Seldonian Toolkit: Building Software with Safe and Fair Machine Learning

Austin Hoag
Berkeley Existential Risk Initiative
USA
austinthomashoag@gmail.com

James E. Kostas, Bruno Castro da Silva, Philip S. Thomas, Yuriy Brun
University of Massachusetts
USA
{jekostas, bsilva, pthomas, brun}@cs.umass.edu

***Abstract*—We present the *Seldonian Toolkit*, which enables software engineers to integrate provably safe and fair machine learning algorithms into their systems. Software systems that use data and machine learning are routinely deployed in a wide range of settings from medical applications, autonomous vehicles, the criminal justice system, and hiring processes. These systems, however, can produce unsafe and unfair behavior, such as suggesting potentially fatal medical treatments, making racist or sexist predictions, or facilitating radicalization and polarization. To reduce these undesirable behaviors, software engineers need the ability to easily integrate their machine-learning-based systems with domain-specific safety and fairness requirements defined by domain experts, such as doctors and hiring managers. The Seldonian Toolkit provides special machine learning algorithms that enable software engineers to incorporate such expert-defined requirements of safety and fairness into their systems, while provably guaranteeing those requirements will be satisfied. A video demonstrating the Seldonian Toolkit is available at https://youtu.be/wHR-hDm9jX4/.**

I. Introduction

The use of machine learning (ML) algorithms has become increasingly commonplace, with a wide range of applications including optimizing user experiences [44], providing decision support for high-risk high-impact applications such as criminal sentencing [5], deciding which loans should be approved [6], deciding which resumes should be evaluated by a human [37], and providing medical decision support [29].

Unfortunately, data-driven software can sometimes produce undesirable behavior, such as unsafe or unfair behavior. IBM Watson, for example, recommended potentially fatal cancer treatments [36], and ML software used in 11 US states to predict whether a person will commit a crime in the future was found to have a racial bias [5].

One of the root causes of these undesirable behaviors is the fact that there is a disconnect between the users and the developers of data-driven software. Users, such as doctors, lawyers, and hiring managers, have the expertise to define what unsafe or unfair behavior means. However, these users are not the people building the software systems that must satisfy safety and fairness requirements. The software engineers that do build such systems, by contrast, are typically neither domain experts nor ML experts, so it is critical that they have the ability to integrate ML algorithms and domain-specific safety and fairness requirements.

We introduce the Seldonian[1] Toolkit, a framework that bridges the gap between ML experts, software engineers, and users. The toolkit implements a *Seldonian ML algorithm* [45] that allows domain-expert users to specify safety and fairness requirements, and trains ML models that are probabilistically verified to satisfy those requirements. The Seldonian Toolkit's key contributions are the support for: (1) specification of custom, domain-specific constraints that can encode safety and fairness properties, (2) training ML models while providing high-confidence guarantees that these models, applied to new data, satisfy the specified safety or fairness constraints, and (3) evaluating the effectiveness of Seldonian algorithms for a given use case via comparison to standard ML approaches and other fairness-aware ML algorithms.

II. The Seldonian Toolkit

The Seldonian Toolkit consists of two Python libraries and a graphical user interface that runs in the browser. The Python libraries are the Seldonian Engine[2] and the Seldonian Experiments Library[3]. The graphical user interface is called the Seldonian Interface GUI (SIGUI)[4]. We first describe the SIGUI (Section II-A), the Seldonian Engine (Section II-B), and then the Experiments library (Section II-C).

With beginners in mind, we provide tutorials and examples (with more in development) on how to use the Seldonian Toolkit, starting from installation and progressing toward real-world end-to-end use cases, such as creating safe and fair deep learning and computer vision models with the toolkit.[5]

We have developed the toolkit with ease of adoption in mind, interfacing with ubiquitous tools, such as NumPy, SciPy, scikit-learn, and PyTorch. Currently, six groups of computer science graduate students at the University of Massachusetts are applying the toolkit to distinct ML problems, providing us with feedback.

[1]The toolkit name [45] is a homage to Isaac Asimov's fictional character, Hari Seldon, a resident of a universe where Asimov's three laws of robotics fail to adequately control agent behavior due to their non-probabilistic requirements, and who formulated and solved a machine learning problem that would likely have required probabilistic constraints [7].

[2]https://github.com/seldonian-toolkit/Engine

[3]https://github.com/seldonian-toolkit/Experiments

[4]https://github.com/seldonian-toolkit/GUI

[5]https://seldonian.cs.umass.edu/Tutorials/tutorials/

979-8-3503-2263-7/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSE-Companion58688.2023.00035

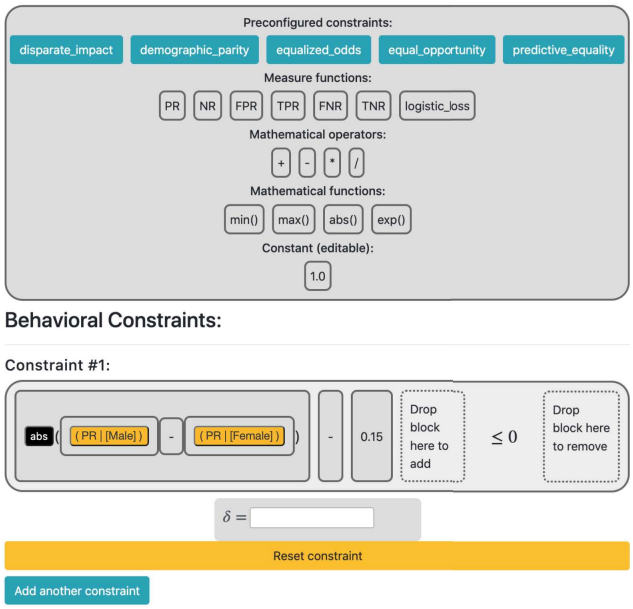

Fig. 1. The Seldonian Interface GUI (SIGUI) helps users specify safety and fairness guarantees by selecting and combining basic building blocks.

A. Seldonian Interface GUI (SIGUI)

The SIGUI is designed to help users specify safety and fairness *behavioral constraint requirements*. The form of these constraints can vary depending on the domain, and we designed the Seldonian Toolkit to be flexible and accept a wide range of such requirements. Many prominent definitions of fairness [40] can be defined as mathematical statements or inequalities. For example, a common desirable fairness definition, known as demographic parity, requires that a positive outcome be given to the same fraction of people of two protected classes. Assume, for instance, that an ML model is designed to decide who should get a loan while not discriminating based on race. In this case, the same fraction of applicants of each race should be given loans, up to some threshold. Not all types of safety constraints can be expressed as mathematical statements. Consider, for example, a user observing a robotic system performing an unwanted behavior, such as a chess robot breaking a child's finger [9]. The mathematical expression required to instruct the robot not to do this is hard to define analytically by non-expert users. The Seldonian Toolkit should, nonetheless, allow users to specify that this behavior is undesirable.

Suppose the user wishes to specify that an application must satisfy the demographic parity requirement. The toolkit's behavioral constraint specifications consist of two parts: (1) constraint strings and (2) confidence levels. Constraint strings are mathematical inequalities that must hold. For example, Figure 1 shows how the constraint string for demographic parity can be specified in the SIGUI. The user first selects "measure functions." These include metrics quantifying properties of a given ML model, such as its positive rate (PR), negative rate (NR), false-positive rate (FPR), etc. Then, the user combines the measure functions with mathematical functions (e.g., `max()`) and operators (e.g., ratio), and associates them with protected classes, such as gender. For example, in Figure 1, "$\mathrm{PR} \mid [\mathrm{Male}]$" means "positive rate for men." To specify demographic parity with respect to (binary) gender, the user specifies "$\mathrm{abs}\left((\mathrm{PR} \mid [\mathrm{Male}]) - (\mathrm{PR} \mid [\mathrm{Female}])\right) - 0.15 \leq 0$," which means "the absolute value of the difference between positive rates for men and women must be less than or equal to 0.15." The toolkit's probabilistic verification requires the user to also specify, for each constraint, the required confidence level, δ. This δ is the acceptable probability of violation of the constraint. The user could, for example, require the Seldonian Toolkit to produce an ML model with at least 99.9% confidence ($\delta = 0.001$) that it will satisfy the above-specified demographic parity constraint when applied to unseen data.

Safety constraints can be defined similarly to fairness constraints. For example, a behavioral constraint on an insulin pump ML-model update could require that the update cause no more instances of hypoglycemia than the original version [45], or that a new model's accuracy is strictly higher, with high confidence.

To simplify the constraint specification process, SIGUI allows the user to select from five commonly-used predefined fairness constraints, modify them, or build their own completely unique constraints using drag-and-drop building blocks. SIGUI also includes a tutorial to speed up the learning process.[6] The user can specify an unlimited number of behavioral constraint requirements for each model.

B. Seldonian Engine

The Seldonian Engine is the core library of the toolkit and implements a general-purpose Seldonian algorithm. Given a dataset D and a set of behavioral constraint requirements (recall Section II-A), a developer can use the Engine to train an ML model that is probabilistically guaranteed to satisfy all of the requirements. (Note that under certain conditions, such as when given insufficient data to train a model, or contradictory requirements, the Engine will explicitly state that no solution could be found, as further discussed below.)

While the Seldonian algorithm [45] is not a contribution of this work (the Toolkit is a usable implementation of a previously-published algorithm), we briefly describe it here. At a high level, the algorithm operates as follows. First, the available dataset D is partitioned into two disjoint sets: candidate data, D_{cand}, and safety data, D_{safety}. D_{cand} is provided as input to a component called `candidate selection`, which trains a single ML model that the algorithm plans to return. This model is called the *candidate solution*. The algorithm attempts to select a model that maximizes performance, while also satisfying the behavioral constraints, but, at this point, provides no guarantees. Next, the `safety test` component probabilistically verifies that the candidate solution satisfies the behavioral constraints. The `safety test` executes the

[6] https://seldonian-toolkit.github.io/GUI/build/html/index.html

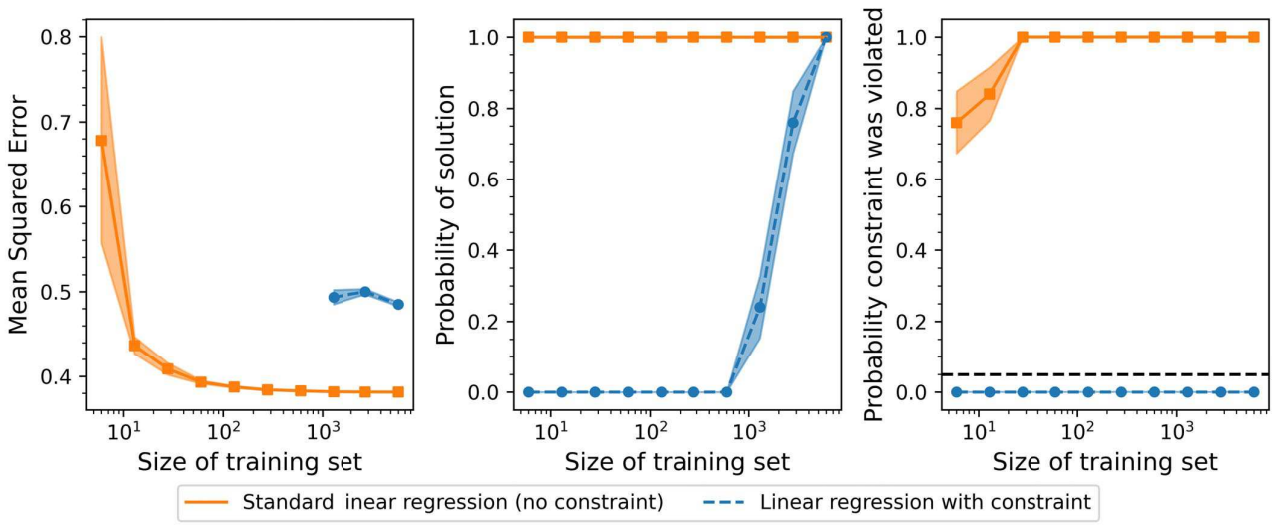

Fig. 2. The Seldonian Experiments Library can help software engineers evaluate trade-offs between using Seldonian and non-Seldonian ML algorithms. Here, models trained using the Seldonian Toolkit are less accurate than ones trained using standard linear regression, and require more data, but always satisfy a fairness requirement that the linear regression model fails.

solution on D_{safety} and checks if the constraints hold on that data. The size of D_{safety} is selected specifically to allow the use of probability bounds, such as Hoeffding's inequality, to estimate the likelihood that the requirement is satisfied not only on D_{safety}, but also on new, unseen data. If the bound satisfies the specified confidence level, the algorithm returns the candidate model. Otherwise, if the model fails the requirement or the algorithm's confidence is insufficient, the algorithm returns "No Solution Found." Our prior work has formally proven that the produced models are probabilistically verified to satisfy the provided safety and fairness constraints and empirically compared Seldonian algorithms to other fairness-aware ML algorithms [19], [31], [45].

To use the Seldonian Engine, the developer needs a set of behavioral constraint requirements (created by, or in coordination with a domain expert), a dataset, and instructions on which ML model the Engine should use internally. The Engine can use any (parametric) ML model, such as a deep neural network or a linear regression model. The Engine uses its implementation of the Seldonian algorithm to train and probabilistically verify the model, and either returns that model or "No Solution Found."

C. Seldonian Experiments Library

Developers and domain-expert users will understandably want to know how well their Seldonian algorithms perform compared to existing ML models that do not necessarily enforce safety or fairness. Satisfying behavioral constraints can involve trade-offs between enforcing safety and, for instance, runtime or accuracy (though our prior work showed that given real-world data, Seldonian algorithms often identify safe and fair solutions without a significant reduction in accuracy [17], [45]).

The Experiments Library helps the developer understand these trade-offs. Consider an engineer tasked with building a system that predicts housing prices within two zip codes. The developer has access to historical data describing how various features, including zip code, size, and the year when it was built, influence housing prices. The stakeholders (or, perhaps, the law) require that the system must have similar accuracy in both zip codes, perhaps because these zip codes correlate with the residents' races, and failing this requirement could lead to discrimination. The developer could use a standard ML model, e.g., linear regression implemented in scikit-learn [35], to achieve good overall accuracy in predicting house prices, but they might find that their predictions are more accurate in one zip code than the other.

The developer can input that scikit-learn model and dataset into the Seldonian Toolkit, along with the behavioral constraint from the stakeholders (formalized using SIGUI). The Engine will train a new model that satisfies the requirement, i.e., it will be similarly accurate in both zip codes. Then, the Experiments Library will produce a set of three diagnostic plots to help the developer understand the potential trade-offs.

We generated a synthetic dataset with a single feature (house size), a single sensitive attribute (zip code), and a single label (house price), and then used the Experiments Library on this dataset with a fairness constraint that enforces similar accuracy in the two zip codes. Figure 2 shows the resulting three plots for this house-price prediction problem. (While the Experiments Library currently makes these plots static, future work can explore if interactive features can help users better understand model behavior.) The left plot presents the accuracy of the two ML models (produced by scikit-learn and the Seldonian algorithm) in terms of mean squared error, as a function of the size of the dataset. The larger the mean squared error, the less accurate the model is in explaining the data. Here, the Seldonian model sacrifices accuracy to be able to satisfy the

fairness constraint with high confidence. Note that the plot shows no accuracy for datasets smaller than 10^3 data points, as we explain next. Here, we stress, again, that while we selected this example to showcase the possible trade-offs that may occur, prior research has shown that accuracy can often remain just as high as that of standard, unsafe ML algorithms when safety or fairness constraints are enforced [17], [45].

The middle plot shows how often the algorithm produces a verified fair model (as opposed to "No Solution Found"), as a function of the size of the dataset. While scikit-learn always produces a model (a very inaccurate one for small datasets), the Seldonian Engine is only able to produce models it can confidently say are fair once it has access to approximately 10^3 data points. (This is the reason the left plot did not contain accuracy information for small datasets). The amount of needed data is another important trade-off factor in choosing algorithms.

Finally, the right plot shows how often the trained models violate the behavioral constraint requirement when applied to new, unseen data. Scikit-learn's linear regression model always violates the fairness constraint, thus discriminating against one zip code. The Seldonian algorithm's model, by contrast, never violates the constraint.

Importantly, external ML models can be used in the toolkit, with little additional configuration. Because reimplementing ML models in a new framework, especially complex ones such as deep neural networks, can be prohibitively time consuming when designing software, the Seldonian Toolkit supports NumPy [20], SciPy [47], scikit-learn [35], and PyTorch models [34], and work is ongoing to add support for Keras, TensorFlow, and other popular ML libraries. Further, the Experiments Library incorporates Microsoft Fairlearn's [2] fairness-aware classification algorithms, and we similarly plan to add support for the Fairness Constraints [51] library. This allows developers to directly compare their Seldonian models to other methods that aim to improve fairness (but which do not provide similar guarantees).

III. Related Work

Recent research has argued that fairness is not only an important aspect of machine learning, but must also be addressed within the software engineering community, developing tools and technology to support software engineers in building fair systems [11]. Several frameworks exist to help data scientists evaluate models with respect to their fairness and other performance measures, including Fairlearn [10], Fairkit-learn [26], [27], AIFairness 360 [24], FairViz [12], FairML [1], and Fairway [13]. Like our Seldonian Toolkit, these frameworks help evaluate trade-offs in models but none provide methods for training provably fair nor safe ML models. Meanwhile, algorithms aimed to enforce fairness in ML models, e.g., [2], [51], or repair models to reduce bias, e.g., [22], [42], similarly do not provide the kinds of high-confidence guarantees the Seldonian Toolkit provides. In fact, two recent systematic literature reviews of research on testing models and systems for fairness [14] and mitigating bias [21], found that other than Seldonian algorithms [45], none provide high-confidence guarantees.

Formal verification of software systems using proof assistants, such as Coq [43] and Isabelle/HOL [32], allows proving software properties but, unlike our work, requires significant manual effort [28], [30]. Recent research has enabled fully automating formal verification by learning models of existing proofs for a set of systems and then using those models to generate proofs of new properties for new systems [3], [8], [15], [16], [18], [23], [25], [33], [38], [39], [50]. However, these approaches to not apply directly to machine learning models and, typically, require the system whose properties are being verified to be written in the language used by the underlying proof assistant.

Frameworks such as FairPrep [41] and FairRover [52] help data scientists follow best practices in software engineering and ML, focusing on responsible and ethical ML uses. Meanwhile, Google's What-If tool helps scientists analyze and understand ML models without writing code [49]. These tools are complementary to our work.

An important complement to building systems that enforce fairness is testing systems for discrimination [4], [17], [46]. When additional data become available, these tools can be used to audit models learned by the Seldonian Tooklit and by other methods.

Finally, Seldonian algorithms have been developed for contextual bandits [31], the setting where the training data and deployment data come from different distributions [19], and to enforce measures of long-term fairness [48], suggesting future extensions of the Seldonian Toolkit.

Acknowledgments

This work is supported by the National Science Foundation under grants no. CCF-1763423, CCF-2018372, and CCF-2210243, and by a gift from the Berkeley Existential Risk Initiative.

References

[1] Julius A. Adebayo. *FairML: ToolBox for diagnosing bias in predictive modeling*. PhD thesis, Massachusetts Institute of Technology, 2016.

[2] Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, John Langford, and Hanna Wallach. A reductions approach to fair classification. In *ICML*, volume PMLR 80, pages 60–69, Stockholm, Sweden, 2018.

[3] Arpan Agrawal, Emily First, Zhanna Kaufman, Tom Reichel, Shizhuo Zhang, Timothy Zhou, Alex Sanchez-Stern, Talia Ringer, and Yuriy Brun. Proofster: Automated formal verification. In *ICSE Demo*, May 2023.

[4] Rico Angell, Brittany Johnson, Yuriy Brun, and Alexandra Meliou. Themis: Automatically testing software for discrimination. In *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) Demonstration Track*, pages 871–875, Lake Buena Vista, FL, USA, November 2018.

[5] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine bias. *ProPublica*, May 2016.

[6] Kumar Arun, Garg Ishan, and Kaur Sanmeet. Loan approval prediction based on machine learning approach. *IOSR Journal of Computer Engineering*, 18(3):18–21, 2016.

[7] Isaac Asimov. *Foundation*. Gnome Press Publishers, New York, NY, USA, 1951.

[8] Kshitij Bansal, Sarah M. Loos, Markus N. Rabe, Christian Szegedy, and Stewart Wilcox. HOList: An environment for machine learning of higher order logic theorem proving. In *International Conference on Machine Learning (ICML)*, volume 97, pages 454–463, Long Beach, CA, USA, 2019. PMLR.

[9] Des Bieler. Chess-playing robot breaks finger of 7-year-old boy during match. *The Washington Post*, Jul 2022.

[10] Sarah Bird, Miro Dudík, Richard Edgar, Brandon Horn, Roman Lutz, Vanessa Milan, Mehrnoosh Sameki, Hanna Wallach, and Kathleen Walker. Fairlearn: A toolkit for assessing and improving fairness in AI. Technical Report MSR-TR-2020-32, 2020.

[11] Yuriy Brun and Alexandra Meliou. Software fairness. In *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) New Ideas and Emerging Results Track*, pages 754–759, Lake Buena Vista, FL, USA, November 2018.

[12] Ángel Alexander Cabrera, Will Epperson, Fred Hohman, Minsuk Kahng, Jamie Morgenstern, and Duen Horng Chau. FairVis: Visual analytics for discovering intersectional bias in machine learning. In *IEEE VAST*, pages 46–56, 2019.

[13] Joymallya Chakraborty, Suvodeep Majumder, Zhe Yu, and Tim Menzies. Fairway: A way to build fair ML software. In *ESEC/FSE*, pages 654–665, 2020.

[14] Zhenpeng Chen, Jie M. Zhang, Max Hort, Federica Sarro, and Mark Harman. Fairness testing: A comprehensive survey and analysis of trends. *CoRR*, abs/2207.10223, 2022.

[15] Emily First and Yuriy Brun. Diversity-driven automated formal verification. In *International Conference on Software Engineering (ICSE)*, pages 749–761, Pittsburgh, PA, USA, May 2022.

[16] Emily First, Yuriy Brun, and Arjun Guha. TacTok: Semantics-aware proof synthesis. *Proceedings of the ACM on Programming Languages (PACMPL) Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) issue*, 4:231:1–231:31, November 2020.

[17] Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. Fairness testing: Testing software for discrimination. In *ESEC/FSE*, pages 498–510, 2017.

[18] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. TacticToe: Learning to reason with HOL4 tactics. In *International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 46, pages 125–143, 2017.

[19] Stephen Giguere, Blossom Metevier, Yuriy Brun, Bruno Castro da Silva, Philip S. Thomas, and Scott Niekum. Fairness guarantees under demographic shift. In *ICLR*, 2022.

[20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[21] Max Hort, Zhenpeng Chen, Jie M. Zhang, Federica Sarro, and Mark Harman. Bias mitigation for machine learning classifiers: A comprehensive survey. *CoRR*, abs/2207.07068, 2022.

[22] Max Hort, Jie M. Zhang, Federica Sarro, and Mark Harman. Fairea: A model behaviour mutation approach to benchmarking bias mitigation methods. In *ESEC/FSE*, pages 994–1006, Athens, Greece, 2021.

[23] Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. GamePad: A learning environment for theorem proving. In *International Conference on Learning Representations (ICLR)*, 2019.

[24] IBM. AI Fairness 360. https://aif360.mybluemix.net, 2019.

[25] Albert Jiang, Konrad Czechowski, Mateja Jamnik, Piotr Milos, Szymon Tworkowski, Wenda Li, and Yuhuai Tony Wu. Thor: Wielding hammers to integrate language models and automated theorem provers. In *Neural Information Processing Systems (NeurIPS)*, New Orleans, LA, USA, 2022.

[26] Brittany Johnson, Jesse Bartola, Rico Angell, Sam Witty, Stephen J. Giguere, and Yuriy Brun. Fairkit, fairkit, on the wall, who's the fairest of them all? Supporting data scientists in training fair models. *CoRR*, abs/2012.09951, 2020.

[27] Brittany Johnson and Yuriy Brun. Fairkit-learn: A fairness evaluation and comparison toolkit. In *ICSE Demo*, pages 70–74, 2022.

[28] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. SeL4: Formal verification of an OS kernel. In *Symposium on Operating Systems Principles (SOSP)*, pages 207–220, Big Sky, MT, USA, 2009.

[29] Matthieu Komorowski, Leo A Celi, Omar Badawi, Anthony C Gordon, and A Aldo Faisal. The artificial intelligence clinician learns optimal treatment strategies for sepsis in intensive care. *Nature Medicine*, 24(11):1716–1720, 2018.

[30] Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM (CACM)*, 52(7):107–115, 2009.

[31] Blossom Metevier, Stephen Giguere, Sarah Brockman, Ari Kobren, Yuriy Brun, Emma Brunskill, and Philip S. Thomas. Offline contextual bandits with high probability fairness guarantees. In *NeurIPS*, pages 14893–14904, 2019.

[32] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: A proof assistant for higher-order logic*, volume 2283. Springer Science & Business Media, 2002.

[33] Aditya Paliwal, Sarah M. Loos, Markus N. Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. In *Conference on Artificial Intelligence (AAAI)*, pages 2967–2974, New York, NY, USA, 2020.

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035, 2019.

[35] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830, 2011.

[36] Casey Ross. IBM's Watson supercomputer recommended 'unsafe and incorrect' cancer treatments, internal documents show, 2018.

[37] Pradeep Kumar Roy, Sarabjeet Singh Chowdhary, and Rocky Bhatia. A machine learning approach for automation of resume recommendation system. *Procedia Computer Science*, 167:2318–2327, 2020.

[38] Alex Sanchez-Stern, Yousef Alhessi, Lawrence Saul, and Sorin Lerner. Generating correctness proofs with neural networks. In *Machine Learning in Programming Languages (MAPL)*, 2020.

[39] Alex Sanchez-Stern, Emily First, Timothy Zhou, Zhanna Kaufman, Yuriy Brun, and Talia Ringer. Passport: Improving automated formal verification using identifiers. *ACM TOPLAS*, 2023.

[40] Suchi Saria and Julia Rubin. Fairness definitions explained. *ACM/IEEE International Workshop on Software Fairness (FairWare)*, 2018.

[41] Sebastian Schelter, Yuxuan He, Jatin Khilnani, and Julia Stoyanovich. FairPrep: Promoting data to a first-class citizen in studies on fairness-enhancing interventions. In *EDBT*, 2020.

[42] Bing Sun, Jun Sun, Long H. Pham, and Jie Shi. Causality-based neural network repair. In *ICSE*, pages 338–349, Pittsburgh, PA, USA, 2022.

[43] The Coq Development Team. Coq, v.8.7. https://coq.inria.fr, 2017.

[44] Georgios Theocharous, Philip S. Thomas, and Mohammad Ghavamzadeh. Personalized ad recommendation systems for life-time value optimization with guarantees. In *IJCAI*, pages 1806–1812, 2015.

[45] Philip S. Thomas, Bruno Castro da Silva, Andrew G. Barto, Stephen Giguere, Yuriy Brun, and Emma Brunskill. Preventing undesirable behavior of intelligent machines. *Science*, 366(6468):999–1004, 22 November 2019.

[46] Florian Tramer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, Jean-Pierre Hubaux, Mathias Humbert, Ari Juels, and Huang Lin. FairTest: Discovering unwarranted associations in data-driven applications. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, Paris, France, April 2017.

[47] Pauli Virtanen et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[48] Aline Weber, Blossom Metevier, Yuriy Brun, Philip S. Thomas, and Bruno Castro da Silva. Enforcing delayed-impact fairness guarantees. *CoRR*, abs/2208.11744, 2020.

[49] James Wexler. The What-If tool: Code-free probing of machine learning models. https://ai.googleblog.com/2018/09/the-what-if-tool-code-free-probing-of.html, 2018.

[50] Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning (ICML)*, Long Beach, CA, USA, 2019.

[51] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna P. Gummadi. Fairness constraints: Mechanisms for fair classification. In *FAT ML*, Lille, France, 2015.

[52] Hantian Zhang, Nima Shahbazi, Xu Chu, and Abolfazl Asudeh. FairRover: Explorative model building for fair and responsible machine learning. In *Workshop on Data Management for End-To-End Machine Learning*, 2021.