DESENVOLUPAMENT D'UN MOTOR D'ESCACS

Ferran Guasch

Resum– En les últimes dècades, els motors d'escacs han passat a dominar el joc de les seixanta-quatre caselles aconseguint un nivell de joc superhumà. Desafortunadament, la seva força es basa en el càlcul profund de variants, en comptes d'una comprensió posicional excel·lent. Des de l'aparició d'*AlphaZero* s'ha començat a donar més importància per aconseguir aquesta capacitat estratègica, però encara s'està lluny d'aconseguir el motor perfecte. Per contraposició al model tradicional, en aquest treball de recerca s'investiga la creació d'un motor basat només amb sentit estratègic. Per realitzar l'estudi s'ha desenvolupat un motor d'escacs completament funcional, i s'han estudiat els resultats de diferents arquitectures neuronals per la funció d'avaluació, amb l'objectiu d'aconseguir un nivell de joc semblant a la d'un motor convencional.

Paraules clau– Escacs, Profunditat De Cerca Limitada, Xarxes Neuronals, Aprenentatge Profund, Stockfish, AlphaZero

Abstract– In recent decades, chess engines have gone on to dominate the game of the sixty-four squares, achieving a superhuman level of play. Unfortunately, its strength is based on the deep calculation of variants rather than an excellent positional understanding. Since the appearance of *AlphaZero*, the focus has begun to shift into developing this strategic ability, nonetheless, we are still far from achieving the perfect engine. In contrast to the traditional approach, this research focuses on the creation of an engine based only on strategic sense. To carry out the study, we have developed a fully functional chess engine, and, with the aim to achieve a similar level of play to a conventional one, we have studied the results of different architectures for the evaluation function.

Keywords– Chess, Limited Look Ahead, Neural Networks, Deep Learning, Stockfish, AlphaZero

1 Introducció

Des del motor d'escacs *Deep Blue* de 1997, que va derrotar a Garry Kasparov en un matx ple de controvèrsia, fins a l'actualitat, tots els motors amb un nivell de joc elevat s'han basat en maximitzar l'exploració i profunditat de l'arbre de moviments per escollir-ne el millor. Per altre banda, els millors jugadors d'escacs es diferencien de la resta de professionals per la seva capacitat de comprensió estratègica, abans que per la seva capacitat de càlcul.

Per aquesta raó, en aquest treball d'investigació es vol observar si es possible crear un motor d'escacs relativament fort, que sigui capaç d'avaluar les posicions de forma estratègica. Per aconseguir-ho s'ha desenvolupat un motor d'escacs totalment funcional, on la funció d'avaluació es realitzada per xarxa neuronal. Per assegurar-nos que es basa principalment en conceptes estratègics, fixarem i reduirem la profunditat de cerca de l'arbre de moviments. Perquè la xarxa neuronal aconsegueixi aquest sentint posicional, l'entrenarem perquè donada una posició ens indiqui la avaluació objectiva de la mateixa, un valor decimal donant pel motor més fort actualment, *Stockfish*[1]. D'aquesta manera, per aconseguir bons resultats la xarxa neuronal haurà de trobar els patrons i les interrelacions entre les peces que fa que una posició sigui millor que una altre.

2 Objectius

L'objectiu principal del projecte consisteix en crear un motor d'escacs funcional, des de zero, i que la seva força de joc estigui definida per la seva capacitat de estratègica. Per aconseguir-ho, es definirà la funció d'avaluació com una xarxa neuronal, amb el pretext de que sigui capaç de

• E-mail de contacte: Ferran.GuaschG@autonoma.cat
• Menció realitzada: Computació
• Treball tutoritzat per: Dimosthenis Karatzas (CVC)
• Curs 2021/2022

codificar cert coneixement posicional i estratègic, amb l'objectiu que donada una posició ens pugui indicar la avaluació de la mateixa o aconsellar la millor jugada segons el motor més fort actualment, *Stockfish*. La idea és que la xarxa neuronal, no veurà ni el passat ni el possible futur de la partida, només la posició actual i a partir d'aquesta haurà de ser capaç de aconseguir una avaluació prou encertada.

Concretament, es volen aconseguir els següents objectius.

- Un motor d'escacs totalment funcional, amb la capacitat de crear moviments com donar avaluacions per posicions d'entrada.
- Una funció d'avaluació basada en una xarxa neuronal, que juntament amb un *Minimax* limitat en profunditat, permeti al motor escollir amb més un 65% de les millors jugades en certes posicions segons *Stockfish*.
- Independentment dels resultats de la secció anterior, realitzar un estudi, amb fonamentació experimental, d'altres arquitectures basades en *Deep Learning* per la funció d'avaluació, i comparar els resultats.

3 METODOLOGIA

Per arribar als objectiu mencionats anteriorment, s'ha decidit desenvolupar el projecte en tres iteracions principals. En la primera iteració s'ha construït el motor pràcticament sencer. En la segona, s'ha creat una primera arquitectura per la funció d'avaluació i s'ha ajuntat en el motor. I en la tercera, s'han estudiat noves arquitectures i comparat els resultats.

Per assegurar el bon desenvolupament del projecte, s'han fet reunions, pràcticament setmanals amb el tutor del projecte. En aquestes s'ha parlat de possibles modificacions i correccions a realitzar per el treball fet prèviament, s'han resolt dubtes i s'ha debatut quines arquitectures per la funció d'avaluació serien més interessant provar.

Pràcticament per tota la generació de codi s'ha utilitzat *Python 3.9* sobre una arquitectura *Linux* i utilitzant *Neovim* com editor principal. El model d'avaluació ha estat escrit amb la llibreria de *Pytorch*, s'han fet servir els serveis de *Google Colab* i *Kaggle* per entrenar-lo gràcies a la capacitat d'utilitzar *GPUs* gratuïtament. S'ha utilitzat *Wandb* per la monitorització dels entrenaments.

De forma general, s'han consultat les fonts Chess Programming Wiki, Chess Programming i per tots els temes relacionat amb els motors d'escacs, i per informació més formal, actualitzada, i sobretot referent a la part d'intel·ligència artificial, s'ha utilitzat Arxiv.org, Github i Pytorch Docs.

4 ESTAT DE L'ART

Degut a la demanda i interes general per part de la comunitat escaquísitica, cada any s'observa un progrés de més de 20 punts d'ELO[4] en la força dels motors, que corresponen a un avanç significatiu tenint en compte que ja tenen un nivell de joc molt elevat. Últimament aquest avenços han sigut gràcies a noves propostes per definir la funció d'avaluació, tot i així, sovint, aquesta millora es donada pels avenços en altres tècniques *software* que suposen una millora en la capacitat de càlcul. Un exemple molt rellevant, seria la introducció del model de representació del taulell basat en *Magic Bitboards* que va suposar una millora molt important davant de la generació de moviments legals per la capacitat d'aconseguir els atacs de cada peça sobre cada casella en una complexitat temporal de $O(1)$.

Respecte al sistema d'avaluació de la posició, component necessària a qualsevol motor i que l'indueix d'aquesta capacitat estratègica, és on trobem més divergències en implementacions i paradigmes, i és on precisament cada motor s'intenta diferenciar. Tot i això podem distingir tres enfocaments diferents. Primer, l'utilitzat per *LcZ*, *LeelaChess Zero*, una re-implementació de les idees publicades al paper de *AlphaZero*[11], que utilitza el paradigma de *Reinforcement Learning* amb una arquitectura basada en *Resnets*[9] i *MonteCarlo Tree Search*[10] per un refinament de les prediccions. Seguidament, l'utilitzat per *Stockfish 14* que incorpora una tècnica innovadora, provinent dels motors de *Shogi*, per sobre d'un sistema d'avaluació convencional. Aquesta és l'anomenada *NNUE* (*EUNN: Efficiently Updatable Neural Networks*) [5] una xarxa neuronal *fully connected* amb només quatre *hidden layers*, molt més petita que la utilitzada per motors com *LcZ*, que s'executa a la *CPU* i és extremadament ràpida, és capaç d'analitzar 60 milions de posicions per segon, a contraposició de les només 40 mil per *LcZ*. La seva propietat característica, de manera molt simplificada, és que aquesta arquitectura és capaç de donar l'avaluació d'una posició sense tornar a fer inferència sobre tot el model, quan té un punt de referencia proper. En altres paraules és capaç de donar-te l'avaluació d'una posició només utilitzant una part de la xarxa, si sap l'avaluació de la posició anterior, per exemple. Aquest fet contribueix de manera important en la seva excepcional velocitat. I finalment, l'utilitzat pels motors més tradicionals, on la funció d'avaluació consta d'unes heurístiques que s'aproximem a com un humà interpreta la posició, amb l'afegit d'altres valors constants i consideracions de la posició que s'han demostrat que milloren l'avaluació.

Respecte a predecessors que hagin desenvolupat la idea que ens proposem en aquest projecte, la creació d'un motor amb una capacitat estratègica predominant, observem els resultats de *Playing Chess With A Limited Look Ahead* [8] de *Arman Maesumi*, on aconseguiren que el seu motor predigues el millor moviment de la posició segons *Stockfish* en un 83% de les posicions del seu set de test. La funció d'avaluació era definida per una xarxa neuronal *fully connected*, com la nostra proposta inicial per la funció d'avaluació. En aquest paper, l'autor aconseguia que el model predigués l'avaluació de la posició, segons *Stockfish* amb una profunditat de 12 nodes, amb només $85cp$[1] de diferencia. Aquests són un resultats excel·lents, que semblen indi-

[1] Cp: Centi-pawns, mesura utilitzada pels motors per tractar amb les avaluacions de les posicions, que indica una centèsima de peó.

car que un model relativament simple ja es capaç de aconseguir una alta compresió estratègica. Per aconseguir-ho va decidir codificar la posició en *bitboards*[6.2] afegint-hi els bits necessaris per codificar el color a jugar, les permissions d'enroc i. curiosament, per indicar si el blanc o el negre es troben en escac, i seguidament va crear un *Encoder* de $775 \rightarrow 512 \rightarrow 256 \rightarrow 128$ nodes per capa i després utilitzar aquestes sortides per entrenar l'*MLP* de $128 \rightarrow 2048 \rightarrow 2048 \rightarrow 2048 \rightarrow 1$[2].

També a *Matthia Sabatelli et al. 2018* [12], on es fa un estudi semblant, arribaren a la conclusió que un *Multilayer perceptron (MLP)* tenia potencialment més capacitat per funcionar com a funció d'avaluació que una *Convolutional Neural Network (CNN)*, i també que s'aconseguiren millors resultats quan el model era entrenat amb la representació del taulell amb *bitboards*[6.2]que quan era representat amb notació algebraica. Tot i així, en aquest cas només aconseguiren bons resultats en tasques de classificació, on les avaluacions donades per *Stockfish* havien estat truncades en posicions guanyades, taules i perdudes. Per altre banda, en els problemes de classificació, on havien dividit les tres classes anteriors en sis més especifiques, van aconseguir aproximadament un 60% en el test en la seva millor arquitectura i dades d'entrada (Bitboards, i MLP) amb $769 \rightarrow 2048 \rightarrow 2048 \rightarrow 2048 \rightarrow 1050 \rightarrow 20$.

5 Coneixements Previs

Per entendre en totalitat aquest projecte és important tindre un coneixement bàsic dels escacs i les seves normes. Respecte als motors, indicar que consten de tres blocs fonamentals. El primer és de la manera que internament representen el taulell. Actualment, la gran majoria de motors utilitzen una representació basada en *bitboards* (observar secció 6.2), ja que és de les més ràpides. Aquesta s'aprofita que el taulell té 64 caselles, representant la posició cada tipus de peça sobre el taulell amb números de 64 bits. El segon bloc correspon a la generació de moviments. Aquí es defineix les regles del joc, de manera que el motor només calculi i jugui moviments legals a la posició. El tercer bloc correspon a la decisió del moviment a fer. Aquí és on s'examina l'arbre de possibles moviments, amb algorismes com *Minimax*[3] o *Alpha Beta Prunning*[2], on junt amb la funció d'avaluació, acaben escollint el següent moviment a fer.

També és important indicar que tot i que aquest motors van ser dissenyats amb la idea d'aconseguir el nivell de joc més alt possible per compatir contra els humans o altres motors, actualment s'utilitzen majoritàriament, i pràcticament de forma exclusiva, per analitzar posicions i crear nova teoria d'obertures. Tan és així, que en els últims campionats del món, els equips particulars del dos participants han dedicat una part del temps i pressupost molt significatives en només validar noves idees i posicions amb els millors motors sobre ordinadors molt potents.

Finalment, recordar que la força dels motors convencionals es basa en la seva capacitat de càlcul i, per tant amb la rapidesa per aconseguir una avaluació numèrica de la posició. En altres paraules, la funció d'avaluació és una peça clau del motor però no necessàriament més important que les altres peces que el composen.

[2]A mitjans del projecte es va contactar amb l'autor de l'article, per diverses raons, però se li va preguntar perquè aquesta arquitectura i simplement va indicar que havia estat prova i error fins arribar a bons resultats.

6 Experimentació

A grans trets, un motor d'escacs està format per una representació del taulell, una funcionalitat per generar tots els moviments legals i, finalment, una funcionalitat per recórrer l'arbre de possibles moviments i escollir-ne el millor, segons una certa funció d'avaluació.

En les següents subseccions es parlarà sobre el funcionament bàsic del motor i de la manera que s'ha decidit desenvolupar.

6.1 Arquitectura

El motor s'ha desenvolupat de manera jeràrquica i sota els principis de desenvolupament *software* de *Single-responsibility*, on cada classe només s'encarrega d'una funció específica, i d'*Open–closed*, on el sistema permet afegir mòduls, que implementen noves funcionalitats, sense canviar el codi controlador.

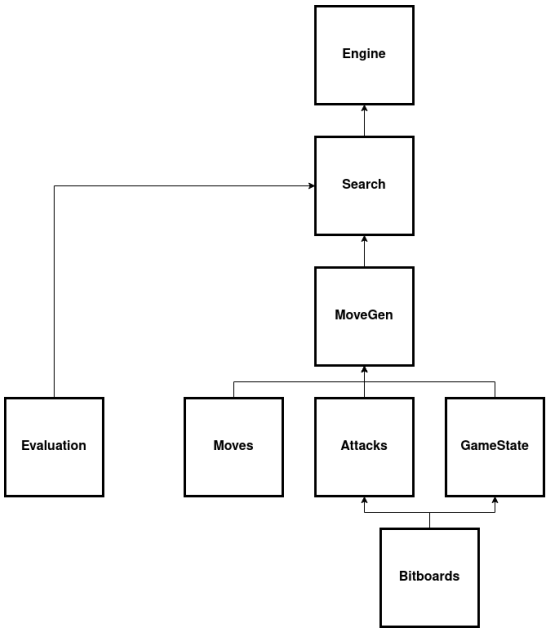


Fig. 1: Model jeràrquic de l'arquitectura *software*

Al nivell més baix de la jerarquia tenim les classes *GameState* que defineix l'estat de la partida, que inclou els bitboards, bits d'estat i altres funcionalitats com serialitzar les dades per passar-les a la funció d'avaluació. La de *Moves:* que conté una llista per guardar els moviments a fer, i disposa de diverses funcions per codificar i descodificar-los. [6.2.2]. I, la d'*Attakcs:* que crea les taules d'atacs per cada peça a cada casella i en qualsevol posició, i que conté les funcionalitats per accedir a aquests valors [6.2.1]. Les primeres dues necessiten informació sobre com són els *bitboards* i les operacions que es poden fer entre ells, junt amb la definició d'altres constants com els *Magic Numbers*.

En el segon nivell trobem el *MoveGen*, que s'encarrega de generar tots els moviments legals d'una posició. Seguidament, tenim la classe identificada com a *Search* en el diagrama, que crea i recorre l'arbre de cerca utilitzant *Negamax* (Variació de *Minimax* que per cada *ply*[3] maximitza la negació de l'avaluació del moviment) , i que depèn del sistema d'avaluació, una classe que conté el model i les funcions necessàries per realitzar inferència sobre el mateix per aconseguir l'avaluació numèrica de la posició.

Finalment, tenim la classe controladora, que com tal, només s'encarrega de definir i començar el joc.

6.2 Representació del Taulell

Des dels primers motors d'escacs que la representació del taulell s'ha basat en un sistema de codificació binària per aconseguir més velocitat. Existeixen moltes representacions possibles, però la més utilitzada es basa en *bitboards*. Aquest tracta de construir un vector dotze números de seixanta-quatre bits, un per cada peça possible i omplert amb uns i zeros segons on es troba la peça indicada a la posició actual. En el següent exemple 2 tenim una posició a la dreta, i a l'esquerra observem un exemple de com serien els *bitboards* pels peons i torres dels dos colors.

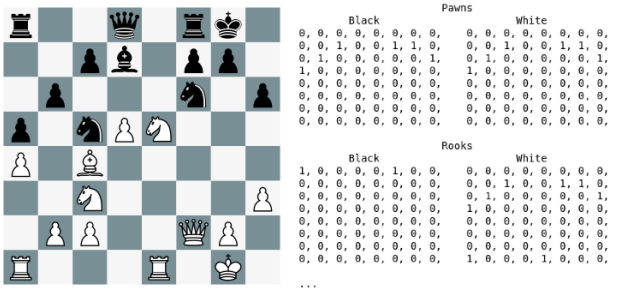


Fig. 2: Representació amb *bitboards*

Aquest guany de velocitat que proporcionen respectes les altres alternatives, és causat per la velocitat d'execució de les operacions entre aquest. Aquestes operacions es defineix com:

- **get_bit**: funció per aconseguir el bit d'una casella en concret d'un *bitboard* donat.
- **set_bit**: funció per posar el bit d'una casella en concret d'un *bitboard* donat.
- **pop_bit**: funció per eliminar el bit d'una casella en concret d'un *bitboard* donat.
- **count_bits**: funció per contar el número de bits en un *bitboard*.
- **ls1b_index**: funció per obtenir el bit menys significatiu amb valor d'u.

Aquestes dos últimes són essencials per la creació de taules d'atac, sobretot per crear el que serien els *bitboards* de peces ocupades[4].

[3] Una *ply* és una jugada entesa com a un sol moviment del blanc o negre, mentre que un *move* es considera una jugada d'un costat i la seva resposta.

[4] *Occupancy bitboards*: *Bitboards* que indiquen l'ocupació, sense distinció de les peces, de cada casella del taulell, per cada color o ambdós. Molt útils per fer operacions binàries entre *bitboards* sencers.

6.2.1 Generació de moviments

Existeixen moltes estratègies per la generació de moviments. Per la realització d'aquest treball s'ha escollit la més famosa en els últims motors degut a la millora de velocitat que proporciona. Aquesta és l'anomenada *MagicBitboards*[7], que permet accedir a unes taules precalculades, a l'inici de l'execució, com si fos una base de dades amb les màscares de bits que indiquen els moviments possibles d'una peça donada una posició.

Per entendre el seu funcionament utilitzarem el següent exemple.

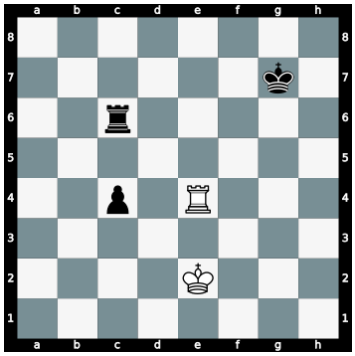


Fig. 3: Posició d'exemple

Imaginem que tenim aquesta posició, i volem que el nostre motor ens digui els moviments legals que té la torre. Per aconseguir-ho, el motor, només necessitarà observar a les taules precalculades i donar-nos el resultat. Per fer-ho, només necessitarà buscar el *"hash"* corresponent. Per obtenir-lo es seguiran els següent passos.

1. S'agafen tots els moviments que podria fer la torre, (menys l'última casella on pot accedir en qualsevol direcció, ja que tan si hi ha una peça en aquestes com si no el recorregut de la mateixa serà el mateix), i el taulell d'ocupació actual.

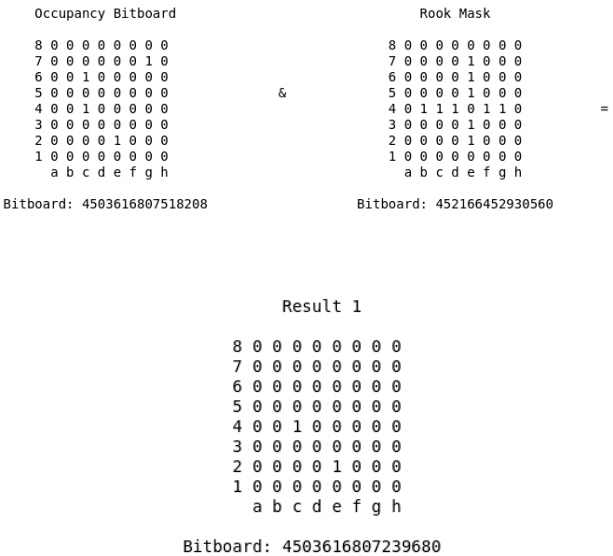
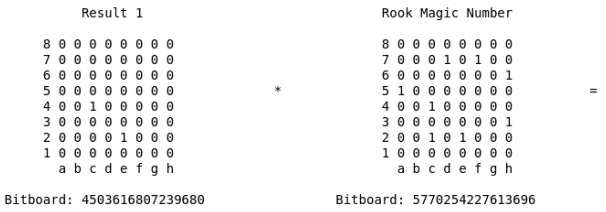


Fig. 4: Primer pas per la obtenció de la màscara d'atacs de la torre.

Observem que la torre, des d'e4, es pot dirigir a les caselles indicades per primer *bitboard*, i que les peces altres peces es torben a g7, c6, c4 i e2.

2. Seguidament, el resultat obtingut es multiplica per un *Magic Number*, que és únic i definit per la casella que es troba la torre.



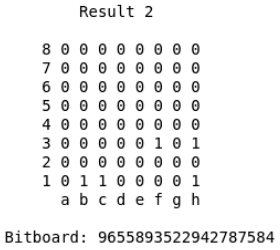


Fig. 5: Segon pas

Com s'observa, tant el *Magic Number* com el resultat de l'operació s'ha d'entendre com un número i no un *bitboard*, ja que aparentment semblen aleatoris. De totes maneres, tant els primers bits del resultat, com del nombre de bits en u que tenia la màscara de la torre, guarden informació útil.

3. Dit això, l'últim pas és agafar aquests bits i utilitzar-los com *hash* en la taula d'atacs possibles de la torre. Obtenint el resultat final.

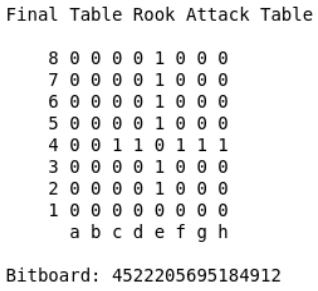


Fig. 6: Resultat de la màscara d'atacs de la torre.

Utilitzar *Magic Bitboards* no només afavoreix el temps d'execució, sinó també l'arquitectura *software* de la funcionalitat de generació de moviments. Aquesta, simplificadament, només necessitarà iterar per totes els *bitboards* de cada peça, i per cada bit en un de cadascun d'aquest *bitboards* haurà de realitzar els càlculs indicats anteriorment, aconseguint tots els moviments pseudolegals. Després haurà de filtrar totes el realment legals i posar-los en una llista.

6.2.2 Codificació moviments

Per aconseguir millor rendiment, dintre del motor també es codifiquen tots els possibles moviments en números enters, en aquest cas de 24 bits.

Fig. 7: Codificació moviments
(Enroc del rei blanc de e1 a g1)

D'aquesta manera el motor tampoc necessita fer conversions entre la informació intrínseca del taulell i dels mateixos *bitboards*, i a canvi, per realitzar la conversió a la notació l'anomenada notació algebraica (ex: e2e4) només necessita aplicar el *righ shift* necessari per recuperar la informació convenient.

6.2.3 Funció d'Avaluació

La comprensió de la posició i el sentit estratègic de qualsevol motor està codificat a la funció d'avaluació. Deguts als diferents estudis analitzats a l'estat de l'art[4], s'ha decidit estudiar la definició d'aquesta funció com una xarxa neuronal basada diferents arquitectures de *Deep Learning*, una primera com una *MLP*, i una segona com una *CNN*. També, segons els estudis realitzats anteriorment, i per la manera que el motor s'ha construït, s'ha decidit, que les entrades a aquestes xarxes neuronals, fossin els mateixos *bitboards* del motor, junt amb els bits d'estat de la posició. També s'ha decidit que la sortida del motor serà un valor decimal, degut a dues raons principals. La primera és perquè existeix un interès molt gran perquè el motor sigui capaç d'avaluar una posició única d'entrada amb una valor numèric[5] continu, per exemple, quan s'analitzen obertures, a partir d'un cert número de moviments es vol saber quin color està millor, i preferiblement de manera quantitativa, ja no t'interessa que el motor et segueixi donant les millors continuacions, ja que un cap humà seria capaç de memoritzar-les. La segona raó és perquè ens interessa avaluar la força del motor utilitzant una certa profunditat de càlcul, cert *Look Ahead*, per exemple quatre moviment endavant, simulant una estructura de pensament més similar a la humana, i per fer-ho ens interessa que hi hagi certa diferencia d'avaluació entre moviments, per molt semblant que sigui el seu efecte per definir el resultat de la partida.

Creació de Dades

Per entrenar el model necessitem la informació d'una certa posició, donada en *bitboards*, i una avaluació. Per aconseguir aquesta informació, primer s'ha descarregat un *PGN*[5] de Lichess Database, de més de 20Gb. D'aquest fitxer s'han extret més de 40 milions de *FENs*[6] i les avaluacions per

[5] Format estàndard per guardar partides d'escacs, els moviments fets i altre informació com dades dels jugadors, comentaris i avaluacions de la posició.

[6] Sistema codificació estàndard d'una posició en una cadena de caràcters.

cadascuna d'elles. Finalment, a partir d'aquestes *FENs*, s'han creat els *bitboards* i s'han guardat com a vectors junt amb 6 bits d'estat, color a jugar, capacitat d'enroc i possibilitat de jugar enpassant, aconseguint 40 milions de vectors de $12 \times 64 + 6 = 774$ elements.

Per millorar la comoditat i escalabilitat del tractament de les dades, s'ha dissenyat un petit sistema jeràrquic per ordenar-les. Aquest sistema adquireix les dades del fitxer *PGN* de manera automàtica, i les divideix en fitxers de com a màxim un milió de vectors. Simultàniament, el sistema enregistra a un *csv* el nom del fitxer que conté els *bitboards* i la quantitat de vectors que conté.

Perquè el xarxa neuronal pugui adquirir les dades, s'ha creat una *Data Loader*, de *Pytorch*, amb la capacitat que l'usuari indiqui quants fitxers de dades vol utilitzar per una sessió d'entrenament i test, i quants vol que es carregin a memòria simultàniament cada cop que el sistema demani una mostra que estigui fora del rang de les quals es troben a memòria en aquell moment. Aquesta arquitectura permet treballar amb tants fitxer de dades com es desitgi, intentant que es perdi el menor temps carregant en memòria els fitxers de dades, ja que cadascun d'aquests ocupa 780Mb de mitjana.

Metodologia d'entrenament

Els models s'han definit utilitzant *Pytorch*, i s'ha entrenat sobre els *notebooks* de *Google Colab* i *Kaggle* degut els serveis gratuïts de *GPU* que proporcionen.

MLP

Degut als resultats observats a l'estudi de l'estat de l'art, s'ha decidit estudiar una *MLP* com a primera arquitectura. De fet, s'ha pres com a idea base la arquitectura proposada per *Arman Maesumi*[8], proposada pel problema de classificació, capaç de distingir entre posicions guanyades, taules o perdudes. Així doncs, em acabat amb la següent arquitectura:

- Estructura: $[774 \rightarrow 512 \rightarrow 512 \rightarrow 256 \rightarrow 64 \rightarrow 1]$
- Funció d'activació *ReLU* en els *hidden layers*, i *TanH* a la sortida, per aconseguir un valor entre -1 i 1.
- Optimitzador: Adam amb $\eta = 0.001, \beta_1 = 0.90, \beta_2 = 0.999$, parametres per defecte.
- Funció cost: *Mean squared error (MSE)*
- Correctesa del model si $abs(\hat{y} - y) <= 0.9$

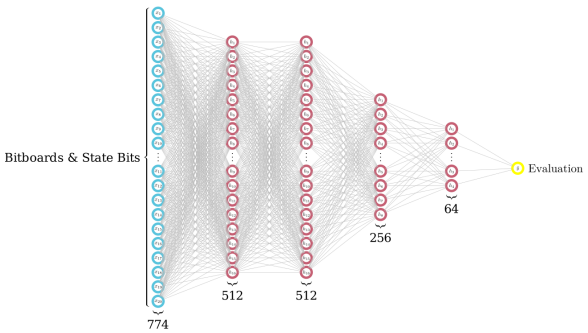


Fig. 8: Xarxa neuronal

En un primera instancia també es van crear amb altres arquitectures amb un número de capes o nodes per capa inferior, però el seu rendiment va ser inferior.

Aquest model va ser pre-entrenat amb els valors de l'avaluació aplicats sobre un llindar, on les posicions amb una avaluació més gran que dos peons d'avantatge passaven a considerar-se victòries i viceversa. Tot i així, després d'entrenar aquesta arquitectura amb més de 10 milions de posicions diferents, amb les dades d'avaluació normals, els resultats obtinguts van ser els següents.

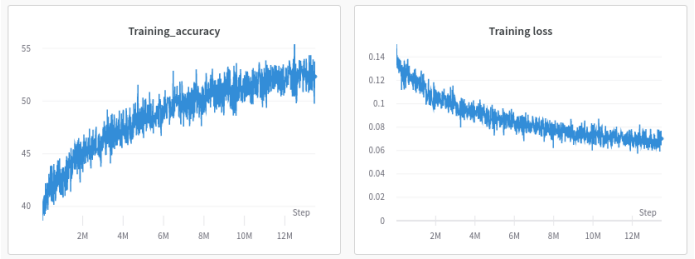


Fig. 9: Gràfiques d' *exactitud* i *pèrdua*

En aquestes gràfiques observem com a cada iteració el model li costa més aprendre, i conseqüentment, tot i anar augmentant el número de posicions noves vistes en la fase d'entrenament, la precisió del mateix augmenta cada cop més lentament. En vista d'aquests resultats és van realitzar entrenaments amb un número d'èpoques elevat amb la intenció que el model aconseguís memoritzar, fer *overfitting*, d'un parell milions de posicions, però no va aconseguir millorar significativament els resultats, indicant que l'arquitectura proposada potser segueix mancant de complexitat.

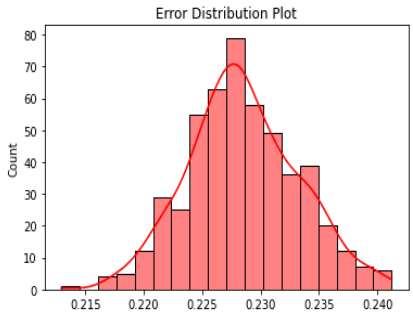


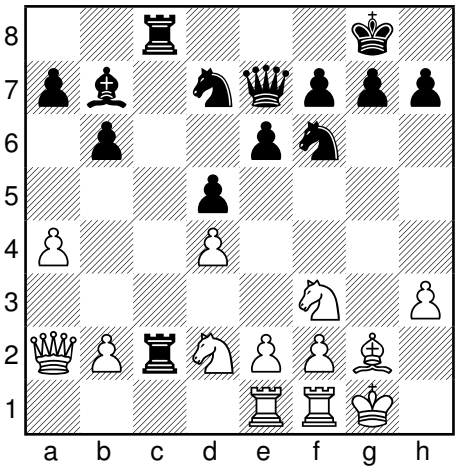
Fig. 10: Gràfica sobre l'error mitjà.

Com s'observa en la figura anterior, el model acostuma a fallar amb un error mitjà de més 0.225 (l'equivalent a més de dos peons de diferència). Aquest resultat és força dolent, ja que indica que ni tan sols és capaç de distingir entre posicions guanyades o perdudes de manera fiable.

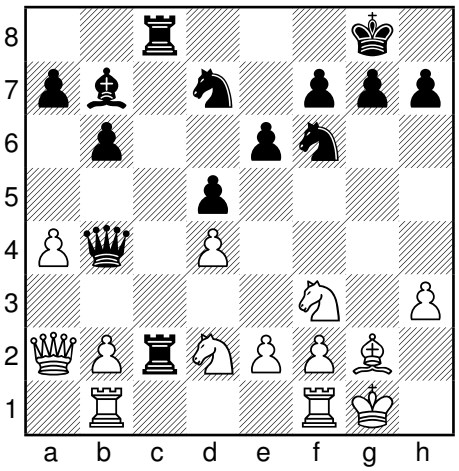
Tot i aquests resultats, s'ha observat de forma experimental, fent una partida contra aquest, que el model ha adquirit un cert coneixement estratègic, això sí, molt limitat. Sembla entendre el concepte de material, evitant deixar-se peces majors, i conceptes com el desenvolupament de les peces. Tot això, utilitzant només un *Look Ahead* de quatre jugades de profunditat.

Exemple:
[--] : Motor ■ : Autor

1 d4 e6 2 c3 d5 3 ♘f3 ♘f6 4 a4 ♗e7 5 ♗g5 O-O 6 ♕c1 ♘bd7 7 ♗h4 c5 8 ♘bd2 b6 9 g4 ♘×g4 10 ♗×e7 ♕×e7 11 h3 ♘gf6 12 ♗g2 ♗b7 13 ♕b1 ♖fc8 14 ♕a2 c×d4 15 c×d4 ♖c2 16 O-O ♖ac8 17 ♖ae1



17... ♕b4 18 ♖b1



Per exemple, observem la penúltima jugada el motor, on decideix portar la torre cap al centre, el que s'entendria com a un moviment actiu, i després de que a la següent jugada vegi que el peó de b2 es troba indefens, torna enrere per defensar-lo. Aquest fet és una indicació que entén la implicació d'un possible avantatge material en la posició. També veiem que portem 18 moviments jugats, i el motor només es troba amb peó de menys (tot i que a les següents jugades el negre guanyi material), i l'estructura de la posició, la distribució de les peces, des del punt de vista humà sembla prou natural.

CNN

La capacitat estratègia es defineix, pràcticament, per la capacitat d'identificar diferents patrons i la seva implicació sobre l'avaluació de la posició. Així, volem que el nostre model sigui capaç de resumir la posició en aquests patrons definitus de l'avaluació. Per tant, per la seva capacitat de resumir les entrades, identificant les característiques més rellevants de les mateixes, s'ha proposat una xarxa convolucional amb la següent arquitectura.

- Estructura
  $[(16, 8, 8) *$
  $(50, 1, 1) *$
  $ReLU(Norm(256, 3, 3)) *$
  $ReLU(Norm(512, 3, 3))$ $*$
  $ReLU(Norm(512, 3, 3)) *$
  $ReLU(Norm(1024, 3, 3))] *$
  $AVGP(1024, 8, 8)$
  $\rightarrow ReLU(1024) \rightarrow ReLU(64) \rightarrow tanh(1)]$
- $Padding = 1$
- Optimitzador: Adam amb $\eta = 0.001, \beta_1 = 0.90, \beta_2 = 0.999$, parametres per defecte.
- Funció cost: *Mean squared error (MSE)*
- Correctesa del model si $abs(\hat{y} - y) <= 0.9$

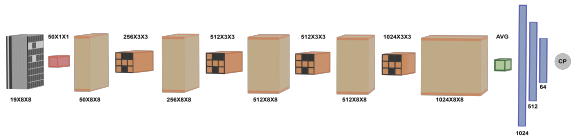


Fig. 11: Arquitectura CNN.

En aquest cas, les entrades del model passen a tindre forma d'imatge. Per aconseguir-la, s'agafa la informació dels dotze *bitboards*, es torna a disposar en forma de taulell, 8×8, i se'l superposa un sobre l'altre. Per serialitzar els bits d'estat en la imatge, aquests s'afegeixen com canals extra com si fossin taulells de tot uns o zeros segons la seva informació.

A causa de la costosa serialització a imatge, i quantitat de memòria que requereix, aquest model només s'ha entrenat amb un milió de posicions.

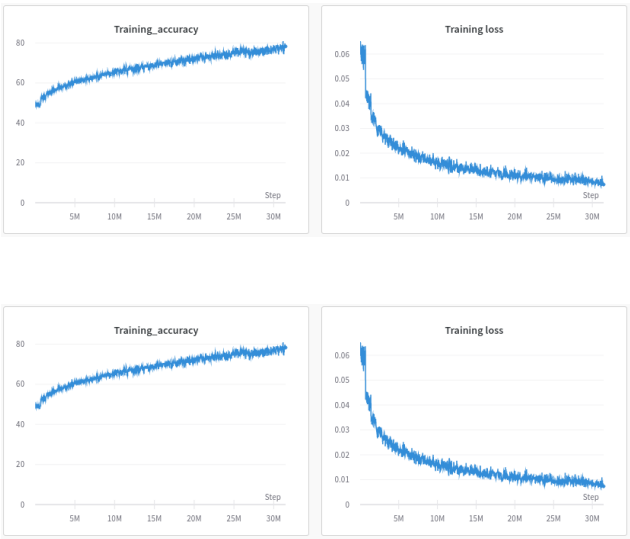


Fig. 12: Gràfiques d' *exactitud* i *pèrdua* en dos entrenaments seguits

D'aquests entrenaments observem que, en aquest cas, el model és realment prou complex, per com a mínim, memoritzar les posicions d'entrada, i no desviar-se cap a la mitjana estadística de la sortida.

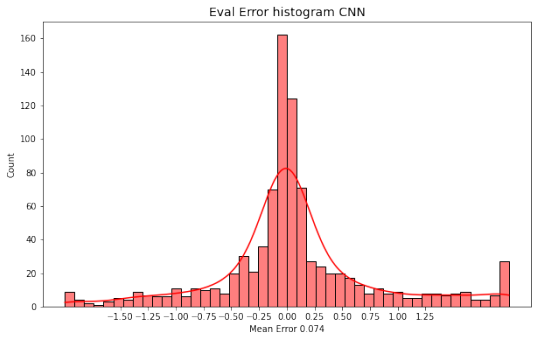


Fig. 13: Gràfica sobre l'error mitjà

Finalment, en la taula anterior, veiem que el model, en les posicions de test, només s'equivoca segons l'avaluació d'*Stockfish*, amb un 0.074, un valor molt bo, equivalent a $74cp$ millorant algun dels resultats aconseguits per l'estat de l'art.

Comparacions amb Look Ahead

En aquesta última secció comparem com de funcionals son els dos models quan s'utilitzen dins del motor com a funció d'avaluació amb una certa profunditat de cerca en l'arbre de moviments, dos *ply*. La informació d'aquest apartat està recopilada a la taula 1 de l'Apèndix.

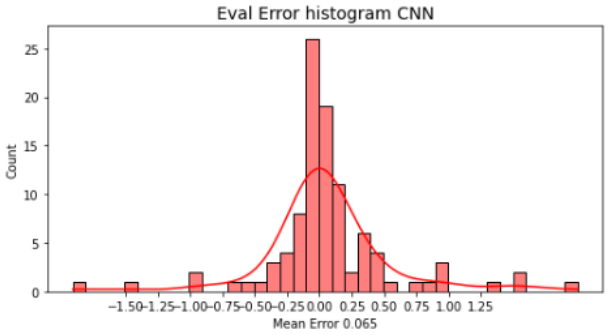


Fig. 14: Gràfica sobre l'error mitjà per CNN i dues jugades de Look Ahead

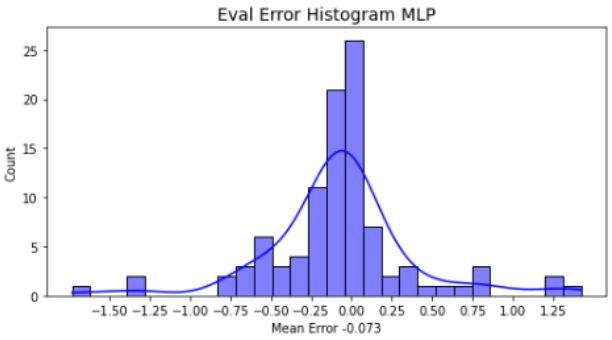


Fig. 15: Gràfica sobre l'error mitjà per MLP i dues jugades de Look Ahead

Els resultats de les gràfiques anteriors ens mostren que els dos models s'afavoreixen de tindre aquesta capacitat de càlcul de dos moviments, tot i això, el model *MLP* millora molt significativament, quasi aconseguit el mateix error mitjà, en valor absolut, que l'arquitectura *CNN*. A més, cal indicar, que l'arquitectura *CNN* és, com a mínim, un ordre de magnitud més lenta que la *MLP*. Aquesta ineficiència no només es deu a la seva complexitat més elevada, sinó també perquè serialitzar els *bitboards* amb les imatges que necessita com a entrada és molt costós.

7 Conclusions

Respecte als objectius del projecte, considerem que els més importants, com construir un motor funcional per comprendre el seu funcionament i provar diferents arquitectures basades en *Deep Learning* per la funció d'avaluació han estat assolits satisfactòriament. La mètrica d'aconseguir predir un 65% dels primers moviments recomanats per *Stockfish* no ha sigut aconseguida, en els resultats de la taula de l'Apèndix s'observa com els dos models encerten menys d'un 30% de les primeres jugades del motor. Aquesta mètrica realment no és tan significativa, ja que en la majoria de posicions acostuma a haver-hi més d'un moviment que diferents motors considerarien com a primera opció. Tot i així s'havia escollit per tindre un segon punt de referència respecte a algunes mètriques de l'estat de l'art. Per altra banda, els models implementats han obtingut certa capacitat estratègica, reflectida en mètriques més fiables, com en els erros mitjans d'avaluació respecte al *ground truth*.

Analitzant els resultats de l'apart anterior, podem concloure que una arquitectura basada en xarxes convolucionals, té molta més capacitat per detectar patrons estratègics en la posició, que un *MLP* , i a més considerem que és una arquitectura amb potencial per aconseguir un nivell estratègic de joc elevat. Aquestes conclusions estan d'acord amb les presentades per *DeepMind* en el seu motor *AlphaZero*, on també utilitzaven xarxes convolucionals per assolir objectius similars. Però, per altra banda, es contraposa als resultats de *Matthia Sabatelli et al*, on indicaven que una *MLP* tindria més potencial per assolir aquesta fita. No només això, sinó que indicaven que la millor codificació dels taulells, perquè els models aprenguessin a identificar patrons, eren els *bitboards*. Amb els nostres resultats no podem denegar aquesta afirmació, de fet creiem que com a molt serà un model basat en estructures semblants, com la nostra entrada per la *CNN*, però tampoc ho podem confirmar. També tot i no haver aconseguit resultats tan bons com a la investigació realitzada per *Arman Maesumi*, creiem que amb una arquitectura més complexa i molt més temps d'entrenament, segurament, arribaríem a obtenir resultats similars, ja que les nostres observacions indiquen que la *MLP* també és capaç de codificar certa capacitat posicional, tot i que de manera limitada.

Finalment, cal indicar que tot i la nostra arquitectura *CNN* tingui una comprensió posicional significativament més elevada que la *MLP*, com a funció d'avaluació pel nostre motor, és molt més potent l'última. Com s'ha indicat a l'apartat de Coneixements Previs[5], la comprensió posicional només és una part del joc, i per causar un impacte diferenciador ha d'estar acompanyada de la capacitat de càlcul suficient, donada en aquest cas per un model més eficient en la inferència.

Futures Línies De Recerca

Existeixen encara molts paradigmes i arquitectures basades en *Deep Learning* per estudiar, que serien interessants implementar, i altres que per qüestions de temps s'han quedat a les portes de ser implementades. Existeix un gran interès per desenvolupar una arquitectura basada en *Transformers*, que pràcticament des d'un principi va ser indicada el tutor del projecte i que creiem que tindria molt de potencial per aconseguir els nostres objectius.
També existeix un cert interès per tornar a programar el motor, utilitzant un llenguatge més eficient com C++, i afegir-li moltes de les millores possibles augmentar la poda de l'arbre de cerca i augmentar la capacitat de càlcul, intentant aconseguir un motor realment fort.

Agraïments

Agrair al tutor del projecte, Dimosthenis Karatzas, per la ser un guia constant en el transcurs del projecte, i per la seva voluntat d'ensenyar-me tot el possible relacionat amb la intel·ligència artificial per tirar endavant el projecte.

Referències

[1] Official-Stockfish .(Sep 26, 2021). Stockfish. Retrived Sep 27, 2021 from GitHub, Github Repository, 21ad356c0900c9eba9b7b1f7453f934eab80f303

[2] Wikipedia contributors. Alpha–beta pruning. Wikipedia. Retrieved Sep 26, 2021, from Wikipedia

[3] Wikipedia contributors. (2021, September 12). Minimax. Wikipedia. Retrieved Sep 26, 2021, from Wikipedia

[4] Scott, G. (2021, October 4). Regression Tests · glinscott/fishtest Wiki. GitHub. Retrieved October 4, 2021, from Github, f8c79daa88ebd87b9831574e55402f0672b9ee5c

[5] Ynasu87, Y. (2018, May 6). GitHub - ynasu87/nnue: Efficiently Updatable Neural-Network-based evaluation functions for computer shogi. GitHub. Retrieved September 28, 2021, from Github, b94987e07a6f5c079d330915b33472bdac179e06

[6] Niklasf, N. (2021, November 3). GitHub - niklasf/python-chess: A chess library for Python, with move generation and validation, PGN parsing and writing, Polyglot opening book reading, Gaviota tablebase probing, Syzygy tablebase probing, and UCI/XBoard engine communication. GitHub. Retrieved November 5, 2021, from Github, 9135f9392bd06dd78bdfd755630c359597cbf77c

[7] Chess Programming Contributors. (2021, June 8). Magic Bitboards - Chessprogramming wiki. Chess Programming Wiki. Retrieved October 11, 2021, from CPW

[8] Maesumi, A. (2020, July 4). Playing Chess with Limited Look Ahead. arXiv.Org. Retrieved July 14, 2021, from h arXiv.Org

[9] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[10] Coulom, R. (2006, May). Efficient selectivity and backup operators in Monte-Carlo tree search. In International conference on computers and games (pp. 72-83). Springer, Berlin, Heidelberg.

[11] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... and Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815.

[12] Sabatelli, M., Bidoia, F., Codreanu, V., and Wiering, M. A. (2018, January). Learning to Evaluate Chess Positions with Deep Neural Networks and Limited Lookahead. In ICPRAM (pp. 276-283).

APÈNDIX

A.1 Secció d'Apèndix

Les avaluacions d'*Stockfish* estan fetes amb un *Look Ahead* de vint jugades aproximadament, mentre que els dos models d'avaluació només poden observar dues jugades de profunditat.

TAULA 1: TAULA DE COMPARACIONS DE 100 POSICIONS

FEN	Stockfish Eval	Stockfish Move	MLP Eval	MLP Move	CNN Eval	CNN Move
rn2r1k1/1bpp1pb1/p5pp/1p1P1n2/7B/1BN2N2/PPP2PPP/R4RK1 w - - 0 14	-0.118	f1e1	-0.037	a1e1	-0.097	h4f6
8/2Q3pk/6rp/5q2/8/4p2P/6P1/3R3K w - - 0 41	-0.287	c7c4	1.0	c7g7	-0.166	c7c4
r4rk1/pp2bppp/2p1pn2/q7/4P3/5N1P/PPP1QPP1/R1B2RK1 w - - 6 14	0.04	a2a4	0.048	c2c3	0.029	c1d2
rnbqkb1r/ppp1ppp1/8/3p3p/3P1Pn1/3BPN2/PPP3PP/RNBQK2R b KQkq - 3 5	-0.07	c7c5	0.023	g7g6	-0.039	h5h4
r3k2r/1p3ppp/p1qpp3/8/1PPP2bb/5N1n/P2N1PK1/R1BQ3R w kq - 1 19	-0.121	h1h3	-0.056	h1h3	-0.19	h1h3
2kr3r/ppp1q1b1/2n1b3/3Pp2p/4N1pP/P2BP1B1/1P2QPP1/2KR3R b - - 0 19	-0.007	e5d5	0.023	e6d5	-0.122	e6d5
r2q1rk1/bQ3ppp/pp1P4/3P4/1Pn2P2/P3B2P/6P1/3R1K1R w - - 1 22	-0.731	e3f2	-0.537	b7a7	-0.508	f1e2
r3k1nr/pp3ppp/3q4/3p1b2/3N4/1QN5/PP3PPP/R3KB1R b KQkq - 0 11	-0.723	d6e5	-0.765	d6e5	-0.638	f5d7
r1bq1rk1/ppp2ppp/3b1n2/8/1nP2P2/2N1P2N/P2P3P/R1BQKB1R w KQ - 1 10	-0.278	a2a3	-0.63	c1b2	-0.358	h3f2
7k/5R2/2p5/p3rp1p/2P1p3/1P2K2P/P7/8 w - - 3 43	0.0	f7d7	0.116	e3d4	0.57	f7f8
2k3r1/1p6/2p1pn2/4Bp2/8/2P4P/PP3PP1/3R2K1 b - - 1 28	-0.579	f6d7	-0.787	f6d5	-0.697	f6d5
r1b2rk1/ppp2ppp/1qn5/b5Q1/3PP2N/2NR3P/PPP3P1/2K2B1R b - - 6 15	-0.148	a5c3	0.541	c6d4	-0.05	f8d8
r5k1/3b3p/n1p1p1pQ/2Bp4/p2n2q1/P2B4/2P3PP/R3R1K1 b - - 3 28	0.753	a6c4	0.811	a6c5	0.338	d4f3
r1bqk2r/pppp2pp/2n2p2/b3p3/2B1N3/2PP4/PP3PPP/RNBQK2R w KQkq - 1 8	0.901	d1h5	1.0	d1h5	0.925	d1h5
r1b3k1/ppp2Bpp/2p5/2b1P3/3r4/2N4P/PPP2qP1/R1BQ3K b - - 0 13	0.905	f2f7	0.88	f2f7	0.743	f2f7
2rr2k1/p4p1p/1p4p1/2p1p3/1PP5/P1KB4/7P/3q4 w - - 0 27	1.0	d3c2	0.224	d3g6	-0.998	d3g6
r2q1rk1/p4pp1/2p1p3/npP3P1/4BB2/2P1PQ2/PP3P2/R3K2R b KQ - 0 16	-1.0	f7f5	-0.975	f7f5	-0.834	d8d7
3n1rk1/4b1p1/1r2qn1p/pPp1p3/P2p4/2NQ1N1P/5PP1/R1B2RK1 w - - 2 25	-0.393	f1e1	-0.446	d3d4	-0.197	d3e2
r3k2r/ppqn1pp1/2nbb3/3pp1Np/5P1P/2P1P3/PP4PB/RN1QKB1R b KQkq - 0 12	0.36	e5f5	0.935	a8c8	0.387	e5f4
1q5k/4bp2/pn2p2p/1p1pP2P/6P1/1P2QN2/P4P2/6K1 b - - 3 31	0.631	h8h7	0.282	h8g7	0.183	b6d7
8/8/2B3p1/pp3k1p/3P2r1/8/PP5K/3R4 b - - 0 39	-0.455	f5e6	-0.417	b5b4	-0.371	g6g5
r2qr1k1/1bp2pp1/1p3n1p/p1p1p3/P3P3/1PNPPN1P/2P3P1/R3QRK1 w - - 0 14	0.097	g2g4	0.61	e1g3	0.014	e1h4
6k1/1pp2p1p/1q4p1/p3p3/8/N1P5/PP3nPP/3B2K1 w - - 0 24	-1.0	g1f1	-0.942	a3c4	-0.958	a3c4
8/1pk5/p7/5R2/3P2n1/PK3R2/1P6/2r5 w - - 5 38	0.692	f5g5	0.964	f3c3	0.737	f5f7
5rk1/R4ppp/2p5/2Pp1q2/3Qb3/4P3/1P3PPP/4KB1R w K - 2 22	0.926	f3e4	1.0	f2f3	0.635	a7e7
r2q1rk1/ppp2ppp/1n1p4/8/2PbQ3/2NB3P/PP3PP1/R4RK1 b - - 1 14	-0.767	d5f2	-0.241	d4c3	0.166	d4c3
B1b4r/p1ppqk1p/1p3Np1/5PP1/2P4P/8/PP1P1P1b/R1BQK3 w Q - 1 17	1.0	e1f1	0.994	f6e4	0.076	e1f1
r4rk1/pp2bppp/3q4/2pbp2P/5PP1/2PBB3/1PPQ4/2KRR3 b - - 0 17	0.064	a8d8	0.156	d5f3	-0.014	d6b6
8/r2r3p/3PkP2/8/8/b3P1P1/P6P/R2R2K1 b - - 0 38	0.03	e6f6	0.247	a7a8	-0.118	a3b2
r3kb1r/pp3ppp/2p2n2/q3p3/2B3b1/2NP2Q1/PPP2PPP/R3K1NR w KQkq - 0 10	-0.019	g2e2	0.023	a2a3	0.019	a2a3
r1bq1rk1/ppp1bppp/2n5/8/2BP4/P1NQpN2/1PP3PP/R4RK1 b - - 0 11	-0.032	c6a5	0.032	e3e2	0.183	c6d4
2Q5/5ppk/p5qp/1p2p3/1P6/P1P1B3/1K3P2/3R4 b - - 0 35	-1.0	g6f6	0.731	g6b1	0.908	g6b1
8/1p6/5k2/p1n4p/P6P/6K1/8/8 b - - 1 48	1.0	c5a4	1.0	c5d7	1.0	c5b3
1r1q1rk1/p1p2ppp/3b1n2/4p3/1P1n4/P1NP2P1/4PPBP/1RBQ1RK1 b - - 0 14	-0.385	h7h5	-0.25	d8c8	-0.041	c7c6
8/p2b4/1p1p4/1Pp1p3/2P1Pp2/P2P1P1k/6p1/R5K1 w - - 0 44	0.791	g2f2	-0.056	a1a2	0.368	a3a4
r4r1k/6b1/pQ1p3p/3BN1q1/5p2/P6b/1B3PPP/R3R1K1 w - - 1 25	0.974	b6d6	1.0	d5a8	0.866	e5f7
2r3k1/pp2qppp/1bp1r3/6p1/3P3P/2P1PB2/PP1Q4/K3R2R w - - 0 22	0.231	h4h5	0.059	h4h5	0.074	d2h2
2r2rk1/5ppp/2n2n2/8/1b1BP3/2N2P2/BP4PP/1K1R3R w - - 3 22	0.504	d4f6	0.93	d4f6	0.459	d4f6
2kr1bnr/ppp1pppp/8/8/3n4/2NB3P/PPPP1PP1/R1B2RK1 b - - 6 10	-0.021	e7e6	0.708	c8b8	0.098	e7e5
6k1/1p3p2/p5pP/3P2R1/2n5/8/KPr1rP2/1R6 w - - 3 35	-0.834	a2b3	-0.57	h6h7	-0.714	a2b3
6k1/p5p1/2p1p3/2PpP2p/P7/5rpr/4K1R1/R3B3 b - - 1 36	0.493	f3b3	0.768	f3b3	0.68	f3b3
4k2r/r1p2ppp/p2p4/1pbBp3/4P3/2PP1n2/PPK3qP/R1BQ3R w k - 2 16	-0.343	c2b1	-0.113	d1e2	-0.658	c1d2
6k1/p4p2/1p4p1/n6p/P1r2P2/5N1P/6P1/R5K1 w - - 1 29	-0.277	b2b3	-0.364	f4f5	-0.172	f3d2
r7/pp5B/3kb1R1/n2pp2P/P1p4b/1PPPP3/1B3K2/RN6 w - - 4 28	0.769	f2g2	0.997	f2e2	0.469	f2f3
r1b2rk1/pp3ppp/2n5/4PN2/1b2pP1q/4B3/PP2B1PP/R2Q2KR b - - 1 16	0.915	c8f5	0.918	c8f5	0.524	c8f5
8/1R6/pbp2k1p/8/PP2B3/3P4/2P1b1PP/5R1K b - - 0 33	-1.0	e2f1	-0.848	e2f3	-0.988	e2f1
R2Q4/5pkp/1p2r1p1/5p2/1P6/6P1/5PKP/4q3 b - - 5 30	0.262	e1e4	0.653	e1b4	0.284	e1e2
8/2np4/pK1k4/1p5P/3P4/PP3N2/8/8 b - - 0 41	-0.737	c7e6	-0.59	c7d5	-0.036	c7d5
rnb1kb1r/pppp1ppp/5n2/4N3/8/1B6/PPPPqPPP/RNB1K2R w KQkq - 0 7	0.0	e1e2	-0.42	e1e2	-0.469	e1e2
4r3/p1p2pkp/1p4q1/5p2/2nP2r1/2P1B1P1/P3RP1Q/5K1R w - - 4 34	0.857	e3h6	-0.408	h2h6	-0.658	e3h6
8/6p1/kp2N3/2bpPr1P/3p4/3P4/4KP2/1R6 b - - 0 42	0.673	f5e5	0.431	f5e5	-0.058	f5e5
r2qk2r/pp4pp/2nnB3/2p1p1b1/8/4PQ2/PP1P1P1P/RNB1K2R b KQkq - 0 13	0.117	c6b4	0.755	a7a6	0.163	a8c8
2kr1b1r/1pp1q1pp/p3p3/3N4/6b1/1P1PBN2/1PP2PPP/R2QR1K1 b - - 0 14	-0.334	d8d5	-0.391	d8d5	-0.29	d8d5
rnbk3r/ppp3pp/3bp3/1N2p3/2P3n1/5N2/PP3PPP/R1B1KB1R w KQ - 6 11	0.135	c1g5	0.576	b5d6	0.155	h2h3
1r5k/2p2p2/2Bp3p/pp2pp2/7r/2PP1N2/PP3b1P/2KR3R b - - 1 25	-0.181	f2e3	0.048	h4h5	0.179	f2e3
8/r1k3p1/4bpB1/1P6/8/1P6/6PP/3R2K1 w - - 0 37	0.691	b5b6	0.885	b5b6	0.392	h2h4
8/4R3/6kp/2p5/P6K/2P4P/1P2p3/5r2 b - - 0 35	0.67	e2e1	-0.048	f1f4	-0.151	f1f4
6k1/1r2qrp1/p1nb1n1p/2p1p1p1/PpPpP2P/1P1P2N1/N3Q3/1RB2RK1 b - - 1 30	0.652	g5h4	0.971	g5h4	0.515	f7f8
rn1qkb1r/ppp1pppp/5n2/8/4N3/3P4/PPP2PPP/RNBbKB1R w KQkq - 0 6	-0.846	e4f6	-0.845	e1d1	-0.802	e4f6
8/p5pp/3p2k1/1p1Q1b2/6q1/2NP1R2/PPP5/R1K5 w - - 3 30	1.0	d5d6	1.0	a2a4	0.995	f3f5
r1bq1rk1/p3nppp/2p1p3/3p4/3P4/2P2N2/PP1N1PPP/R2Q1RK1 b - - 3 11	0.024	f7f6	0.023	a7a6	0.03	c6c5
r3kb1r/ppp1p1pp/2n1pn2/8/3qpP2/6N1/PPPP2PP/RNBQ1R1K b kq - 5 8	0.136	h7h5	0.889	e8c8	0.249	e8c8
1r1q1rk1/1b3p2/4p2p/p5pQ/1bBP4/2nNP1B1/5PPP/RR4K1 b - - 5 27	-0.074	b8c8	0.023	c3b1	-0.044	c3b1
rn3rk1/pq3ppp/1p2pn2/4P3/8/2N2N2/PP2QPPP/R4RK1 b - - 0 15	-0.155	f6d5	-0.098	f6d7	-0.136	f6d5

FEN	Stockfish Eval	Stockfish Move	MLP Eval	MLP Move	CNN Eval	CNN Move
rn1q1rk1/ppp1pp1p/3p1bp1/8/2BPP1b1/2N2N1P/PPP2PP1/R2QK2R b KQ - 0 8	0.148	g4f3	0.115	g4f3	0.028	g4f3
8/5Np1/8/2kb4/5PPP/p7/1p1R4/4K3 w - - 0 39	-0.856	d2c2	0.523	d2b2	0.623	d2d5
rnb2k1r/p3n3/1pp1qp2/6pp/3PN3/1N1B4/P3QPPP/R4RK1 w - - 1 23	0.438	f2f4	0.957	b3c5	0.693	f1e1
r1br2k1/p1p2ppp/2p2n2/1p6/4P3/3Bq2P/PP1N1PP1/R1B1K2R w KQ - 0 14	-0.308	f2e3	-0.199	f2e3	-0.27	f2e3
5r2/p7/bp6/2p5/2P1k1K1/1P1r4/P7/4q3 w - - 0 36	1.0	g4g5	1.0	b3b4	1.0	b3b4
8/5P2/8/8/6k1/3p3p/8/4K3 b - - 0 59	0.0	h6h7	0.023	h3h2	0.915	d3d2
r4r2/p1pb1k2/1p3n2/1N1Pn1pp/PP2p3/4P3/R2Q1PPP/5RK1 w - - 0 22	0.733	b5c7	1.0	b5c7	0.691	d2d4
rnbk1bnr/pppp2pp/4p3/5p2/3P4/5N2/PPP1PPPP/RN1QKB1R b KQ - 1 4	-0.908	f8e7	-0.992	d7d5	-0.854	f8b4
r1bq1rk1/ppp2ppp/2n2n2/3p4/Pb1PpP2/2N1P3/1PPBB1PP/R2QK1NR w KQ - 3 8	-0.156	c3a2	-0.278	c3a2	-0.221	e2f1
r3k2N/pp4pp/2p5/2bpp3/B3P2q/7P/PPPPN3/R1B2Q1K b q - 0 17	-0.972	h4e5	-0.894	h4e4	-0.883	b7b5
r1b4N/1pqkb1N1/pnp1p2p/8/2pP4/P2BP3/1PQ2PPP/R3K2R w KQ - 0 18	1.0	d3f5	1.0	d3g6	0.742	h8g6
8/8/3k4/7K/7p/8/7P/8 b - - 1 50	0.0	h4h3	0.048	h4h3	-0.937	d6d5
1r4k1/p4p1p/1pn1rqp1/3bp3/1Pp5/P1PP1NPP/5PB1/R2QR1K1 w - - 0 19	0.0	d3c4	0.023	d3d4	-0.015	b4b5
r1bq1bnr/p1p1k1pp/np1p1p2/1B1Pp1N1/4P3/2P1B3/PP3PPP/RN1QK2R w KQ - 0 10	0.316	d1e2	0.886	b5c6	0.58	g5e6
7r/ppk4p/8/5ppP/2nP4/2P2rP1/P2K4/4R2R w - - 2 25	-0.839	d2c1	-0.941	d2e2	-0.888	d2c1
r1bq1rk1/1p2npb1/p1n1p1pp/3p4/8/BPN1PP2/P1PPN1PP/R2Q1RK1 b - - 1 12	0.843	d8a5	0.995	f7f5	0.783	h6h5
rnb2rk1/2p2p1p/6p1/p2p2qP/pb2n1P1/5p2/2PP4/RNBQKBNR w - - 0 15	-1.0	d1f3	-0.383	g1h3	-0.537	g1f3
r1bq1r2/1pp2pkp/p2p2p1/4n2n/4P3/2N2P2/PPPQ2P1/2KR1BNR b - - 0 12	-0.083	f8h8	-0.023	d8f6	-0.249	e5d7
8/pQpk1rp1/3b2p1/6B1/8/2P2P2/PPq2P2/R3K3 w Q - 1 30	0.671	b7b5	0.495	a1c1	-0.309	b7e4
r2qkb1r/4pppp/p1p2n2/2p5/3pP3/3P1Q1P/PPP2PP1/R1BNK2R b KQkq - 0 10	-0.152	e7e5	0.023	e7e6	-0.086	d8c7
8/1k6/1P1Kp3/pP2P1p1/6P1/8/8/8 b - - 1 42	1.0	a5a4	-0.266	b7a8	-0.573	b7b6
rn1qkb1r/3bn1pp/pp3p2/1N2p3/P2p3N/1B1P4/1PP2PPP/R1BQ1RK1 w kq - 0 12	1.0	b5d6	1.0	b5d6	9.898	b5d6
3r1r1k/2pq1pp1/p1np2b1/1p1Bp1Q1/4P3/2PP2NP/PP3PP1/R2R2K1 b - - 0 21	-0.231	c6e7	-0.409	c6e7	-0.579	c6e7
r1b2r2/2pp1pk1/pp1b2p1/3P2q1/8/2N1P2Q/PPn3PP/3R1RK1 w - - 2 20	-1.0	d1d3	-0.865	f1f7	-0.648	c3e4
6k1/5pp1/4b2p/8/3B4/2R2P2/q3r1PP/3R2K1 w - - 2 30	1.0	g1h1	-0.421	c3c2	-0.394	g1f1
r2q3r/p1pb2pp/3b2k1/3B1p2/3Q4/8/PPP2PPP/R1B2RK1 b - - 1 16	-0.229	d7c6	-0.793	c7c6	-0.737	c7c6
2r4r/1pqkb2p/p4p1n/Nn1PB3/2p3P1/5N1P/2Q2P2/R3R1K1 b - - 0 24	-0.146	f6e5	-0.181	f6e5	0.012	f6e5
r2r4/1p2Rpkp/p2p1np1/3P4/2P2R2/1PN5/P5PP/6K1 w - - 4 30	0.326	e7b7	0.629	e7b7	0.024	e7b7
8/6p1/Q1bkqn1p/2p5/3p1P2/7P/PP4P1/4R1K1 w - - 0 33	0.8	e1e6	1.0	e1e6	0.731	e1e6
8/8/8/1p6/1Pp5/2K5/4k3/8 b - - 4 48	1.0	e2e3	1.0	e2e3	0.979	e2e3
3r1b1r/p3pk1p/Qp1p1ppn/3P4/4P3/5NBP/PP3PP1/2R2RK1 b - - 2 20	-1.0	f8g7	-0.999	h6f5	-0.964	h8g8
rn1q1bn1/ppp2k2/8/3pp1p1/6br/3BPN2/PPPN1PPP/R2QK2R w KQ - 0 11	0.63	f3e5	0.263	f3e5	0.631	f3e5
r2qr3/1pp2pkp/p2p1np1/3P4/2P3n1/2N5/PP1QBPPP/R4RK1 w - - 3 16	0.021	h2h3	0.579	h2h3	0.071	f2f3
7r/ppp1r3/2nk1p2/6pB/1P1P1bPp/2P1pP2/P3R1P1/R1B3K1 w - - 8 27	0.117	b4b5	-0.711	e2e3	-0.027	g2g3
r1b3k1/ppp3pp/8/4q3/3Bp3/2NP1r1P/PPP5/R4BKR w - - 0 19	0.822	d4d5	0.98	d4e5	0.704	d4e5
3q1k1r/2p1p1b1/4Qn2/1p1p2p1/3P4/4BP2/PPP5/2KR2NR w - - 0 22	1.0	h1h8	1.0	h1h8	0.986	h1h8