Katakomba: Tools and Benchmarks for Data-Driven NetHack

Vladislav Kurenkov*
Tinkoff
v.kurenkov@tinkoff.ai

Alexander Nikulin*
Tinkoff
a.p.nikulin@tinkoff.ai

Denis Tarasov
Tinkoff
den.tarasov@tinkoff.ai

Sergey Kolesnikov
Tinkoff
s.s.kolesnikov@tinkoff.ai

Abstract

NetHack is known as the frontier of reinforcement learning research where learning-based methods still need to catch up to rule-based solutions. One of the promising directions for a breakthrough is using pre-collected datasets similar to recent developments in robotics, recommender systems, and more under the umbrella of offline reinforcement learning (ORL). Recently, a large-scale NetHack dataset was released; while it was a necessary step forward, it has yet to gain wide adoption in the ORL community. In this work, we argue that there are three major obstacles for adoption: resource-wise, implementation-wise, and benchmark-wise. To address them, we develop an open-source library[2] that provides workflow fundamentals familiar to the ORL community: pre-defined D4RL-style tasks, uncluttered baseline implementations, and reliable evaluation tools with accompanying configs and logs synced to the cloud.

1 Introduction

Reinforcement learning (Arulkumaran et al., 2017; Levine et al., 2020) led to remarkable progress in the decision-making problems in recent years: robotics (Smith et al., 2022; Kumar et al., 2021), recommender systems (Chen et al., 2022a), traffic control (Chen et al., 2020), energy management (Yu et al., 2019), combinatorial optimization (Mazyavkina et al., 2021), and videogames (Baker et al., 2022; Hafner et al., 2023). Many of those are considered solved or close to solved problems, and the arguably next untapped frontier for RL algorithms is the NetHack (Küttler et al., 2020).

NetHack[3] is considered one of the most challenging games for humans, even more to learning agents. This game requires strong generalization capabilities, as the levels are procedurally generated with each reset, and extraordinary memory capacity, as the episodes may last for 100k steps requiring to remember what happened a thousand steps before to act optimally. Moreover, the high level of uncertainty and its dependence on the initial character configuration further hardens the problem. Indeed, at the moment, the best agent, AutoAscend (Hambro et al., 2022a), is essentially rule-based, and yet reinforcement learning agents cannot even come close in terms of the game score or levels reached. While there were various attempts to advance online reinforcement learning agents, this task seems out of reach for them.

*Contributed equally.

[2]Source code: https://github.com/corl-team/katakomba

[3]For a brief introduction to the game, we recommend excellent game wiki, as well as the original publication by Küttler et al. (2020), which introduced the NetHack environment.

37th Conference on Neural Information Processing Systems (NeurIPS 2023) Track on Datasets and Benchmarks.

Recently, a promising direction was outlined – re-using large-scale datasets of human/bot demonstrations to bootstrap RL agents, either in the pre-training phase or in the fine-tuning stage (Hambro et al., 2022b; Kumar et al., 2022b). This research direction holds great promise to advance the community's progress in solving the NetHack and attacking the weak sides of the RL agents. While the proposed tools and datasets are necessary, their implementations, support, and ease of adoption still need to be improved. What is essential, the use of NetHack promises the democratization of research efforts, providing way more efficient tools compared to both StarCraft (Vinyals et al., 2019) and Minecraft (Guss et al., 2019) learning environments. However, while the open-sourced datasets are a significant first step, the provided tools are very far from being readily adopted by other researchers (Section 2).

In this work, we aim to address this gap and provide a set of instruments that are conducive to future research endeavors, smooth the learning curve, and define a precise set of tasks to be evaluated against when training offline RL agents on the NetHack dataset, where contributions are as follows:

- **D4RL-Style Benchmark** A set of small-scale datasets based on the large-scale data source (Hambro et al., 2022b) for faster experimentation, including several data loaders for different speed-memory tradeoffs with a familiar interface to the ORL community alike Fu et al. (2020).
- **Clean Recurrent Offline RL Baselines** Straightforward implementations of popular offline RL baselines augmented with recurrence: BC (Michie et al., 1990), CQL (Kumar et al., 2020), AWAC (Nair et al., 2020), REM (Agarwal et al., 2020), IQL (Kostrikov et al., 2021). Each implementation is separated into single files akin to Huang et al. (2021); Tarasov et al. (2022) to smooth the learning curve.
- **Evaluation Guideline** We use reliable evaluation tools for comparing offline RL agents in the natural presence of high variability. For this, we employ recent RLiable tools (Agarwal et al., 2021) that avoid comparing against mean values (which are especially deceptive in the case of the NetHack environment). In order to facilitate the comparisons, we also release the raw results we obtained in our experiments so that future work may re-use them for a fair comparison.
- **Open-Sourced Training Logs and Configs** Moreover, we keep a public track of our experiments: including configs, implementations, and learning dynamics using Weights&Biases (Biewald, 2020), so that it is transparent for inspection and reproduction.

We believe our work is a logical next step for advancing reinforcement learning in the NetHack environment, providing a conducive set of tools, algorithms, and utilities for experimenting with offline RL algorithms. Moreover, as we plan to support the benchmark, we intend to extend it with offline-to-online tools and benchmarks. By no means this work serves as a converged fixed point. We view this as a starting step in building reliable tools for solving NetHack with prior data and commit to further support and development.

2 What Hinders the Adoption of Data-Driven NetHack?

In this section, we describe the main obstacles and motivate our work. We outline our experience with the recently released dataset (Hambro et al., 2022b), accompanying tools and divide the main issues into three categories: implementational, resources, and benchmark-wise. These are needed to illustrate that the initial release of the dataset is a significant step. However, it is only the first step, and further development is needed, which we will address in further sections.

Implementation-wise First, when trying to reproduce the experiments, we found that installing the released tools is error-prone: where the provided Dockerfile was not operable and required fixing the moolib library with a CMake modification, which is probably undesirable for broader adoption of practitioners. Second, when we attempted to reproduce the scores reported in Hambro et al. (2022b), the only configuration file provided was for the IQL algorithm[4] and its modification with hyperparameters for other algorithms from the paper required additional effort to assign proper

[4]Consider line 38 for an example of hyperparameters ambiguity: `https://github.com/dungeonsdatasubmission/dungeonsdata-neurips2022/blob/main/experiment_code/hackrl/dqn_ttyrec_config.yaml`. When changing the crop dimensions, the implementation crashes.

values. Consequently, the reproduction did not result near what was reported in the paper (note that a similar pattern was observed in Piterbarg et al. (2023)). While we do not claim that the reproduction is impossible, we could not find the proper fix in a reasonable amount of time and consider this as another obstacle in adoption – practitioners should either have access to the original training logs or be able to replicate the results reliably.

Beyond the replication frictions, another issue lies in the design of implementations. Offline and offline-to-online settings are interleaved within just one file (1500 lines of code), where the algorithms are realized using the framework for distributed reinforcement learning not typical for the ORL community (Mella et al., 2022). Consequently, understanding the underlying logic is hard, and some performance-related implementation tricks can be hard to spot. For example, q-learning-based algorithms do not use the last element of the sequence, i.e., every $sequence\_length$ game tuple is not used for training (but only for bootstrapping). While all of these may seem like minor issues, their sum brings a significant burden upon practitioners interested in the data-driven NetHack research.

Resource-wise The great addition to the released datasets was an accompanying data loader that operates over compressed game episodes (we refer to it as TTYRec throughout the text). While the Hambro et al. (2022b) demonstrated that the proposed loader could operate at approximately 5 hours per epoch, we observed a slower rate when adapting it for recurrent algorithms. The main issue lies in the underlying format where the data is fetched the following way – (s_t, a_t, r_{t-1})[5]. While this may seem non-significant, it results in the need to fix the tuples sequence to obtain a suitable format (s_t, a_t, s_{t+1}, r_t). To better demonstrate the impact of this problem, consider Table 1, where we test the original data loader and the one with the fix, observing a significant increase in the loading time as both batch size and sequence length increase. While this can be avoided with the original loader by simply discarding each $sequence\_length$ element (as done in the original work), the problem persists as the original implementations do not work without reward shaping with potentials requiring a scan over the sequence, not to mention that such data rejection is not justified beyond performance reasons. As an additional but important issue to resource-constrained practitioners, one must first download the entire 90GB AA dataset, even if they aim to work on a subset of data.

Table 1: Loading times for different storage formats averaged over 500 iterations. The right part of the table, with HDF5 columns, depicts the loading time for datasets repacked within Katakomba. Note that the transitions are in the proper format for them by design.

Variants	TTYRec	TTYRec, Proper Tuples	HDF5 (Memmap)	HDF5 (RAM)	HDF5 (Compressed)
batch_size=64, seq_len=16	15ms	17ms	2ms	1ms	516ms
batch_size=256, seq_len=32	74ms	132ms	12ms	9ms	2.39s
batch_size=256, seq_len=64	255ms	372ms	18ms	14ms	2.42s

Benchmark-wise Arguably, one of the driving forces in Deep Learning and Reinforcement Learning, in general, is a well-defined set of tasks. To demonstrate how the proposed large-scale dataset could be utilized for tasks definition, Hambro et al. (2022b) described two settings: learning on the whole dataset for all role-race-alignment-sex combinations and learning on the subset of data for the Monk-Human-Neutral setting. While this is a good entry point and could be of great use to practitioners interested in large-scale data regimes, there was no detailed proposal on how one should further define tasks which is indeed an open question in the NetHack learning community (Küttler et al., 2020). Moreover, the original raw large-scale dataset requires practitioners to manually define SQL queries for extracting the data of interest, which is flexible but could be an overkill.

More importantly, the proposed comparison metrics were mean and median statistics of average episode returns over training seeds. One may argue that the median is a robust statistic. However, in this case, it is a median of *average* episode returns over training seeds and not of the whole set of evaluation episodes. It is well known in the RL community (Agarwal et al., 2021) that those are not reliable due to the highly-variable nature of RL agents. As of the NetHack, this further amplifies by the extremely-variable nature of the game itself as both demonstrated in Küttler et al. (2020); Hambro et al. (2022b). Therefore, to reduce the noise in the progress of RL agents in this domain, one would need a different evaluation toolset that is better suited for it.

[5]Notation standard for the RL community, where s is a state, a is an action, and r is a reward.

3 Katakomba

Given the issues described in the previous section, we are ready to present Katakomba – a set of tools, benchmarks, and memory-based offline RL baselines. In this section, we gradually introduce the components of the library with accompanying code examples to better demonstrate the ease of use. All the numbers and performance metrics discussed in the text were measured on the same set of hardware: 14CPU, 128GB RAM, NVMe, 1xA100; for more details, please, see Appendix D.

3.1 Benchmark

Listing 1: Usage example: Tasks are defined via the character field that is then used by the offline wrapper for dataset loading and score normalization.

```
from katakomba.env import NetHackChallenge
from katakomba.env import OfflineNetHackChallengeWrapper

# The task is specified using the character field
env = NetHackChallenge(
    character="mon-hum-neu",
    observation_keys=["tty_chars", "tty_colors", "tty_cursor"]
)

# A convenient wrapper that provides interfaces for
# dataset loading and score normalization
env = OfflineNetHackChallengeWrapper(env)
```

Decomposition In our benchmark, we re-use the dataset collected by the AutoAscend bot (Hambro et al., 2022b,a). While this bot is highly-capable, it still is considered an early-game contender because it can not descend further than two levels in half of the episodes. As the dataset becomes an early game bridgehead, we divide the tasks based on the character configurations: role, race, and alignment. In the NetHack community, these are known to be the most important (and even having a dramatic effect), requiring to utilize varying abilities of each role or race[6]. Overall, in opposition to merging all combinations, this decomposition allows more focus on the characters' gameplay and the size reduction one needs to download for both playing-around and committed research, as we will describe further.

Categories This results in 38 datasets in total. However, it may not be possible for researchers to examine each of them as the training times can be an obstacle. To this end, we further divide the tasks into three categories: Base, Extended, and Complete. We separate each category based on the wisdom of the NetHack community, i.e., that roles have a more substantial effect on the gameplay than race, and race has more effect than alignment. The datasets and categories are listed in Table 2. Base tasks consist of all possible roles for the human race; we choose a neutral alignment where possible; otherwise, we pick the alternative one (for humans, if there is no neutral alignment, there is only one alternative). We include all other role-race combinations for Extended tasks. For Complete, we add tasks that were not included in the Base and Extended categories. Note that these three categories combined cover all of the allowed role-race-alignment combinations.

Data Selection As demonstrated in the previous section, the original TTYRec dataset is actually slower than expected when it is used for learning agents. Moreover, one may be unable to download the 90GB dataset. Therefore, we take a different path by subsampling and repacking the original dataset task-wise, averaging 680 episodes and 1.3GB per task. The subsampling procedure is stratified by the game score (for more details, please see the script[7]). This allows for more versatility: one can download the needed datasets on demand as in D4RL (Fu et al., 2020); furthermore, this permits us to address the rolling issue as we repacked the transition tuples in the way suitable to typical ORL pipeline as a part of the release. To ensure the reliable accessibility of the data, we host it on the HuggingFace Hub (Engstrom et al., 2020).

[6] https://nethackwiki.com/wiki/Player

[7] https://github.com/corl-team/katakomba/blob/main/scripts/generate_small_dataset.py

Table 2: Katakomba tasks. We split the large-scale AutoAscend dataset into 38 problems dividing them into three categories. The splits are justified by the early-game nature of the AutoAscend data, where roles have a higher impact on the gameplay, then races, and then alignments. Note that the whole benchmark covers all possible role-race-alignment combinations.

Tasks	# Transitions	Median Turns	Median Score	Median Deathlvl	Size (GB)	Compressed Size (GB)
Base (Role-Centric)	-	-	-	-	-	-
arc-hum-neu	24527163	32858.0	4802.5	2.0	94.5	1.3
bar-hum-neu	26266771	35716.0	11964.0	4.0	101.1	1.7
cav-hum-neu	21674680	30361.0	8152.0	4.0	83.5	1.3
hea-hum-neu	14473997	18051.0	2043.0	1.0	55.7	0.8
kni-hum-law	22287283	28246.0	6305.0	3.0	85.8	1.5
mon-hum-neu	33741542	42400.0	11356.0	4.0	129.9	2.1
pri-hum-neu	18376473	26796.5	5366.5	2.0	70.8	1.1
ran-hum-neu	17625493	25354.0	6168.0	2.0	67.9	1.0
rog-hum-cha	14284927	19334.0	3005.5	1.0	55.0	0.8
sam-hum-law	22422537	32951.0	7850.0	4.0	86.3	1.3
tou-hum-neu	13376498	17955.5	2554.5	1.0	51.5	0.8
val-hum-neu	27784788	35250.0	11402.5	4.0	107.0	1.8
wiz-hum-neu	14343449	19808.5	3132.5	1.0	55.2	0.8
Extended (Race-Centric)	-	-	-	-	-	-
pri-elf-cha	18796560	26909.5	4718.5	2.0	72.4	1.1
ran-elf-cha	18238686	26607.0	7583.0	4.0	70.2	1.1
wiz-elf-cha	15277820	19512.0	2988.5	1.0	58.8	0.9
arc-dwa-law	25100788	34669.0	4026.0	1.0	96.7	1.5
cav-dwa-law	22871890	32261.0	7158.0	3.0	88.1	1.5
val-dwa-law	32787658	33973.0	8652.5	3.0	126.6	2.5
arc-gno-neu	24144048	34432.0	4077.5	1.0	93.0	1.4
cav-gno-neu	21624779	29860.0	6446.0	3.0	83.3	1.4
hea-gno-neu	14884704	18518.0	1980.5	1.0	57.3	0.9
ran-gno-neu	17571659	25970.0	5326.0	2.0	67.7	1.1
wiz-gno-neu	14193637	19206.0	2736.0	1.0	54.7	0.9
bar-orc-cha	27826356	39291.0	10499.0	4.0	107.2	1.8
ran-orc-cha	18127448	26707.0	5460.0	2.0	69.8	1.1
rog-orc-cha	16674806	22351.0	3103.0	1.0	64.2	1.0
wiz-orc-cha	15994150	22570.5	3241.5	1.0	61.6	1.0
Complete (Alignment-Centric)	-	-	-	-	-	-
arc-hum-law	23422383	31446.0	4188.0	1.0	90.2	1.3
cav-hum-law	22328494	31039.0	8174.0	4.0	86.0	1.3
mon-hum-law	30782317	39647.0	10855.0	4.0	118.5	1.9
pri-hum-law	18298816	27192.0	4833.0	1.0	70.5	1.1
val-hum-law	30171035	34570.5	9707.0	4.0	116.2	2.1
bar-hum-cha	25362111	35925.0	12574.0	5.0	97.7	1.6
mon-hum-cha	33662420	41730.5	11418.0	4.0	129.6	2.1
pri-hum-cha	18667816	28204.5	5847.0	2.0	71.9	1.1
ran-hum-cha	16999630	24698.5	6236.0	2.0	65.6	1.0
wiz-hum-cha	14635591	20257.0	3294.0	1.0	56.4	0.9

3.2 Data Loaders for Speed-Memory Tradeoff

As we outlined in Section 2, while the original large-scale dataset is well-compressed, its iteration speed integrated with proper sequential loaders is still an obstacle for fast experimentation loop. Moreover, it requires at least one full download for the entire large-scale dataset, which may not be suitable for resource-constrained scenarios. In Katakomba, we address this by providing three different loaders trading-off memory and speed allowed by the deliberate task division. In each of them, the compressed task's dataset is automatically downloaded and decompressed if one aims for speed. Moreover, we provide a Python interface for an automatic clean-up if one does not want to store the decompressed dataset beyond the code execution.

HDF5, In-Memory (Decompressed, RAM) This is the fastest format that relies on the decompression of the dataset into the RAM, which may require from 51GB to 129GB depending on the dataset listed in Table 2. This option is quite demanding regarding the memory but comes with the advantage of the fastest data access (see Table 1).

HDF5, Memory Map (Decompressed, Disk) This option is a middle-ground that allows the datasets to be efficiently utilized without requiring large RAM. When one specifies this preference, the dataset will be decompressed on one's hard drive, and the consequent reads will be conducted from it. The advantage of this approach compared to the loader from Hambro et al. (2022b) is that there is no need to decompress on the fly, and this is done precisely once at the start of the training process (taking from three to ten minutes on average).

Listing 2: Usage example: Katakomba provides different loaders that can be chosen depending on one's speed-memory tradeoff.

```
# Decompress the dataset into RAM
dataset = env.get_dataset(mode="in_memory")

# Decompression on-read
dataset = env.get_dataset(mode="compressed")

# The original dataloader introduced in Hambro et al, 2022
dataset = env.get_dataset(scale="big")

# Decompress the dataset on disk
dataset = env.get_dataset(mode="memmap")

# If you want to delete the decompressed dataset
# This will not affect the compressed version
dataset.close()
```

HDF5, In-Disk (Compressed, Disk) This mode is the most cheap one but slow. Essentially, the dataset is read on from disk and decompressed on-the-fly. We found this useful for debugging purposes where one does not need the whole training process to be run.

TTYRec, In-Disk (Compressed, Disk) In case one finds the original approach to data loading more suitable, we also provide a convenient interface that wraps the source large-scale dataset and loader. One can also set it up in more detail using keyword arguments from the original TTYRec data loader. However, this option comes with the downsides described in Section 2, i.e., slower iteration speed and the need to download the 90GB dataset at least once.

In addition to the dataset interfaces, we also provide an implementation of a replay buffer suitable for sequential offline RL algorithms for bootstrapping practitioners with ready-to-go tools.

3.3 Evaluation Methodology under High Variability

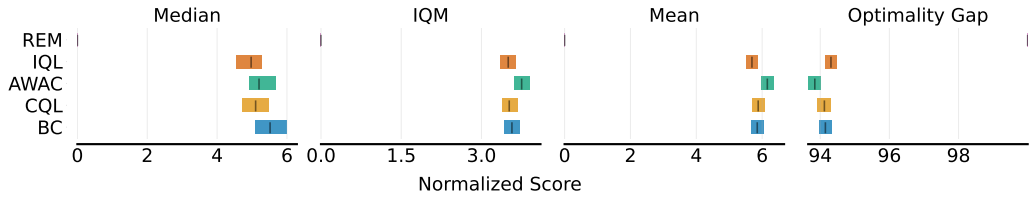
In Hambro et al. (2022b), authors used an average episode return across seeds when comparing baselines. While this is a standard practice in the RL and ORL communities, it was recently shown to be unreliable (Agarwal et al., 2021) as the algorithms are known to be highly variable in performance. This problem amplifies even more for NetHack, where the score distribution is typically quite wide for humans and bots (Küttler et al., 2020).

To address this, we argue that the evaluation toolbox from Agarwal et al. (2021) is more appropriate and suggest using it when comparing NetHack learning agents. We use these tools for two dimensions: in-game score and death level. The first dimension corresponds to what one typically optimizes with ORL agents but is considered a proxy metric (Küttler et al., 2020). While the latter lower bounds the early-game progress and is more indicative of the game completion.

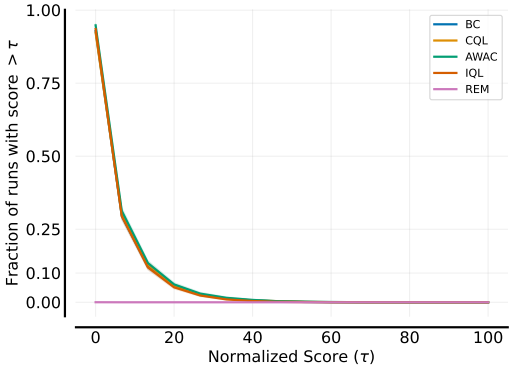
Furthermore, similar to Fu et al. (2020), we also suggest reporting normalized scores to capture how far one is from the AutoAscend bot. We use mean scores per dataset as a normalization factor and rescale to [0, 100] range after normalization, similar to D4RL (Fu et al., 2020). This functionality is also provided as a part of the offline wrapper for the NetHackChallenge environment. Please refer to Appendix D for precise values.

4 Benchmarking Recurrent Offline RL Algorithms

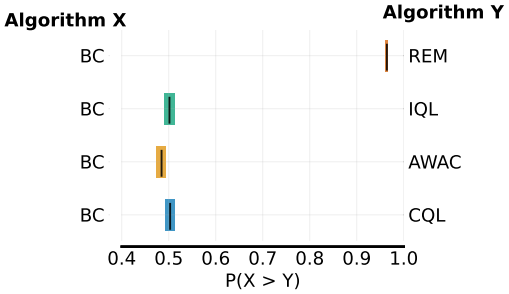
Algorithms For our benchmark, we rely on the following set of algorithms: Behavioral Cloning (BC) (Michie et al., 1990), Implicit Q-Learning (IQL) (Kostrikov et al., 2021), Conservative Q-Learning (CQL) (Kumar et al., 2020), Randomized Ensemble Mixture (REM) (Agarwal et al., 2020), and Advantage-Weighted Actor-Critic (AWAC) (Nair et al., 2020). These are known as either the most competitive in the continuous control setting (Tarasov et al., 2022) or were shown to be competitive in the discrete control (Agarwal et al., 2020; Kumar et al., 2022a). Similar to Hambro et al. (2022a,b), we build upon Chaotic-Dwarven-GPT-5 architecture that converts the tty observation into an image



(a) Bootstrapped point estimates. It can be seen that except REM, which failed, all algorithms perform about the same across all metrics and are generally far behind the AutoAscend algorithm, which mean scores per dataset were used for normalization.



(b) Performance profiles. Notably, around 5% of episodes can surpass the normalized score of 20.



(c) Probability of improvement of BC to other algorithms.

Figure 1: Normalized performance under the Katakomba benchmark for all 38 datasets. Each algorithm was run for three seeds and evaluated over 50 episodes resulting in 5700 points for constructing these graphs. As one can see, there is not much improvement beyond naive behavioral cloning.

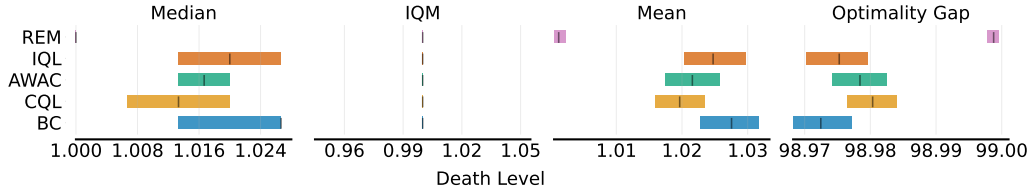
and then feeds it into the CNN layers followed by the LSTM (Hochreiter & Schmidhuber, 1997). Ultimately, we test five common ORL algorithms augmented with recurrence. To the best of our knowledge, there are no other open-sourced alternatives beyond the Hambro et al. (2022a) that also do not implement the AWAC algorithm.

Experimental Setup We train every algorithm for 500k gradient steps, resulting in around 20 epochs per dataset. We report the scores of the last trained policy over 50 evaluation episodes as standard in the ORL community. Important to highlight that while this amount may seem small for NetHack, this number is adequate when used in conjunction with stratified datasets and RLiable evaluation tools due to the underlying bootstrapping mechanism. For specific hyperparameters used, please either see Appendix E or the configuration files provided in the code repository.

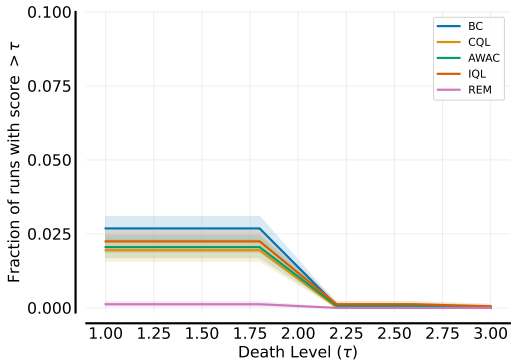
Replicability and Reproducibility Statement To ensure that our experiments' results are replicable and can easily be reproduced and inspected, we rely on the Weights&Biases (Biewald, 2020). In the provided code repository, one can find all the logs, including configs, dependencies, code snapshots, hyperparameters, system variables, hardware specifications, and more. Moreover, to reduce the efforts of interested parties required for inspection, we structurize the results using the Weights&Biases public reports.

Results The outcomes are twofold. First, the results achieved are similar to the already observed by the Piterbarg et al. (2023) and Hambro et al. (2022b), but on a larger number of offline RL algorithms tested. As shown in Figure 1 and Figure 2, all algorithms were unable to replicate the scores of the AutoAscend bot, reaching normalized scores below 6.0 on average and not progressing beyond the first level on the majority of training runs. Notably, only 5% of the episodes resulted in a normalized score of around 20.0 (Figure 1b). Moreover, REM has not been able to achieve even the non-zero normalized score. Second, perhaps surprisingly, the only algorithm that does not rely in any way on policy constraints is also the only algorithm that completely failed. This, and also the high correlation

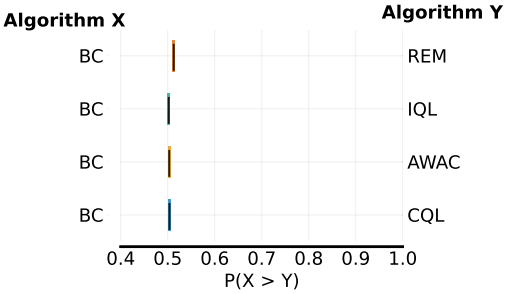
in the performance profiles (Figure 1b), gives us a hint that all other methods showing non-zero results actually rely primarily on behavioral cloning in one form or another, such as advantage weighted regression in IQL and AWAC or KL-divergence as in CQL. Indeed, the most successful hyperparameters in our experiments proved to be those that significantly increase the weight of losses that encourage similarity to the behavioral policy (see Table 10 in the Appendix E). Furthermore, as shown in the Figure 1c and Figure 1c Behavioral Cloning algorithm is not worse than all the other more sophisticated offline RL algorithms. Thus, NetHack remains a major challenge for offline RL algorithms, and Katakomba can serve as a starting point and testbed for offline RL research in this direction. For graphs stratified by the Base, Extended, and Complete categories, see Appendix F.



(a) Bootstrapped point estimates.



(b) Performance profiles. Notably, only 2.5% of episodes progress to the second level.



(c) Probability of improvement of BC to other algorithms.

Figure 2: Death level under the Katakomba benchmark for all 38 datasets. Each algorithm was run for three seeds and evaluated over 50 episodes resulting in 570 points for constructing these graphs. As one can clearly see, there is not much improvement beyond the naive behavioral cloning.

5 Related Work

Offline RL In recent years, there was considerable interest of the reinforcement learning community in the offline setting, which resulted in numerous and diverse algorithms specifically tailored for this setup (Nair et al., 2020; Kumar et al., 2020; Kostrikov et al., 2021; Chen et al., 2021; Fujimoto & Gu, 2021; An et al., 2021; Nikulin et al., 2023; Tarasov et al., 2023). The core idea behind most of them is to ensure the resulting policy stays close to the behavioral one. This could be achieved via different ways: penalizing value function (Kumar et al., 2020; An et al., 2021; Nikulin et al., 2023), constraining the policy outputs (Fujimoto & Gu, 2021; Tarasov et al., 2023), or even training directly in the conservative latent space (Zhou et al., 2020; Chen et al., 2022b; Akimov et al., 2022). Due to the benchmark-centricity of the RL field, most of the proposed ORL algorithms are for continuous control with a few exceptions (Agarwal et al., 2020; Kumar et al., 2020, 2022a). The de-facto standard benchmark is the D4RL (Fu et al., 2020), which provides a suite of datasets focused on continuous control with proprioceptive states under different regimes, such as sparse-reward or low-data regimes. Also, few benchmarks move the focus from proprioceptive states to images or other more complex entities (Qin et al., 2022; Lu et al., 2022; Agarwal et al., 2020).

RL for NetHack NetHack as a testbed for RL agents was introduced in Küttler et al. (2020). To further advance the RL agents in this domain, the NetHack Challenge Competition (NHC) was held

(Hambro et al., 2022a) that resulted in two of the most performant agents – learning-based Chaotic-Dwarven-GPT-5, and a rule-based AutoAscend (AA). Notably, the latter outperformed learning-based agents by a wide margin. Consequently, this solution was used to collect the large-scale NetHack Learning Dataset (Hambro et al., 2022b). The closest concurrent work is by Piterbarg et al. (2023) – the authors released another AA dataset but accompanied with the hierarchical labels, which arise due to the nature of the AutoAscend bot, demonstrating their usefulness in cloning the AA bot. However, in contrast to our work, Piterbarg et al. (2023) focuses on the large-scale setup similar to Hambro et al. (2022b).

6 Discussion, Limitations, and Future Work

In this work, we focused on building reliable tools and benchmarks for offline RL setting using the recently released AutoAscend large-scale dataset (Hambro et al., 2022b). While this does not cover the whole spectrum of NetHack's community interests, such as offline-to-online regime or learning from human demonstrations, we believe our effort is helpful in establishing reliable and open instruments for data-driven NetHack. Moreover, our contributions could be of interest to the part of the ORL community studying discrete control, memory, and adaptivity (Ghosh et al., 2022).

Given our results and experience with the NetHack Learning Environment, we believe fruitful future research may lie along the following directions: finding a better state-encoder, as the current one presents a bottleneck in both efficiency (rendering is expensive) and locality (only the small part of the terminal is used). Another interesting research direction would be to assess recently appeared recurrence mechanisms such as Linear Recurrent Unit (Orvieto et al., 2023), which might also speed up the training process without hurting the performance. Also, as the interest in generalization properties will appear, it would be a great addition to include more datasets that will provide metadata on the seeds used for data generation, as it will allow to assess trained agents on both seen and unseen seeds to quantify the generalization gap more systematically.

Overall, we firmly believe that NetHack provides a nice playground for investigating how to build a next generation of reinforcement learning agents using prior data that would encompass stronger generalization and memory capabilities. To this end, we plan to continuously maintain the benchmark, accompanying tools, and curate new datasets if considered useful for further advancements.

References

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pp. 104–114. PMLR, 2020.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

Dmitriy Akimov, Vladislav Kurenkov, Alexander Nikulin, Denis Tarasov, and Sergey Kolesnikov. Let offline rl flow: Training conservative agents in the latent space of normalizing flows. *arXiv preprint arXiv:2211.11096*, 2022.

Gaon An, Seungyong Moon, Jang-Hyun Kim, and Hyun Oh Song. Uncertainty-based offline reinforcement learning with diversified q-ensemble. *Advances in neural information processing systems*, 34:7436–7447, 2021.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654, 2022.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com.

Chacha Chen, Hua Wei, Nan Xu, Guanjie Zheng, Ming Yang, Yuanhao Xiong, Kai Xu, and Zhenhui Li. Toward a thousand lights: Decentralized deep reinforcement learning for large-scale traffic signal control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3414–3421, 2020.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Minmin Chen, Can Xu, Vince Gatto, Devanshu Jain, Aviral Kumar, and Ed H. Chi. Off-policy actor-critic for recommender systems. *Proceedings of the 16th ACM Conference on Recommender Systems*, 2022a.

Xi Chen, Ali Ghadirzadeh, Tianhe Yu, Yuan Gao, Jianhao Wang, Wenzhe Li, Bin Liang, Chelsea Finn, and Chongjie Zhang. Latent-variable advantage-weighted policy optimization for offline rl. *arXiv preprint arXiv:2203.08949*, 2022b.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, L. Rudolph, and Aleksander Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *ICLR*, 2020.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

Dibya Ghosh, Anurag Ajay, Pulkit Agrawal, and Sergey Levine. Offline rl policies should be trained to be adaptive. In *International Conference on Machine Learning*, pp. 7513–7530. PMLR, 2022.

William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. Minerl: a large-scale dataset of minecraft demonstrations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pp. 2442–2448, 2019.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Eric Hambro, Sharada Mohanty, Dmitrii Babaev, Minwoo Byeon, Dipam Chakraborty, Edward Grefenstette, Minqi Jiang, Jo Daejin, Anssi Kanervisto, Jongmin Kim, et al. Insights from the neurips 2021 nethack challenge. In *NeurIPS 2021 Competitions and Demonstrations Track*, pp. 41–52. PMLR, 2022a.

Eric Hambro, Roberta Raileanu, Danielle Rothermel, Vegard Mella, Tim Rocktäschel, Heinrich Kuttler, and Naila Murray. Dungeons and data: A large-scale nethack dataset. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022b.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, and Jeff Braga. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *arXiv preprint arXiv:2111.08819*, 2021.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

Aviral Kumar, Anikait Singh, Stephen Tian, Chelsea Finn, and Sergey Levine. A workflow for offline model-free robotic reinforcement learning. In *5th Annual Conference on Robot Learning*, 2021. URL https://openreview.net/forum?id=fy4ZBWxYbIo.

Aviral Kumar, Rishabh Agarwal, Xinyang Geng, G. Tucker, and Sergey Levine. Offline q-learning on diverse multi-task data both scales and generalizes. *ArXiv*, abs/2211.15144, 2022a.

Aviral Kumar, Rishabh Agarwal, Xinyang Geng, George Tucker, and Sergey Levine. Offline q-learning on diverse multi-task data both scales and generalizes. *arXiv preprint arXiv:2211.15144*, 2022b.

Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Cong Lu, Philip J. Ball, Tim G. J. Rudner, Jack Parker-Holder, Michael A. Osborne, and Yee Whye Teh. Challenges and opportunities in offline reinforcement learning from visual observations, 2022.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

Vegard Mella, Eric Hambro, Danielle Rothermel, and Heinrich Küttler. moolib: A Platform for Distributed RL. 2022. URL https://github.com/facebookresearch/moolib.

Donald Michie, Michael Bain, and J Hayes-Miches. Cognitive models from subcognitive skills. *IEE control engineering series*, 44:71–99, 1990.

Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Alexander Nikulin, Vladislav Kurenkov, Denis Tarasov, and Sergey Kolesnikov. Anti-exploration by random network distillation. *arXiv preprint arXiv:2301.13616*, 2023.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences, 2023.

Ulyana Piterbarg, Lerrel Pinto, and Rob Fergus. Nethack is hard to hack. *arXiv preprint arXiv:2305.19240*, 2023.

Rong-Jun Qin, Xingyuan Zhang, Songyi Gao, Xiong-Hui Chen, Zewen Li, Weinan Zhang, and Yang Yu. Neorl: A near real-world benchmark for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 35:24753–24765, 2022.

Laura Smith, Ilya Kostrikov, and Sergey Levine. A Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning, August 2022.

Denis Tarasov, Alexander Nikulin, Dmitry Akimov, Vladislav Kurenkov, and Sergey Kolesnikov. CORL: Research-oriented deep offline reinforcement learning library. In *3rd Offline RL Workshop: Offline RL as a "Launchpad"*, 2022. URL https://openreview.net/forum?id=SyAS49bBcv.

Denis Tarasov, Vladislav Kurenkov, Alexander Nikulin, and Sergey Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning. *arXiv preprint arXiv:2305.09836*, 2023.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Liang Yu, Weiwei Xie, Di Xie, Yulong Zou, Dengyin Zhang, Zhixin Sun, Linghua Zhang, Yue Zhang, and Tao Jiang. Deep reinforcement learning for smart home energy management. *IEEE Internet of Things Journal*, 7(4):2751–2762, 2019.

Wenxuan Zhou, Sujay Bajracharya, and David Held. Plas: Latent action space for offline reinforcement learning. In *Conference on Robot Learning*, 2020.

A What Is Inside the Datasets?

Every dataset is repacked into HDF5 files similar to Fu et al. (2020). The data keys are described in Table 3; along to the typical (s_t, a_t, r_t, d_t) tuples, the metadata is also provided as the datasets' attributes with a comprehensive information about specific trajectories similar to Hambro et al. (2022b). The re-packing script is provided at `https://github.com/corl-team/katakomba/tree/main/scripts/generate_small_dataset.py`.

Table 3: The re-packed datasets constitute of transformed data from Hambro et al. (2022b). Dissimilar the the large scale dataset, the repacked data is now in the format familiar to the ORL practitioners. We also save the metadata for each trajectory, for a comprehensive description, please, see Appendix F in Hambro et al. (2022b).

Name	Type	Shape	Description
`tty_chars`	`np.uint8`	[B, T, H, W]	s_t: The on-screen characters (default screen size 80 x 24).
`tty_colors`	`np.int8`	[B, T, H, W]	s_t: The on-screen colors for each character.
`tty_cursor`	`np.int16`	[B, T, 2]	s_t: The coordinates of the on-screen cursor.
`actions`	`np.uint8`	[B, T]	a_t: The NLE actions the player made in response to the s_t.
`rewards`	`np.int32`	[B, T]	r_t: The difference between in-game scores at states s_t and s_{t-1}. Note that this was used in all implementations of the algorithms provided in Hambro et al. (2022b). We also found that without this reward-shaping, all offline RL algorithms failed completely.
`dones`	`np.uint8`	[B, T]	d_t: An indicator whether the current state is the last one in the trajectory.

B License

Our codebase and repacked datasets are released under the NETHACK GENERAL PUBLIC LICENSE. The original NetHack Learning environment (Küttler et al., 2020) and large-scale datasets (Hambro et al., 2022b) are also released under NETHACK GENERAL PUBLIC LICENSE.

C General Ethic Conduct and Potential Negative Societal Impact

To the best of our knowledge, our work does not present any direct potential negative societal impact.

As of the general ethic conduct, we believe that the most relevant issue to be discussed is the "Consent to use or share the data". Our work is largely built upon both the NetHack Learning Environment (Küttler et al., 2020) and the coresponding large-scale dataset (Hambro et al., 2022b), and as already described in the Appendix B both are distributed under the NETHACK GENERAL PUBLIC LICENSE that explicitly allows for re-usage and re-distribution.

D Resources and Statistics

We used 64 separated computational nodes with 1xA100, 14CPU, 128GB RAM, and the NVMe as long-term storage for all our experiments. All the values reported in the paper were also obtained under this configuration. One can also find more detailed information inside the Weights&Biases logs in the code repository.

Table 4: Scores used for Normalization. You can also find them at `https://github.com/corl-team/katakomba/blob/main/katakomba/utils/scores.py`. For other statistics, please, see Table 2 in the main text.

Tasks	Minimum Score	Maximum Score	Mean Score
Base (Role-Centric)	-	-	-
arc-hum-neu	0.0	138103.0	6636.44
bar-hum-neu	0.0	292342.0	17836.68
cav-hum-neu	0.0	258978.0	12113.87
hea-hum-neu	0.0	64337.0	4068.27
kni-hum-law	0.0	419154.0	14137.06
mon-hum-neu	0.0	171224.0	17456.05
pri-hum-neu	0.0	114269.0	7732.69
ran-hum-neu	0.0	54874.0	8067.99
rog-hum-cha	0.0	68628.0	4818.20
sam-hum-law	0.0	155163.0	11009.36
tou-hum-neu	0.0	59484.0	4211.47
val-hum-neu	16.0	313858.0	18624.77
wiz-hum-neu	0.0	71709.0	5323.48
Extended (Race-Centric)	-	-	-
pri-elf-cha	0.0	83744.0	7109.35
ran-elf-cha	0.0	66690.0	9014.18
wiz-elf-cha	0.0	71664.0	5005.16
arc-dwa-law	0.0	83496.00	5445.69
cav-dwa-law	0.0	161682.0	11893.48
val-dwa-law	0.0	1136591.0	23473.61
arc-gno-neu	0.0	110054.0	5316.57
cav-gno-neu	0.0	142460.0	10083.06
hea-gno-neu	0.0	69566.0	3783.93
ran-gno-neu	0.0	58137.0	6965.04
wiz-gno-neu	0.0	37376.0	4317.51
bar-orc-cha	0.0	164296.0	17594.38
ran-orc-cha	3.0	69244.0	7608.48
rog-orc-cha	0.0	54892.0	4897.69
wiz-orc-cha	0.0	40871.0	5016.74
Complete (Alignment-Centric)	-	-	-
arc-hum-law	2.0	84823.0	5826.35
cav-hum-law	0.0	156966.0	12462.82
mon-hum-law	7.0	190783.0	16091.57
pri-hum-law	0.0	99250.0	6847.99
val-hum-law	0.0	428274.0	26103.03
bar-hum-cha	0.0	164446.0	18228.11
mon-hum-cha	0.0	223997.0	18353.30
pri-hum-cha	0.0	58367.0	8262.56
ran-hum-cha	3.0	62599.0	8378.50
wiz-hum-cha	0.0	55185.0	5316.82

E Hyperparameters

For all algorithms, hyperparameters have been reused from previous work whenever possible. For BC, CQL, and IQL reference values, see Appendix I.4 in the Hambro et al. (2022b). For AWAC, hyperparameters from IQL were reused due to the very similar policy updating scheme. For REM, hyperparameters were taken from the original work (see Agarwal et al. (2020)).

As in Hambro et al. (2022b), and in contrast to the original CQL implementation, we multiply the TD loss by the α coefficient instead of the CQL loss, as we observed better results with such a scheme. We performed a search for $\alpha \in [0.0001, 0.0005, 0.001, 0.01, 0.05, 0.1, 0.5, 1.0]$ with best value $\alpha = 0.0001$.

Table 5: BC hyperparameters.

Parameter	**Value**
optimizer	AdamW (Kingma & Ba, 2014; Loshchilov & Hutter, 2017)
training iterations	500000
batch size	64
sequence length	16
learning rate	3e-4
weight decay	0.0
state encoder	Chaotic-Dwarven-GPT-5(Hambro et al., 2022a,b)
LSTM hidden dim	2048
LSTM layers	2
LSTM dropout	0.0
use previous action	True

Table 6: CQL hyperparameters. Note that in our implementation, the α coefficient multiplies the TD loss.

Parameter	**Value**
optimizer	AdamW (Kingma & Ba, 2014; Loshchilov & Hutter, 2017)
training iterations	500000
batch size	64
sequence length	16
learning rate	3e-4
weight decay	0.0
state encoder	Chaotic-Dwarven-GPT-5(Hambro et al., 2022a,b)
LSTM hidden dim	2048
LSTM layers	2
LSTM dropout	0.0
use previous action	True
tau (τ)	5e-3
gamma (γ)	0.999
reward clip range	[-10.0, 10.0]
alpha (α)	1e-4

Table 7: IQL hyperparameters.

Parameter	Value
optimizer	AdamW (Kingma & Ba, 2014; Loshchilov & Hutter, 2017)
training iterations	500000
batch size	64
sequence length	16
learning rate	3e-4
weight decay	0.0
state encoder	Chaotic-Dwarven-GPT-5(Hambro et al., 2022a,b)
LSTM hidden dim	2048
LSTM layers	2
LSTM dropout	0.0
use previous action	True
tau (τ)	5e-3
gamma (γ)	0.999
reward clip range	[-10.0, 10.0]
expectile	0.8
temperature	1.0
advantage clip max	100

Table 8: AWAC hyperparameters.

Parameter	Value
optimizer	AdamW (Kingma & Ba, 2014; Loshchilov & Hutter, 2017)
training iterations	500000
batch size	64
sequence length	16
learning rate	3e-4
weight decay	0.0
state encoder	Chaotic-Dwarven-GPT-5(Hambro et al., 2022a,b)
LSTM hidden dim	2048
LSTM layers	2
LSTM dropout	0.0
use previous action	True
tau (τ)	5e-3
gamma (γ)	0.999
reward clip range	[-10.0, 10.0]
temperature	1.0
advantage clip max	100

Table 9: REM hyperparameters.

Parameter	Value
optimizer	AdamW (Kingma & Ba, 2014; Loshchilov & Hutter, 2017)
training iterations	500000
batch size	64
sequence length	16
learning rate	3e-4
weight decay	0.0
state encoder	Chaotic-Dwarven-GPT-5(Hambro et al., 2022a,b)
LSTM hidden dim	2048
LSTM layers	2
LSTM dropout	0.0
use previous action	True
tau (τ)	5e-3
gamma (γ)	0.999
reward clip range	[-10.0, 10.0]
ensemble heads	200.0

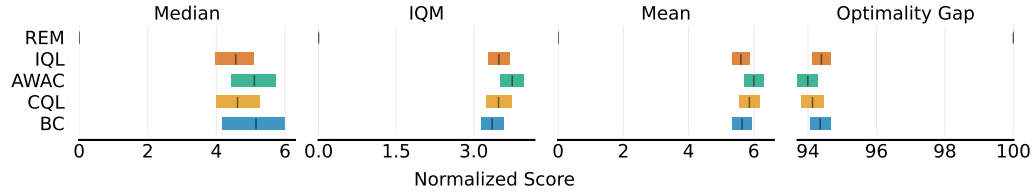
Table 10: The effect of CQL policy constraint strength on the performance. Results, which are averaged across 3 seeds, are sorted based on the final unnormalized game score.

Alpha (α)	**Return**
0.0001	526.72 ± 71.37
0.0005	396.50 ± 39.35
0.001	395.56 ± 139.92
0.05	226.55 ± 196.45
0.01	32.42 ± 43.69
0.1	14.04 ± 10.29
0.5	0.00 ± 0.00
1.0	0.59 ± 0.45

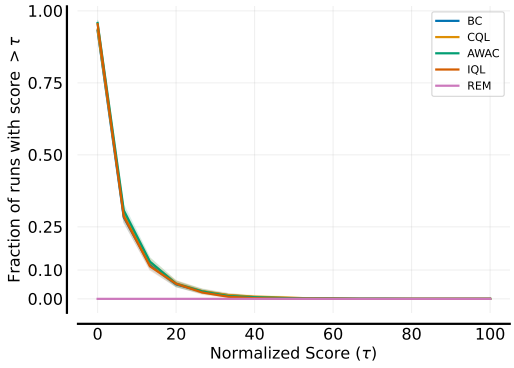
F Results per Benchmark Categories

In this section, we report the results stratified by the introduced categories. If one is willing to inspect specific datasets, we organized all training logs into Weights&Biases public reports, found at `https://wandb.ai/tlab/NetHack/reports`.

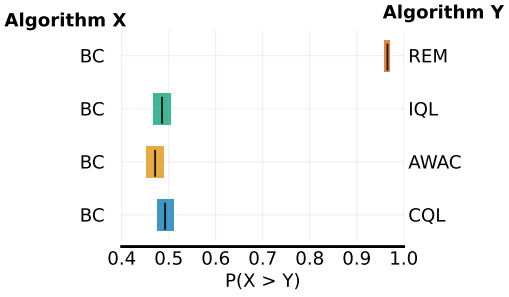
Note that one can find all the evaluation scores (for more than one checkpoint) within the runs and use them for any evaluation tools of interest. Also, we provide convenient scripts for constructing RLiable (Agarwal et al., 2021) graphs based on the provided runs that can be configured for one's purposes as well (see `https://github.com/corl-team/katakomba/tree/main/scripts/rliable_report.py`).



(a) Bootstrapped point estimates.

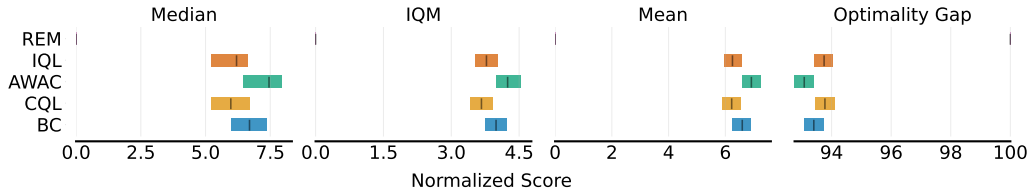


(b) Performance profiles.

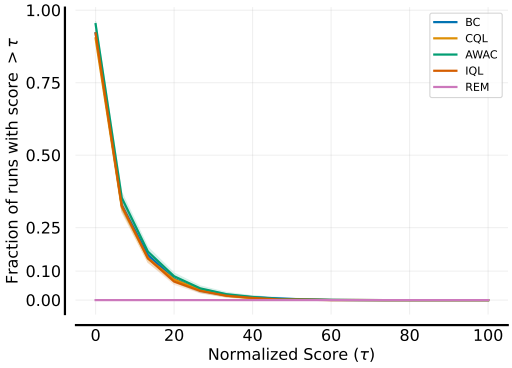


(c) Probability of improvement of BC to other algorithms.

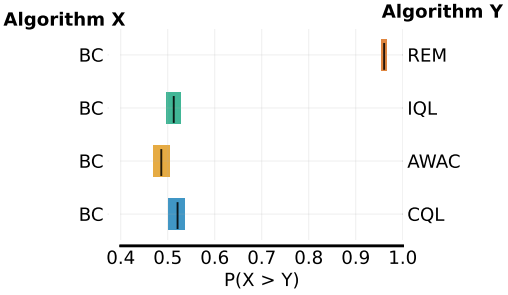
Figure 3: Normalized performance under the Katakomba benchmark for Base datasets. Each algorithm was run for three seeds and evaluated over 50 episodes resulting in 1950 points for constructing these graphs.



(a) Bootstrapped point estimates.

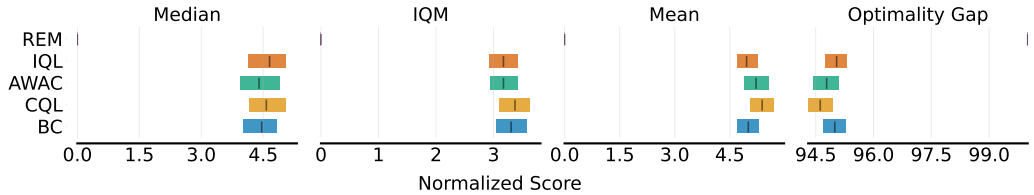


(b) Performance profiles.

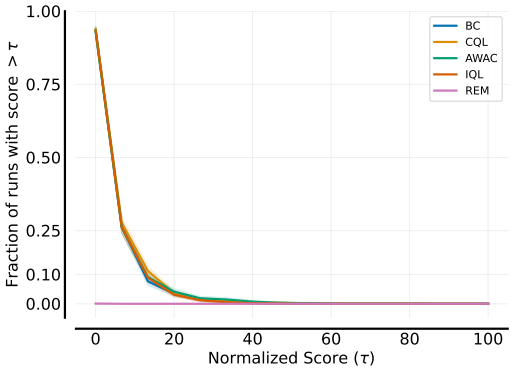


(c) Probability of improvement of BC to other algorithms.

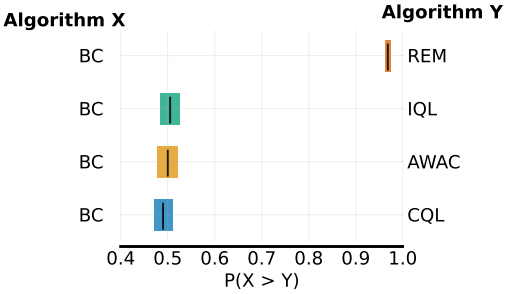
Figure 4: Normalized performance under the Katakomba benchmark for Extended datasets. Each algorithm was run for three seeds and evaluated over 50 episodes resulting in 2250 points for constructing these graphs.



(a) Bootstrapped point estimates.

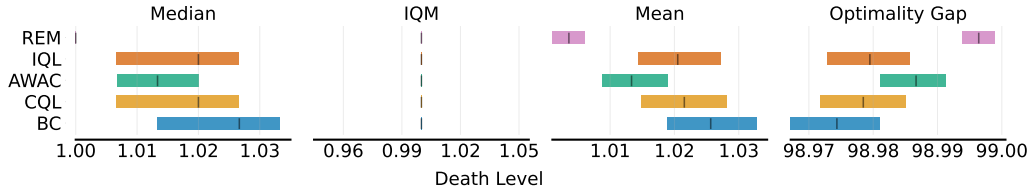


(b) Performance profiles.

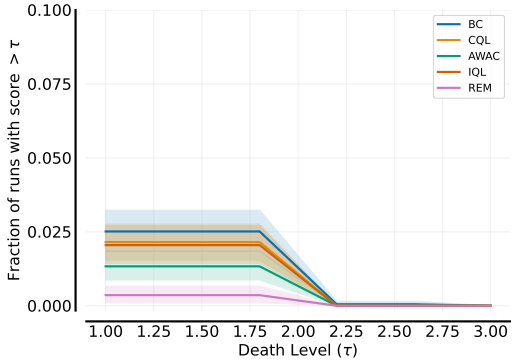


(c) Probability of improvement of BC to other algorithms.

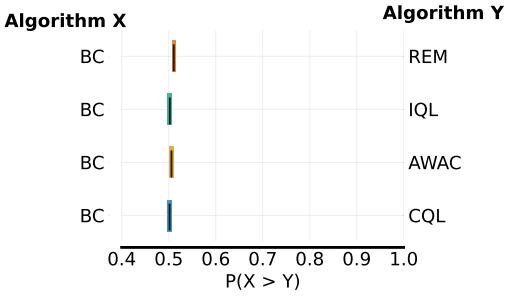
Figure 5: Normalized performance under the Katakomba benchmark for Complete datasets. Each algorithm was run for three seeds and evaluated over 50 episodes resulting in 1500 points for constructing these graphs.



(a) Bootstrapped point estimates.

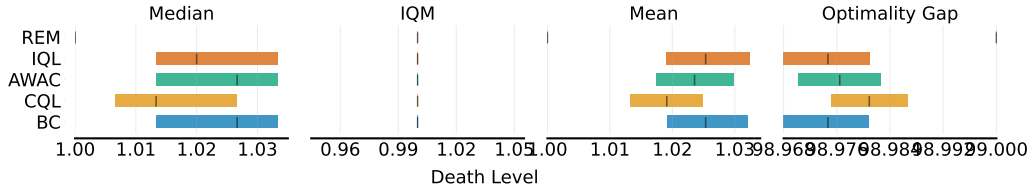


(b) Performance profiles.

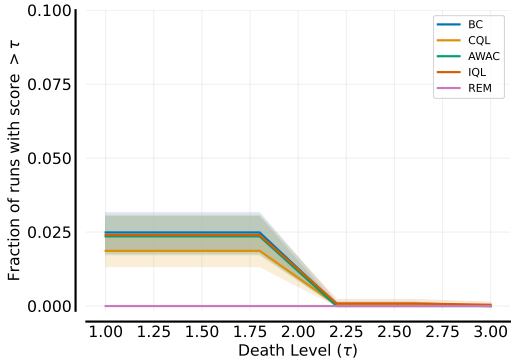


(c) Probability of improvement of BC to other algorithms.

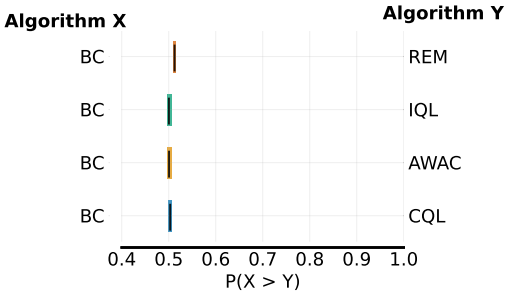
Figure 6: Death levels under the Katakomba benchmark for Base datasets. Each algorithm was run for three seeds and evaluated over 50 episodes resulting in 1950 points for constructing these graphs.



(a) Bootstrapped point estimates.

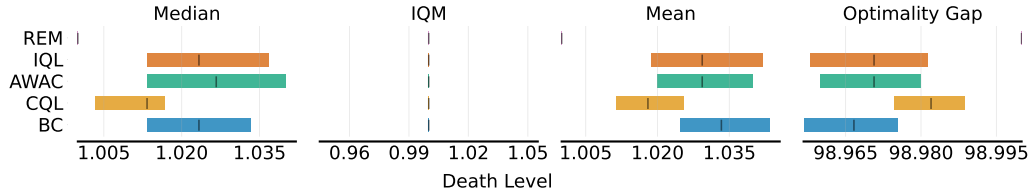


(b) Performance profiles.

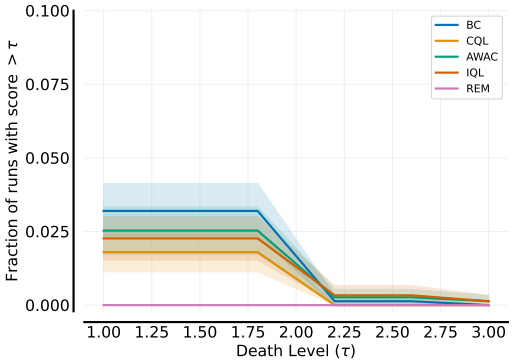


(c) Probability of improvement of BC to other algorithms.

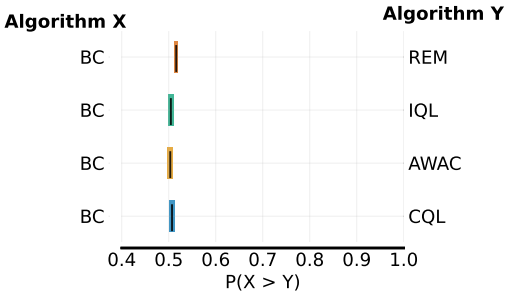
Figure 7: Death level under the Katakomba benchmark for Extended datasets. Each algorithm was run for three seeds and evaluated over 50 episodes resulting in 2250 points for constructing these graphs.



(a) Bootstrapped point estimates.

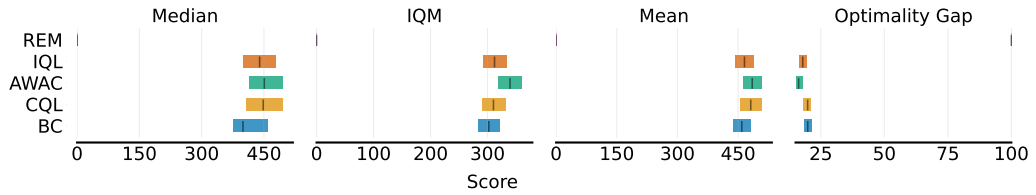


(b) Performance profiles.

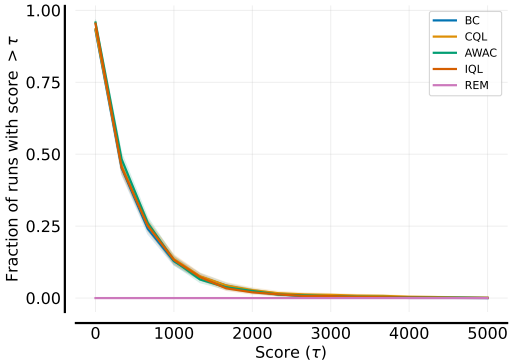


(c) Probability of improvement of BC to other algorithms.

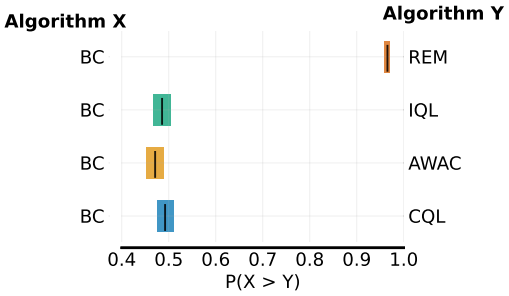
Figure 8: Death levels under the Katakomba benchmark for Complete datasets. Each algorithm was run for three seeds and evaluated over 50 episodes resulting in 1500 points for constructing these graphs.



(a) Bootstrapped point estimates.

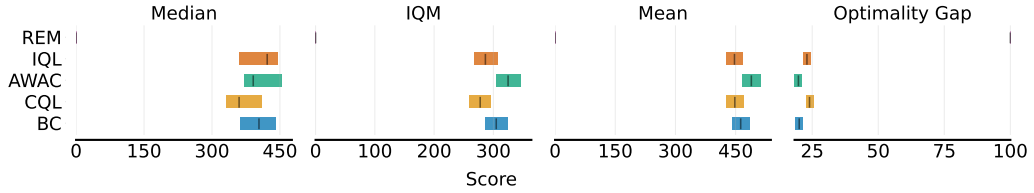


(b) Performance profiles.

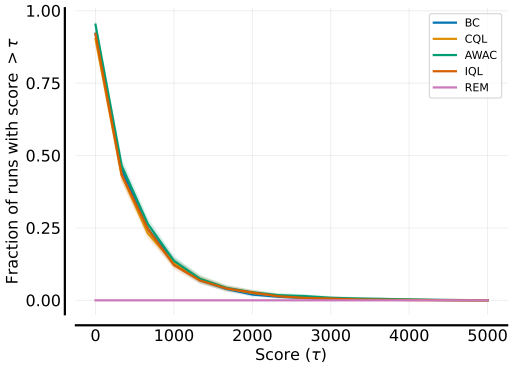


(c) Probability of improvement of BC to other algorithms.

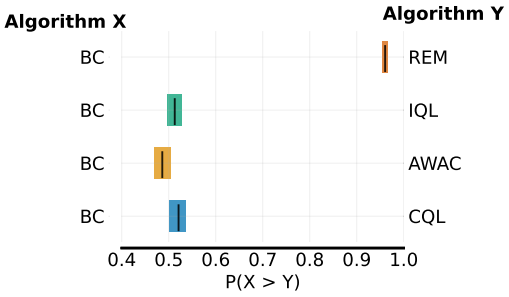
Figure 9: Unnormalized in-game score under the Katakomba benchmark for Base datasets. Each algorithm was run for three seeds and evaluated over 50 episodes resulting in 1950 points for constructing these graphs.



(a) Bootstrapped point estimates.

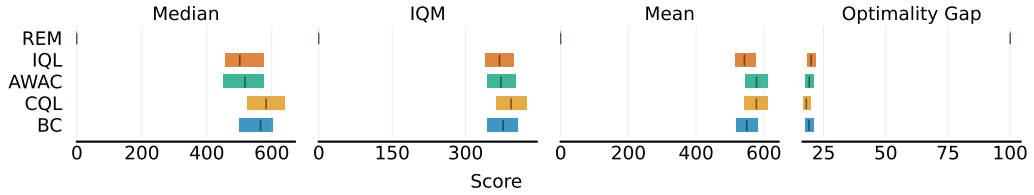


(b) Performance profiles.

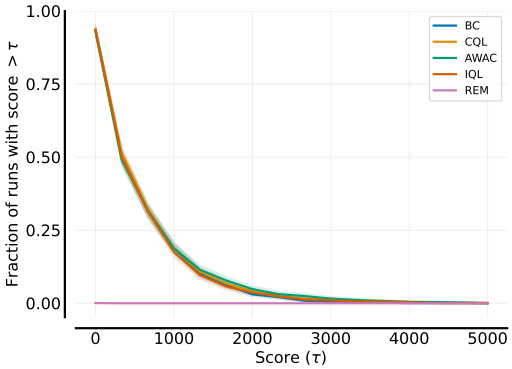


(c) Probability of improvement of BC to other algorithms.

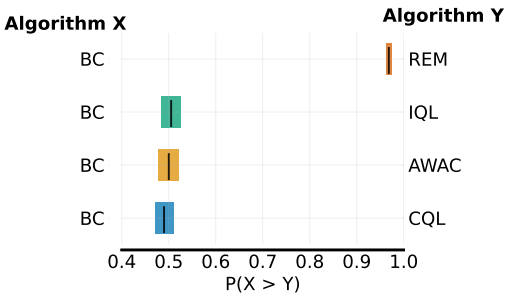
Figure 10: Unnormalized in-game score under the Katakomba benchmark for Extended datasets. Each algorithm was run for three seeds and evaluated over 50 episodes resulting in 2250 points for constructing these graphs.



(a) Bootstrapped point estimates.



(b) Performance profiles.



(c) Probability of improvement of BC to other algorithms.

Figure 11: Unnormalized in-game score under the Katakomba benchmark for Complete datasets. Each algorithm was run for three seeds and evaluated over 50 episodes resulting in 1500 points for constructing these graphs.