

# Deckard: A Declarative Tool for Machine Learning Robustness Evaluations

Charles Meyers  <sup>1\*</sup>

<sup>1</sup> Department of Computing Science, Umeå University, Umeå  \* These authors contributed equally.

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

## Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

## Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#))

## Summary

The software package presented, called deckard, is a modular software toolkit designed to streamline and standardize experimentation in machine learning (ML) with a particular focus on the adversarial scenario. It provides a flexible, extensible framework for defining, executing, and analyzing end-to-end ML pipelines in the context of a malicious actor. As it is built on top of the Hydra configuration system, deckard supports declarative YAML-based configuration of data preprocessing, model training, and adversarial attack pipelines, enabling reproducible, framework-agnostic experimentation across diverse ML settings.

In addition to configuration management, deckard includes a suite of utilities for distributed and parallel execution, automated hyperparameter optimisation, visualisation, and result aggregation. The tooling abstracts away much of the engineering overhead typically involved in adversarial ML research, allowing researchers to focus on algorithmic insights rather than implementation details. The presented software facilitates rigorous benchmarking by maintaining an auditable trace of configurations, random seeds, and intermediate outputs throughout the experimental lifecycle.

The system is compatible with a variety of ML frameworks and several classes of adversarial attacks, making it a suitable backend for both large-scale automated testing and fine-grained empirical analysis. By providing a unified interface for experimental control, deckard accelerates the development and evaluation of robust models, and helps close the gap between research prototypes and verifiable, reproducible results.

## Statement of need

While tools such as mlflow ([Zaharia et al., 2018](#)), Weights & Biases ([Biewald, 2020](#)), optuna ([Akiba et al., 2019](#)), and Kubernetes ([Kubernetes, 2019](#)) provide essential infrastructure for model tracking and experiment management, deckard occupies a different position in the ML ecosystem—focusing specifically on configurable, adversarially robust experimentation.

Unlike MLflow and Weights & Biases, which emphasize logging, visualization, and reproducibility for various ML frameworks, deckard enforces reproducibility by construction through its declarative, YAML-driven configuration system built on Facebook's hydra ([Yadan, 2019](#)) configuration management tool. In contrast to cloud-management software like Kubernetes—which is a general-purpose container orchestration platform—deckard abstracts away orchestration details and offers native support for parallel and distributed experimentation, tailored to ML workflows involving attack/defense cycles, model retraining, or optimisation. While deckard integrates tightly with IBM's Adversarial Robustness Toolbox ([Nicolae et al., 2018](#)), the software is designed to be easily extensible to other attack frameworks. The human- and machine-readable parameter configuration system allows researchers to declaratively define end-to-end pipelines that span data sampling, preprocessing, model training, attack generation,

41 defense evaluation, multi-objective optimisation, and visualisation. Tools like ray (Moritz et  
42 al., 2018), optuna (Akiba et al., 2019), or nevergrad (Bennet et al., 2021) offer components  
43 of this pipeline (e.g., hyperparameter search or configuration management), but lack unified  
44 support for adversarial ML, verification, or auditability at scale. While deckard complements  
45 these existing tools, and in many cases can be integrated with them, its primary contribution  
46 is in automating and verifying adversarial ML experiments in a way that is both extensible and  
47 framework-agnostic.

## 48 Usage

49 Various versions of this software have been used in several recently published and not-yet-  
50 published works by the author of this paper, all of which are available in the examples  
51 folder in the source code repository [here](#). One published work, now reproducible via the  
52 examples/attack\_defence\_survey folder, includes a large survey of attacks and defences  
53 against canonical datasets and models (C. Meyers et al., 2023). Another work analysed the  
54 run-time requirements of attacks against a particular model before and after retraining against  
55 those attacks (C. Meyers, Löfstedt, et al., 2024) (reproducible via examples/retraining). The  
56 next paper formalised a method for estimating the time-to-failure of a given model against  
57 a suite of attacks and introduce a metric that quantifies the ratio of attack and training  
58 cost (Meyers et al., 2023) (reproducible via examples/survival\_heuristic). Furthermore, a  
59 not yet published work uses this time-to-failure model as a mechanism for analysing the cost  
60 efficacy of various hardware choices in the context of adversarial attacks (reproducible via  
61 examples/power) (C. Meyers, Sedghpour, et al., 2024). Another work exploits the tooling to  
62 train a custom model that is designed to run client-side by using compression algorithms to  
63 measure the distance between text (reproducible via examples/compression).

## 64 Experiment Management

65 Typically ML projects are composed of long and complex pipelines that are highly dependent on  
66 a number of parameters that must be configured by either the model builder or attacker. Due  
67 to the large scale and cost associated with training popular models (e.g., neural networks), it is  
68 often necessary to tune a model using hundreds or thousands of individual model configurations  
69 (often called *hyper-parameters*). In addition, even more simple models are often part of various  
70 long and complex data and software pipelines. In short, the typical ML model has a very large  
71 number of hyper-parameters that must be configured, managed, and evaluated. Generally,  
72 one of many benchmark datasets is first sampled, then preprocessed, sent to a model, with  
73 optional pre- and post-processing defences, and then scored according to some chosen metric  
74 which may include the performance against any number of adversarial attacks. Each stage in  
75 this example pipeline might include tens or hundreds of possible sets of hyper-parameters that  
76 must be exhaustively tested. Furthermore, this problem scales drastically as we include more  
77 and more stages in a pipeline since each additional stage introduces a new combinatorial layer  
78 of complexity, rapidly expanding the total number of potential configurations that must be  
79 evaluated for robustness and be reproducible for posterity. Not only does deckard provide a  
80 standard way to document and configure these hyper-parameters, it gives each experiment an  
81 auditable identifier.

## 82 Reproducibility and Auditability

83 For ML, various regulatory and legal frameworks govern safety (*IEC 61508 Safety and Functional*  
84 *Safety*, 2010; *IEC 62304 Medical Device Software - Software Life Cycle Processes*, 2006; *ISO*  
85 *26262-1*, 2018; *The Parliament of the European Union*, 2024), privacy [The Parliament of the  
86 European Union (2024); European Parliament & Council of the European Union (2016); Legisla-  
87 ture of the United States (1996); Legislature of the United States (1998);] and/or transparency

(The Legislature of California, 2024; The Parliament of the European Union, 2024). The software package presented here provides a machine- and human-readable format for creating reproducible and auditable experiments as required by various regulations. In addition, several examples connected to both published and not-yet-published work live in the examples folder in the repository, allowing for easy reproducibility of several extensive sets of experiments across several popular ML software frameworks. The power example provides a reproducible way to run a suite of adversarial tests using popular cloud-based platforms and the retraining and survival\_heuristic examples provide examples of both CPU and GPU-based parallelisation, respectively.

The basics subfolder provides a minimum working example for each of the supported ML frameworks: tensorflow (Abadi et al., 2015), pytorch (Paszke et al., 1912), scikit-learn (Pedregosa et al., 2011), and keras (Chollet, 2015). The basics folder also provides examples of various classes of adversarial examples: *poisoning* attacks that change model behaviour by injecting data during training (Biggio et al., 2012), *inference* attacks (Li & Zhang, 2021) that attempt to reverse engineer properties of the training data, *extraction* attacks that attempt to reverse engineer the model (Jagielski et al., 2020), and *evasion* attacks that induce errors of classification during run-time (C. Meyers et al., 2023). The parameters file for each experiment ensures that a given pipeline can be reproduced and the standardised format allows us to derive a hash value that is hard to forge but easy to verify. Not only does this hash serve as an identifier to track the state of an experiment, but also serves as a way to audit the parameters file for tampering. Likewise, by using dvc (DVC Authors, 2023) to track any input or output files specified in the parameters file, the software associates each score file with a identifier that is easy to track and verify, but hard to forge—ensuring that forged or modified results are easy to spot in version-controlled experiment repository.

## Parallel and Distributed Design

Since ML projects can exploit specialized hardware such as multi-core processors or GPUs, and often rely on clusters of machines for large-scale data processing, it was necessary to enable parallel and distributed experiment execution and model optimization. By leveraging the hydra configuration framework, deckard automatically supports optimization libraries like nevergrad (Bennet et al., 2021), Adaptive Experimentation (A. Developers, 2025), and optuna (Akiba et al., 2019), making the software modular and extensible. Additionally, experiments can be managed using a variety of popular job schedulers, including Ray (Moritz et al., 2018), Redis Queue (Stamps, 2025), and slurm (Yoo et al., 2003) for distributed jobs or joblib (J. Developers, 2025) for jobs on a single machine.

By using a declarative design, a given set of experiments can be specified once and executed seamlessly across different backends without modification to the underlying codebase. This makes deckard both adaptable and scalable, suitable for use on personal laptops, multi-node servers, or large-scale, high-performance clusters. When configured appropriately, experiment batches can be parallelized, enabling massive parameter sweeps, ensemble evaluations, or adversarial robustness tests to be executed in parallel—reducing turnaround time while maintaining strong guarantees on reproducibility and auditability. The design of the presented software prioritizes clarity and maintainability by capturing each experimental configuration as a YAML artifact, making both successful and failed runs equally traceable and shareable. This approach transforms experiment tracking from an afterthought into a first-class component of the trustworthy ML workflow.

## Funding

Financial support has been provided in part by the Knut and Alice Wallenberg Foundation grant number 2019.0352 and by the eSSSENCE Programme under the Swedish Government's

136 Strategic Research Initiative.

## 137 Acknowledgements

138 The author would like to thank Aaron MacSween, Abel Souza, and Mohammad Saledghpour  
139 Reza for their guidance in software design principles. In particular, the author appreciates  
140 Mohammad's code and documentation regarding cloud-based and other Kubernetes deploy-  
141 ments. The author would like to thanks his advisors, Erik Elmroth and Tommy Löfstedt for  
142 their research expertise, funding, and patience.

## 143 References

- 144 Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis,  
145 A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia,  
146 Y., Jozefowicz, R., Kaiser, L., Kudlur, M., ... Zheng, X. (2015). *TensorFlow: Large-scale*  
147 *machine learning on heterogeneous systems*. <https://www.tensorflow.org/>
- 148 Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A next-generation  
149 hyperparameter optimization framework. *Proceedings of the 25th ACM SIGKDD Interna-*  
150 *tional Conference on Knowledge Discovery & Data Mining*, 2623–2631.
- 151 Bennet, P., Doerr, C., Moreau, A., Rapin, J., Teytaud, F., & Teytaud, O. (2021). Nevergrad:  
152 Black-box optimization platform. *ACM SIGEVOlution*, 14(1), 8–15.
- 153 Biewald, L. (2020). *Experiment tracking with weights and biases*. <https://www.wandb.com/>
- 154 Biggio, B., Nelson, B., & Laskov, P. (2012). Poisoning attacks against support vector machines.  
155 *International Conference on Machine Learning*.
- 156 Chollet, F. (2015). Keras. In *GitHub repository*. <https://github.com/fchollet/keras>; GitHub.
- 157 Developers, A. (2025). AX: Adaptive experimentation platform. [https://ax.readthedocs.io/en/](https://ax.readthedocs.io/en/stable/#)  
158 [stable/#](https://ax.readthedocs.io/en/stable/#)
- 159 Developers, J. (2025). Joblib: Running python functions as pipeline jobs. [https://joblib.](https://joblib.readthedocs.io/en/stable/)  
160 [readthedocs.io/en/stable/](https://joblib.readthedocs.io/en/stable/)
- 161 DVC Authors. (2023). DVC–Data Version Control. Github. [https://github.com/iterative/dvc.](https://github.com/iterative/dvc.org)  
162 [org](https://github.com/iterative/dvc.org)
- 163 European Parliament, & Council of the European Union. (2016, May 4). *Regulation (EU)*  
164 *2016/679 of the European Parliament and of the Council. Of 27 April 2016 on the*  
165 *protection of natural persons with regard to the processing of personal data and on the*  
166 *free movement of such data, and repealing Directive 95/46/EC (General Data Protection*  
167 *Regulation)*. <https://data.europa.eu/eli/reg/2016/679/oj>
- 168 IEC 61508 safety and functional safety (2nd ed.). (2010). (2nd ed.) [Computer software].  
169 International Electrotechnical Commission.
- 170 IEC 62304 medical device software - software life cycle processes (2nd ed.). (2006). (2nd ed.)  
171 [Computer software]. International Electrotechnical Commission.
- 172 ISO 26262-1:2011, road vehicles — functional safety. (2018). [https://www.iso.org/standard/](https://www.iso.org/standard/43464.html)  
173 [43464.html](https://www.iso.org/standard/43464.html) (visited 2022-04-20).
- 174 Jagielski, M., Carlini, N., Berthelot, D., Kurakin, A., & Papernot, N. (2020). High accuracy  
175 and high fidelity extraction of neural networks. *29th USENIX Security Symposium (USENIX*  
176 *Security 20)*, 1345–1362.
- 177 Kubernetes. (2019). *Kubernetes—an open source system for managing containerized applica-*

- 178 tions. Github. <https://github.com/kubernetes/kubernetes>
- 179 Legislature of the United States. (1996). *Health insurance portability and accountability act*.
- 180 Legislature of the United States. (1998). *Children's online privacy protection act*.
- 181 Li, Z., & Zhang, Y. (2021). Membership leakage in label-only exposures. *Proceedings of the*  
182 *2021 ACM SIGSAC Conference on Computer and Communications Security*, 880–895.
- 183 Meyers, C., Löfstedt, T., & Elmroth, E. (2023). Safety-critical computer vision: An empirical  
184 survey of adversarial evasion attacks and defenses on computer vision systems. *Artificial*  
185 *Intelligence Review*, 1–35.
- 186 Meyers, C., Löfstedt, T., & Elmroth, E. (2024). Massively parallel evasion attacks and the  
187 pitfalls of adversarial retraining. *EAI Endorsed Transactions on Internet of Things*, 10.
- 188 Meyers, C., Sedghpour, M. R. S., Löfstedt, T., & Elmroth, E. (2024). *A training rate and*  
189 *survival heuristic for inference and robustness evaluation (TRASHFIRE)*. [https://arxiv.org/](https://arxiv.org/abs/2401.13751)  
190 [abs/2401.13751](https://arxiv.org/abs/2401.13751)
- 191 Meyers, Reza, Löfstedt, & Elmroth. (2023). A systematic approach to robustness modelling.  
192 *Springer Artificial Intelligence Review*.
- 193 Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z.,  
194 Paul, W., Jordan, M. I., & others. (2018). Ray: A distributed framework for emerging {AI}  
195 applications. *13th USENIX Symposium on Operating Systems Design and Implementation*  
196 *(OSDI 18)*, 561–577.
- 197 Nicolae, M.-I., Sinn, M., Tran, M. N., Buesser, B., Rawat, A., Wistuba, M., Zantedeschi, V.,  
198 Baracaldo, N., Chen, B., Ludwig, H., & others. (2018). Adversarial robustness toolbox v1.  
199 0.0. *arXiv Preprint arXiv:1807.01069*.
- 200 Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin,  
201 Z., Gimelshein, N., Antiga, L., & others. (2019). Pytorch: An imperative style, high-  
202 performance deep learning library. *arXiv 2019. arXiv Preprint arXiv:1912.01703*, 10.
- 203 Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M.,  
204 Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D.,  
205 Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python.  
206 *Journal of Machine Learning Research*, 12, 2825–2830.
- 207 Stamps. (2025). *RQ: Easy job queues for python*. Stamps, an Indonesian CRM company.  
208 <https://python-rq.org/>
- 209 The Legislature of California. (2024). *AB-2013 generative artificial intelligence: Training data*  
210 *transparency*.
- 211 The Parliament of the European Union. (2024). *High-level summary of the AI act*.
- 212 Yadan, O. (2019). *Hydra – a framework for elegantly configuring complex applications*. Github.  
213 <https://github.com/facebookresearch/hydra>
- 214 Yoo, A. B., Jette, M. A., & Grondona, M. (2003). Slurm: Simple linux utility for resource  
215 management. *Workshop on Job Scheduling Strategies for Parallel Processing*, 44–60.
- 216 Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S.,  
217 Nykodym, T., Ogilvie, P., Parkhe, M., & others. (2018). Accelerating the machine learning  
218 lifecycle with MLflow. *IEEE Data Eng. Bull.*, 41(4), 39–45.