deckard: A Declarative Tool for Machine Learning Robustness Evaluations

15 April 2025

Summary

The software package presented, called `deckard`, is a modular toolkit that streamlines and standardizes machine learning (ML) experimentation, with a particular emphasis on evaluating models under worst-case perturbations(Goodfellow, Shlens, and Szegedy 2014). It offers a flexible, extensible framework for defining, running, and analysing end-to-end ML pipelines in settings where inputs may be manipulated by an attacker or corrupted by unexpected noise (Biggio et al. 2013).

Built on the Hydra configuration system (Yadan 2019), `deckard` supports declarative YAML-based configuration for data preprocessing, model training, and adversarial attack pipelines. This makes it easy to run reproducible, framework-agnostic experiments across a wide range of ML domains.

Beyond configuration management, `deckard` provides tools for distributed and parallel execution, automated hyperparameter optimization, visualization, and result aggregation. By handling much of the engineering overhead common in adversarial ML research, it lets researchers focus on developing ideas rather than juggling infrastructure. The system also ensures rigorous benchmarking by keeping an auditable record of configurations, seeds, and intermediate outputs throughout each experiment.

Compatible with multiple ML frameworks and a variety of attack methods, `deckard` works both as a backend for large-scale automated testing and as a platform for detailed empirical analysis. With its unified interface for experiment control, the toolkit accelerates the development of robust models and helps bridge the gap between exploratory research and reproducible, trustworthy results .

To avoid any ambiguity about contributions, we clarify that `deckard` is a method and software system introduced in this work. Although the implementation is publicly available on GitHub, the modular design, configuration-driven framework, and experimental methodology are original to this paper. While the prior works outlined in the Usage Section can be fully reproduced with this software, this paper is the first to present the system as a cohesive method, fully specified, documented, and intended for broad reuse.

Statement of need

The software package presented, called `deckard`, is a modular toolkit that streamlines and standardizes machine learning (ML) experimentation, with a particular emphasis on evaluating models under worst-case perturbations (Goodfellow, Shlens, and Szegedy 2014). It offers a flexible, extensible framework for defining, running, and analysing end-to-end ML pipelines in settings where inputs may be manipulated by an attacker or corrupted by unexpected noise (Biggio et al. 2013).

Built on the Hydra configuration system (Yadan 2019), `deckard` supports declarative YAML-based configuration for data preprocessing, model training, and adversarial attack pipelines. This makes it easy to run reproducible, framework-agnostic experiments across a wide range of ML domains.

Beyond configuration management, `deckard` provides tools for distributed and parallel execution, automated hyperparameter optimization, visualization, and result aggregation. By handling much of the engineering overhead common in adversarial ML research, it lets researchers focus on developing ideas rather than

juggling infrastructure. The system also ensures rigorous benchmarking by keeping an auditable record of configurations, seeds, and intermediate outputs throughout each experiment.

Compatible with multiple ML frameworks and a variety of attack methods, `deckard` works both as a backend for large-scale automated testing and as a platform for detailed empirical analysis. With its unified interface for experiment control, the toolkit accelerates the development of robust models and helps bridge the gap between exploratory research and reproducible, trustworthy results .

To avoid any ambiguity about contributions, we clarify that `deckard` is a method and software system introduced in this work. Although the implementation is publicly available on GitHub, the modular design, configuration-driven framework, and experimental methodology are original to this paper. While the prior works outlined in the Usage Section can be fully reproduced with this software, this paper is the first to present the system as a cohesive method, fully specified, documented, and intended for broad reuse.

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, Löfstedt, and Elmroth 2023). Another work analysed the run-time requirements of attacks against a particular model before and after retraining against those attacks (Charles Meyers and Elmroth 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 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-parameters*). 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 (The Parliament of the European Union 2024; "ISO 26262-1:2011, Road Vehicles — Functional Safety" 2018; *IEC 61508 Safety and Functional Safety* 2010; *IEC 62304 Medical Device Software - Software Life Cycle Processes* 2006), privacy (The Parliament of the European Union 2024; European Parliament and Council of the European Union 2016; Legislature of the United States 1996, 1998) and/or transparency (The Parliament of the European Union 2024; The

Legislature of California 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, Nelson, and Laskov 2012), *inference* attacks (Li and 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, Löfstedt, and Elmroth 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, Jette, and Grondona 2003) for distributd 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 eSSENCE Programme under the Swedish Government's Strategic Research Initiative.

Acknowledgements

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

References

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, et al. 2015. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems."

Akiba, Takuya, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. "Optuna: A Next-Generation Hyperparameter Optimization Framework." In *Proceedings of the 25th ACM SIGKDD*, 2623–31.

Bennet, Pauline, Carola Doerr, Antoine Moreau, Jeremy Rapin, Fabien Teytaud, and Olivier Teytaud. 2021. "Nevergrad: Black-Box Optimization Platform." *ACM SIGEVOlution* 14 (1): 8–15.

Biggio, Battista, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. "Evasion Attacks Against Machine Learning at Test Time," 387–402.

Biggio, Battista, Blaine Nelson, and Pavel Laskov. 2012. "Poisoning Attacks Against Support Vector Machines." *International Conference on Machine Learning.*

Charles Meyers, Tommy Löfstedt, and Erik Elmroth. 2024. "Massively Parallel Evasion Attacks and the Pitfalls of Adversarial Retraining." *EAI Endorsed Transactions on Internet of Things* 10.

Chollet, François. 2015. "Keras." *GitHub Repository.* https://github.com/fchollet/keras; GitHub.

Developers, Ax. 2025. "AX: Adaptive Experimentation Platform." https://ax.readthedocs.io/en/stable/#.

Developers, Joblib. 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, and Council of the European Union. 2016. "Regulation (EU) 2016/679 of the European Parliament and of the Council." May 4, 2016. https://data.europa.eu/eli/reg/2016/679/oj.

Goodfellow, Ian J, Jonathon Shlens, and Christian Szegedy. 2014. "Explaining and Harnessing Adversarial Examples." *arXiv:1412.6572.*

IEC 61508 Safety and Functional Safety. 2010. 2nd ed. International Electrotechnical Commission.

IEC 62304 Medical Device Software - Software Life Cycle Processes. 2006. 2nd ed. International Electrotechnical Commission.

"ISO 26262-1:2011, Road Vehicles — Functional Safety." 2018. https://www.iso.org/standard/43464.html (visited 2022-04-20).

Jagielski, Matthew, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. 2020. "High Accuracy and High Fidelity Extraction of Neural Networks." In *29th USENIX Security Symposium (USENIX Security 20)*, 1345–62.

Legislature of the United States. 1996. "Health Insurance Portability and Accountability Act."

———. 1998. "Children's Online Privacy Protection Act."

Li, Zheng, and Yang Zhang. 2021. "Membership Leakage in Label-Only Exposures." In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 880–95.

Meyers, Charles, Tommy Löfstedt, and Erik Elmroth. 2023. "Safety-Critical Computer Vision: An Empirical Survey of Adversarial Evasion Attacks and Defenses on Computer Vision Systems." *Artificial Intelligence Review* 56 (Suppl 1): 217–51.

Meyers, Charles, Mohammad Reza Saleh Sedghpour, Tommy Löfstedt, and Erik Elmroth. 2024. "A Training Rate and Survival Heuristic for Inference and Robustness Evaluation (TRASHFIRE)." https://arxiv.org/abs/2401.13751.

Meyers, Reza, Löfstedt, and Elmroth. 2023. "A Systematic Approach to Robustness Modelling." *Springer Artificial Intelligence Review.*

Moritz, Philipp, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, et al. 2018. "Ray: A Distributed Framework for Emerging {AI} Applications." In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 561–77.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, et al. 1912. "Pytorch: An Imperative Style, High-Performance Deep Learning Library. arXiv 2019." *arXiv Preprint arXiv:1912.01703* 10.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. 2011.

“Scikit-Learn: Machine Learning in Python.” *Journal of Machine Learning Research* 12: 2825–30.

Stamps. 2025. “RQ: Easy Job Queues for Python.” Stamps, an Indonesian CRM company. https://python-rq.org/.

The Legislature of California. 2024. “AB-2013 Generative Artificial Intelligence: Training Data Transparency.”

The Parliament of the European Union. 2024. “High-Level Summary of the AI Act.”

Yadan, Omry. 2019. “Hydra – a Framework for Elegantly Configuring Complex Applications.” Github. https://github.com/facebookresearch/hydra.

Yoo, Andy B, Morris A Jette, and Mark Grondona. 2003. “Slurm: Simple Linux Utility for Resource Management.” In *Workshop on Job Scheduling Strategies for Parallel Processing*, 44–60. Springer.