




Imitation Learning Datasets: A Toolkit For Creating Datasets, Training Agents and Benchmarking

Demonstration Track

Nathan Gavenski 
King's College London
London, United Kingdom
nathan.schneider_gavenski@kcl.ac.uk

Michael Luck 
University of Sussex
Sussex, United Kingdom
michael.luck@sussex.ac.uk

Odinaldo Rodrigues 
King's College London
London, United Kingdom
odinaldo.rodrigues@kcl.ac.uk




ABSTRACT

Imitation learning field requires expert data to train agents in a task. Most often, this learning approach suffers from the absence of available data, which results in techniques being tested on its dataset. Creating datasets is a cumbersome process requiring researchers to train expert agents from scratch, record their interactions and test each benchmark method with newly created data. Moreover, creating new datasets for each new technique results in a lack of consistency in the evaluation process since each dataset can drastically vary in state and action distribution. In response, this work aims to address these issues by creating *Imitation Learning Datasets*, a toolkit that allows for: (i) curated expert policies with multithreaded support for faster dataset creation; (ii) readily available datasets and techniques with precise measurements; and (iii) sharing implementations of common imitation learning techniques. **Demonstration link:** <https://nathangavenski.github.io/#/il-datasets-video>

KEYWORDS

Imitation Learning; Benchmarking; Dataset

ACM Reference Format:

Nathan Gavenski , Michael Luck , and Odinaldo Rodrigues . 2024. Imitation Learning Datasets: A Toolkit For Creating Datasets, Training Agents and Benchmarking: Demonstration Track. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 3 pages.

1 INTRODUCTION

Creating imitation learning (IL) techniques requires researchers to gather *expert* samples to train an agent in the desired task. This process can be arduous since collecting expert samples usually involves either recording humans performing the task or training a new agent from scratch using another learning paradigm. Moreover, creating new datasets for each new technique does not allow for evaluation consistency across different IL approaches [9].

When creating IL datasets, we must also consider which experts are used to collect data. Most often, when a new expert is created, no information is provided about the quality of the data collected [1], which can drastically affect the performance across different IL approaches [9]. Moreover, researchers must find available code and run it in any newly created dataset for consistent comparison.

Evaluating IL techniques in new datasets is time-consuming since: (i) not all techniques have code readily available; (ii) implementations might contain bugs; and (iii) published versions might not support the environment the user is experimenting with. Testbeds, such as Gym [8], help researchers to overcome problems with different environment support. However, it is up to researchers to create code that supports all available environments, such as those with continuous and discrete actions and states spaces.

In light of these issues, we have developed *Imitation Learning Datasets* (IL-Datasets) [4], a toolkit that aims to help researchers through: (i) creating new datasets by allowing for faster *multi-threaded* creation and curated expert policies; (ii) assisting the training of IL agents by sharing *readily available datasets* with easily customisable data; and (iii) benchmarking by *providing results* for IL techniques in a diverse set of environments. Figure 1 shows each stage in a typical pipeline implementation. In Step (i) users can use a ‘Controller’ class that allows them to create datasets using *expert policies* hosted on HuggingFace [3] or *custom-made* ones following the ‘Policy’ interface. With the dataset selected (based on the data provided on HuggingFace or locally), the user can instantiate the ‘BaselineDataset’ to create training and evaluation datasets to train an agent in step (ii). Lastly, in step (iii), users can specify a benchmarking strategy, including data and hyperparameters, and the toolkit will ensure that no *leakage* (which refers to training data being present during test) will occur, recording the benchmarking details for reproducibility and better comparison with other work.

2 DATASET CREATION

The multithreaded ‘Controller’ class allows users to execute functions that record the ‘Policy’ experiences asynchronously. This is a lightweight class since it creates a thread pool with a fixed number of threads (informed by the user) and spawns objects that will be executed for each episode instead of processes waiting for available resources. Therefore, as the execution of each episode ends, the execution of a new episode starts ensuring nearly 100% uptime in

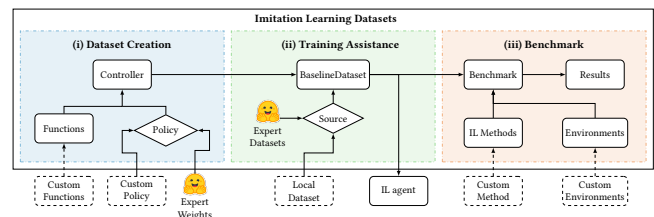


Figure 1: Visualisation of typical pipeline implementation.

all threads. IL-Datasets is agnostic to environment implementation (e.g., vectorized environments) since threads do not share memory pointers. IL-Datasets allows users for the creation of new datasets using already curated policies ensuring for lower behaviour divergence across different datasets [1]. In addition, users can also define custom-made policies to create new datasets while still benefiting from IL-Datasets features, and make them available for other users.

To create a new dataset a user simply needs to provide an ‘enjoy’ function, which uses a policy to interact with the environment and collect samples during an episode; and a ‘collate’ function that creates a single dataset file from all the recordings of the ‘enjoy’ function. IL-Datasets provides functions for converting the newly created dataset into a HuggingFace dataset, which can later be updated to the platform if the user desires. The following code snippet illustrates the creation of a new dataset using IL-Datasets.

```
1 from imitation_datasets.controller import Controller
2 from imitation_datasets.functions import baseline_enjoy
3 from imitation_datasets.functions import baseline_collate
4 Controller(
5     baseline_enjoy, baseline_collate, episodes=1000, threads=4
6 ).start({"game": "walker", "mode": "all"})
```

The ‘Controller’ class (Lines 4-6) uses the already provided ‘enjoy’ and ‘collate’ functions, Lines 2 and 3, respectively. Therefore, creating a new dataset only requires 6 lines of code (without losing the asynchronous multithread benefit). We provide ‘enjoy’ and ‘collate’ functions (based on the StableBaselines [6] pattern) for fast prototyping of new agents, where these functions will create a dataset containing ‘state’, ‘action’, ‘reward’, ‘accumulated episode reward’, and ‘episode start’ (signalling which states are the first in each episode). Line 6 starts the asynchronous multithread process for the register entry ‘walker’, which has a curated expert for the ‘Walker2d-v3’ environment. IL-Datasets provides a list of registered environments with expert policies.¹ Lastly, we allow customisable ‘enjoy’ and ‘collate’ functions to avoid the pitfalls from Imitations [5] and StableBaselines [6], which have dataset creation functions but only support a strict format not suitable for most common state-of-the-art code available.

3 TRAINING ASSISTANCE

IL-Datasets provides a ‘BaselineDataset’ class that allows researchers to use custom-made (Line 2) or hosted data (Lines 3).

```
1 from src.imitation_datasets.dataset import BaselineDataset
2 local = BaselineDataset("/path/to/local/file.npz")
3 hf = BaselineDataset("/path/to/hosted/data", source="hf")
```

It is important to note that even though these datasets are hosted on HuggingFace, once downloaded, the whole process can be executed offline if so needed. The ‘BaselineDataset’ class inherits the PyTorch Dataset class [7] and returns a tuple of (s_t, a_t, s_{t+1}) , where s_t and s_{t+1} are the current and next states, and a_t is the action responsible for the state transition. ‘BaselineDatasets’ can also be inherited from other classes to support other formats, e.g., sequential data². By using IL-Datasets’s³ data, researchers can use up to 1,000 episodes for each of the available environments. These episodes can be divided between *train* (Line 2) and *evaluation* (Line 3) splits:

```
1 from src.imitation_datasets.dataset import BaselineDataset
2 dataset_train = BaselineDataset(..., n_episodes=100)
3 dataset_eval = BaselineDataset(..., n_episodes=100, split="eval")
```

In Line 2, ‘n_episodes’ denotes the number of training episodes, e.g., [0, 100), and in Line 3, it refers to the evaluation set interval, e.g., [100, 1,000). Each dataset published contains the expert policies used during creation and provides the average reward in cases where the *performance* metric (also provided by IL-Datasets) is desired. Moreover, all of these features exist within IL-Dataset to ensure *consistency* in IL experiments. By using the same dataset and splits, researchers can be sure that all trajectories remain the same through different runs.

4 BENCHMARKING

The final IL-Datasets feature is *benchmarking*, with which we aim to implement and test different IL techniques based on the available datasets. We publish all data to help researchers reduce the amount of work to create IL techniques and to reduce the entry barrier to new researchers, but users’ benchmarks will only be published if they desire to do so. The benchmark trains each technique with the available data for 100,000 epochs and, afterwards, evaluates each one using a specific set of seeds guaranteeing reproducibility and consistency across multiple executions. These seeds are selected to reduce *data leakage* and to make sure that the first state is not present in the training datasets. Each technique is evaluated by executing the best model (according to the original work selection criteria) in each environment, and the *average episodic reward* and *performance* metrics are displayed in a list with all benchmark results published in the IL-Datasets page. Furthermore, when training these techniques, IL-Datasets also uses specific seeds to guarantee that the training results will be the same for each method across multiple executions. We note that this is done outside of Gym [8] environments since they do not support random number generators anymore. Therefore, training IL-Datasets implementations without using these seeds will not guarantee the same results.

5 CONCLUSION

In this paper, we described Imitation Learning Datasets: a toolkit to help researchers implement, train and evaluate IL agents. IL-Datasets can be used to reduce comparison efforts while increasing consistency between published work. It achieves this by offering: (i) fast and lightweight dataset creation through asynchronous multi-thread processes with curated expert policies that allow for no prior expert training and low behaviour divergence between different creations; (ii) readily available datasets, for fast prototyping of new techniques letting users only worry about model implementation; and (iii) benchmarking results for IL techniques. We believe that IL-Datasets will help facilitate the integration of new researchers and improve consistency across different IL work.

ACKNOWLEDGMENTS

This work was supported by UK Research and Innovation [grant number EP/S023356/1], in the UKRI Centre for Doctoral Training in Safe and Trusted Artificial Intelligence (www.safeandtrusted.ai.org) and made possible via King’s Computational Research, Engineering and Technology Environment (CREATE) [2].

¹<http://github.com/NathanGavenski/IL-Datasets>

²<https://gist.github.com/NathanGavenski/ec904c7c3bf06b6361a0897b798206ac>

³<https://nathangavenski.github.io/#/IL-Datasets-data>

REFERENCES

- [1] Suneel Belkhal, Yuchen Cui, and Dorsa Sadigh. 2023. Data Quality in Imitation Learning. *arXiv* (2023). arXiv:2306.02437v1 [cs.RO]
- [2] King's College London e Research team. 2023. King's computational research, engineering and technology environment (CREATE). <https://doi.org/10.18742/rmvf-m076>
- [3] Hugging Face. 2023. Hugging Face. Web Page. <https://huggingface.co/>
- [4] Nathan Gavenski. 2023. Imitation Learning Datasets. GitHub Repository. <https://github.com/NathanGavenski/IL-Datasets>
- [5] Adam Gleave, Mohammad Taufeeque, Juan Rocamonde, Erik Jenner, Steven H. Wang, Sam Toyer, Maximilian Ernestus, Nora Belrose, Scott Emmons, and Stuart Russell. 2022. imitation: Clean Imitation Learning Implementations. arXiv:2211.11972v1 [cs.LG]. arXiv:2211.11972 [cs.LG] <https://arxiv.org/abs/2211.11972>
- [6] ashley hill, antonin raffin, maximilian ernestus, adam gleave, anssi kanervisto, rene traore, prafulla dhariwal, christopher hesse, oleg klimov, alex nichol, matthias plappert, alec radford, john schulman, szymon sidor, and yuhuai wu. 2018. stable baselines. <https://github.com/hill-a/stable-baselines>.
- [7] 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. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>
- [8] mark towers, jordan k terry, ariel kwiatkowski, john u. balis, gianluca de cola, tristan deleu, manuel goulão, andreas kallinteris, arjun kg, markus kimmel, rodrigo perez vicente, andrea pierré, sander schulhoff, jun jet tai, andrew tan jin shen, and omar g. younis. 2023. gymnasium. <https://doi.org/10.5281/zenodo.8127026>
- [9] Boyuan Zheng, Sunny Verma, Jianlong Zhou, Ivor W. Tsang, and Fang Chen. 2022. Imitation Learning: Progress, Taxonomies and Challenges. *IEEE Transactions on Neural Networks and Learning Systems* (2022), 1–16. <https://doi.org/10.1109/tnnls.2022.3213246>