education", and 10, "reduced inequalities" [United Nations and Development 2015].

2. Literature review

Online environments that automatically correct source code are commonly used as pedagogical support tools in programming courses, mainly due to their speed in providing student feedback. Table 1 gathers automatic code correction environments for Python language taken from systematic literature reviews conducted by [Ihantola et al. 2015] and [Luxton-Reilly et al. 2018] (CloudCoder [Hovemeyer and Spacco 2013, Papancea et al. 2013], CodeWorkout [Panamalai Murali 2016], PCRS [Zingaro et al. 2013], UUhistle [Sorva and Sirkiä 2010], Pythy [Edwards et al. 2014], Web-CAT [Edwards and Perez-Quinones 2008], PEEF [Araujo et al. 2021]), as well as two environments developed by Brazilian universities that were not included in the conducted review (CodeBench [Galvão et al. 2016] and Beecrowd [Bez et al. 2012, Bez et al. 2013]), a popular commercial web-system (Repl.it), and the Machine Teaching environment mentioned in this paper. These environments' functionalities were analyzed and compared in a previous paper [Moraes et al. 2022]. In this paper, we investigate whether these tools are suitable for use in classes with a large number of students, severe budget restrictions, and students with deficient access to computers and the internet. These tools are analyzed and compared based on five criteria[3]:

1. Is the online learning environment still active (recent developments)? When was the last update?
2. Can the environment be accessed using a smartphone?
3. Is the environment distributed as open source?
4. Does the environment provide a dedicated area for instructors with statistics and graphs so that they can have both an overview of the class and specific students? What is available?
5. Was the environment evaluated? By which method?

The second question assesses if the tool can be used in a smartphone. According to the Brazilian Institute of Geography and Statistics (IBGE), 99.6% of the individuals with internet access, do it using a smartphone in contrast with 42.7% that do it using a computer (an individual can use both). It means that only half of the people with internet access also have access to a computer with internet. But almost everyone with internet access has access to a smartphone with the internet as well[4]. Nowadays, most web systems are built upon javascript libraries and HTML5 that provide them with a responsive design and allow them to function and automatically adjust to different screen sizes and devices. Therefore, all the tools that were developed as web systems ran perfectly on smartphones. The tools that did not work were developed as desktop applications or were not available online.

To consider budget restrictions, we analyzed if the tools are distributed as open source and if they still have active developments (questions three and one, respectively). A tool distributed as open source is guaranteed to be always accessible (institutions could always manage the system locally) and to never have a fee that could derail the tool's

[3]This review is limited in comparing the tool's availability and features. The analyzed papers do not present their architecture, so they could not be compared.

[4]Data obtained from the 2021 Continuous National Household Sample Survey. Available at https://sidra.ibge.gov.br/tabela/7311

Table 1. Online environments to autocorrect Python code. Dashboard features: G=grades; S=submissions; ST=statistics; D=dropout prediction

Environment	1. Active / last update	2. Smart-phone	3. Open Source	4. Dashboards	5. Evaluation
CloudCoder	No / Feb 2016	N/A	Yes	G	Not evaluated
CodeWorkout	Yes / Nov 2021	Yes	Yes	G S	Questionnaire
PCRS	Yes / Jan 2023	Yes	Yes	G S	Not evaluated
UUhistle	No / N/A	No	No	No dashboard	Questionnaire and control group
Pythy	No / Feb 2020	Yes	Yes	No dashboard	Questionnaire
Web-CAT	Yes / Sep 2020	Yes	Yes	G S	Control group
PEEF	No / Abr 2021	No	No	G S ST	Not evaluated
CodeBench	Yes / N/A	Yes	No	G S ST	Questionnaire and control group
Beecrowd	Yes / May 2023	Yes	No	G S ST	Questionnaire
Repl.it	Yes / N/A	Yes	No	G S	N/A
Machine Teaching	Yes / Sep 2022	Yes	Yes	G S ST D	Questionnaire

adoption. Also, recent developments indicate that the system development is active and it still has support for bugs and improved functionalities.

Finally, as analyzed by the fourth question, the investigated tools support the use by instructors and students. For students, the tools generally function as an integrated development environment (IDE), where students code and receive real-time feedback. However, the features for instructors vary significantly. In UUhistle, for example, instructors use the system for simulations and examples. Most tools allow instructors to see whether or not a student has passed a question. Additionally, Beecrowd and the Brazilians CodeBench and Machine Teaching provide student submissions with in-depth analysis through dashboards with graphs or statistics about students and classes. To efficiently deal with a large number of students, the tool must provide consolidated statistics about the questions, classes, and students, instead of only providing a final grade for a question. The statistics allow instructors and course administrators to pinpoint students' difficulties throughout the semester before their final exams and grades.

Machine Teaching differs from these systems because, in addition to supporting the regular use by students, it focuses on data collection for decision support. The dashboards presented in the system provide information so that instructors can reflect on the syllabus' order, the content difficulty, and activities' deadlines, encouraging reflection and supporting the continuous course update. It also provides a dropout prediction feature to alert instructors about students at risk of low performance or dropout.

3. Machine Teaching Web System

This section presents the developed online learning environment to support instructors and students in programming classes. The system's main functionalities were already presented at [Moraes et al. 2022]. This paper focuses on the new requirements and functionalities concerning the support given to instructors in dealing with a great number of students and the differences in access across university students in the Brazilian scenario. We present our data analysis functionalities: dashboards for students' and instructors' awareness, and student dropout prediction. Then, the new system architecture designed to allow an increased number of simultaneous access and access from low computational power machines is presented.

After logging in the system, the students have access to an integrated development environment, where they are presented with a problem, and they should write the expected answer in a free-text coding format. For each exercise, a test case function generator was defined to correct the results. The students receive feedback every time they submit an answer, and they can see whether they passed or failed a unit test case. If they get all of them correct, the task is considered done, and the student may move on to another problem. The system saves a state every time a student submits an answer. The set of submissions is used to generate aggregated statistics regarding problems, content, students, instructors, course, among others.

3.1. Data analysis

3.1.1. Dashboards for awareness

The system offers three interfaces for instructors to analyze student submissions at the individual level or aggregated by class. Figs. 1 and 2 present the interface with detailed information for each student, including their codes, outcomes, and the timestamp of each submission. In particular, the interface in Fig. 2 was requested by the instructors to be able to compare individual student submissions. This interface displays all student submissions side by side, along with the respective submission timestamp, and the percentage of successful test cases for a single problem. In this way, the instructor can identify common errors among students.

To support analysis by class, the dashboard shown in Fig. 3 was proposed. In this dashboard, instructors can verify the content and problems that took the longest for students to solve or required more attempts, which may indicate the need for adjustments in the teaching activity. Consolidated information is also available about individual students. These dashboards are designed to make the instructor's work more efficient and to reduce the time needed to access a class overview and each student's evolution. These features and the system's usability and effectiveness were evaluated in a previous paper [Moraes et al. 2022].

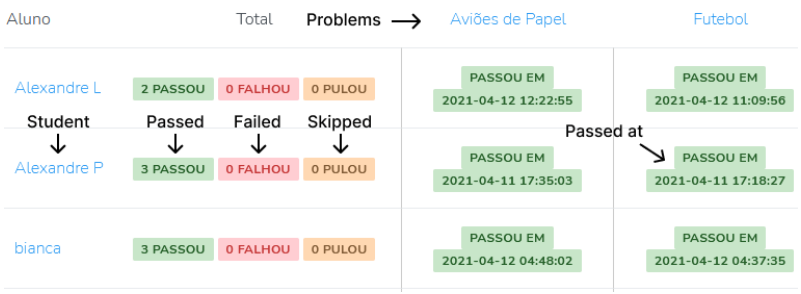

Figure 1. Interface with each student's outcome for a specific chapter.

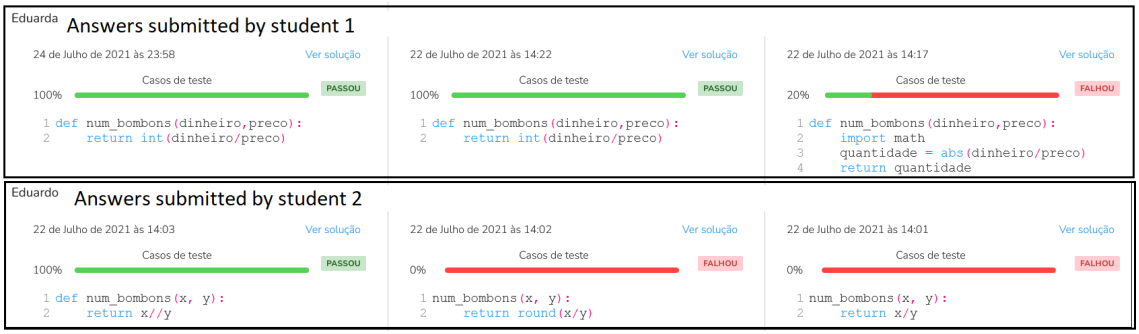

Figure 2. Interface with the detailed solutions for a specific problem for each student.

3.1.2. Dropout prediction

Dropout prediction used to be a simpler task when class sizes were smaller and it was possible to assess student attendance by observing class routines, student behavior, and outcomes in assignments and tests. However, this observation is limited in classes with a massive number of students per instructor. Therefore, we propose a dropout prediction feature that alerts instructors to potential risk situations.

Previous research has achieved positive outcomes in dropout prediction using decision trees and providing interpretable results [Pereira et al. 2019, Damasceno et al. 2019, Al-Shabandar et al. 2018]. We utilized the CART algorithm with the Gini Index as the splitting criterion and 10-fold cross-validation, with 80% of the observations in the train set and the remaining 20% in the test set. In this application, it is more harmful to misclassify the "likely to dropout" class if a student who is likely to drop out is classified as a false-negative. Therefore, the model was optimized for the F1 metric, aiming at balancing sensitivity and specificity for the "likely to dropout" class. Analysis of the resulting tree yielded an F1 score of 0.81 and a Recall of 0.88 for this class, meaning that the model can recover most of the true-positive "likely to dropout" observations and not provide too many false negatives. Weekly success rates emerged as the strongest predictors, followed by the number of attempts per question and unique days of exercise submission, meaning that positive feedback from the system and student continuous attendance are important factors when determining the student dropout risk.

Figure 3. Dashboard for a class. Contains overall class progress, indications of problems and content that required more time or more attempts, and per-student completion statistics.

3.2. Architecture

The initial implementation of Machine Teaching encompasses a peculiar architectural element: the code processing, as submitted by the student, occurs within the student's browser. This client-side processing demands that the employed hardware satisfies minimum requirements. In the year 2022, a refined version of Machine Teaching was conceptualized to bypass the restrictive element of the client-side architecture inherent in the prior version. This novel implementation resolves the issue through the establishment of a remote environment dedicated to code execution, encompassing an auto-scalable server cluster. This adaptation allows students to undertake exercises from any device with internet connectivity. Furthermore, the processing time will no longer be a function of the student's hardware, as a dedicated server cluster will facilitate the processing of exercises submitted through Machine Teaching.

3.2.1. Cluster Management

The solution implemented in the refined version of Machine Teaching is delineated in the diagram presented in Fig 4. The platform adaptation was facilitated by employing the Kubernetes container orchestration system. Each container in the cluster operates as an

isolated environment for executing the students' code. The key factors underscoring the decision to utilize Kubernetes encompass:

- **Load Balancing**: Kubernetes exhibits the ability to balance the load and manage network traffic distribution. Consequently, student submissions will be directed towards the available container.
- **CPU and Memory Bounds**: The system permits the definition of CPU and memory (RAM) limitations for each cluster container. This prevents a surge in workload, whether due to malicious code or non-performant code, from adversely affecting the execution environment of other students.
- **Self-healing**: Kubernetes autonomously manages the restart of containers that encounter failure, replacing containers when necessary, and discarding containers that do not respond to health checks.

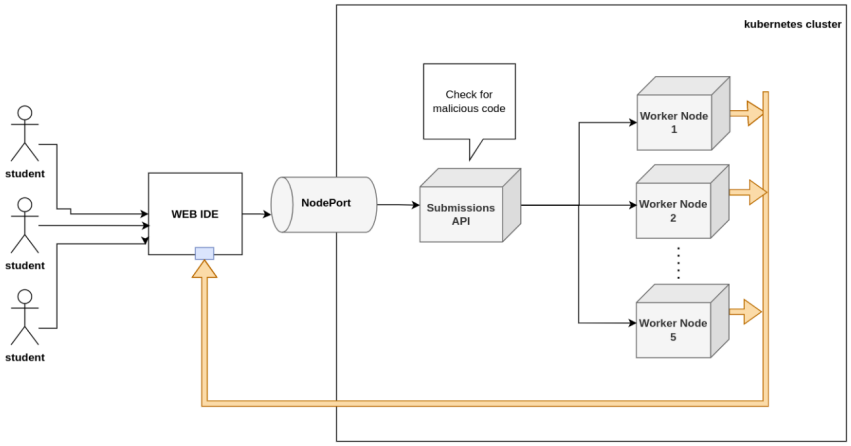

Figure 4. Architecture using a container orchestration system and source code scanning stage.

Beyond alterations in the student's code execution environment, business rules were instituted to ensure scalability and continued security. Evaluation of the implemented system was performed, taking into consideration two principal aspects: scalability and security. The security analysis was predicated on the three pillars of information security, as per [Beal 2005]: confidentiality, integrity, and availability. Conversely, the scalability analysis was conducted with the intent to estimate the maximum number of simultaneous users that the system could accommodate without compromising its operational speed. Details of each evaluation can be seen at [de Castro Xara Wanderley 2023].

3.2.2. Security Evaluation

To safeguard against server compromise, a source code scanning stage was incorporated to identify potentially malicious code or vulnerabilities. This procedure, known as SAST (Static Application Security Testing), leverages several extensively utilized solutions, both open-source and proprietary.

1. **Confidentiality:** Confidentiality is infringed upon when data, expected to be accessed exclusively by a pre-defined set of users, are inadvertently exposed to unauthorized users. To guarantee data confidentiality, the Machine Teaching system is

required to prevent system users from accessing data generated during another user's session. Through a series of tests, it was ascertained that file manipulation and read operations are identified as potential threats and subsequently obstructed. Thus, a malicious user would be unable to access the code submitted by another user or the files generated by them.

2. **Integrity:** The system's integrity would be at risk if a malicious user or agent managed to improperly manipulate any system file. As corroborated by the confidentiality test, breaching the system's integrity proved impossible, as it effectively detects and precludes any form of file manipulation.
3. **Availability:** Lastly, to assess the system's availability, the possibility of compromising it was considered in the event of a code submission rendering one of the servers unavailable, thereby impacting the processing of other users' submissions. Currently, the system lacks any interruption mechanism for submissions that excessively consume CPU and memory resources. Hence, it has been empirically demonstrated that compromising the system's availability is relatively straightforward.

3.2.3. Scalability Evaluation

The evaluation of the system's scalability and resilience includes studying its performance under a load of multiple users and identifying any constraints that may manifest as the number of simultaneous requests increases. A load test was conducted to evaluate the maximum number of users the system could accommodate without the response time exceeding one second. The metric of RPS (requests per second) was employed, assuming each user would generate approximately 0.3 requests per second. The result can be seen in Table 2.

If there is only one available container (all student submissions are sent to the same container in a queue), the system can handle only five requests per second, resulting in 17 simultaneous students submitting exercises. By increasing the number of containers to five, the system can handle up to 18 requests per second, which results in 60 simultaneous users submitting their code. The number of containers can increase or decrease depending on the expected number of simultaneous users and requests.

It is crucial to highlight that the system did not employ threads during this evaluation. Consequently, the system processed the code synchronously. However, it would be misleading to assume that each container managed only one request at a time. In reality, each container provided the submissions API via a WSGI (Web Server Gateway Interface), which utilized a prefork worker to spawn and manage multiple application instances. For this evaluation, the WSGI was configured to spawn 5 instances of the submissions API for each container.

Table 2. Scalability results achieved varying the number of available containers

No. containers	Requests per second (RPS)	Maximum No. simultaneous users
1	5	17
3	15	50
5	18	60

The performance metrics revealed a noteworthy pattern. As the number of containers increased from 1 to 3, there was a predictable increase in requests per second (RPS). However, the transition from 3 to 5 containers did not achieve the expected linear scaling; the server reached only 18 RPS instead of the projected 25 RPS. The underlying cause for this performance deviation remains uncertain, but preliminary investigations suggest that it could be linked to the load-balancing strategy. Within the Kubernetes cluster, the kube-proxy component is responsible for routing requests to an operational pod. A potential limitation here is that kube-proxy lacks insight into a container's internal status. This could result in requests being directed to containers that are already managing their full quota of 5 requests, as dictated by the number of WSGI forks. Further analysis would be required to confirm this hypothesis.

4. Conclusion and future work

In pursuit of fulfilling the UN's sustainable goal of "quality education", policies to expand access to public universities were implemented in Brazil. This changed the scenario in classrooms and brought many challenges, including classes with a large number of students and students without adequate access to computers and the internet. This work presented a learning environment web system to support instructors in programming classes. We considered the instructor's difficulties in managing such a large class and presented functionalities to mitigate these problems, such as automated student feedback in real-time, dashboards with consolidated statistics per class and student, and alerts indicating disengaged students who are likely to drop out.

The presented system had an initial architecture that did not take into account students' low computational power machines by automatically correcting source code on the client. Therefore, we also presented the system's new architecture which establishes a safe remote environment for code execution, releasing the workload from the client machine. It is based on an auto-scalable server cluster and incorporates a source code scanning stage to increase server security. We ran a security evaluation on this architecture and it passed two out of three criteria. The system's confidentiality and integrity were not compromised, but its availability was. Future work will implement an interruption mechanism if an executed code takes longer than expected. The new architecture scalability was also tested. As the number of containers increases, the maximum number of simultaneous users also increases. In our tests, we achieved 60 simultaneous users with a response time of less than one second by deploying just five workers. Another advantage of the proposed architecture is that it promotes a separation of responsibilities. The Web IDE, used to process the student's code in the browser itself, now plays the role of code editor, so that future editions in this module will aim to improve the user experience. The source code execution and validation were separated, allowing updates to expand the coverage of security checks, without the need to alter the cluster that executes the code.

5. Online Resources

1. Machine Teaching: `www.machineteaching.tech`
2. Github: `https://github.com/MachineTeachingEdu/`

6. Acknowledgements

This work was carried out with the support of CNPq (PIBIC UFRJ) and the Instituto Reditus.

References

Al-Shabandar, R., Hussain, A. J., Liatsis, P., and Keight, R. (2018). Analyzing Learners Behavior in MOOCs: An Examination of Performance and Motivation Using a Data-Driven Approach. *IEEE Access*, 6:73669–73685.

Araujo, L. G., Bittencourt, R., and Chavez, C. (2021). Python enhanced error feedback: Uma ide online de apoio ao processo de ensino-aprendizagem em programação. In *Anais do Simpósio Brasileiro de Educação em Computação*, pages 326–333, Porto Alegre, RS, Brasil. SBC.

Beal, A. (2005). *Segurança da Informação: Princípios e Melhores Práticas para a Proteção dos Ativos de Informação nas Organizações*. Atlas.

Bez, J. L., Ferreira, C. E., and Tonin, N. (2013). Uri online judge academic: A tool for professors. In *Proceedings of the 2013 International Conference on Advanced ICT and Education*, pages 744–747.

Bez, J. L., Tonin, N., and Zanin, F. (2012). Enhancing traditional algorithms classes using uri online judge. In *2012 International Conference on e-Learning and e-Technologies in Education (ICEEE)*, pages 110–113.

Castioni, R., Melo, A. A. S. d., Nascimento, P. M., and Ramos, D. L. (2021). Universidades federais na pandemia da covid-19: acesso discente à internet e ensino remoto emergencial. *Ensaio: Avaliação e Políticas Públicas em Educação*, 29(111):399–419.

Cunha, J. K. A., Oliveira, B. R. d., and Fernandes, N. R. (2023). Assistência estudantil na educação superior: A trajetória do programa nacional de assistência estudantil na universidade federal de ouro preto. *Revista Tempos e Espaços em Educação*, 16(35):e18808.

Damasceno, A., Almeida, C., Fernandes, W., Lopes, H., and Barbosa, S. (2019). What Can Be Found from Student Interaction Logs of Online Courses Offered in Brazil. *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação - SBIE)*, 30(1):1641.

de Castro Xara Wanderley, G. M. (2023). *Arquitetura resiliente e escalável para um sistema de execução de código remoto*. Trabalho de conclusão de curso, Universidade Federal do Rio de Janeiro.

Edwards, S. H. and Perez-Quinones, M. A. (2008). Web-CAT: automatically grading programming assignments. In *Proc. 13th Annu. Conf. on Innovation and Technology Computer Science Education*, page 328, Madrid, Spain.

Edwards, S. H., Tilden, D. S., and Allevato, A. (2014). Pythy: improving the introductory python programming experience. In *Proc. 45th ACM technical symposium on Computer science education*, pages 641–646, Atlanta, GA, USA.

Galvão, L., Fernandes, D., and Gadelha, B. (2016). Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação. *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação - SBIE)*, 27(1):140–149.

Hipólito, J., Shirai, L. T., Diele-Viegas, L. M., Halinski, R., Pires, C. S. S., and Fontes, E. M. G. (2022). Brazilian budget cuts further threaten gender equality in research. *Nature Ecology & Evolution*, 6:234.

Hovemeyer, D. and Spacco, J. (2013). Cloudcoder: A web-based programming exercise system. *J. Comput. Sci. in Colleges*, 28:30.

Ihantola, P., Vihavainen, A., Ahadi, A., Butler, M., Börstler, J., Edwards, S. H., Isohanni, E., Korhonen, A., Petersen, A., Rivers, K., Rubio, M. A., Sheard, J., Skupas, B., Spacco, J., Szabo, C., and Toll, D. (2015). Educational data mining and learning analytics in programming: Literature review and case studies. In *Proc. 2015 ITiCSE Working Group Report*, pages 41–63, Vilnius, Lithuania.

Lima, M. E. O., da Costa Neves, P. S., and e Silva, P. B. (2014). A implantação de cotas na universidade: paternalismo e ameaça à posição dos grupos dominantes. *Revista Brasileira de Educação*, 19:141 – 163.

Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., and Szabo, C. (2018). Introductory programming: A systematic literature review. In *Proc. Companion 23rd Annu. ACM Conf. Innovation and Technology in Computer Science Education*, ITiCSE 2018 Companion, page 55–106, Larnaca, Cyprus.

Moraes, L., Delgado, C., Freire, J., and Pedreira, C. (2022). Machine teaching: uma ferramenta didática e de análise de dados para suporte a cursos introdutórios de programação. In *Anais do II Simpósio Brasileiro de Educação em Computação*, pages 213–223, Porto Alegre, RS, Brasil. SBC.

Moraes, L. O., Pedreira, C. E., Delgado, C., and Freire, J. P. (2021). Supporting decisions using educational data analysis. In *Anais Estendidos do XXVII Simpósio Brasileiro de Sistemas Multimídia e Web*, pages 99–102, Porto Alegre, RS, Brasil. SBC.

Panamalai Murali, K. (2016). *CodeWorkout: Design and implementation of an online drill-and-practice system for introductory programming*. Thesis, Virginia Tech.

Papancea, A., Spacco, J., and Hovemeyer, D. (2013). An open platform for managing short programming exercises. In *Proc. 2013 ACM Conf. Int. Computing Education Research*, pages 47–52, San Diego, CA, USA.

Paul J. Baker, R. B. and Tolone, W. (1974). Diversifying learning opportunities: A response to the problems of mass education. *Research in Higher Education*, 2:251–263.

Pereira, F. D., Oliveira, E., Cristea, A., Fernandes, D., Silva, L., Aguiar, G., Alamri, A., and Alshehri, M. (2019). Early Dropout Prediction for Programming Courses Supported by Online Judges. In Isotani, S., Millán, E., Ogan, A., Hastings, P., McLaren, B., and Luckin, R., editors, *Artificial Intelligence in Education*, Lecture Notes in Computer Science, pages 67–72, Cham. Springer International Publishing.

Sorva, J. and Sirkiä, T. (2010). UUhistle: a software tool for visual program simulation. In *Proc. 10th Koli Calling Int. Conf. Computing Education Research*, pages 49–54, Koli, Finland.

United Nations, D. o. E. and Development, S. A. S. (2015). Transforming our world: the 2030 agenda for sustainable development.

Zingaro, D., Cherenkova, Y., Karpova, O., and Petersen, A. (2013). Facilitating code-writing in PI classes. In *Proc. 44th ACM Technical Symp. Computer Science Education*, pages 585–590, Denver, CO, USA.