A SETUP DETAILS

A.1 R2D2 DISTRIBUTED SYSTEM SETUP

Following R2D2, the distributed system consists of several parts: actors, a replay buffer, a learner, and an evaluator. Additionally, we introduce a centralized batched inference process to make more efficient use of actor resources.

Actors: We use 512 processes to interact with independent copies of the environment, called actors. They send the following information to a central batch inference process:

- x_t: the observation at time t.
- r_{t-1}: the reward at the previous time, initialized with $r_{-1} = 0$.
- a_{t-1}: the action at the previous time, a_{-1} is initialized to 0.
- h_{t-1}: recurrent state at the previous time, is initialized with $h_{-1} = 0$.

They block until they receive $Q(x_t, a; \theta)$. The l-th actor picks a_t using an ϵ_l-greedy policy. As R2D2, the value of ϵ_l is computed following:

$$\epsilon_l = \epsilon^{1+\alpha\frac{l}{L-1}}$$

where $\epsilon = 0.4$ and $\alpha = 7$. After that is computed, the actors send the experienced transition information to the replay buffer.

Batch inference process: This central batch inference process receives the inputs mentioned above from all actors. This process has the same architecture as the learner with weights that are fetched from the learner every 0.5 seconds. The process blocks until a sufficient amount of actors have sent inputs, forming a batch. We use a batch size of 64 in our experiments. After a batch is formed, the neural network of the agent is run to compute $Q(x_t, a, \theta)$ for the whole batch, and these values are sent to their corresponding actors.

Replay buffer: it stores fixed-length sequences of *transitions* $T = (\omega_s)_{s=t}^{t+L-1}$ along with their priorities p_T, where L is the trace length we use. A transition is of the form $\omega_s = (r_{s-1}, a_{s-1}, h_{s-1}, x_s, a_s, h_s, r_s, x_{s+1})$. Concretely, this consists of the following elements:

- r_{s-1}: reward at the previous time.
- a_{s-1}: action done by the agent at the previous time.
- h_{s-1}: recurrent state (in our case hidden state of the LSTM) at the previous time.
- x_s: observation provided by the environment at the current time.
- a_s: action done by the agent at the current time.
- h_s: recurrent state (in our case hidden state of the LSTM) at the current time.
- r_s: reward at the current time.
- x_{s+1}: observation provided by the environment at the next time.

The sequences never cross episode boundaries and they are stored into the buffer in an overlapping fashion, by an amount which we call the *replay period*. Finally, concerning the priorities, we followed the same prioritization scheme proposed by Kapturowski et al. (2018) using a mixture of max and mean of the TD-errors in the sequence with priority exponent $\eta = 0.9$.

Evaluator: the evaluator shares the same network architecture as the learner, with weights that are fetched from the learner every episode. Unlike the actors, the experience produced by the evaluator is not sent to the replay buffer. The evaluator acts in the same way as the actors, except that all the computation is done within the single CPU process instead of delegating inference to the batch inference process. At the end of 5 episodes the results of those 5 episodes are average and reported. In this paper we report the average performance provided by such reports over the last 5% frames (for example, on Atari this is the average of all the performance reports obtained when the total frames consumed by actors is between 190M and 200M frames).

Learner: The learner contains two identical networks called the online and target networks with different weights θ and θ' respectively (Mnih et al., 2015). The target network's weights θ' are updated to θ every 400 optimization steps. θ is updated by executing the following sequence of instructions:

- First, the learner samples a batch of size 64 (batch size) of fixed-length sequences of transitions from the replay buffer, with each transition being of length L: $T_i = (\omega_s^i)_{s=t}^{t+L-1}$.
- Then, a forward pass is done on the online network and the target with inputs $(x_s^i, r_{s-1}^i, a_{s-1}^i, h_{s-1}^i)_{s=t}^{t+H}$ in order to obtain the state-action values $\{(Q(x_s^i, a; \theta), Q(x_s^i, a; \theta')\}$.
- With $\{(Q(x_s^i, a; \theta), Q(x_s^i, a; \theta')\}$, the $Q(\lambda)$ loss is computed.
- The online network is used again to compute the auxiliary contrastive loss.
- Both losses are summed (with by weighting the auxiliary loss by 0.1 as described in C), and optimized with an Adam optimizer.
- Finally, the priorities are computed for the sampled sequence of transitions and updated in the replay buffer.

A.2 V-MPO DISTRIBUTED SETUP

For on-policy training, we used a Podracer setup similar to (Hessel et al., 2021) for fast usage of experience from actors by learners.

TPU learning and acting: As in the Sebulba setup of (Hessel et al., 2021), acting and learning network computations were co-located on a set of TPU chips, split into a ratio of 3 cores used for learning for every 1 core used for inference. This ratio then scales with the total number of chips used.

Environment execution: Due to the size of the recurrent states used by COBERL and stored on the host CPU, it was not possible to execute the environments locally. To proceed we used 64 remote environment servers which serve only to step multiple copies of the environment. 1024 concurrent episodes were processed to balance frames per second, latency between acting and learning, and memory usage of the agent states on the host CPUs.

A.3 COMPUTATION USED

R2D2 We train the agent with a single TPU v2-based learner, performing approximately 5 network updates per second (each update on a mini-batch of 64 sequences of length 80 for Atari and 120 for Control). We use 512 actors, using 4 actors per CPU core, with each one performing ~ 64 environment steps per second on Atari. Finally for the batch inference process a TPU v2, which allows all actors to achieve the speed we have described. In particular, we used 8 TPU cores for learning and 2 for inference.

V-MPO We train the agent with 4 hosts each with 8 TPU v2 cores. Each of the 8 cores per host was split into 6 for learning and 2 for inference. We separately used 64 remote CPU environment servers to step 1024 concurrent environment episodes using the actions returned from inference. The learner updates were made up of a mini-batch of 120 sequences, each of length 95 frames. This setup enabled 4.6 network updates per second, or 53.4k frames per second.

A.4 COMPLEXITY ANALYSIS

As stated, the agent consists of layers of convolutions, transformer layers, and linear layers. Therefore the complexity is $max\{O(n^2 \cdot d), O(k \cdot n \cdot d^2)\}$, where k is the kernel size in the case of convolutions, n is the size of trajectories, and d is the size of hidden layers.

B ARCHITECTURE DESCRIPTION

B.1 ENCODER

As shown in Fig. 1, observations O_i are encoded using an encoder. In this work, the encoder we have used is a ResNet-47 encoder. Those 47 layers are divided in 4 groups which have the following characteristics:

- An initial stride-2 convolution with filter size 3x3 ($1 \cdot 4$ layers).
- Number of residual bottleneck blocks (in order): $(2, 4, 6, 2)$. Each block has 3 convolutional layers with ReLU activations, with filter sizes 1x1, 3x3, and 1x1 respectively ($(2{+}4{+}6{+}2){\cdot}3$ layers).
- Number of channels for the last convolution in each block: $(64, 128, 256, 512)$.
- Number of channels for the non-last convolutions in each block: $(16, 32, 64, 128)$.
- Group norm is applied after each group, with a group size of 8.

After this observation encoding step, a final 2-layer MLP with ReLU activations of sizes $(512, 448)$ is applied. The previous reward and one-hot encoded action are concatenated and projected with a linear layer into a 64-dimensional vector. This 64-dimensional vector is concatenated with the 448-dimensional encoded input to have a final 512-dimensional output.

B.2 TRANSFORMER

As described in Section 2, the output of the encoder is fed to a Gated Transformer XL. For Atari and Control, the transformer has the following characteristics:

- Number of layers: 8.
- Memory size: 64.
- Hidden dimensions: 512.
- Number of heads: 8.
- Attention size: 64.
- Output size: 512.
- Activation function: GeLU.

For DmLab the transformer has the following characteristics:

- Number of layers: 12.
- Memory size: 256 for COBERL and 512 for gTrXL.
- Hidden dimensions: 128.
- Number of heads: 4.
- Attention size: 64.
- Output size: 512.
- Activation function: ReLU.

the GTrXL baseline is identical, but with a Memory size of 512.

B.3 LSTM AND VALUE HEAD

For both R2D2 and V-MPO the outputs of the transformer and encoder are passed through a GRU transform to obtain a 512-dimensional vector. After that, an LSTM with 512 hidden units is applied. The the value function is estimated differently depending on the RL algorithm used.

R2D2 Following the LSTM, a Linear layer of size 512 is used, followed by a ReLU activation. Finally, to compute the Q values from that 512 vector a dueling head is used, as in Kapturowski et al. (2018), a dueling head is is used, which requires a linear projection to the number of actions of the task, and another projection to a unidimensional vector.

V-MPO Following the LSTM, a 2 layer MLP with size 512 and 30 (i.e. the number of levels in DMLab) is used. In the MLP we use ReLU activation. As we are interested in the multi-task setting where a single agent learns a large number of tasks with differing reward scales, we used PopArt (van Hasselt et al., 2016) for the value function estimation (see Table. 13 for details).

B.4 Critic Function

For DmLab-30 (V-MPO), we used a 2 layer MLP with hidden sizes 512 and 128. For Atari and Control Suite (R2D2) we used a single layer of size 512.

C HYPERPARAMETERS

For the experiments in Atari57 and the DeepMind Control suite, COBERL uses the R2D2 distributed setup. We use 512 actors for all our experiments. We do not constrain the amount of replay done for each experience trajectory that actors deposit in the buffer. However, we have found empirical replay frequency per data point to be close among all our experiments (with an expected value of 1.5 samples per data point). We use a separate evaluator process that shares weights with our learner in order to measure the performance of our agents. We report scores at the end of training. The hyperparameters and architecture we choose for these two domains are the same with two exceptions: i) we use a shorter trace length for Atari (80 instead of 120) as the environment does not require a long context to inform decisions, and ii) we use a squashing function on Atari and the Control Suite to transform our Q values (as done in (Kapturowski et al., 2018)) since reward structures vary highly in magnitude between tasks.

C.1 ATARI AND DMLAB PRE-PROCESSING

We use the commonly used input pre-processing on Atari and DMLab frames, shown on Tab. 8. One difference with the original work of Mnih et al. (2015), is that we do not use frame stacking, as we rely on our memory systems to be able to integrate information from the past, as done in Kapturowski et al. (2018). ALE is publicly available at `https://github.com/mgbellemare/Arcade-Learning-Environment`.

Hyperparameter	**Value**
Max episode length	30 min
Num. action repeats	4
Num. stacked frames	1
Zero discount on life loss	$false$
Random noops range	30
Sticky actions	$false$
Frames max pooled	3 and 4
Grayscaled/RGB	Grayscaled
Action set	Full

Table 7: Atari pre-processing hyperparameters.

C.2 CONTROL SUITE PRE-PROCESSING

As mentioned in 4, we use no pre-processing on the frames received from the control environment.

C.3 DMLAB PRE-PROCESSING

Hyperparameter	**Value**
Num. action repeats	4
Num. stacked frames	1
Grayscaled/RGB	RGB
Image width	96
Image height	72
Action set	as in Parisotto et al. (2020)

Table 8: DmLab pre-processing hyperparameters.

C.4 CONTROL ENVIRONMENT DISCRETIZATION

As mentioned, we discretize the space assigning two possibilities (1 and -1) to each dimension and taking the Cartesian product of all dimensions, which results in 2^n possible actions. For the `cartpole` tasks, we take a diagonal approach, utilizing each unit vector in the action space and

then dividing each unit vector into 5 possibilities with the non-zero coordinate ranging from -1 to 1. The amount of actions this results in is outlined on Tab. 9.

Task	Action space size	Total amount of actions
Acrobot	1	2
Cartpole	1	5
Cup	2	4
Cheetah	6	64
Finger	2	4
Fish	5	32
Pendulum	1	2
Reacher	2	4
Swimmer	5	32
Walker	6	64

Table 9: Control discretization action spaces.

C.5 Hyperparameters Used

We list all hyperparameters used here for completeness.

We started by optimizing the hyperparameters of GTrXL highlighted in bold in Tab 10 by doing a sweep over 10 Atari games: Seaquest, Qbert, Frostbite, Ms Pacman, Space Invaders, Gravitar, Solaris, Hero, Venture, Montezuma Revenge. Following this, the hyperparameters were kept fixed throughout all the experiments.

Table 10 reports all the hyperparameters of the R2D2 experiments, both the fixed ones and the ones with the ranges over which we did the sweep. The fixed hyper-parameters were taken from Kapturowski et al. (2018). We then choose a set of hyper-paramters (both for the architecture and the algorithm) to sweep over to maximise the performance of gTrXL in this off-policy setting, given that there was not prior literature on this. The hyper-paramters over which we did the sweep for the algorithm were chose in accordance to Kapturowski et al. (2018) and the related sweep. And for the architecture hyper-parameters we based our choice on Parisotto et al. (2020). We believe that in this way we ensure to have a properly tuned baseline that enforces a fair comparison. Table 11 reports the chosen hyperparameters that we found to optimize the performance of GTrXL on the 10 Atari games. We then moved to CoBERL. CoBERL, in comparison to the baseline GTrXL has two extra hyperparamters: 'Contrastive loss weight' and 'Contrastive loss mask rate'. The former was tuned, whereas the latter we kept equal to 0.15 as done in [11]. Consequently, we re-ran the same procedure as before, but we fixed all the previous hyperparameters optimized for GTrXL (see Tab. 11 and we perform a grid search over the same 10 Atari games to find the value of 'Contrastive loss weight' that maximized performance. The values over which we did the search are 0.01, 0.1 and 1. We ended-up picking 1, although the difference between 0.1 and 1 was minimal. Table 12 reports the 2 extra parameters used for CoBERL.

For DmLAB we optimized the hyperparameters of GTrXL on all the 30 games. Table 13 reports both the fixed ones and the ones with the ranges over which. The fixed hyper-parameters were taken directly from Parisotto et al. (2020) and we sweep over ''Epsilon Alpha", "Target Update Period" and 'Memory size' to make sure we maximised performance of this baseline. Table 14 reports the chosen hyperparameters that we found to optimize the performance of GTrXL on DmLAB. We then moved to CoBERL. Again, by keeping fixed all the previous hyperparameters optimized for GTrXL we perform a grid search over all the 30 games to find the value of 'Contrastive loss weight' that maximized performance. The values over which we did the search were 0.1 and 1. We did not find any significant difference between the two values, so we left it equal to 1 such that the two losses would have the same effect. Table 15 reports the 2 extra parameters used for CoBERL and the reduced memory size, in accordance with our hypothesis that the LSTM on top of Transformer would help reducing the size of the memory especially in last.

Hyperparameter	**Value**
Optimizer	Adam
Learning rate	$\{\mathbf{0.0001}, \mathbf{0.0003}\}$
Q's λ	$\{\mathbf{0.8}, \mathbf{0.9}\}$
Adam epsilon	10^{-7}
Adam beta1	0.9
Adam beta2	0.999
Adam clip norm	40
Q-value transform (non-DMLab)	$h(x) = sign(x)(\sqrt{\lvert x\rvert + 1} - 1) + \epsilon x$
Q-value transform (DMLab)	$h(x) = x$
Discount factor	0.997
Batch size	32
Trace length (Atari)	80
Trace length (non-Atari)	120
Replay period (Atari)	40
Replay period (non-Atari)	60
Replay capacity	80000 sequences
Replay priority exponent	0.9
Importance sampling exponent	0.6
Minimum sequences to start replay	5000
Target Q-network update period	400
Evaluation ϵ	0.01
Target ϵ	0.01
Number of layers	$\{\mathbf{6}, \mathbf{8}, \mathbf{12}\}$
Memory size	$\{\mathbf{64}, \mathbf{128}\}$
Number of heads	$\{\mathbf{4}, \mathbf{8}\}$
Attention size	$\{\mathbf{64}, \mathbf{128}\}$

Table 10: GTrXL Hyperparameters used in all the R2D2 experiments with range of sweep.

Hyperparameter	**Value**
Optimizer	Adam
Learning rate	$\mathbf{0.0003}$
Q's λ	$\mathbf{0.8}$
Adam epsilon	10^{-7}
Adam beta1	0.9
Adam beta2	0.999
Adam clip norm	40
Q-value transform (non-DMLab)	$h(x) = sign(x)(\sqrt{\lvert x\rvert + 1} - 1) + \epsilon x$
Q-value transform (DMLab)	$h(x) = x$
Discount factor	0.997
Batch size	32
Trace length (Atari)	80
Trace length (non-Atari)	120
Replay period (Atari)	40
Replay period (non-Atari)	60
Replay capacity	80000 sequences
Replay priority exponent	0.9
Importance sampling exponent	0.6
Minimum sequences to start replay	5000
Target Q-network update period	400
Evaluation ϵ	0.01
Target ϵ	0.01
Number of layers	$\mathbf{8}$
Memory size	$\mathbf{64}$
Number of heads	$\mathbf{8}$
Attention size	$\mathbf{64}$

Table 11: GTrXL Hyperparameters choosen for all the R2D2 experiments.

Hyperparameter	**Value**
Contrastive loss weight	1.0
Contrastive loss mask rate	0.15

Table 12: Extra hyperparameters for CoBERL for the R2D2 experiments

Hyperparameter	**Value**
Batch Size	120
Unroll Length	95
Discount	0.99
Target Update Period	$\{\mathbf{10},\ \mathbf{20},\ \mathbf{50}\}$
Action Repeat	4
Initial η	1.0
Initial α	5.0
ϵ_η	0.1
ϵ_α	$\{\mathbf{0.001},\ \mathbf{0.002}\}$
Popart Step Size	0.001
Memory size	$\{\mathbf{256}, \mathbf{512}\}$

Table 13: GTrXL Hyperparameters used in all the VMPO experiments with range of sweep.

Hyperparameter	**Value**
Batch Size	120
Unroll Length	95
Discount	0.99
Target Update Period	50
Action Repeat	4
Initial η	1.0
Initial α	5.0
ϵ_η	0.1
ϵ_α	0.002
Popart Step Size	0.001
Memory size	$\mathbf{512}$

Table 14: GTrXL Hyperparameters choosen for all the VMPO experiments.

Hyperparameter	**Value**
Contrastive loss weight	1.0
Contrastive loss mask rate	0.15
Memory size	256

Table 15: Extra hyperparameters for CoBERL for the VMPO experiments

D ADDITIONAL ABLATIONS

Table 16 shows the results of several gating mechanisms that we have investigated. As we can observe the GRU gate is a clear improvement especially on DMLab, only being harmful in median on the reduced ablation set of Atari games.

		CoBERL	'Sum' gate	'Concat' gate	w/o Gate
DMLab	Mean	$\mathbf{113.39\% \pm 3.64\%}$	$108.70\% \pm 3.23\%$	$106.31\% \pm 5.37\%$	$84.07\% \pm 5.71\%$
	Median	$\mathbf{112.02\%}$	108.95%	108.54%	104.33%
Atari	Mean	$\mathbf{698.0\% \pm 53.84\%}$	$548.66\% \pm 11.16\%$	$653.20\% \pm 59.13\%$	$591.33\% \pm 91.25\%$
	Median	276.6%	$\mathbf{437.85\%}$	325.96%	320.09%

Table 16: Gate ablations. Human normalized scores on Atari-57 ablation tasks and DMLab tasks. For the mean we include standard error over seeds.

DM Suite	CoBERL	R2D2-GTrXL	R2D2	D4PG-Pixels	CURL	Dreamer	Pixel SAC
acrobot swingup	**359.75 ± 3.47**	215.39 ± 122.82	327.16 ± 5.35	81.7 ± 4.4	-	-	-
fish swim	**624.40 ± 54.91**	91.32 ± 277.15	345.63 ± 227.44	72.2 ± 3.0	-	-	-
fish upright	**942.33 ± 6.12**	849.52 ± 23.01	936.09 ± 11.58	405.7 ± 19.6	-	-	-
pendulum swingup	**836.63 ± 9.77**	743.65 ± 52.44	831.86 ± 61.54	680.9 ± 41.9	-	-	-
swimmer swimmer6	**447.60 ± 51.51**	225.97 ± 60.67	329.61 ± 26.77	194.7 ± 15.9	-	-	-
finger spin	**985.05 ± 1.58**	977.41 ± 8.91	980.85 ± 0.67	**985.7 ± 0.6**	926 ± 45	796 ± 183	179 ± 166
reacher easy	**983.05 ± 2.47**	981.64 ± 1.99	**982.28 ± 9.30**	967.4 ± 4.1	929 ± 44	793 ± 164	145 ± 30
cheetah run	**525.06 ± 44.59**	115.15 ± 133.95	365.45 ± 50.40	**523.8 ± 6.8**	518 ± 28	**570 ± 253**	197 ± 15
walker walk	780.54 ± 26.48	595.96 ± 77.59	687.18 ± 18.15	**968.3 ± 1.8**	902 ± 43	897 ± 49	42 ± 12
ball in cup catch	978.28 ± 6.56	975.21 ± 1.77	**980.54 ± 1.94**	980.5 ± 0.5	959 ± 27	879 ± 87	312 ± 63
cartpole swingup	798.66 ± 7.72	837.31 ± 4.15	816.23 ± 2.93	**862.0 ± 1.1**	841 ± 45	762 ± 27	419 ± 40
cartpole swingup sparse	732.51 ± 18.60	747.94 ± 8.61	**762.57 ± 6.71**	482.0 ± 56.6	-	-	-

Table 17: Results on tasks in the DeepMind Control Suite. CoBERL, R2D2-GTrXL, R2D2, and D4PG-Pixels are trained on 100M frames, while CURL, Dreamer, and Pixel SAC are trained on 500k frames. We show these three other approaches as reference and not as a directly comparable baseline.

E GAME SCORES

Atari (ablation games)	CoBERL	R2D2-gTrXL	R2D2	CoBERL-auxiliary loss
beam rider	22246.68 ± 7078.73	**61478.38 ± 27336.64**	34708.13 ± 11513.28	16318.78 ± 12438.73
enduro	2312.58 ± 35.59	2173.92 ± 135.85	**2346.15 ± 12.69**	2300.61 ± 75.25
breakout	**421.88 ± 1.50**	393.88 ± 31.14	336.19 ± 119.23	420.72 ± 9.86
pong	**21.00 ± 0.00**	**21.00 ± 0.00**	**21.00 ± 0.00**	**21.00 ± 0.00**
qbert	36932.00 ± 5498.71	21616.94 ± 3377.11	25129.37 ± 7139.03	**50362.28 ± 14109.22**
seaquest	167183.79 ± 112932.87	**326714.40 ± 51904.47**	124417.45 ± 128759.58	174867.68 ± 123876.30
space invaders	**34112.19 ± 10216.42**	21669.97 ± 6219.26	3712.64 ± 82.30	20192.85 ± 20815.81

Atari (ablation games)	CoBERL	CoBERL with LSTM before	CoBERL w/o LSTM
beam rider	22246.68 ± 7078.73	19233.70 ± 9849.79	**54345.65 ± 8111.23**
enduro	**2312.58 ± 35.59**	2304.59 ± 47.39	2227.61 ± 110.91
breakout	421.88 ± 1.50	**424.05 ± 6.83**	422.69 ± 7.58
pong	**21.00 ± 0.00**	21.00 ± 0.00	21.00 ± 0.00
qbert	**36932.00 ± 5498.71**	34773.82 ± 8972.64	33854.14 ± 5762.10
seaquest	**167183.79 ± 112932.87**	94254.91 ± 57966.74	151011.38 ± 93597.94
space invaders	**34112.19 ± 10216.42**	16980.01 ± 18410.59	4098.37 ± 938.44

Atari (ablation games)	CoBERL	No Skip Conn.	Sum gate	Concat
beam rider	22246.68 ± 7078.73	53379.23 ± 6229.05	**72882.42 ± 21239.63**	51371.38 ± 12560.94
enduro	**2312.58 ± 35.59**	2083.99 ± 382.91	2247.23 ± 82.22	2288.13 ± 58.59
breakout	**421.88 ± 1.50**	285.54 ± 181.62	403.36 ± 53.69	357.52 ± 72.63
pong	**21.00 ± 0.00**	**21.00 ± 0.00**	**21.00 ± 0.00**	**21.00 ± 0.00**
qbert	36932.00 ± 5498.71	31786.70 ± 5012.64	33807.00 ± 4294.93	**43487.35 ± 14518.46**
seaquest	167183.79 ± 112932.87	180119.87 ± 56159.52	294976.55 ± 41827.55	**399817.75 ± 36729.78**
space invaders	**34112.19 ± 10216.42**	27620.20 ± 17966.22	10387.59 ± 2858.63	20933.98 ± 13072.95

Atari (ablation games)	CoBERL with CURL	CoBERL with SimCLR
beam rider	25998.99 ± 18557.89	**27631.98 ± 31908.43**
enduro	2331.72 ± 36.46	**2344.57 ± 6.37**
breakout	328.38 ± 48.22	**381.42 ± 46.31**
pong	19.31 ± 2.38	**21.00 ± 0.00**
qbert	16073.04 ± 321.56	**18531.50 ± 3906.27**
seaquest	145340.79 ± 31778.00	**233181.04 ± 146617.39**
space invaders	21621.09 ± 15373.01	**28785.96 ± 18516.74**

Control Suite	CoBERL	gTrXL	R2D2
acrobot swingup	**359.75 ± 3.47**	215.39 ± 122.82	327.16 ± 5.35
fish swim	**624.40 ± 54.91**	91.32 ± 277.15	345.63 ± 227.44
fish upright	**942.33 ± 6.12**	849.52 ± 23.01	936.09 ± 11.58
pendulum swingup	**836.63 ± 9.77**	743.65 ± 52.44	831.86 ± 61.54
swimmer swimmer6	**447.60 ± 51.51**	225.97 ± 60.67	329.61 ± 26.77
finger spin	**985.05 ± 1.58**	977.41 ± 8.91	980.85 ± 0.67
reacher easy	**983.05 ± 2.47**	981.64 ± 1.99	982.28 ± 9.30
cheetah run	**525.06 ± 44.59**	115.15 ± 133.95	365.45 ± 50.40
walker walk	**780.54 ± 26.48**	595.96 ± 77.59	687.18 ± 18.15
ball in cup catch	978.28 ± 6.56	975.21 ± 1.77	**980.54 ± 1.94**
cartpole swingup	798.66 ± 7.72	**837.31 ± 4.15**	816.23 ± 2.93
cartpole swingup sparse	732.51 ± 18.60	747.94 ± 8.61	**762.57 ± 6.71**

Atari-57	CoBERL	R2D2-gTrXL	R2D2
alien	10229.89 ± 7932.26	9655.65 ± 1819.53	**10718.62 ± 4599.62**
amidar	2656.00 ± 966.37	**3883.37 ± 640.23**	2142.70 ± 241.96
assault	5469.38 ± 2607.77	10242.96 ± 4234.94	**13817.82 ± 1503.31**
asterix	**980283.74 ± 25765.32**	666449.34 ± 173208.21	724279.78 ± 195506.02
asteroids	**108985.96 ± 66922.29**	104932.39 ± 26450.55	74148.67 ± 49306.65
atlantis	**1091347.38 ± 37782.18**	979337.34 ± 8121.73	983110.27 ± 41978.78
bank heist	1117.91 ± 790.60	1318.51 ± 48.90	**1328.12 ± 456.18**
battle zone	77501.43 ± 39229.60	**98554.44 ± 43709.86**	94385.32 ± 13045.67
beam rider	22246.68 ± 7078.73	**61478.38 ± 27336.64**	34708.13 ± 11513.28
berzerk	**1756.21 ± 278.30**	626.60 ± 156.19	1466.54 ± 422.70
bowling	**184.32 ± 44.08**	42.33 ± 59.86	96.33 ± 30.66
boxing	**100.00 ± 0.00**	100.00 ± 0.00	99.71 ± 0.40
breakout	**421.88 ± 1.50**	393.88 ± 31.14	336.19 ± 119.23
centipede	66669.74 ± 6479.47	**76325.85 ± 16594.15**	74513.40 ± 12696.62
chopper command	**506146.56 ± 303260.36**	36993.73 ± 35157.10	33945.00 ± 13504.01
crazy climber	120806.63 ± 57107.29	120684.30 ± 6872.59	**157946.90 ± 42953.93**
defender	410044.34 ± 8847.63	293804.22 ± 72002.55	**462135.87 ± 12678.03**
demon attack	**137934.41 ± 5473.26**	131514.55 ± 9979.81	117580.02 ± 21157.45
double dunk	**24.00 ± 0.00**	19.93 ± 2.89	24.00 ± 0.00
enduro	2312.58 ± 35.59	2173.92 ± 135.85	**2346.15 ± 12.69**
fishing derby	**52.89 ± 2.89**	42.41 ± 11.36	49.92 ± 6.79
freeway	**34.00 ± 0.00**	34.00 ± 0.00	33.67 ± 0.47
frostbite	**8723.86 ± 1321.24**	7323.81 ± 2407.78	6909.40 ± 747.58
gopher	90684.67 ± 10244.87	**104482.57 ± 5853.11**	100203.22 ± 20908.66
gravitar	**6315.32 ± 223.45**	4282.56 ± 1705.21	5643.37 ± 631.75
hero	**20786.34 ± 139.43**	14010.16 ± 173.98	17996.97 ± 2841.71
ice hockey	19.82 ± 21.47	17.74 ± 11.55	**22.90 ± 10.41**
jamesbond	5576.58 ± 2595.65	**7962.54 ± 873.51**	7727.43 ± 2489.21
kangaroo	12173.70 ± 3129.91	13520.79 ± 2068.49	**14436.53 ± 116.90**
krull	37813.60 ± 18826.63	**58459.18 ± 38117.13**	12285.18 ± 572.14
kung fu master	**126648.43 ± 5685.80**	70772.67 ± 28598.85	102387.20 ± 17781.04
montezuma revenge	**833.33 ± 1178.51**	0.00 ± 0.00	133.33 ± 188.56
ms pacman	**11295.44 ± 4623.95**	11146.67 ± 902.33	9893.29 ± 1172.58
name this game	25044.23 ± 6659.51	**26944.23 ± 1315.96**	24348.53 ± 1917.48
phoenix	**514890.69 ± 169407.85**	322625.85 ± 92449.70	194688.45 ± 178633.63
pitfall	**0.00 ± 0.00**	0.00 ± 0.00	0.00 ± 0.00
pong	**21.00 ± 0.00**	**21.00 ± 0.00**	**21.00 ± 0.00**
private eye	10520.09 ± 4986.07	**15057.95 ± 42.35**	4988.65 ± 6938.00
qbert	**36932.00 ± 5498.71**	21616.94 ± 3377.11	25129.37 ± 7139.03
riverraid	24894.12 ± 3079.76	23368.64 ± 2389.28	**28057.87 ± 576.75**
road runner	279422.07 ± 223362.40	**518566.46 ± 62912.72**	0.00 ± 0.00
robotank	**82.36 ± 8.68**	50.70 ± 33.29	67.41 ± 21.91
seaquest	167183.79 ± 112932.87	**326714.40 ± 51904.47**	124417.45 ± 128759.58
skiing	-29958.52 ± 2.39	**-22984.50 ± 9882.73**	-29963.55 ± 1.82
solaris	**4931.47 ± 2344.05**	1661.77 ± 1032.29	2610.71 ± 1573.21
space invaders	**34112.19 ± 10216.42**	21669.97 ± 6219.26	3712.64 ± 82.30
star gunner	**106292.95 ± 29670.23**	104395.14 ± 16821.38	93412.15 ± 10284.46
surround	**9.30 ± 0.50**	9.25 ± 0.54	8.78 ± 0.50
tennis	**15.61 ± 11.17**	8.00 ± 11.31	0.00 ± 0.00
time pilot	**39261.92 ± 7400.70**	14303.91 ± 1695.58	23611.05 ± 3357.51
tutankham	38.79 ± 49.73	23.11 ± 16.91	**84.30 ± 46.88**
up n down	397836.59 ± 111520.35	289177.29 ± 135467.48	**422332.40 ± 41201.70**
venture	**1873.20 ± 30.56**	1782.00 ± 83.88	1640.10 ± 141.12
video pinball	**276228.36 ± 133392.82**	58865.88 ± 51845.02	206756.24 ± 72958.76
wizard of wor	**18707.03 ± 21128.79**	14862.25 ± 8267.18	16548.31 ± 10862.27
yars revenge	**322255.03 ± 171716.17**	201576.03 ± 23183.30	316415.89 ± 176059.44
zaxxon	14420.27 ± 10309.48	20132.38 ± 3941.10	**31116.74 ± 1966.53**

DmLab Levels	coberl	gtrxl
rooms collect good objects test	$\mathbf{9.75 \pm 0.12}$	9.67 ± 0.22
rooms exploit deferred effects test	$\mathbf{56.56 \pm 4.01}$	54.87 ± 0.87
rooms select nonmatching object	$\mathbf{64.04 \pm 10.51}$	58.54 ± 4.07
rooms watermaze	$\mathbf{58.35 \pm 1.65}$	50.27 ± 3.37
rooms keys doors puzzle	32.74 ± 5.62	$\mathbf{34.42 \pm 8.53}$
language select described object	$\mathbf{627.55 \pm 0.72}$	624.50 ± 14.14
language select located object	$\mathbf{614.88 \pm 3.14}$	578.00 ± 2.82
language execute random task	$\mathbf{226.23 \pm 27.81}$	191.70 ± 15.61
language answer quantitative question	$\mathbf{330.66 \pm 5.03}$	293.00 ± 9.89
lasertag one opponent large	$\mathbf{14.16 \pm 4.48}$	10.50 ± 4.24
lasertag three opponents large	$\mathbf{31.39 \pm 4.48}$	28.83 ± 5.89
lasertag one opponent small	$\mathbf{29.83 \pm 1.52}$	27.75 ± 3.88
lasertag three opponents small	$\mathbf{46.0 \pm 1.40}$	35.0 ± 0.01
natlab fixed large map	$\mathbf{44.66 \pm 7.28}$	42.66 ± 13.85
natlab varying map regrowth	$\mathbf{28.33 \pm 6.93}$	25.39 ± 8.39
natlab varying map randomized	$\mathbf{45.88 \pm 9.02}$	32.08 ± 12.31
platforms hard	$\mathbf{57.53 \pm 7.17}$	35.00 ± 25.92
platforms random	$\mathbf{86.03 \pm 0.45}$	73.45 ± 2.64
psychlab continuous recognition	$\mathbf{57.38 \pm 4.14}$	53.73 ± 5.42
psychlab arbitrary visuomotor mapping	51.36 ± 1.94	$\mathbf{58.02 \pm 0.71}$
psychlab sequential comparison	31.43 ± 1.42	31.83 ± 0.94
psychlab visual search	79.91 ± 0.14	79.58 ± 0.58
explore object locations small	$\mathbf{83.71 \pm 6.87}$	74.10 ± 7.10
explore object locations large	$\mathbf{64.72 \pm 3.84}$	61.12 ± 2.65
explore obstructed goals small	$\mathbf{260.37 \pm 6.97}$	234.16 ± 10.60
explore obstructed goals large	$\mathbf{114.44 \pm 2.45}$	76.25 ± 8.83
explore goal locations small	$\mathbf{370.00 \pm 2.50}$	339.10 ± 3.43
explore goal locations large	$\mathbf{142.50 \pm 9.34}$	127.50 ± 14.14
explore object rewards few	$\mathbf{76.93 \pm 13.61}$	70.23 ± 0.61
explore object rewards many	$\mathbf{105.80 \pm 3.37}$	95.50 ± 3.53

F LEARNING CURVES

F.1 ATARI LEARNING CURVES

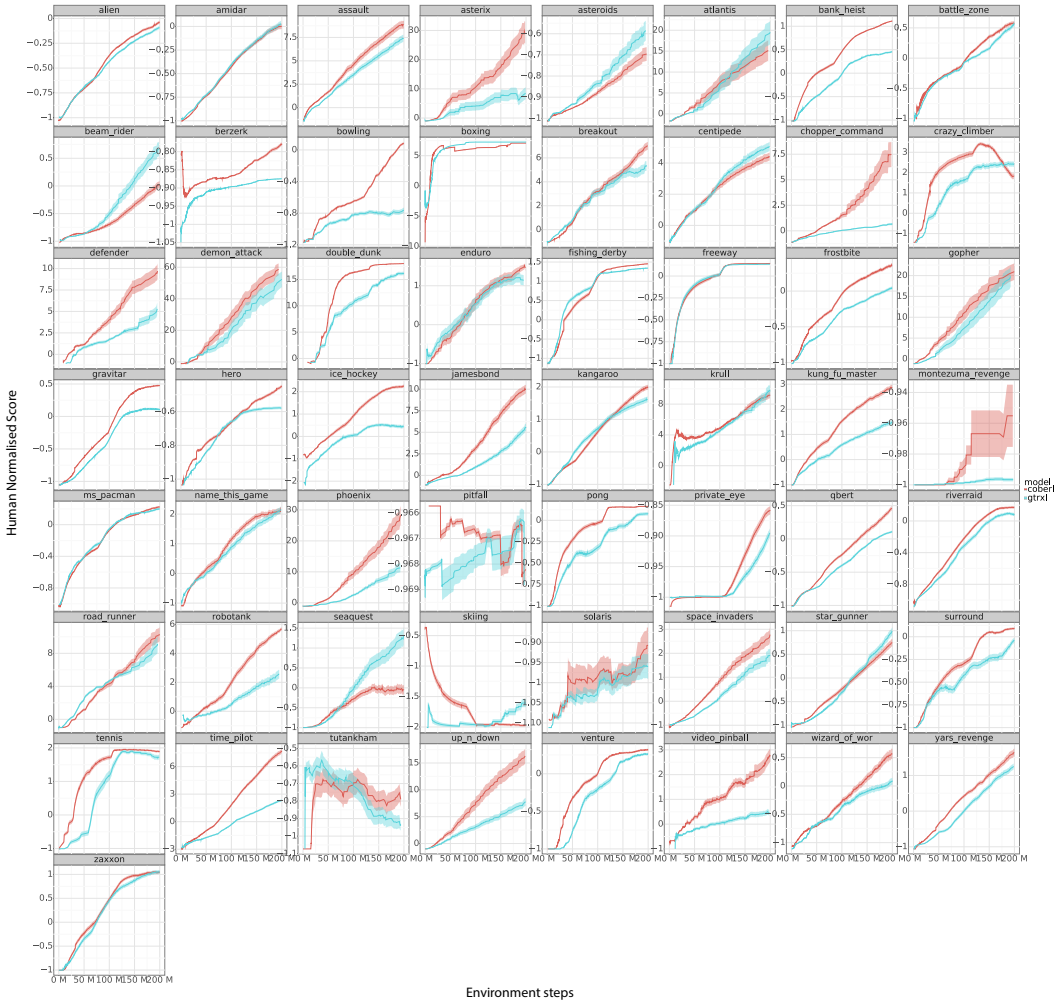

Figure 5: Learning Curves for Atari. The x-axis represents number of environment steps in millions. The y-axis represent the Human Normalised score. The error represents the 95% confidence interval. Red is COBERL, BLUE is GTrXL

F.2 DMCONTROL LEARNING CURVES

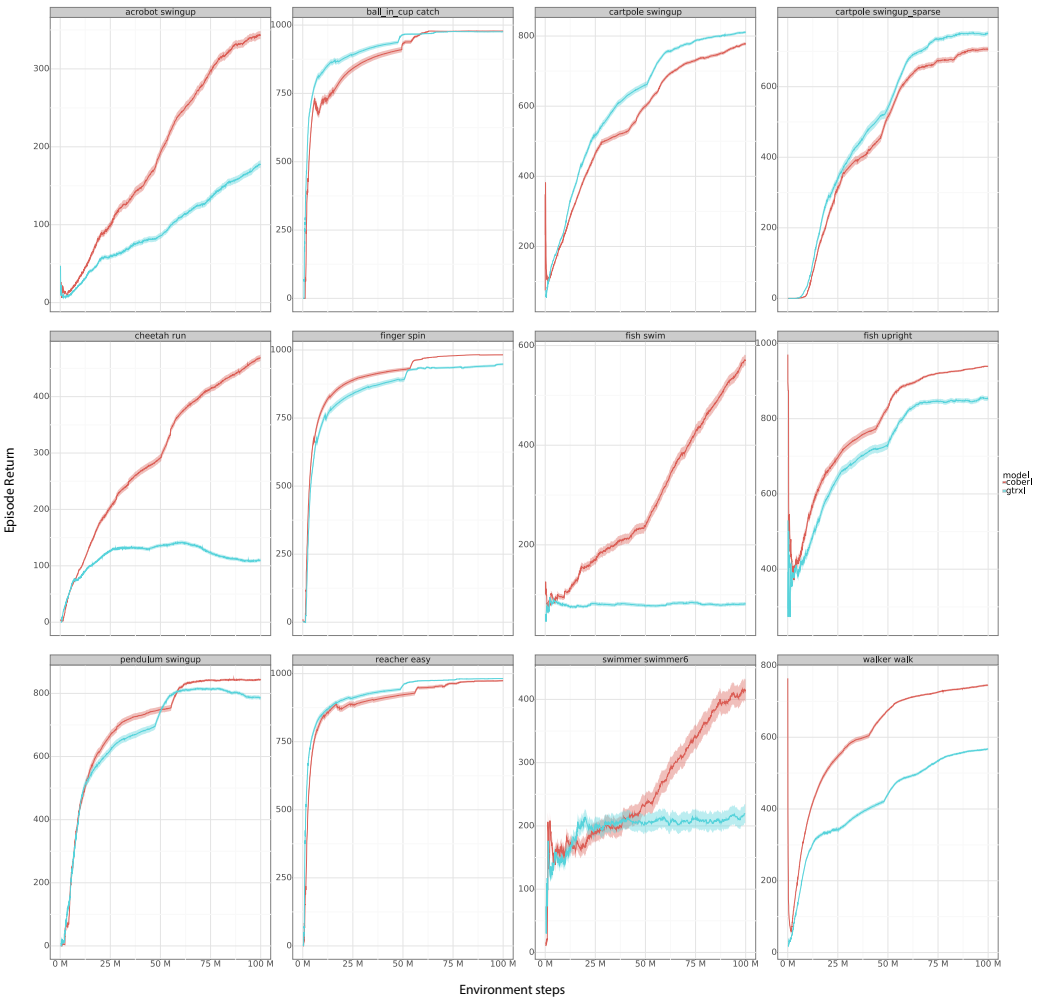

Figure 6: Learning Curves for DMControl. The x-axis represents number of environment steps in millions. The y-axis represent the Episode return. The error represents the 95% confidence interval. Red is COBERL, BLUE is GTrXL

F.3 DMLAB LEARNING CURVES

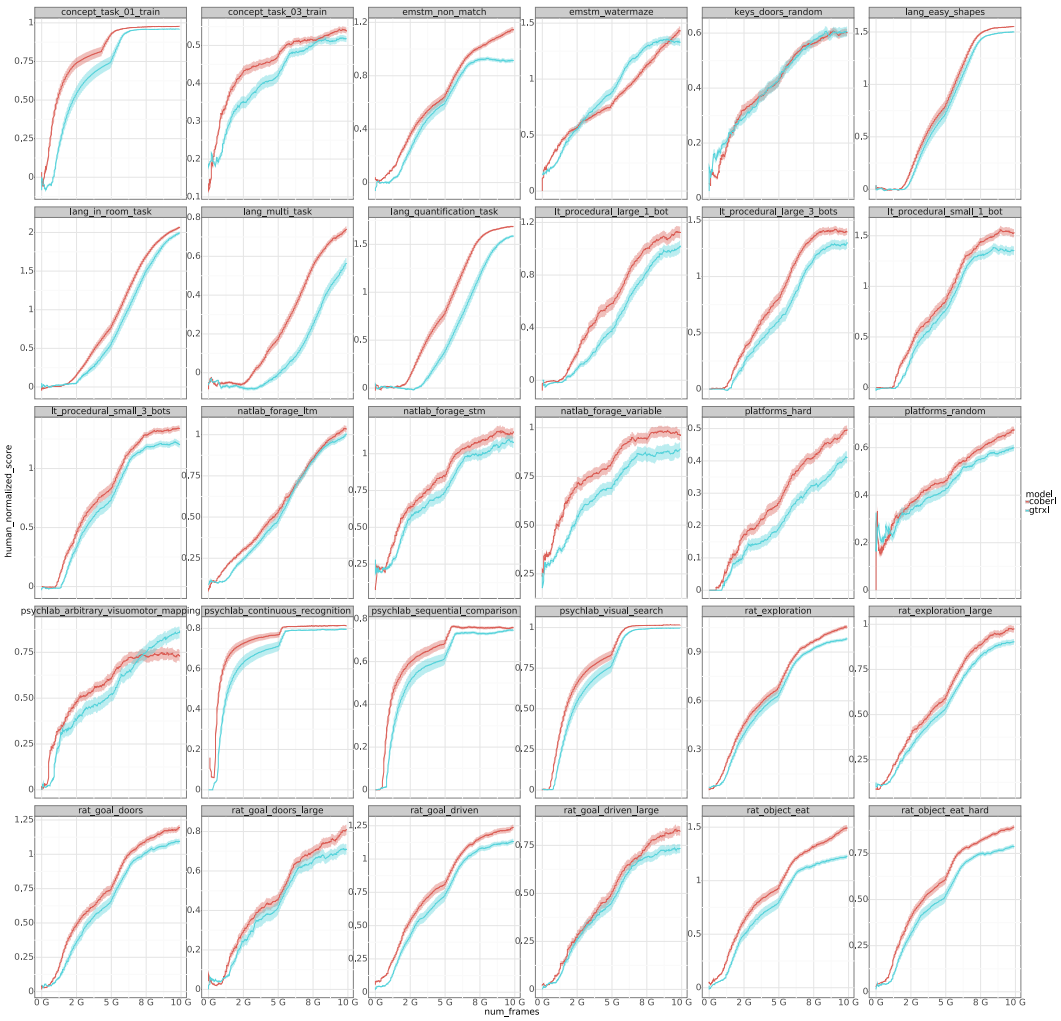

Figure 7: Learning Curves for DMLab. The x-axis represents number of environment steps in billions. The y-axis represent the Human Normalised score. The error represents the 95% confidence interval. Red is COBERL, BLUE is GTrXL

G LICENSES

The The Arcade Learning Environment Bellemare et al. (2013) is released as free, open-source software under the terms of the GNU General Public License. The latest version of the source code is publicly available at: http://arcadelearningenvironment.org

DeepMind Control Suite Tassa et al. (2018) is released as free, open-source software under the terms of Apache-2.0 License. The latest version of the source code is publicly available at: `https://github.com/deepmind/dm_control/blob/master/dm_control/suite/README.md`

DmLab Beattie et al. (2016) is released as free, open-source software under the terms of Apache-2.0 License. The latest version of the source code is publicly available at: `https://github.com/deepmind/lab/tree/master/game_scripts/levels/contributed/dmlab30`

H PSEUDO-CODE

Algorithm 1 Pseudo-code for CoBERL

```
training_iterations ← 0
initialise_weights(VisualEncoder, TransformerStack, Gate, ValueNetwork)
while training_iterations ≤ max_training_iterations do
    sampled_batch ← sample_experience()
    encoded_images ← VisualEncoder(sampled_batch)
    encoded_actions ← OneHotActionEncoder(sampled_batch)
    combined_inputs ← concat(encoded_images, previous_rewards, encoded_actions)
    transformer_inputs, transformer_targets, mask ← create_masks_targets(combined_inputs)
    output_transformer_contrastive ← TransformerStack(transformer_inputs, causal_mask=False)
    contrastive_loss ← compute_aux_loss(output_transformer_contrastive, transformer_targets,
mask)
    output_transformer_RL ← TransformerStack(transformer_inputs, causal_mask=True)
    gated_output ← Gate(combined_inputs, output_transformer_RL)
    lstm_output ← LSTM(gated_output)
    combined_inputs_for_value_net ← concat(lstm_output, combined_inputs)
    value_estimation ← ValueNetwork(combined_inputs_for_value_net)
    rl_loss ← compute_rl_loss(value_estimation, extra_args)
    total_loss ← rl_loss + contrastive_loss
end while
```

Pseudo-code for the auxiliary loss calculation

```
def compute_aux_loss(input1, input2, mask_ext):

  batch_size, seq_dim, feat_dim = input1.shape
  input1 = reshape(input1, [-1, feat_dim])
  input2 = reshape(input2, [-1, feat_dim])

  input1 = l2_normalize(input1, axis=-1)
  input2 = l2_normalize(input2, axis=-1)

  # Compute labels index
  labels_idx = arange(input1.shape[0])
  labels_idx = labels_idx.astype(jnp.int32)
  # Compute pseudo-labels for contrastive loss.
  labels = one_hot(labels_idx, input1.shape[0] * 2)
  # Mask out the same image pair.
  mask = one_hot(labels_idx, input1.shape[0])
  # Compute logits.
  logits_11 = matmul(input1, jnp.transpose(input1))
  logits_22 = matmul(input2, jnp.transpose(input2))
  logits_12 = matmul(input1, jnp.transpose(input2))
  logits_21 = matmul(input2, jnp.transpose(input1))
  # Calculate invariance penalty.
  inv_penalty = kl_with_logits(  # [B * T]
      stop_gradient(logits_11), logits_22, axis=-1)
  inv_penalty += kl_with_logits(
      stop_gradient(logits_12), logits_22, axis=-1)
  inv_penalty += kl_with_logits(
      stop_gradient(logits_21), logits_11, axis=-1)
  inv_penalty += kl_with_logits(
      stop_gradient(logits_12), logits_21, axis=-1)
  inv_penalty = inv_penalty / 4.  # [B * T]

  logits_11 = logits_11 - mask * 1e9
  logits_22 = logits_22 - mask * 1e9

  logits_1211 = concatenate([logits_12, logits_11], axis=-1)
  logits_2122 = concatenate([logits_21, logits_22], axis=-1)

  loss_12 = -sum(labels * log_softmax(logits_1211), axis=-1)
  loss_21 = -sum(labels * log_softmax(logits_2122), axis=-1)

  loss = reshape(loss_12 + loss_21, [batch_size, seq_dim]) * mask_ext
  inv_penalty = reshape(inv_penalty, [batch_size, seq_dim]) * mask_ext

  loss = mean(loss + inv_penalty)

  return loss
```

I AREA UNDER THE CURVE

For all the levels we calculated the AUC by integrating composite Simpson's rule with a delta(x) of 5 steps. We use the *intergate* package from scipy (Virtanen et al., 2020).

J LIMITATIONS AND FUTURE WORK

A limitation of our method is that it relies on single time step information to compute its auxiliary objective. Such objective could naturally be adapted to operate on temporally-extended patches, and/or action-conditioned inputs. Also, as done in the original BERT (Devlin et al., 2019), it could be possible to add a CLS token at the beginning of each sequence sent to the Transformer and then train the CLS token with RL gradients. In this way it would be possible to directly use the embeddings

of the CLS token as a sequence summary and hence provide more context to the policy estimation network. We regard those ideas as promising future research avenues.