第34回世界コンピュータ将棋選手権

Ari Shogi and フレンズ　アピール文書

兵頭優空

概要

Ari Shogi and フレンズは、DL系のAI「Ari Shogi」とNNUE系のAIを組み合わせる事で「比較的低コストであらゆる状況に対応できるAI」を目指しているハイブリッド型の将棋AIです。

(12月末に今まで使っていたノートPCが壊れたので)1月末から新しいPCに変わり、計算資源にかなり余裕ができたので、NNUEの学習など今まで計算資源の都合で取り組めなかった事にも取り組みつつ、良い成績を目指したいと思っています。

目次

名前について

「Ari Shogi and フレンズ」という名前の「Ari Shogi」の部分は牡羊座(Aries)から、「and フレンズ」の部分は、複数のAIで協力するイメージからつけています。

牡羊座から名前を付けたのは、「自分の星座が牡羊座だから」と「某電気羊のように、何回か大きく進化していつか電竜になってほしいという願いを込めている」の2つの理由があります。(2つ目は後付けです)

アピールポイントなど

・DL系のAIの「終盤が弱い」など弱点を単体で克服するのを放棄し、苦手な部分はNNUE系AIに任せます。

・DL評価関数では、精度を上げるために複数の評価関数を使ってアンサンブルを行います。

第4回電竜戦(以降、DR4)でAri Shogi and フレンズは、「複数の評価関数の出力の平均を1つの評価関数の出力として扱う」という機能を大会直前に実装し、「自分が学習させた中で精度の高いモデル上位3つ」を搭載して参加していました。

この機能は、

1. かなり前から「複数の評価関数で行えば(速度は大きく落ちるものの)簡単に精度を上げられそう」という事を考えていた(実際に手を付けることはなかった)

2. 11月に「夏頃から新しく使い始めた入力特徴量」に大きなバグが見つかった。その特徴量を使って学習している途中だったモデルは良い感じの精度まで上がっていたが、結構酷いバグだったので新しいモデルを1から学習させる事になった。

3. 2のせいで予定が狂い、大会で使うモデルの精度が予定よりもずっと低くなってしまった。(これ関連で入玉モデル、進行度モデルの学習もほとんど行えなかったので、それらの出力による合議のモード切替は全然上手くいかなかった)

4. 大会直前に「今から精度を上げる方法はないか」と考えていたところ、アンサンブルの件を思い出したので、それ関連で実装が1番簡単そうだった「出力を平均する」というのを実装した。(当時は学習はGoogleColab無料版などの「ネットで無料で使えるGPU」に頼ってい

たが、使える分を全部使いきってしまった後だったのでそんなにできる事がなかった。当時のメインマシンのノートPCにはGPUが乗っていたが、GPUに負荷をかけるとバッテリー周りの挙動が怪しくなるようになっていたので、GPUに負荷が猛烈にかかる学習は怖くてできなかった。)

という経緯で実装された急場しのぎのものでしたが、評価関数の精度が(少しだけですが)ちゃんと上がり、デメリットの速度低下も

・探索部がPythonで書かれているので、C++で書かれているものに比べると元々かなり遅い

・(速度低下が1番悪影響を及ぼしそうな)中終盤はNNUE系AIに丸投げしている

・最善手をNNUE系AIでチェックし、NNUE系AIが考える最善手との評価値から大きく悪化している場合は、NNUE系AIの最善手を指すという機能がある

といった理由であまり気にならなかったので、割と上手くいった改造だったと考えています。

WCSC34バージョンのAri Shogi and フレンズは、DR4バージョンで上手くいった方針を発展させ、より強力な手法を採用等に取り組み、精度のさらなる向上を目指します。

また、アンサンブルに使うモデルはある程度多様性があったほうが良いと考えているので、今使っているResNet系以外の系統のモデルも学習させてアンサンブルに使う予定です。

(3月末の時点では、
https://tokumini.hatenablog.com/entry/2021/09/04/120000

を参考にVision Transformer(ViT)を実装するところまで終わっているが、計算資源が足りなくて学習は回せていない)

・NNUE評価関数を自作の学習部で学習させます

DR4までは、「NNUEの学習もやってみたいなー」とは思ってはいたものの、リソースの都合等から取り組む事はなく、公開されているNNUE評価関数をそのまま利用していましたが、今回からは自分で学習させた標準型のNNUE評価関数を使用する予定です。

学習については、普通に学習させるだけでなく色々な実験(DLモデル

からの蒸留とか)をしたいと考えています。(が、リソースの都合であまりできなさそう)

使い慣れたプログラムをベースにしている方が効率が良く実験できると考えたので、実験用プログラムのベースとしてpython-dlshogi2の学習部をベースにしたNNUE学習部を製作しました。

学習部は公開してほしいという要望があったため、テストで作った振り飛車評価関数と学習ログと一緒に
https://github.com/YuaHyodo/Ari_Shogi_NNUE_train

https://github.com/YuaHyodo/Haojian_nnue

で公開しています。(厳密には、非公開のものから分岐させて個人用のメモとかを消した後のものなので同じものではないです。また、元々公開する予定のなかったものなので、超絶汚いです)

上の学習部(のもとになった非公開の学習部)でHáoを水匠1000万ノードのデータで1epochだけ追加学習した"Haojian_240317"という評価関数が、CPU: i5-13400F, 探索部: やねうら王7.50,1スレッド, ハッシュ512MB, たややん互角局面集24手目～という設定で、

```
///
VS Háo / 1手1000ms
対局数5000 先手勝ち2606(52.7%) 後手勝ち2336(47.3%) 引き分け58
engine1
勝ち2558(51.8% R12.1 +-9.6) 先手勝ち1349(27.3%) 後手勝ち1209(24.5%)
宣言勝ち8 先手宣言勝ち3 後手宣言勝ち5 先手引き分け33 後手引き分け25
engine2
勝ち2384(48.2%) 先手勝ち1257(25.4%) 後手勝ち1127(22.8%)
宣言勝ち3 先手宣言勝ち2 後手宣言勝ち1 先手引き分け25 後手引き分け33
///
VS Háo / 1手5000ms
対局数5000 先手勝ち2478(50.0%) 後手勝ち2475(50.0%) 引き分け47
engine1
勝ち2630(53.1% R21.4 +-9.7) 先手勝ち1315(26.5%) 後手勝ち1315(26.5%)
宣言勝ち15 先手宣言勝ち10 後手宣言勝ち5 先手引き分け21 後手引き分け26
engine2
```

勝ち2323(46.9%) 先手勝ち1163(23.5%) 後手勝ち1160(23.4%)

宣言勝ち6 先手宣言勝ち3 後手宣言勝ち3 先手引き分け26 後手引き分け21

///

VS BLOSSOM / 1手1000ms

対局数5000 先手勝ち2486(50.2%) 後手勝ち2470(49.8%) 引き分け44

engine1

勝ち2914(58.8% R61.2 +-9.8) 先手勝ち1461(29.5%) 後手勝ち1453(29.3%)

宣言勝ち22 先手宣言勝ち13 後手宣言勝ち9 先手引き分け22 後手引き分け22

engine2

勝ち2042(41.2%) 先手勝ち1025(20.7%) 後手勝ち1017(20.5%)

宣言勝ち1 先手宣言勝ち0 後手宣言勝ち1 先手引き分け22 後手引き分け22

///

VS BLOSSOM / 1手5000ms

対局数5000 先手勝ち2597(52.5%) 後手勝ち2346(47.5%) 引き分け57

engine1

勝ち2876(58.2% R56.7 +-9.8) 先手勝ち1502(30.4%) 後手勝ち1374(27.8%)

宣言勝ち9 先手宣言勝ち4 後手宣言勝ち5 先手引き分け30 後手引き分け27

engine2

勝ち2067(41.8%) 先手勝ち1095(22.2%) 後手勝ち972(19.7%)

宣言勝ち1 先手宣言勝ち0 後手宣言勝ち1 先手引き分け27 後手引き分け30

///

VS 水匠5 / 1手1000ms

対局数5000 先手勝ち2580(52.5%) 後手勝ち2331(47.5%) 引き分け89

engine1

勝ち2548(51.9% R12.9 +-9.6) 先手勝ち1338(27.2%) 後手勝ち1210(24.6%)

宣言勝ち56 先手宣言勝ち30 後手宣言勝ち26 先手引き分け47 後手引き分け42

engine2

勝ち2363(48.1%) 先手勝ち1242(25.3%) 後手勝ち1121(22.8%)

宣言勝ち1 先手宣言勝ち0 後手宣言勝ち1 先手引き分け42 後手引き分け47

///

VS 水匠5 / 1手5000ms

対局数5000 先手勝ち2515(51.1%) 後手勝ち2402(48.9%) 引き分け83

engine1

勝ち2596(52.8% R19.1 +-9.6) 先手勝ち1323(26.9%) 後手勝ち1273(25.9%)

宣言勝ち56 先手宣言勝ち29 後手宣言勝ち27 先手引き分け34 後手引き分け49

engine2

勝ち2321(47.2%) 先手勝ち1192(24.2%) 後手勝ち1129(23.0%)

宣言勝ち2 先手宣言勝ち1 後手宣言勝ち1 先手引き分け49 後手引き分け34

///

計測はTanuki Coliseum(https://github.com/nodchip/TanukiColiseum / 使ったのは古いバージョン)で行った。

いずれもengine1がHaojian_240317

FV_SCALEは、水匠5とBLOSSOMが24、Háoが20、Haojian_240317は軽く計測して1番強そうだった値に設定。

対BLOSSOMは設定をミスっているのか、相性の問題なのか、開始局面の問題なのか、異様に成績が良い。

という結果になっていて、floodgateでの計測でもHáoより少しだけレートが高いという結果で、少なくとも弱くはなっていさそうなので学習部に致命的レベルのバグはないと思います。多分

Haojianという名前は、ベースになっているHáoと、某ゲームに出てくる災いの剣からつけています。

・自動定跡生成時の局面評価で、「方策出力のみで指すDL系AIによる連続対局の結果」と「浅い探索のNNUE系AIによる連続対局の結果」と「長い時間探索したAIによる連続対局の結果」を組み合わせる予定です。

「方策出力のみで指すDL系AIによる連続対局」には「GPUを用いる事でそこそこの棋力で同時に何千局も行える」という大きなメリットがあるため、定跡生成や教師データ生成の一部で使用できると考えています。

詳細は未定ですが、

「方策出力のみで指すDL系AIによる連続対局の勝率」と「浅い探索の

NNUE系AIによる連続対局の勝率」が大きく違なる場合、局面の難易度が高いと判断し「長い時間探索したAIによる連続対局」を多めに行う

といった事も行いたいと考えています。

本番までに余裕があったらやる事

・持ち時間管理部を改造する

形成に大きな差がついてからは持ち時間に差があっても逆転する事は難しいと思うので、それを念頭に置いた持ち時間管理を行いたいと考えています。

詳細は決まってないですが、DR4バージョンの序盤モードの持ち時間管理をベースに作ろうと今のところは考えています。

[DR4バージョンの序盤モードの持ち時間管理(ざっくり)]

1.事前に決めておいた「中終盤のために残しておく時間」を現在の残り時間から引く

2.「現在の局面の終局までの推定手数」から、事前に決めておいた「終局までの推定手数がこの値以下になった時に中終盤モードに移行する値」を引いたものを「中盤までの推定手数」とする

3.残り時間を「中盤までの推定手数」で割ったものに、加算時間や秒読みを足したものを「今の手番で使う時間(のベース)」に設定

4.一定の条件を満たした場合は予定の時間より早く打ちきったり、逆に延長したりする。(時間は余裕を持って計算しているので足りなくなる事はあまりない)

ちなみに、本番では「終局までの推定手数」の精度が低かったせいで思うように行きませんでした。

~~・局面生成AIを教師データ生成などに応用する~~

~~DR4では簡単な局面を生成できるところまではいったものの、色々あってGPU資源がなくなってしまったために結局応用する事はできませんでした。~~

~~いつかは覚えてないですが、DR4のアピール文書~~
~~(https://drive.google.com/file/d/16P0Gcv4K0yo8v4GGH2zo2QQ290EnWeHd/view)に書いていた「評価関数の弱点の克服方法」について、より良い案を思いついたので余裕があったら実装したいと考えています。~~

=> 間に合いそうにないので今回はやりません。

ただ、個人的には局面生成AIにゼロからの強化学習くらいロマンを感じるので、いつかリベンジする予定です。

使用予定のライブラリ

・python-dlshogi2(https://github.com/TadaoYamaoka/python-dlshogi2)

シンプルな構造でそれなりに強く、かつ使い慣れているため、Ari ShogiやNNUE学習部のベースとして採用しています。

・やねうら王(https://github.com/yaneurao/YaneuraOu)

自分が知っている「NNUEが利用できる探索部」のなかで最強なので、NNUE系の探索部として採用しています。

・dlshogi(https://github.com/TadaoYamaoka/DeepLearningShogi)

探索部のパラメータ調整スクリプトや、教師データの変換スクリプトが便利なので、その部分だけ利用しています。

・KomoringHeights(https://github.com/komori-n/KomoringHeights)

USIプロトコルで利用できて、かつ性能が良さそうなので、詰め将棋エンジンとして採用しています。

使用予定のデータ / 公開モデル

・AobaZeroのデータ(http://www.yss-aya.com/aobazero/)

・floodgateのデータ(http://wdoor.c.u-tokyo.ac.jp/shogi/index.html)

・水匠1手200万手のデータ
(https://drive.google.com/file/d/1R9kI3xDKeoIjyFPDORS6K-1wwck

o75fr/view ）

・水匠1000万ノードの公開データ
（ https://drive.google.com/file/d/1VyP4MX_AuQhvy8sesymgPVf9sUnQoGPl/view ）

・dlshogi_with_GCTのデータ
（ https://tadaoyamaoka.hatenablog.com/entry/2021/05/06/223701 ）

・書籍「強い将棋ソフトの創りかた」のデータ

いずれのデータも、質も量も十分以上なため採用しています。

・Qhapaq Pretty Derbyのデータ
（ https://qhapaq.hatenablog.com/entry/2021/11/23/220251 ）

採用理由：振り飛車評価関数を簡単に作れるから

振り飛車をコンセプトにするわけではないですが、振り飛車評価関数を一部で採用する予定なので、その振り飛車評価関数の学習に利用します。

~~・shogi_suisho5のデータ_depth9（ https://huggingface.co/datasets/nodchip/shogi_suisho5_depth9 ）~~

~~・shogi_hao_depth9のデータ（ https://huggingface.co/datasets/nodchip/shogi_hao_depth9 ）~~

~~ゼロからNNUEを学習させる事ができる量/質である事が分かっているデータなので、ゼロからNNUEを学習させる場合は採用します。~~

SSDの容量が足りないし、ゼロからNNUEを学習するのを断念したので多分使いません。

・Háo（ https://github.com/nodchip/tanuki-/releases/tag/tanuki-.halfkp_256x2-32-32.2023-05-08 ）

・BLOSSOM（ https://twitter.com/senninha_a/status/1654675205464395

9872 ）

・水匠5(https://github.com/mizar/YaneuraOu/releases/tag/v7.5.0 ）

・Lí(https://github.com/nodchip/tanuki-/releases/tag/tanuki-wcsc33-2023-05-04 ）

~~ゼロからのNNUE評価関数を学習させるのを断念した場合は、NNUE評価関数のベースとして採用する予定です。(単体でベースにする以外にも、キメラ化したものをベースにする可能性もある ）~~

~~特にHáoは、3月末の時点で1番強い評価関数(Háoよりレート10～20ほど強い)のベースになっているので、採用される確率が1番高いです。~~

採用理由：強いから

3月末の時点の状況だと、NNUEをゼロから学習させるのは無理そうなのでこれらの評価関数を組み合わせて作ったキメラ評価関数をベースにする予定です。

標準型NNUEを作るのに標準型以外のNNUEを素材として使用しり予定になっているのは、ネットワーク構造がある程度違うNNUEを使ったキメラも作れる方法を採用する予定だからです。

その他

・本当は会場に行って参加したいですが、家から遠いので今年もオンライン参加です。

・アピール文書は後で追記したものに差し替える予定です

以下、おまけなので文章とかより一層雑です

おまけ1：モデル精度について色々

モデル名	パラメータ数	方策正解率	価値正解率	備考
WCSC33のモデル	1687306	0.443736	0.7185835	テストデータの一部が学習データに含まれている
23年11月モデルA	3213186	0.445989	0.7198876	テストデータの一部が学習データに含まれている
23年11月モデルB	3213186	0.4552351	0.7279302	テストデータの一部が学習データに含まれている
DR4で使ったアンサンブルモデル	8113678	0.463778	0.7344486	WCSC33のモデル + 23年11月モデルA + 23年11月モデルB
SplatBrella_231219のアンサンブルモデル	12296782	0.475275	0.7415995	WCSC33のモデル + 23年11月モデルB + python-dlshogi2のモデル
WCSC34に向けて育成中のモデルの1つ	7164218	0.463992	0.727845	学習途中でまだまだ伸びそう
python-dlshogi2のリポジトリのモデル	7396290	0.4785397	0.7409203	
GCT電竜	?	0.46163161	0.73495564	データは https://tadaoyamaoka.hatenablog.com/entry/2021/09/11/155938 より引用
dlshogi with GCT	?	0.48964214	0.75278598	データは https://tadaoyamaoka.hatenablog.com/entry/2021/09/11/155938 より引用
dlshogi dr2_exhi	?	0.52322546	0.76564239	データは https://tadaoyamaoka.hatenablog.com/entry/2021/09/11/155938 より引用
tanuki-wcsc28	32117089	-	0.66378727	方策出力が無いので価値正解率のみ
水匠5	32117089	-	0.68430669	方策出力が無いので価値正解率のみ
BLOSSOM	32117089	-	0.68488171	方策出力が無いので価値正解率のみ
Háo	32117089	-	0.68491119	方策出力が無いので価値正解率のみ

・データはGCTの公開ノートブック
(https://tadaoyamaoka.hatenablog.com/entry/2020/11/26/203912)で使われているテストデータと同じ

・パラメータ数は
https://rheinmetall.hatenablog.com/entry/2021/05/14/000000 のコードで測定

・「23年11月モデルA」は「入力特徴量がバグっていたのに気づいて途中で学習を止めたモデル」

・「python-dlshogi2のリポジトリのモデル」は
https://github.com/TadaoYamaoka/python-dlshogi2/blob/main/checkpoints/checkpoint.pth のモデルの事。精度を比較したり、アンサンブルの効果を確かめたりするために利用している。

・比較のために、GCT電竜、dlshogi with GCT、dlshogi dr2 exhiのデータを https://tadaoyamaoka.hatenablog.com/entry/2021/09/11/155938 から引用している。

・昔の記録を発掘して、そこに書いてある数値を確認せずにコピペしているだけなので、間違っている可能性がある

・SplatBrella_231219は、モデルとパラメータを軽く弄って(ついでにいらない機能をOFFにして)floodgateに放流していた Ari Shogi and フレンズ。ペタショック定跡に(有利な先手番でだが)1発入れた時のもの。

http://wdoor.c.u-tokyo.ac.jp/shogi/view/2023/12/19/wdoor+floodgate-300-10F+SplatBrella_231219+YOV800-peta4M-3700X+20231219190001.csa

SplatBrellaという名前は、某イカのTPSゲームで私がここ数年愛用している傘の英名からとっています。

[感想とか考察とか]

・DR4のモデルでは、単体での精度が1番高いモデルから正解率が方策/価値ともに上昇している。探索速度は落ちたものの、DR4での構成では探索速度の低下によるデメリットをあまり感じなかったので、大会直前にこれを実装したのは成功だったと思う。

・SplatBrellaのモデルは、単体での精度が1番高いモデルからほとんど変わっていない。(むしろ若干下がっている)

1番精度の高いモデルと2番目以降のモデルの精度に大きな差があったので、足を引っ張ってしまったのかもしれない。(が、より複雑な方法でアンサンブルを行った場合は、これでも精度が上がる可能性があると自分は考えている。この辺は余裕があったら実験する。)

=>(追記)「1番精度の高いモデルと2番目以降のモデルの精度に大きな差があったため、足を引っ張って精度が上がらなかった」という説を確かめるための実験を行った。

モデル名	パラメータ数	方策正解率	価値正解率	備考
WCSC33のモデル	1687306	0.443736	0.7185835	テストデータの一部が学習データに含まれている
23年11月モデルA	3213186	0.445989	0.7198876	テストデータの一部が学習データに含まれている
23年11月モデルB	3213186	0.4552351	0.7279302	テストデータの一部が学習データに含まれている
DR4で使ったアンサンブルモデル	8113678	0.463778	0.7344486	WCSC33のモデル + 23年11月モデルA + 23年11月モデルB
SplatBrella_231219のアンサンブルモデル	12296782	0.475275	0.7415995	WCSC33のモデル + 23年11月モデルB + python-dlshogi2のモ
WCSC34に向けて育成中のモデルの1つ	7164218	0.463992	0.727845	学習途中でまだまだ伸びそう
python-dlshogi2のリポジトリのモデル	7396290	0.4785397	0.7409203	
GCT電竜	?	0.46163161	0.73495564	データは https://tadaoyamaoka.hatenablog.com/entry/2021/09/11/155
dlshogi with GCT	?	0.48964214	0.75278598	データは https://tadaoyamaoka.hatenablog.com/entry/2021/09/11/155
dlshogi dr2_exhi	?	0.52322546	0.76564239	データは https://tadaoyamaoka.hatenablog.com/entry/2021/09/11/155
tanuki-wcsc28	32117089	-	0.66378727	方策出力が無いので価値正解率のみ
水匠5	32117089	-	0.68430669	方策出力が無いので価値正解率のみ
BLOSSOM	32117089	-	0.68488171	方策出力が無いので価値正解率のみ
Háo	32117089	-	0.68491119	方策出力が無いので価値正解率のみ
WCSC34に向けて育成中のモデルの1つ(その後)	7164218	0.4717394	0.7350589	伸びしろはもうそんなに残ってなさそうだが、まだ少しは伸び
2024年3月30日の実験モデル	14560508	0.4842954	0.7458868	pydlshogi2のモデル + 育成中のモデル(その後)

(表の赤色のところが新しく調べた場所)

「2024年3月30日の実験モデル」は、「WCSC34に向けて育成中のモデルの1つ(その後)」と「python-dlshogi2のリポジトリのモデル」の2つのモデルの出力を単純に平均しただけのものだが、ちゃんと正解率が上昇している。

なので、「SplatBrellaのモデル精度が、python-dlshogi2のモデル単体からほとんど上がらなかったのは、他のモデルが足を引っ張っていたから」という説は多分あっている。

ただ、あるモデルが"有望ではない"としてほぼ0%を出力した指し手でも、他のモデルがある程度の値を付けた場合は探索される可能性が出てくるため、実際に探索に組み込んだ際はテストデータでの精度以上に違いが出る(良い方向でも悪い方向でも)のではないか、と考えている。(試せてはない)

・DR4で使った3つのモデルは「2023年に作ったDL評価関数ランキングtop3」だが意外と精度が出ていない。

「WCSC34に向けて育成中のモデル」は、学習させ始めてから3日くらいしか経ってないのに、DR4のアンサンブルモデルと同じくらいの精度が出ている。(DR4のモデルはテストデータの一部が学習データに含まれているというハンデがあるので、実際の精度は育成中のモデルのほうが高い可能性もある)

(2024年1月の末にPCが新しくなるまでは)GoogleColab無料版などの「無料で利用できるGPU」を使ってチマチマと学習を進めていた(一応、メインのノートPCにはGPUが載っていたが、酷使しすぎたせいか、GPUに負荷をかけるとバッテリー周りの挙動が怪しくなるようになっていたので、あまり学習には投入できなかった)のが、新しくなってからはRTX4060Ti(メモリ16GB)がほぼ24時間フル稼働、と環境が大きく変わったのはあるが、それにしても「3日学習させたモデル >= 去年1年間の集大成ともいえるモデル」というのは少し残念。

おまけ2: DR4バージョンのAri Shogi and フレンズのfloodgateでの計測

ハードウェア含め本番と同じ構成で計測。対局数が少ないのでレートはあまり信頼できないが、少なくとも4000は超えていると思う。

asa35	4257	61	35	0.635	2023-08-08	4168
SV-037	4254	75	47	0.614	2023-09-03	4165
test_13900k	4252	39	15	0.712	2023-06-29	4163
b22sd	4242	45	30	0.598	2023-12-05	4153
ECLIPSE-20230523_R9-5900HX	4241	24	9	0.728	2023-07-25	4152
GloriousMoment	4240	22	5	0.799	2023-11-28	4151
FAREWELL_R9-5900HX	4234	221	136	0.619	on line	4145
Sagittarius	4232	104	68	0.603	2023-12-02	4143
VW201	4229	16	7	0.691	2023-06-18	4140
Ari_Shogi_and_Friends_DR4	4221	12	4	0.735	2023-12-05	4132
S.novi-belgii	4219	16	1	0.933	2023-10-15	4130
GGG	4217	52	29	0.640	2023-09-14	4128
ECLIPSE-20230717_R9-5900HX	4215	31	16	0.659	2023-07-25	4126
ECLIPSE_Ryzen9-5900HX	4214	26	14	0.648	2023-07-03	4125
dltest	4210	14	4	0.758	2023-10-17	4121
KANROJI_MITSURI	4209	31	15	0.662	2023-11-03	4120
Hao-s-book_black-i5-13400	4204	21	2	0.913	2023-12-06	4115
nnue-tanuki-dr3_5800u	4203	51	32	0.616	2023-11-22	4114
SV-q044	4200	222	36	0.859	2023-10-19	4111
B1112	4199	343	240	0.588	on line	4110
HoneyWaffle-37-003	4195	879	666	0.569	2023-12-02	4106

引用元：http://wdoor.c.u-tokyo.ac.jp/shogi/x/rating/players-floodgate-20231207.html

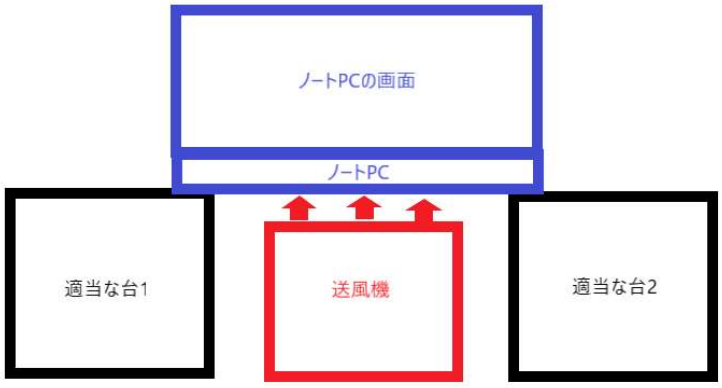
おまけ3. 第4回電竜戦本戦の感想とか

・大会前は「モデル精度が低い」「モード切替機能が微妙」「定跡がない」「探索部や合議パラメータを本来は連続対局の結果を元に自動調整する予定だったが、リソース不足でちゃんとできなかった」「無理な運用を続けたせいでハードウェアが劣化してきている」といった理由から、「あまり良い結果にはならなさそう」と予想していた

・特にハードウェアは、大会前/大会中に故障しないかヒヤヒヤしていた。長期間無理な運用をしてきたせいなのか、GPUに強い負荷をかけると「バッテリーが認識されなくなる」や「電源に接続されているはずなのにバッテリーで稼働中(電源に接続されていない)と表示される」といった現象が発生したし、比較的軽めの負荷でも長期間稼働させると「いつの間にか電源が落ちている」や「画面が真っ暗になり、一切の操作を受け付けなくなる(が、

本体は高温になっているしファンも稼働しているので恐らく電源はついている)」といった現象が発生した。

・大会中も、ずっと電源に接続しているのにも関わらず「バッテリーが検出されません」みたいな表示と「充電中」という表示が高速で切り替わる、という事が起きていた。多分そのせいで、大会中は下の画像のようにしてPCを冷やしていたのに、CPUのクロックがかなり落ちてしまっていた。



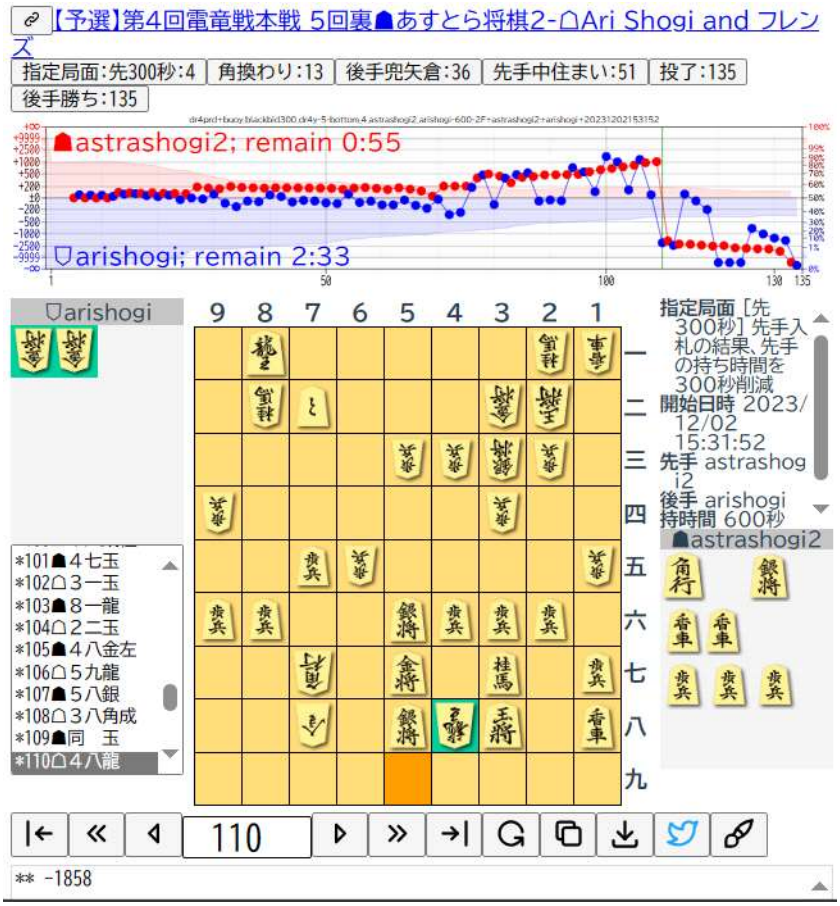
・大会で使ったPCは、大会後の12月の末に「動作中に異臭がする」という事が起きてしまったので封印する事になった。一応起動もできるし、データは無事で取り出す事もできるが、火事になるのは絶対に避けなければならないので、(データを取り出すなどの)理由がない限りは起動しないようにしている。

PCが壊れたせいで、1ヵ月くらい開発速度が大きく低下し、モチベーションも激減したが、「気づかないうちにPCから出火して家が火事になる」とか「大会前/大会中に壊れる」とか「(バックアップ取ってないのに)データが全部消える」とか、そういう事が無かったのでまだ良かったと思う。

・それでもB級に進出し、しかもB級最下位を回避したのは、非常に運が良かったからだと思う。(実際、C級上位陣や、2日目に参加しなかったQhapaqチームやアストラ将棋チームのほうが強いと思う)

WCSC33も強運で1次予選を突破していたし、2023年は将棋AIに関しては非常に運がよかった。(2023年の初詣で引いたおみくじの結果はたしか"凶"だったけど)

・個人的に1番嬉しかった勝ちは1日目5回裏の対あすとら将棋2戦。相手が格上だったので、5回表の先手千日手で満足していたのだが、終盤に相手が読み抜けしたので、2戦1.4勝0.6敗という想像もしなかったレベルの非常に良い結果に終わった。



引用元: https://denryu-sen.jp/denryusen/dr4_production/dist/#/dr4prd+buoy_blackbid300_dr4y-5-bottom_4_astrashogi2_arishogi-600-2F+astrashogi2+arishogi+20231202153152/110

・個人的に1番悲しかった負けは2日目の5回表の対ECLIPSE戦。後手番で600以上まで行ったのに、そこから逆転して負けた。

引用元：https://denryu-sen.jp/denryusen/dr4_production/dist/#/dr4prd+buoy_blackbid300_dr4b-5-top_4_eclipse_arishogi-600-2F+eclipse+arishogi+20231203141035/91