

## A Model

We follow the model structure of Gumbel MuZero [7] as far as necessary for our case of non-intermediate rewards. As an improvement, we add the self-supervised consistency loss introduced in Efficient Zero [31]. To increase reproducibility, we go into as much detail as necessary.

### A.1 Inference

External inputs to inference are the observation  $o_t$  and the action  $a_t$  at time  $t$ .

The results of the network functions are state  $s_t^\tau$ , policy  $\mathbf{p}_t^\tau$  and value  $v_t^\tau$  derived from an observation  $o_t$ , one initial inference step and  $\tau$  sequential recurrent inference steps. We call  $\tau$  in-mind time.

The *initial inference* function  $I_\theta^{\text{initial}}$  produces an in-mind starting point at the in-mind time  $\tau = 0$ . The *recurrent inference* function  $I_\theta^{\text{recurrent}}$  takes one step into the future along  $\tau$ .  $\theta$  denotes the network parameters.

$$s_t^0, \mathbf{p}_t^0, v_t^0 = I_\theta^{\text{initial}}(o_{0:t}) \quad (7)$$

$$s_t^{\tau+1}, \mathbf{p}_t^{\tau+1}, v_t^{\tau+1} = I_\theta^{\text{recurrent}}(s_t^\tau, a_{t+\tau}) \quad (8)$$

The inference functions have the network functions *representation*  $h_\theta$ , *generation*  $g_\theta$  and the *predictions*  $p_\theta, v_\theta$  as building blocks:

$$s_t^0 = h_\theta(o_{0:t}) \quad (9)$$

$$s_t^{\tau+1} = g_\theta(s_t^\tau, a_{t+\tau}) \quad (10)$$

$$\mathbf{p}_t^\tau = p_\theta(s_t^\tau) \quad (11)$$

$$v_t^\tau = v_\theta(s_t^\tau) \quad (12)$$

Two additional building blocks are the *similarity projector*  $P_{1,t}^\tau$  and the *similarity predictor*  $P_{2,t}^\tau$  used in EfficientZero [31] to measure and increase similarity<sup>2</sup> of  $s_t^\tau$  and  $s_{t+\tau}^0$ :

$$P_{1,t}^\tau = P_{1,\theta}(s_t^\tau) \quad (13)$$

$$P_{2,t}^\tau = P_{2,\theta}(P_{1,t}^\tau) \quad (14)$$

The high-level breakdown of the inference functions  $I_\theta^{\text{initial}}$  and  $I_\theta^{\text{recurrent}}$  together with the top-down description of the building blocks in the source papers [7, 31] provide a description that may help to reproduce the network but may leave details unclear. One approach to avoid potential ambiguity is a mathematically precise functional description as in [15].

We take a different approach, revealing not only the exact structure of the model but also its trained parameters: The open-source implementation we provide allows the model to be exported into the language-neutral Open Neural Network Exchange (ONNX) format [2], which can be visualised by an application such as Netron [16], as the following links illustrate for the building blocks 9-14:

- <https://netron.app?url=https://.../onnx/MuZero-TicTacToe-Representation.onnx>
- <https://netron.app?url=https://.../onnx/MuZero-TicTacToe-Prediction.onnx>
- <https://netron.app?url=https://.../onnx/MuZero-TicTacToe-Generation.onnx>
- <https://netron.app?url=https://.../onnx/MuZero-TicTacToe-SimilarityProjector.onnx>
- <https://netron.app?url=https://.../onnx/MuZero-TicTacToe-SimilarityPredictor.onnx>

In addition to viewing the model in ONNX format, it can also be used to run the model on an ONNX runtime.

<sup>2</sup>Remark: A similarity measure for in-mind states could be used as an equality measure for in-mind states, opening the possibility of moving from a decision tree to an acyclic decision graph [5] even when using in-mind states.

## 365 A.2 Training

366 The training's loss function:

$$l_t(\theta) = l_t^0 + \frac{1}{\tau_{unroll}} \sum_{\tau=1}^{\tau_{unroll}} l_t^\tau + c_4 \|\theta\|^2 \quad (15)$$

367

$$l_t^\tau = c_1 l^p(p_{t+\tau}^{target}, p_t^\tau) + c_2 l^v(v_{t+\tau}^{target}, v_t^\tau) + c_3 l^{sim}(s_{t+\tau}^0, s_t^\tau) \quad (16)$$

$$l^{sim}(s_{t+\tau}^0, s_t^\tau) = \begin{cases} 0 & \text{if } \tau = 0 \\ l^{sim}(stopgradient(P_{1,t+\tau}^0), P_{2,t}^\tau) & \text{if } \tau > 0 \end{cases} \quad (17)$$

368 As in MuZero [17] the gradient is scaled at the start of the dynamics function by  $1/2$ .

369 The value function does not change in the final, absorbing state. In the absorbing state, no loss forces  
370 are on the policy and arbitrary actions do not put any loss force on the value.

## 371 B Planning - Input and Output

372 We use Gumbel MuZero [7] for planning - in the variant Full Gumbel with deterministic action  
373 selection at non-root nodes. Given the model and a recurrent inference step budget as input, the  
374 outputs of the planning optimiser at a time  $t$  are a chosen action  $a_{\mathbf{g}}$ , the improved policy  $\mathbf{p}_{improved}$   
375 and the improved value  $v_{improved}$  - here in the notation of Gumbel MuZero [7]:

$$\mathbf{p}_{improved} = \text{softmax}(\text{logits} + \sigma(Q_{completed})) \quad (18)$$

$$v_{improved} = v_{mix} \quad (19)$$

376 During training, we always use the Gumbel noise  $\mathbf{g}$ . For a greedy playout with maximum exploitation,  
377 the planning optimiser could be entered without Gumbel noise  $\mathbf{g} = 0$ , resulting in a deterministic  
378 action selection  $a_{\mathbf{g}=0}$ , see Appendix C.4.

## 379 C Proofs Relevant for Policy Improvement

### 380 C.1 Softmax and Equivalent Logits

381 From  $p$ , logits  $\in \mathbb{R}^k$  and

$$p = \text{softmax}(\text{logits}) \quad (20)$$

382 it follows

$$\ln(p) \equiv \text{logits} \quad (21)$$

383 where we call logits<sub>A</sub> and logits<sub>B</sub> equivalent logits<sub>A</sub>  $\equiv$  logits<sub>B</sub> if  $p_A = p_B$ .

384 *Proof:*

$$\begin{aligned} \ln(p_i) &= \ln\left(\frac{\exp(\text{logits}_i)}{\sum_j \exp(\text{logits}_j)}\right) \\ &= \text{logits}_i + C \\ &\equiv \text{logits}_i \end{aligned}$$

385 with the constant  $C \in \mathbb{R}$ .

## 386 C.2 Policy Improvement Proof by Planning With Gumbel and a Prior Temperature

387 Gumbel MuZero [7] proved that

$$p_{improved} = \text{softmax}(\text{logits} + \sigma(Q_{completed})) \quad (22)$$

388 produces a policy improvement compared to  $p = \text{softmax}(\text{logits})$ .

389 It follows that

$$p_{improved,T} = \text{softmax}\left(\frac{\text{logits} + \sigma(Q_{completed})}{T}\right) \quad (23)$$

390 with  $T > 0$  produces a policy improvement compared to  $p_T = \text{softmax}(\text{logits}/T)$  if one enters the  
391 optimization search with  $\text{logits}/T$  instead of  $\text{logits}$  and  $\sigma/T$  instead of  $\sigma$ .

392 *Proof:* Gumbel MuZero [7] states that  $\sigma$  can be any monotonically increasing transformation. If  $\sigma$  is  
393 monotonically increasing so is  $\sigma/T$ .

## 394 C.3 Policy Improvement by Planning With Gumbel and a Prior Temperature $T = 0$

395 Doing the planning optimization in the limit  $T \rightarrow 0$  could be done by setting the Gumbel  $g$  to 0.

396 *Proof:* The optimization process uses expressions of the type

$$\text{argmax}_a (g(a) + \text{logits}(a) + \sigma(q(a))) \quad (24)$$

397 Entering the optimization search with  $\text{logits}/T$  instead of  $\text{logits}$  and  $\sigma/T$  instead of  $\sigma$  we have  
398 expressions of the type

$$\text{argmax}_a \left( g(a) + \frac{\text{logits}(a) + \sigma(q(a))}{T} \right) \quad (25)$$

399 In the limit  $T \rightarrow 0$  we get

$$\lim_{T \rightarrow 0} \text{argmax}_a \left( g(a) + \frac{\text{logits}(a) + \sigma(q(a))}{T} \right) = \text{argmax}_a (\text{logits}(a) + \sigma(q(a))) \quad (26)$$

400 which is the same as setting Gumbel  $g$  to 0 in equation 24.

## 401 C.4 Improving Planning in an Eager Playout

402 Let  $p = \text{softmax}(\text{logits})$ ,  $p_T = \text{softmax}(\text{logits}/T)$  and  $p(a_1) > p(a_2) > 0$ , then it follows that

$$\frac{p_T(a_1)}{p_T(a_2)} > \frac{p(a_1)}{p(a_2)} \quad (27)$$

403 for  $0 < T < 1$ . Therefore, introducing the temperature  $T$  in this range emphasises the relative  
404 magnitude of the probability. Additionally  $p_T$  is improved by planning using  $\sigma/T$  instead of  $\sigma$   
405 according to Appendix C.2. In the limit  $T \rightarrow 0$   $p_T$  is improved by setting the Gumbel value to 0 in  
406 planning according to Appendix C.3.

## 407 D Experimental Details

### 408 D.1 Training and Testing

409 During the training of the model in each play and training cycle, which we call an epoch, the agent  
410 plays 1000 games before the model is updated 40 times by drawing batches from the last 10000  
411 games of size 256 and using game symmetry to increase the batch size by the symmetry group size 8  
412 on the training device to 2048. To evaluate the strength of the games, the acyclic-directed decision  
413 graph is fully unfolded, all relevant decision nodes are identified, and the network decisions are  
414 compared with the optimal decisions for each epoch, once without and once with tree search.

## 415 D.2 Counting Bad Decisions

416 To illustrate the counting of bad decisions: Adding the entries from the epoch-1028 row, exploration-  
417 off column and all the decision nodes from Table 1 gives 411.

Table 1: Illustrating the counting of bad decisions on the nodes where a decision matters in Figure 5.

epoch	exploration	all decision nodes				decision nodes on optimal path			
		X : I	O : I	X : P	O : P	X : I	O : I	X : P	O : P
1028	off	110	91	104	106	12	23	6	19
1028	on	0	0	0	0	0	0	0	0

<sup>a</sup>

<sup>a</sup>X is short for player x, O for player o, I for intuition - initial inference only, P for planning - doing tree search.

418 We count the decision from a game state only once, even if the decision tree node appears more than  
419 once in the decision tree - we count the nodes in terms of a decision graph. But there is a double  
420 count because we test once with and once without tree search and add the numbers. For a closer look  
421 at distinguishing between bad decisions after initial inference only (I) and the bad decisions after tree  
422 search (P) see Appendix D.3.

## 423 D.3 Testing With and Without Tree Search

424 In Appendix D.2 we described how we count bad decisions when testing a network. Here we take a  
425 closer look at the difference in bad decisions with and without tree search.

426 In the MuZero learning process, the tree search produces better-rewarded decisions than the decisions  
427 from the model just doing initial inference. The model becomes better as it learns from the experience  
428 of this improved policy and therefore makes better decisions than before. This is reflected in Figure 9.

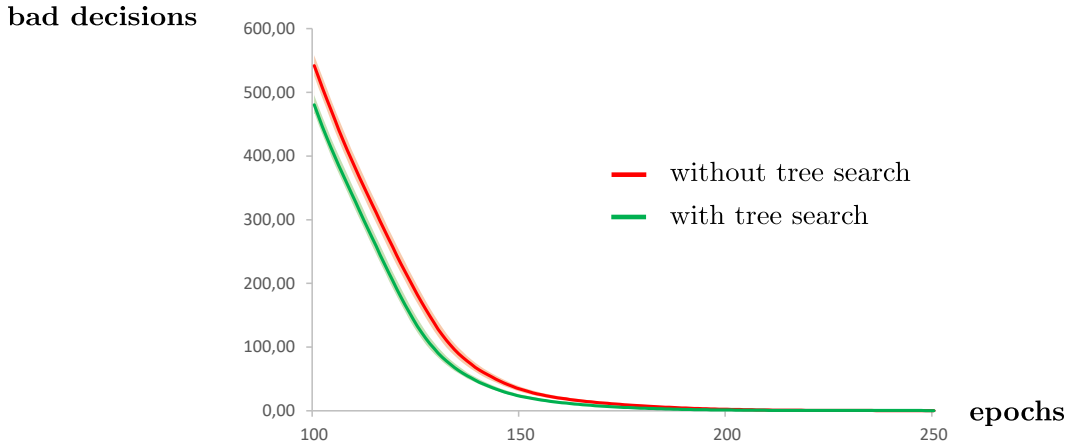


Figure 9: Bad decisions with and without tree search on the models trained with exploration-on in Figure 5, a rolling average of the last 100 epochs, 10 samples, 95% confidence intervals.

429 At training epochs beyond about 250 the training does not show an improvement for the training  
430 parameters used, see Figure 10.

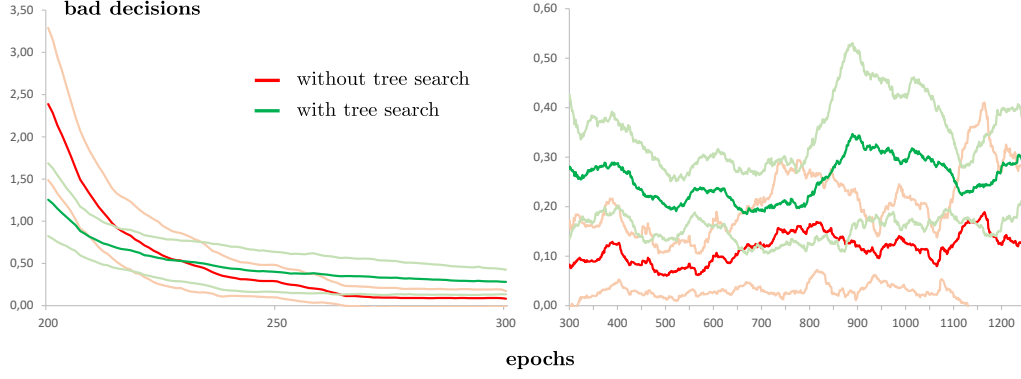


Figure 10: Extended view of the graph shown in Figure 9 - Bad decisions with and without tree search, a rolling average of the last 100 epochs, 10 samples, 95% confidence interval limits.

#### 431 D.4 Statistics

432 In the experiments presented in Figures 5, 7, 8 and 12, we train a sample of 10 networks separately.  
 433 For this sample size, we assume a Student's t-distribution for the mean values [24] to calculate the  
 434 confidence intervals.

#### 435 D.5 Training With Exploration-on - Compound Error

436 If we replace in equation 5 the target value  $v_{initialInference,t}$  with the improved value from planning  
 437  $v_{improved,t}$  from planning, equation 19, we introduce a compound error. Figure 11 illustrates how  
 438 this compound error increases back in time.

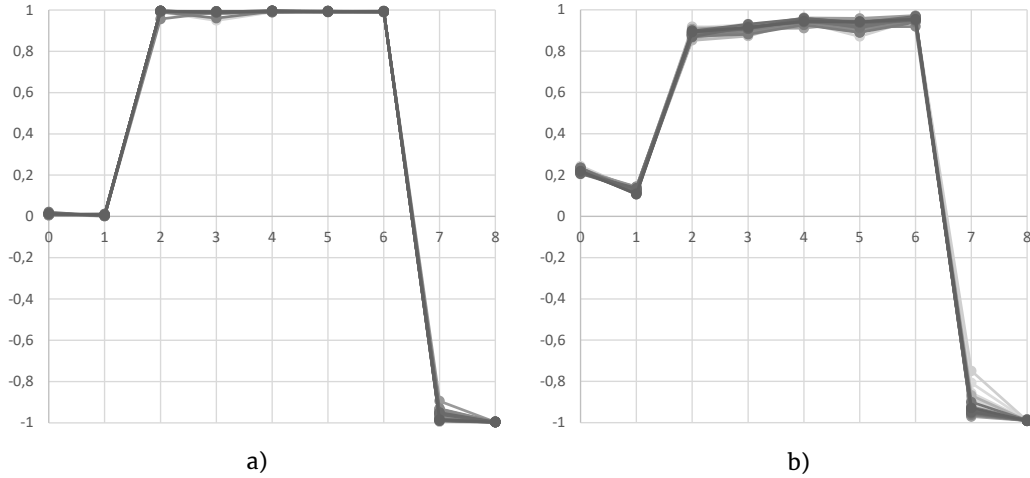


Figure 11:  $v_t^0$  for models from "epoch 1200: light grey" to "epoch 1250: dark grey" trained in two configurations for  $t < t_{startNormal}$ : a)  $v_{initialInference,t}$  and b)  $v_{improved,t}$ .

## 439 D.6 Hyperparameters

440 Table 2 lists the common hyperparameters and Table 3 the specific hyperparameters used in the  
 441 experiments.

Table 2: Common Hyperparameters.

Parameter	Setting
model: number of residuals	6
model: broadcast layers	none
model: channels	256
model: bottleneck channels	128
model: channels of hidden layer in similarity projector	500
model: channels of output layer in similarity projector	500
model: channels of hidden layer in similarity predictor	250
model: channels of output layer in similarity predictor	500
planning	Gumbel MuZero
planning: initial m	4
planning: cVisit	20
planning: cScale	1.0
planning: number of simulations	20
planning: root noise: Dirichlet alpha	1.2
planning: root noise: exploration fraction	0.25
decision: temperature: exploration-on	5
decision: temperature: exploration-off	1
experience: window size	10K
training: steps	50K
training: training steps per epoch	40
training: number of unrolling steps $\tau_{unroll}$	5
training: batch size before symmetry	256
training: symmetry	square
training: loss: $c_1$	1
training: loss: $c_2$	1
training: loss: $c_3$	2
training: optimizer	Adam
training: optimizer: learning rate	0.0001
training: optimizer: weight decay <sup>a</sup>	0.0001

<sup>a</sup> $c_4$  is part of weight decay attribute used in the Adam optimizer.

Table 3: Experiment-specific hyperparameters in planning.

Experiment	Parameter	Setting
Figures 5, 12, 9, 10	root noise: exploration fraction	0.25
	Gumbel value during test ployout with tree search	$\mathbf{g} \sim \text{Gumbel}(0)$
	Temperature during test ployout with no tree search	$T = 0$
Figure 7	root noise: exploration fraction	0.25
	Gumbel value during test ployout with tree search	$\mathbf{g} = 0$
	Temperature during test ployout with no tree search	$T = 0$
Figure 8	root noise: exploration fraction	0
	Gumbel value during test ployout with tree search	$\mathbf{g} \sim \text{Gumbel}(0)$
	Temperature during test ployout with no tree search	$T = 0$

## 442 D.7 Dirichlet Noise and Entropy

443 The entropy of the model’s policy prediction  $p_t^\tau$  at time  $t$  and in-mind time  $\tau$  is

$$H_t^\tau = - \sum_a p_t^\tau(a) \ln(p_t^\tau(a)) \quad (28)$$

444 In Tic-Tac-Toe at the very beginning of the game player x could do any move without influencing the  
 445 outcome of the game. To get some insight into why the models trained with Dirichlet noise produce  
 446 better decisions in our experiments we investigate the entropy  $H_0^0$  for this initial state, see Figure 12.

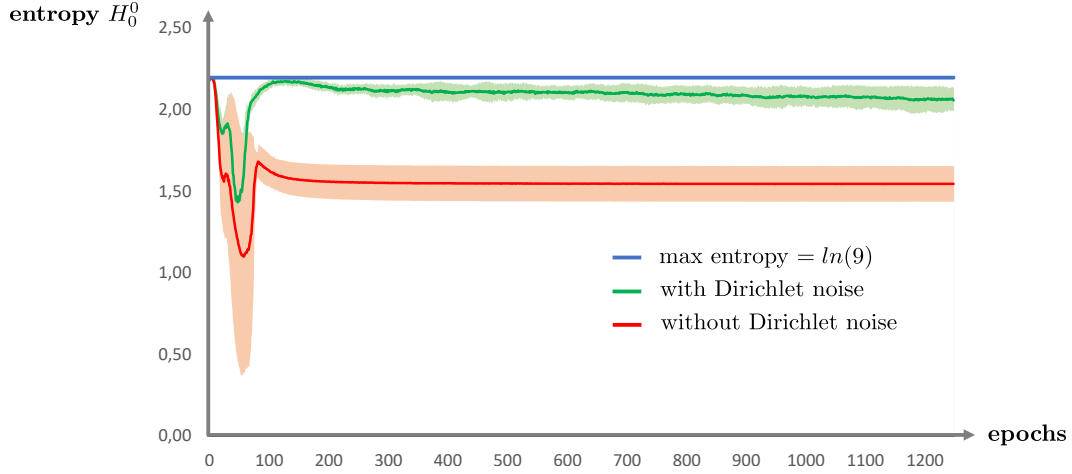


Figure 12: entropy  $H_0^0$  during training from epoch 0 to epoch 1250 - mean value over 10 samples with 99% confidence intervals. One model is trained with Dirichlet noise (green) and another model is trained without Dirichlet noise (red).

447 Adding Dirichlet noise seems to push the entropy higher where there is no reason concerning the  
 448 outcome of the game to favour one action over the other. As a consequence, the agent experiences  
 449 more regions of the decision tree which may be the cause of why it produces better decisions trained  
 450 with Dirichlet noise.

## 451 E Open Source Implementation

452 The source code is available at <https://github.com/...>

453 The open-source implementation stores in the subdirectory *reproduce* all relevant information to  
 454 reproduce the experimental results shown in the paper. All hyperparameters needed to train a network  
 455 are stored in one text file with the parameters hierarchically organised as key/value pairs.

### 456 E.1 Architecture

457 Agent and environment are implemented as a Java Spring Boot command line application. The  
 458 hyperparameters can be configured using Spring Boot configuration properties.

459 Agent and environment are separated by the interface described in Figure 1. This allows plugging the  
 460 agent into different environments. Figure 13 details the structuring of the agent into components and  
 461 their interplay.

462 During training, a loop iterates over epochs. Each epoch has an *experience episodes phase* and a *train*  
 463 *model phase*.

464 The *experience episodes phase* runs episodes in parallel. Each episode iterates discrete timesteps.  
 465 Each timestep begins with an *observation* of the environment. Next, the action to be taken is *decided*.

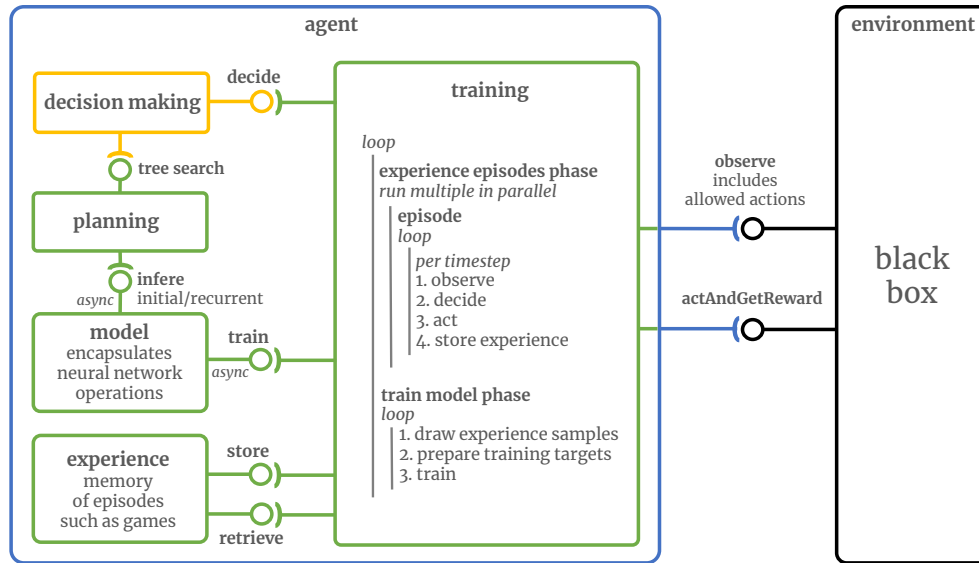


Figure 13: Structuring of the environment and the agent into interacting components.

This is the responsibility of a decision component, which calls planning for the tree search but is free in how it uses planning. Planning calls the model for inference results. Then the agent acts against the environment and stores the experience it has gained at this time step.

During the *train model phase*, samples are taken from the experience and the model component is called to perform batch training.

All calls to the model are asynchronous. This decouples the callers from the model. The model component encapsulates all neural network operations. The decoupling is not only in functionality but also in parallelization. The model component builds batch tasks and sends them via the DeepJavaLibrary (DJL) and Java Native Interface (JNI) to PyTorch, where they are processed using NVIDIA CUDA on a single GPU<sup>3</sup>

The implementation provides export functionality to the ONNX exchange format, which we use to run the model on WebAssembly directly in a browser, or to navigate the visualised network graph using Netron [16].

The build tool is Gradle, which packages the application with all Java dependencies into a single jar file at compile time. C-based dependencies for PyTorch and CUDA are dynamically loaded using the standard DJL approach.

All hyperparameters used can be fully configured by the Spring Boot application via a single YAML file or as command line parameters. Concrete properties files are provided with the source code.

## E.2 Reference Stack and Running Times

The implementation runs on a consumer pc. We use the following reference stack:

- Spring Boot 3.1
- DJL 0.22.1
  - PYTORCH 2.0.0
- Java: Corretto-17.0.6
- CUDA
  - cudnn 8.9

<sup>3</sup>DJL and PyTorch support processing on multiple GPUs out of the box. The use of a single GPU only reflects the hardware we are using.



492           – CUDA SDK 11.8  
493           – GPU Driver 528.24  
494       • operating system: Microsoft Windows 11  
495       • hardware:  
496           – GPU: NVIDIA GeForce RTX 4090  
497           – CPU: Intel Core i9-13900K  
498           – RAM: 128 GB

499   On this stack doing the training of the network for 1250 epochs takes about 7 hours. After each epoch  
500   a network is stored. Testing these 1250 networks on all relevant decisions in the Tic-Tac-Toe decision  
501   graph takes about 3.5 hours.