Deckard: A Declarative Tool for Machine Learning Robustness Evaluations

Charles Meyers [1]

1 Department of Computing Science, Umeå University, Umeå ROR

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,

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

Usage

Various versions of this software have been used in several recently published and not-yet-published works by the author of this paper, all of which are available in the examples folder in the source code repository https://github.com/simplymathematics/deckard. One published work, now reproducible via the examples/attack_defence_survey folder, includes a large survey of attacks and defences against canonical datasets and models ([C. Meyers et al., 2023](#)). Another work analysed the run-time requirements of attacks against a particular model before and after retraining against those attacks ([C. Meyers, Löfstedt, et al., 2024](#)) (reproducible via examples/retraining). The next paper formalised a method for estimating the time-to-failure of a given model against a suite of attacks and introduce a metric that quantifies the ratio of attack and training cost ([Meyers et al., 2023](#)) (reproducible via examples/survival_heuristic). Furthemore, a not yet published work uses this time-to-failure model as a mechanism for analysing the cost efficacy of various hardware choices in the context of adversarial attacks (reproducible via examples/power) ([C. Meyers, Sedghpour, et al., 2024](#)). Another work exploits the tooling to train a custom model that is designed to run client-side by using compression algorithms to measure the distance between text (reproducible via examples/compression).

Experiment Management

Typically ML projects are composed of long and complex pipelines that are highly dependent on a number of parameters that must be configured by either the model builder or attacker. Due to the large scale and cost associated with training ML models, it is often necessary to tune a model using many indivudal model configurations (often called *hyper-paremeters*). To determine adversarial robustness, one of many benchmark datasets is first sampled, then preprocessed, sent to a model, with optional pre- and post-processing defences, and then scored according to some chosen metric which may include the performance against any number of adversarial attacks. Each stage in this example pipeline might include tens or hundreds of possible sets of hyper-parameters that must be exhaustively tested. Furthermore, this problem scales drastically as we include more and more stages in a pipeline since each additional stage introduces a new combinatorial layer of complexity, rapidly expanding the total number of potential configurations that must be evaluated for robustness and be reproducible for posterity. Not only does deckard provide a standard way to document and configure these hyper-parameters, it gives each experiment an auditable identifier.

Reproducibility and Auditability

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

88 published and not-yet-published work live in the examples folder in the repository, allowing for
89 easy reproducibility of several extensive sets of experiments across several popular ML software
90 frameworks. The power example provides a reproducible way to run a suite of adversarial tests
91 using popular cloud-based platforms and the retraining and survival_heuristic examples
92 provide examples of both CPU and GPU-based parallelisation, respectively.

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

Parallel and Distributed Design

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

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

Funding

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., … Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems*. https://www.tensorflow.org/

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2623–2631.

Bennet, P., Doerr, C., Moreau, A., Rapin, J., Teytaud, F., & Teytaud, O. (2021). Nevergrad: Black-box optimization platform. *ACM SIGEVOlution*, *14*(1), 8–15.

Biewald, L. (2020). *Experiment tracking with weights and biases*. https://www.wandb.com/

Biggio, B., Nelson, B., & Laskov, P. (2012). Poisoning attacks against support vector machines. *International Conference on Machine Learning*.

Chollet, F. (2015). Keras. In *GitHub repository*. https://github.com/fchollet/keras; GitHub.

Developers, A. (2025). *AX: Adaptive experimentation platform*. https://ax.readthedocs.io/en/stable/#

Developers, J. (2025). *Joblib: Running python functions as pipeline jobs*. https://joblib.readthedocs.io/en/stable/

DVC Authors. (2023). *DVC–Data Version Control*. Github. https://github.com/iterative/dvc.org

European Parliament, & Council of the European Union. (2016, May 4). *Regulation (EU) 2016/679 of the European Parliament and of the Council. Of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. https://data.europa.eu/eli/reg/2016/679/oj

IEC 61508 safety and functional safety (2nd ed.). (2010). (2nd ed.) [Computer software]. International Electrotechnical Commission.

IEC 62304 medical device software - software life cycle processes (2nd ed.). (2006). (2nd ed.) [Computer software]. International Electrotechnical Commission.

ISO 26262-1:2011, road vehicles — functional safety. (2018). https://www.iso.org/standard/43464.html (visited 2022-04-20).

Jagielski, M., Carlini, N., Berthelot, D., Kurakin, A., & Papernot, N. (2020). High accuracy and high fidelity extraction of neural networks. *29th USENIX Security Symposium (USENIX Security 20)*, 1345–1362.

Kubernetes. (2019). *Kubernetes–an open source system for managing containerized applications*. Github. https://github.com/kubernetes/kubernetes

Legislature of the United States. (1996). *Health insurance portability and accountability act*.

176 Legislature of the United States. (1998). *Children's online privacy protection act*.

177 Li, Z., & Zhang, Y. (2021). Membership leakage in label-only exposures. *Proceedings of the*
178   *2021 ACM SIGSAC Conference on Computer and Communications Security*, 880–895.

179 Meyers, C., Löfstedt, T., & Elmroth, E. (2023). Safety-critical computer vision: An empirical
180   survey of adversarial evasion attacks and defenses on computer vision systems. *Artificial*
181   *Intelligence Review*, 1–35.

182 Meyers, C., Löfstedt, T., & Elmroth, E. (2024). Massively parallel evasion attacks and the
183   pitfalls of adversarial retraining. *EAI Endorsed Transactions on Internet of Things*, *10*.

184 Meyers, C., Sedghpour, M. R. S., Löfstedt, T., & Elmroth, E. (2024). *A training rate and*
185   *survival heuristic for inference and robustness evaluation (TRASHFIRE)*. https://arxiv.org/
186   abs/2401.13751

187 Meyers, Reza, Löfstedt, & Elmroth. (2023). A systematic approach to robustness modelling.
188   *Springer Artificial Intelligence Review*.

189 Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z.,
190   Paul, W., Jordan, M. I., & others. (2018). Ray: A distributed framework for emerging {AI}
191   applications. *13th USENIX Symposium on Operating Systems Design and Implementation*
192   *(OSDI 18)*, 561–577.

193 Nicolae, M.-I., Sinn, M., Tran, M. N., Buesser, B., Rawat, A., Wistuba, M., Zantedeschi, V.,
194   Baracaldo, N., Chen, B., Ludwig, H., & others. (2018). Adversarial robustness toolbox v1.
195   0.0. *arXiv Preprint arXiv:1807.01069*.

196 Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin,
197   Z., Gimelshein, N., Antiga, L., & others. (1912). Pytorch: An imperative style, high-
198   performance deep learning library. arXiv 2019. *arXiv Preprint arXiv:1912.01703*, *10*.

199 Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M.,
200   Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D.,
201   Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python.
202   *Journal of Machine Learning Research*, *12*, 2825–2830.

203 Stamps. (2025). *RQ: Easy job queues for python*. Stamps, an Indonesian CRM company.
204   https://python-rq.org/

205 The Legislature of California. (2024). *AB-2013 generative artificial intelligence: Training data*
206   *transparency*.

207 The Parliament of the European Union. (2024). *High-level summary of the AI act*.

208 Yadan, O. (2019). *Hydra – a framework for elegantly configuring complex applications*. Github.
209   https://github.com/facebookresearch/hydra

210 Yoo, A. B., Jette, M. A., & Grondona, M. (2003). Slurm: Simple linux utility for resource
211   management. *Workshop on Job Scheduling Strategies for Parallel Processing*, 44–60.

212 Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S.,
213   Nykodym, T., Ogilvie, P., Parkhe, M., & others. (2018). Accelerating the machine learning
214   lifecycle with MLflow. *IEEE Data Eng. Bull.*, *41*(4), 39–45.