Ares reinforcement-learning pipeline

User manual for AresRL, AresGameRunner, AresGameConvert and AresNetworkTrainer

Options, file locations, the iteration cycle and recommended starting parameters for training Ares 10×10 draughts networks by self-play.

September 2026
Projects: D:\Development\ares-game-generator, D:\Development\ares-network-trainer

Contents

1 Overview

The pipeline improves an Ares network by repeated self-play. Each **iteration** plays a batch of games with the current best (*generating*) network, turns the games into training records, trains the network further on the most recent records, and lets the new network play a match against the generating network. If the new network scores well enough it becomes the generating network for the next iteration.

Four programs are involved. AresRL is the driver; it only starts the other three and keeps track of what has been done, so a run can be stopped and resumed at any time.

Program	Project	Role
AresRL.exe	ares-game-generator	Driver: runs the iteration cycle from one configuration file (rl.ini), resumes after interruption, writes progress.csv.
AresGameRunner.exe	ares-game-generator	Plays games in parallel (1–32 workers) with a material or network evaluation; also plays matches between two evaluators. Writes engine-PDN.
AresGameConvert.exe	ares-game-generator	Replays PDN games and exports quiet positions as 84-byte training records (and optionally FEN text), with game-result or blended labels.
AresNetworkTrainer.exe	ares-network-trainer	LibTorch/CUDA trainer for the 16-pattern Ares network (kings2 or counts4); exports the engine weight file.

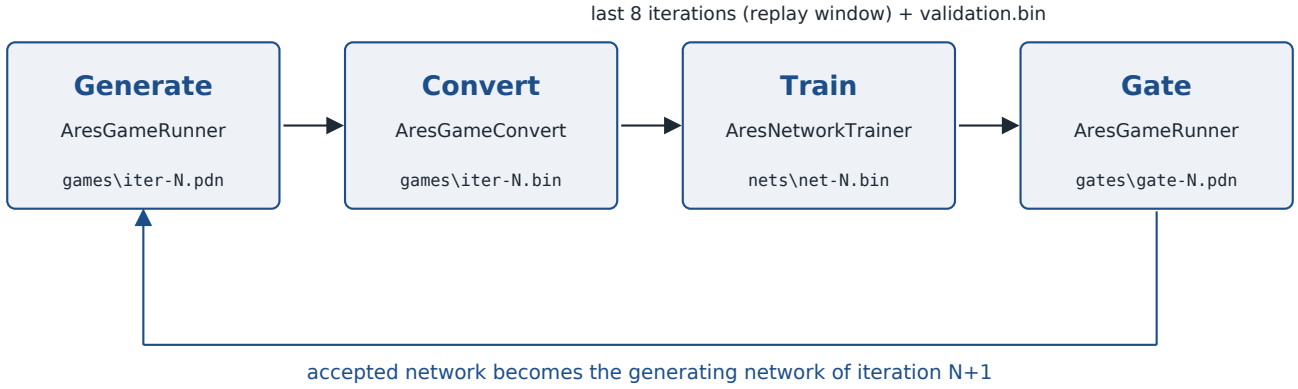

Figure 1. One iteration. The trainer always continues from its own checkpoint; only the generating network is subject to the gate.

1.1 Project folders

Folder (D:\Development\…)	Contents
ares-game-generator	Sources of AresGameRunner, AresGameConvert and AresRL; build output in out\release; rl.example.ini; README.md; this manual in docs\.
ares-network-trainer	Sources of AresNetworkTrainer; build output in out\build\vs2026-release (with the LibTorch DLLs copied next to the executable).
Ares-AI-v2.0.1	Engine sources. The generator uses its move generator and neural.h (ARES_ENGINE_DIR); network.bin is the current engine network (kings2).
ballots	Ballot1.fen – Ballot3.fen (9, 79 and 495 openings), embedded in AresGameRunner at build time (ARES_BALLOT_DIR).
standalone	AresTablebaseProbe.cpp/.h for EGDB adjudication (ARES_TABLEBASE_DIR).

The generator locates the other folders through CMake cache variables whose defaults are these sibling folders: `ARES_ENGINE_DIR`, `ARES_BALLOT_DIR`, `ARES_TABLEBASE_DIR` and `ARES_TRAINER_DIR` (for BinaryDataset.h). To override one:

```
cmake --preset release -DARES_ENGINE_DIR=D:/Other/src
```

2 Building and testing

Both projects must be built with the **64-bit** compiler: the move generator uses BMI2/PEXT and LibTorch is x64-only. Use the “x64 Native Tools Command Prompt”, or start 64-bit PowerShell as follows. A build log must show `Hostx64\x64\cl.exe`; both CMakeLists.txt files stop at configure time with a clear message otherwise.

```
& "C:\Program Files\Microsoft Visual Studio\18\Insiders\Common7\Tools\Launch-VsDevShell.ps1" `
    -Arch amd64 -HostArch amd64

cd D:\Development\ares-game-generator
cmake --preset release
cmake --build --preset release          # AresGameRunner, AresGameConvert, AresRL
ctest --preset release                  # three self-tests

cd D:\Development\ares-network-trainer
cmake --preset vs2026-release
cmake --build --preset vs2026-release
.\out\build\vs2026-release\AresNetworkTrainer.exe --self-test --net kings2
```

Note. CMake remembers the compiler of a build folder. If a folder was once configured from a 32-bit prompt, rename or delete it and configure again. `ctest` does not rebuild: build first, otherwise the tests run the old executables.

clang-cl. The generator project also has `clang-release` and `clang-debug` presets (output in out\clang-release). They need the Visual Studio component “C++ Clang Compiler for Windows” (or another LLVM, given with -DCMAKE_CXX_COMPILER) and the same 64-bit shell. CMake then adds `-mbmi -mbmi2 -mlzcnt -mpopcnt -mfma`, because Clang only allows the PEXT/LZCNT intrinsics with those features enabled. Use `cmake --preset clang-release`, `cmake --build --preset clang-release` and `ctest --preset clang-release`; in rl.ini, point runner and converter to out\clang-release to use these builds.

2.1 What the self-tests cover

- **AresGameRunner --self-test**: ballots, parallel dispatch, EGDB adjudication rules, perft (start, tactical, Woldouby), alpha-beta against an exhaustive reference, draw rules, node/time limits, random-margin candidate sets against exact scores, and both network layouts against an independent scalar implementation plus a bit-identical comparison with the engine’s `neural::evaluate`. Add `--network FILE` to compare a real network with the engine as well.
- **AresGameConvert --self-test**: result orientation, filters, move replay, malformed PDN, encoding and mirror symmetry, score-comment parsing and blended labels.
- **AresRL --self-test**: configuration parsing (including comments after values), Windows argument quoting, and parsing of the other programs’ summary lines.
- **AresNetworkTrainer --self-test**: CUDA forward/backward, both network types and their export sizes, checkpoint round trips, the batch pipeline, and records split over several files.

3 Quick start

3.1 Recommended: a starting model

The trainer cannot read engine .bin files, so a run either trains from scratch or starts from a trainer model (.pt). Training from scratch works, but the first iterations are then wasted: the new networks lose their gates against the existing network.bin until they have caught up. It is faster to pre-train a kings2 model once on your existing data and give it to AresRL as `initial_model`:

```
.\out\build\vs2026-release\AresNetworkTrainer.exe D:\Traindata\wdl-training-data-16x8.bin `
  --validation-file D:\Traindata\wdl-validation-data-16x8.bin `
  --net kings2 --loss mse --batch 262144 --epochs 20 --output-dir D:\RL\pretrain
# then in rl.ini:  initial_model = D:\RL\pretrain\ares-network.pt
```

3.2 Smoke test

Copy `rl.example.ini` to a new folder, for example `D:\RL\smoke\rl.ini`, and reduce the sizes so that one iteration takes a few minutes:

```
batches = 1
validation_batches = 1
steps = 50
iterations = 2
gate_batches = 2
```

```
cd D:\RL\smoke
D:\Development\ares-game-generator\out\release\AresRL.exe rl.ini
```

Check progress.csv afterwards: it shows the game results, record counts, both losses and the gate result per iteration, and the timings needed to size a full run (section 7).

3.3 Full run, stopping and resuming

```
mkdir D:\RL\run1
copy D:\Development\ares-game-generator\rl.example.ini D:\RL\run1\rl.ini
cd D:\RL\run1
D:\Development\ares-game-generator\out\release\AresRL.exe rl.ini            # run / resume
D:\Development\ares-game-generator\out\release\AresRL.exe --dry-run rl.ini  # show progress only
```

- Stop at any moment with Ctrl+C, by closing the window, or by ending AresRL.exe in Task Manager. The running child program (runner, converter or trainer) always stops with it: AresRL keeps its children in a Windows job object that is closed when AresRL ends. Start the same command again to continue at the first unfinished step.
- With `iterations = 0` the loop runs until stopped; with `iterations = 50` it stops once 50 iterations have been completed in total. You can raise the value later and resume.
- Settings may be changed between runs (for example `nodes` or `steps`); they apply from the next step that has not been done yet. Keep `net`, `pin_men`, `loss` and the trainer's `--king-weight` unchanged within a run: the trainer refuses to resume with a different objective.

4 rl.ini reference

Format: `key = value`, one per line. Lines starting with # or ; are comments, and a # preceded by a space or tab starts a comment after a value. Relative paths are relative to the folder of rl.ini. Unknown or duplicate keys are errors, so a typo never falls back silently to a default. An empty value means “use the default”. “Start” is the recommended initial value from rl.example.ini where it differs from, or needs to be filled in beyond, the built-in default.

4.1 Folders, programs and starting point

Key	Default	Start	Meaning
`work_dir`	required	.	Work folder for this run (section 5). Created if needed.
`runner`	required	...\AresGameRunner.exe	Path of AresGameRunner.
`converter`	required	...\AresGameConvert.exe	Path of AresGameConvert.
`trainer`	required	...\AresNetworkTrainer.exe	Path of AresNetworkTrainer.
`initial_network`	material	Ares-AI-v2.0.1\network.bin	Generating network before the first accepted network: an engine .bin (kings2 or counts4, detected from the size) or *material*. Also generates the validation set.
`initial_model`	empty	pre-trained .pt	Trainer model to start the first training round from (--initial-model). Empty = random initialisation.
`net`	kings2	kings2	Network type for the trainer: kings2 (engine PinnedMen=false) or counts4 (PinnedMen=true).
`pin_men`	false	false	Train counts4 with the fixed 100-point man baseline (--pin-men). Requires net = counts4; the engine adds that baseline to every counts4 network.
`iterations`	0	0	Stop after this many completed iterations in total; 0 = until stopped.
`seed`	1	1	Base seed. Iteration N uses seed·1000003 + N·7919 for its games and that value + 1 for its gate; the validation set uses seed·1000003 + 999983.

4.2 Game generation

Key	Default	Start	Meaning
threads	1	24	Parallel game workers (1–32). Use the number of physical cores.
ballot	3	3	Opening set: 1 = 9, 2 = 79, 3 = 495 ballots.
batches	10	10	Passes over the ballot per iteration: 10 × 495 = 4950 games, about 330k records.
nodes	20000	20000	Node budget per move at the start (--nodes); deterministic, independent of CPU load.
nodes_max	0	160000	Automatic increase (section 7.5): above nodes, the budget is multiplied by nodes_factor whenever progress stalls, up to this value. 0 = fixed nodes. Needs gate = true.
nodes_factor	2	2	Multiplier for each automatic increase.
nodes_stall_window	6	6	Number of gates since the previous change that are averaged.
nodes_stall_score	51	51	Raise when the mean gate score of that window is below this percentage.
nodes_schedule	empty	empty	Alternative to nodes_max: fixed steps as iteration:nodes pairs, e.g. 30:40000 60:80000 (from iteration 30 use 40,000, from 60 use 80,000).
random_plies	12	12	Random choice among near-best moves while fewer than this many plies have been played (ballot moves included).
random_margin	20	20	Near-best margin in evaluation points (one man ≈ 100). Both random settings must be 0 or both positive.
egdb	empty	F:\AresTablebases 6P-V7	EGDB folder for adjudication of quiet positions with ≤ 6 pieces. Empty = play games out.
validation_batches	1	1	Ballot passes for the fixed validation set, made once before iteration 1.
runner_extra	empty	empty	Extra AresGameRunner arguments for generation, validation and gate games (space-separated, no quoting), e.g. --hash-mb 32 --egdb-cache-blocks 256.

4.3 Conversion and training

Key	Default	Start	Meaning
score_weight	0.5	0.5	Label = (1−w)·result + w·tanh(score/scale); 0 = game result only.
score_scale	200	200	Evaluation points per tanh unit (200 = one network output unit).
converter_extra	empty	empty	Extra AresGameConvert arguments, e.g. --threats exclude.
window	8	8	Replay window: the trainer uses the records of the last 8 iterations.
steps	300	300	Batches per training round (--steps). See section 7.3 for sizing.
batch	16384	16384	Batch size.
learning_rate	0.001	0.001	AdamW learning rate of the first round only; later rounds resume the optimizer and its learning rate from the checkpoint.
loss	mse	mse	mse, bce or huber. With mse the network output is the expected result (−1…+1).
trainer_extra	empty	empty	Extra trainer arguments, e.g. --lr-decay 0.5 --lr-decay-every 20 (counts training rounds) or --king-weight 2. Keep them the same for a whole run.

4.4 Gating

Key	Default	Start	Meaning
gate	true	true	Play the new network against the generating network after each training round. false = every new network becomes the generating network.
gate_ballot	3	3	Opening set for the gate.
gate_batches	2	2	Even number of ballot passes; colours alternate per pass, so 2 plays every opening once with each colour (990 games).
gate_nodes	0	0	Node budget per move in gate games; 0 = the same budget as the iteration’s games (so it follows automatic increases).
gate_random_plies / gate_random_margin	0 / 0	0 / 0	Optional randomisation of gate games; the ballot openings and root shuffling already make them varied.
gate_threshold	50	50	Minimum score (percent of decided games, draws count half) for the new network to become the generating network.

5 The work folder

Everything a run produces is stored under `work_dir`. Iteration numbers have four digits (0001, 0002, ...).

Path	Written by	Contents
`validation.pdn / validation.bin`	runner / converter	Fixed validation games and records, made once with initial_network. Used by every training round (--validation-file), so losses are comparable across iterations.
`games\iter-N.pdn`	runner	Self-play games of iteration N (engine-PDN with score comments).
`games\iter-N.bin`	converter	Training records of iteration N (84 bytes each).
`window.txt`	AresRL	List of the .bin files of the current replay window (--inputs-from).
`train\`	trainer	The trainer's output folder: ares-network.state.pt (full checkpoint incl. optimizer), ares-network.pt (model), ares-network.bin + .meta.txt (engine export), ares-network-best.* (lowest validation loss so far).
`nets\net-N.bin (+ .meta.txt)`	AresRL	Copy of the trainer export after round N. The .meta.txt records network type and man baseline.
`gates\gate-N.pdn`	runner	Gate games of net-N against the generating network.
`status\iter-N.<step>.txt`	AresRL	Summary of a completed step (generate, convert, train, gate); its existence marks the step as done.
`status\validation.txt`	AresRL	Marks the validation set as complete.
`logs\iter-N-<step>.log, logs\validation.log`	AresRL	Full command line and output of every program run.
`progress.csv`	AresRL	One row per iteration; rebuilt from the status files after every step.

5.1 Resuming and crash safety

- Every output is first written under a temporary name (.tmp, or .partial by the converter) and renamed when the program has finished successfully. A status file is written only after that. On restart, leftover temporary files are removed and the first step without a status file is run again.
- The generating network after iteration N is the *best* entry of the last gate status file (or the last network when gate = false, or initial_network before any accepted network).
- Only one window is not covered: if AresRL is stopped after the trainer has finished but before status\iter-N.train.txt exists, that training round is repeated on resume (one extra round of steps).
- To redo a step deliberately, delete its status file (and later ones) and resume.

5.2 progress.csv

Columns	Meaning
`iteration`	Iteration number.
`generate_generator_network`	Network that played the iteration's games.
`generate_nodes, gate_nodes`	Nodes per move used for the games and for the gate.
`generate_white_wins, _black_wins, _draws, _unfinished, _seconds`	Self-play results and time.
`convert_records, _blended, _seconds`	Records exported, how many had a blended search score, time.
`train_window_files, _window_records`	Size of the replay window used in this round.
`train_train_loss, _validation_loss, _seconds`	Losses of the round (validation on validation.bin) and time.
`gate_wins, _draws, _losses, _score`	Gate result from the new network's point of view; score in percent.
`gate_accepted, gate_best, gate_seconds`	Whether net-N was accepted, the generating network afterwards, time.

5.3 Disk space

Measured sizes: about 2.9 KB of PDN and 5.6 KB of records per self-play game (about 67 records of 84 bytes), and 26.9 MB per network. With the start parameters one iteration therefore needs about 14 MB (games) + 28 MB (records) + 3 MB (gate) + 27 MB (network) ≈ 72 MB, so 100 iterations take about 7 GB. Old games*.pdn, gates*.pdn and rejected nets*.bin may be deleted; keep the .bin records of the current window and the status files.

6 What happens in one iteration

For iteration N with generating network G, AresRL runs the following commands (shown with the start parameters; paths shortened). Each command's output is shown on the console and appended to the step's log.

Generate

```
AresGameRunner --threads 24 --ballot 3 --nodes 20000 --shuffle-root --seed <seed·1000003+N·7919>
  --network G --random-plies 12 --random-margin 20 --egdb F:\AresTablebases6P-V7 [runner_extra]
  --batches 10 --output games\iter-N.pdn.tmp
```

Convert

```
AresGameConvert games\iter-N.pdn --binary-output games\iter-N.bin
  --score-weight 0.5 --score-scale 200 [converter_extra]
```

Train

```
AresNetworkTrainer --inputs-from window.txt --validation-file validation.bin --output-dir train
  --net kings2 --steps 300 --batch 16384 --loss mse [--pin-men]
  first round:  --learning-rate 0.001 [--initial-model initial_model]
  later rounds: --resume train\ares-network.state.pt
  [trainer_extra]
then copy train\ares-network.bin (+ .meta.txt) to nets\net-N.bin
```

Gate

```
AresGameRunner --threads 24 --ballot 3 --nodes 20000 --shuffle-root --seed <generate seed + 1>
  --network nets\net-N.bin --egdb … [runner_extra]
  --opponent G --batches 2 --output gates\gate-N.pdn.tmp
accepted if (wins + draws/2) / (wins + draws + losses) ≥ gate_threshold / 100
```

The trainer keeps training its own model every round, whether or not the gate passed. A rejected network is not lost: the next round continues from it with more data, and it plays the next gate against the same G.

7 Starting parameters and why

These values come from the tests during development and are a starting point, not a tuned optimum. Measure your own throughput with the smoke test and adjust with the formulas below.

7.1 Measurements

Test	Result
Network speed in the generator search	About 3 million nodes per second per core with network.bin; the network evaluation is bit-identical to the engine.
Search depth at 20,000 nodes	Depth 5–9 in the middle game (this search has no pruning or score reuse).
Self-play, ballot 3 (495 games), 10k nodes, 12 random plies, margin 20	1 min 42 s on 2 cores; 33,275 records (≈ 67 per game); 3,229 moves chosen at random; 77 draws.
network.bin against material, 158 games	75.2% at 10k nodes, 86.4% at 50k nodes.
network.bin against itself, 158 games	49.1% and 55.4% (two seeds): the normal spread of a small match.

7.2 Games, nodes and randomness

- **nodes = 20000**: deterministic games of reasonable quality at low cost. More nodes give better labels but fewer games; with a stronger network the gain from depth grows.
- **random_plies = 12, random_margin = 20**: with a network, exact score ties almost never occur, so root shuffling alone no longer creates different games. Choosing among moves within 20 points (a fifth of a man) during the first 12 plies keeps the openings varied while avoiding clear mistakes. Every in-margin move gets an exact score, so the choice is truly among near-equal moves.
- **batches = 10**: 4950 games ≈ 330k records per iteration. Scaled from the measurement above, this takes a few minutes on 24 cores; the smoke test gives the real figure.
- **EGDB**: games reaching ≤ 6 pieces in a quiet position are adjudicated with the exact WDL result, which gives correct labels for the late middle game. Positions with ≤ 6 pieces are not exported.

7.3 Labels, window and training steps

- **score_weight = 0.5**: half game result, half tanh(search score / 200). The search score is much less noisy than the game result for early positions. Ballot moves, fallback moves and final positions keep the plain result.
- **window = 8**: about 8 × 330k ≈ 2.6M records. Each record is used for 8 rounds, and the data reflect the last few generating networks rather than only the latest one.
- **steps and batch**: the number of passes over the window per round is steps × batch / window records. With 300 × 16384 ≈ 4.9M samples on 2.6M records that is about 2 passes, so each record is seen about 15 times during its 8 rounds. Keep passes per round around 1–3: more overfits the current window. Formula: steps ≈ passes × window × records per iteration / batch.
- **learning_rate = 0.001** with AdamW. When the validation loss flattens, add a decay through trainer_extra (for example --lr-decay 0.5 --lr-decay-every 20) or lower it in a new run.

- **loss = mse** matches the engine: the output is the expected result, and 200 × output is the evaluation.

7.4 The gate

- **990 games (ballot 3, 2 passes)**: the standard error of the score is about 1.5 percentage points. With gate_threshold = 50, a network that is equally strong passes about half of the time; that is acceptable because training continues regardless.
- For a stricter gate use 52–55, or more gate_batches. For a faster loop at the cost of more noise, use gate_ballot = 2 (158 games per 2 batches).
- The gate score is also a progress measure: a long series of accepted networks with scores above 50% shows that the loop is improving.

7.5 Growing the node budget

A fixed node budget eventually limits progress: the labels are only as good as the search that produced them. With `nodes_max` set, AresRL raises the budget automatically when the gates show that the networks have stopped improving at the current budget.

- After every gate, AresRL looks at the gate scores since the last change. Once there are nodes_stall_window of them, it averages the last nodes_stall_window scores. If that mean is below nodes_stall_score, the next iteration uses nodes × nodes_factor (at most nodes_max) and the count starts again.
- With the start values (6 gates, 51%): while the networks improve, gate scores are well above 50% and nothing changes. At a plateau the scores fluctuate around 50% (±1.5 each), so the mean of six falls below 51% and the budget doubles: 20k → 40k → 80k → 160k.
- Every doubling roughly doubles the generation and gate time per iteration. Raise nodes_max when the run has reached it and still plateaus.
- The budget for an iteration is computed from rl.ini and the gate results of the earlier iterations only, so a resume gives the same values. Adding nodes_max to a running rl.ini applies the rule to the existing gate history: the next iteration may therefore start at a raised budget.
- nodes_schedule gives fixed steps instead, for runs that should be reproducible regardless of gate outcomes.
- Keep gate_nodes = 0 so that both networks in a gate search with the budget the new network was trained for.

8 AresGameRunner reference

Usage: `AresGameRunner [options]`. Without an opening set it plays one game from the start position (or --fen).

Option	Meaning
`--threads N`	Parallel game workers 1–32 (default 1). Game-level parallelism; each worker has two private search objects.
`--ballot N`	Built-in opening set: 0 = off, 1 = 9, 2 = 79, 3 = 495 openings. The ballot moves are replayed and written into the PDN.
`--openings FILE`	External opening set (plain FEN lines or [FEN "..."] tags). Not together with --ballot/--fen.
`--batches N`	Full passes over the opening set (default 1). The BatchId tag records the pass.
`--games N`	Number of games from --fen or the start position (only without an opening set).
`--fen FEN`	Starting position (default W:W31-50:B1-20).

Option	Meaning
`--network FILE`	Evaluate with an Ares network instead of material. 26,882,244 bytes = kings2, 26,900,676 bytes = counts4 (+100 per man, as the engine with PinnedMen). A FILE.meta.txt with men_baseline=0|100 overrides the baseline.
`--opponent X`	Match mode: the --network player (or material) plays X (a network file or material). Colours alternate per batch (per game with --games); the summary adds player 1's wins/draws/losses and score.
`--nodes N`	Node budget per move; disables the clock, so games are reproducible for a seed.
`--move-ms X`	Time per move in milliseconds (default 100); after --nodes it adds the clock as a second limit.
`--depth N`	Maximum iteration depth 1–128 (default 64).
`--fixed-depth N`	Search exactly to depth N without a clock.
`--random-plies N`	Random choice among near-best moves while fewer than N plies have been played (default 0).
`--random-margin X`	Near-best margin in evaluation points (0–1000). Both random options together.
`--shuffle-root`	Shuffle the root move order once per search (breaks exact ties randomly).
`--seed N`	Base seed (default 1); each game derives its own seed from it and its GameId.
`--hash-mb N`	Move-ordering table per player in MiB (default 16; 0 = off).
`--max-plies N`	Safety limit (default 500); such games are unfinished (*) and skipped by the converter.
`--egdb DIR`	EGDB folder for adjudication. --egdb-max-pieces N (2–6, default 6) and --egdb-cache-blocks N (default 64 per stream per worker) tune it.
`--output FILE`	New PDN file (existing files are refused). Without it, PDN is printed.
`--self-test`	Correctness tests; with --network FILE also compares that network with the engine.

8.1 PDN output

Each game has the tags Event, White, Black (the evaluator names), Result (2-0, 0-2, 1-1 or * — authoritative), Termination, Seed, GameId, OpeningId, BatchId, Ballot, MoveTimeMs, NodeLimit, RandomPlies, RandomMargin, SearchDepth, RootShuffle, HashMiBPerPlayer and EGDB settings. After every searched move a comment gives the search result of the position before the move, from the mover's side:

```
1. 31-26 {s=7 d=5 r=9} 20-24 {s=-3 d=6 r=8} 2. 36-31 {s=7 d=6 r=5} ... 5. 28-22 {s=58 d=8}
```

Field	Meaning
`s=S`	Score in evaluation points (for a random move: the score of the best move, i.e. of the position).
`d=D`	Last completed search depth.
`r=K`	The move was chosen at random among K moves within the margin.
`fb`	Fallback move: no iteration completed, the score is not usable.

9 AresGameConvert reference

Usage: `AresGameConvert INPUT.pdn --binary-output FILE [--fen-output FILE] [options]`. Outputs are written as FILE.partial and renamed at the end; existing outputs are refused.

Option	Meaning
`--binary-output FILE`	Trainer records (84 bytes each, headerless).
`--fen-output FILE`	Text: compact FEN, a tab and the label (+1/0/−1, or a decimal for blended labels).
`--threats MODE`	include (default), exclude or only: positions where the opponent has a capture or promotion.
`--score-weight W`	Blend the search score into the label: (1−W)·result + W·tanh(s/K), W 0–1 (default 0).
`--score-scale K`	Evaluation points per tanh unit (default 200).
`--self-test`	Converter tests.

9.1 Which positions become records

- Only finished games (Result tag 2-0, 0-2 or 1-1); unfinished games are skipped.
- Only positions with more than six pieces, with legal moves and without a capture for the side to move.
- Labels are from the side to move: +1 win, 0 draw, −1 loss (or the blended value).
- No deduplication or shuffling; the trainer shuffles. Split by game (as the validation file does), not by position.

9.2 Record format

Field	Encoding
16 × int32 patterns	Local 8-square pattern indices 0–6560 (black man 0, empty 1, white man 2). With black to move the board is rotated 180° and colours swapped.
4 × int32 nwm, nwk, nbm, nbk	Men and kings of the side to move and the opponent; kings clamped at 5. kings2 uses only the two king counts.
float32 label	Target in −1…+1.

10 AresNetworkTrainer reference

Usage: `AresNetworkTrainer <records.bin> [more.bin …] [options]`. All inputs form one dataset in the given order.

Option	Meaning
`--inputs-from FILE`	Read more input paths from FILE (one per line, # comments, relative to the list file).
`--validation-file FILE`	Separate validation records; all inputs are then trained. Without it, the last --validation fraction (default 0.02) of the data is used.
`--net TYPE`	kings2 (engine PinnedMen=false; 2 × 6 king-count rows) or counts4 (PinnedMen=true; 4 × 21 rows). Default counts4; on --resume the checkpoint's type is used.
`--pin-men`	counts4 only: add the fixed 0.5 × (own − opponent men) baseline (100 points per man).
`--steps N`	Batches per epoch instead of a full pass (successive reshuffled passes); implies --epochs 1.
`--epochs N`	Epochs per run (default 10).
`--batch N`	Batch size (default 8192).

Option	Meaning
`--learning-rate X`	AdamW learning rate (default 0.001).
`--weight-decay X`	AdamW weight decay (default 0.00001).
`--lr-decay X, --lr-decay-every N`	Multiply the learning rate by X after every N epochs (rounds with --steps).
`--loss MODE`	bce (default), mse or huber (--huber-delta X, default 0.5).
`--labels MODE`	minus-one-zero-one (default) or zero-half-one.
`--king-weight X`	Training weight for positions with kings (default 1).
`--output-dir DIR`	Folder for all outputs (remembered in the checkpoint).
`--output, --export, --checkpoint, --best-output, --best-export`	Individual output names (defaults ares-network.pt, .bin, .state.pt, -best.pt, -best.bin).
`--initial-model FILE`	Start from a model-only .pt with a fresh optimizer; pass the matching --net.
`--resume FILE`	Continue a full checkpoint (model, optimizer, learning rate, epoch count). Objective options (--net, --pin-men, --loss, --king-weight) must match.
`--seed N`	Shuffle seed.
`--cpu, --no-prefetch`	Train without CUDA; disable background batch preparation.
`--validate-only FILE`	Check a record file and report its label distribution.
`--self-test`	Built-in tests (use with --net kings2 or counts4).

10.1 Network types and the engine

Type	Count inputs	Export size	Engine setting
kings2	min(5, own kings), min(5, opponent kings)	26,882,244 bytes	PinnedMen = false
counts4 + --pin-men	own men, own kings, opponent men, opponent kings	26,900,676 bytes	PinnedMen = true (adds 100 per man)
counts4 without --pin-men	as counts4	26,900,676 bytes	No matching engine setting (the trainer warns); AresGameRunner can use it through the .meta.txt baseline.

Every export has a sidecar FILE.meta.txt with network_type, engine_pinned_men, file_size, output_scale=200 and men_baseline. Checkpoints are version 7 and record the network type; version 5/6 checkpoints load as counts4.

11 Troubleshooting

Symptom	Cause and remedy
“identifier not found” for _pext_u64, _tzcnt_u64 …, or std::min ambiguities in the trainer	32-bit compiler (Hostx86\x86). Use a 64-bit prompt, rename the build folder, configure again.
“Unknown option” for a new option	The executable is older than the sources: rebuild (ctest does not build).
“Output file already exists”	The runner and converter never overwrite. AresRL removes its own stale temporary files; outside AresRL choose a new name.
“Network … has N bytes; expected …”	Not an Ares network file, or a truncated copy.
Trainer refuses to resume	--net, --pin-men, --loss or --king-weight differ from the checkpoint. Keep them constant within a run.
The first gates are all rejected	Expected when training starts from scratch against an existing strong network; use a pre-trained initial_model (section 3.1).

Symptom	Cause and remedy
Gate scores jump around 50%	Normal noise (±1.5 points for 990 games). Look at the trend in progress.csv.
Validation loss rises while training loss falls	Too many passes per round: lower steps or raise batches/window.
AresRL stopped with an error	The message names the log file in logs\. Fix the cause and run the same command again; completed steps are not repeated.

Appendix A rl.example.ini

```
# AresRL configuration. Copy to e.g. D:\RL\run1\rl.ini and adjust.
# Relative paths are relative to this file. Run:  AresRL.exe rl.ini
# Stop with Ctrl+C at any time; running AresRL again with the same file resumes.

# --- Folders and programs ---
work_dir  = .
runner    = D:\Development\ares-game-generator\out\release\AresGameRunner.exe
converter = D:\Development\ares-game-generator\out\release\AresGameConvert.exe
trainer   = D:\Development\ares-network-trainer\out\build\vs2026-release\AresNetworkTrainer.exe

# --- Starting point ---
# Network that generates the first games and that new networks must beat
# (an engine .bin, or "material").
initial_network = D:\Development\Ares-AI-v2.0.1\network.bin
# Optional trainer model (.pt) to start training from; empty = train from scratch.
initial_model =
net = kings2
pin_men = false

# --- Loop ---
iterations = 0        # stop after this many iterations in total; 0 = run until stopped
seed = 1

# --- Game generation (per iteration) ---
threads = 24
ballot = 3
batches = 10          # 10 x 495 = 4950 games, about 330k records
nodes = 20000         # nodes per move at the start
# Automatic increase: when the mean gate score of the last nodes_stall_window gates
# (since the previous change) is below nodes_stall_score, multiply nodes by
# nodes_factor, up to nodes_max. nodes_max = 0 keeps nodes fixed.
nodes_max = 160000
nodes_factor = 2
nodes_stall_window = 6
nodes_stall_score = 51
# Alternative to nodes_max: fixed steps, e.g.  nodes_schedule = 30:40000 60:80000
nodes_schedule =
random_plies = 12
random_margin = 20
egdb = F:\AresTablebases6P-V7
validation_batches = 1  # fixed validation set, made once with initial_network

# --- Conversion ---
score_weight = 0.5
score_scale = 200

# --- Training (per iteration) ---
window = 8            # replay window: records of the last 8 iterations
steps = 300           # about 2 passes over a full window of ~2.6M records
batch = 16384
learning_rate = 0.001 # first iteration only; later iterations resume the optimizer
loss = mse

# --- Gating: new network against the current generating network ---
gate = true
gate_ballot = 3
gate_batches = 2      # 2 x 495 = 990 games, colours alternate per batch
gate_nodes = 0        # 0 = same nodes as the iteration's games
gate_random_plies = 0
gate_random_margin = 0
gate_threshold = 50   # percent score needed to replace the generating network

# Extra arguments passed unchanged (space-separated, no quoting):
runner_extra =
converter_extra =
trainer_extra =
```