

Automatização da Plataforma de Poker do NIAS-IA: Orquestração para o ensino prático de algoritmos

Luana O. Cabral e Rodolpho V. A. Neves*

Resumo—A aplicação de jogos de *poker* no ensino de Inteligência Artificial consolidou-se como uma estratégia metodológica, porém a avaliação dos modelos nesta disciplina enfrentava dificuldades logísticas derivadas de uma infraestrutura pré-existente baseada em MATLAB. O antigo motor de avaliação, denominado `oPoker.mexw64`, operava de forma fechada e síncrona, demonstrando-se suscetível a falhas de execução provocadas por códigos estudantis que apresentavam exceções ou loops infinitos. Visando solucionar esse problema, este trabalho apresenta o desenvolvimento e a validação da plataforma *web* automatizada `poker_nias`. A metodologia consistiu na migração do motor de avaliação DEnTS para a linguagem Python, empregando a biblioteca `treys` para a otimização de cálculos combinatórios por meio de operações *bitwise*. Ao mesmo tempo, foi estruturada uma camada de orquestração utilizando o *framework* Django, aliada ao isolamento das submissões dos alunos em *sandboxes* gerados via contêineres Docker. A eficácia da arquitetura proposta foi comprovada por meio da simulação de um torneio real envolvendo 17 algoritmos (*bots*) antigos, cujo processamento integral ocorreu com êxito em 64,84 segundos. Conclui-se que o sistema desenvolvido garante a automação do fluxo de correção, democratiza o acesso a tecnologias *open-source* e permite redirecionar a carga horária da disciplina para o foco acadêmico para o ensino de Inteligência Artificial.

Index Terms—Autograding, Contêineres Docker, Engenharia de Software, Inteligência Artificial, Orquestração Web.

I. INTRODUÇÃO E JUSTIFICATIVA

A integração da Inteligência Artificial (IA) no ambiente acadêmico tem impulsionado a aplicação de metodologias ativas e desafios práticos para o ensino de algoritmos de inteligência computacional e tomada de decisão. Nesse cenário, o uso de jogos de informação imperfeita, sobretudo o *poker*, consolidou-se como um instrumento pedagógico. Por envolverem tomada de decisão sob incerteza, análise probabilística e dinâmica adversária, tais jogos oferecem um ambiente de testes para a modelagem e a avaliação de agentes autônomos

[1], [2]. A utilização desses cenários competitivos em disciplinas de exatas estimula o engajamento discente e aproxima a teoria matemática da engenharia de *software* aplicada [3].

Apesar dos benefícios pedagógicos, a operacionalização de torneios avaliativos automatizados enfrenta desafios técnicos e de infraestrutura. No contexto antigo da disciplina ELT 451, o sistema de avaliação dos algoritmos (*bots*) submetidos pelos alunos dependia de um motor proprietário desenvolvido em MATLAB, encapsulado no arquivo binário `oPoker.mexw64`. Em primeiro lugar, a exigência de licenças do MATLAB, cujos valores anuais chegam a USD 330 para uso acadêmico e ultrapassam USD 1.000 para uso comercial padrão, limitava a democratização do acesso aos recursos e prejudicava a escalabilidade do ambiente fora dos laboratórios institucionais. Em termos de engenharia de *software*, a infraestrutura sofria com a execução síncrona das partidas, que demandava cerca de 2,7 horas, somada à dependência de intervenção manual do corpo docente a cada rodada de simulação para configurar e executar os torneios. O principal problema arquitetônico residia na ausência de isolamento computacional. A execução nativa e concorrente de múltiplos códigos estudantis no mesmo escopo expunha o ambiente a falhas, de modo que erros em um único algoritmo submetido, a exemplo de *loops* infinitos ou erros de segmentação de memória, eram capazes de corromper e paralisar todo o fluxo da simulação.

Para resolver as vulnerabilidades dessa infraestrutura anterior, este trabalho apresenta o desenvolvimento da plataforma *web* automatizada denominada `poker_nias`. O projeto consiste na transição tecnológica integral da lógica do motor de avaliação DEnTS, que operava originalmente no formato MATLAB, sendo traduzida, refatorada e otimizada para a linguagem Python. A nova plataforma encapsula esse motor de avaliação computacional por meio de uma arquitetura containerizada, fundamentada em três pilares. A orquestração das requisições *web* e do banco de dados é gerida pelo *framework* Django; o processamento matemático das rodadas e a simulação das

*Este Trabalho de Conclusão de Curso foi desenvolvido no Departamento de Engenharia Elétrica e foi apresentado no dia 02 de julho de 2026.

partidas são executados com o auxílio da biblioteca nativa `treys`; e, para garantir a segurança, a submissão estudantil opera dentro de um ecossistema isolado em *sandboxes* providos por contêineres Docker. Essa containerização atua como uma camada de isolamento contra códigos com falhas lógicas ou de sintaxe, viabilizando um *pipeline* de avaliação automática contínuo e que reduz a ocorrência de falhas de execução.

A justificativa para essa reestruturação é a transição de ferramentas proprietárias para linguagens open-source, como a adoção do Python em detrimento do MATLAB. Conforme argumentado por [4], [5], essa adoção no meio científico elimina dependências de licenças comerciais proprietárias, reduzindo a barreira de custo, democratizando o acesso e promovendo a integração com diversas bibliotecas de inteligência artificial. Além disso, estudos sobre a performance comparativa demonstram que o ecossistema Python favorece a estabilidade, a reprodutibilidade de sistemas dinâmicos e a portabilidade multiplataforma [5]. Paralelamente, no escopo das tecnologias de educação, o uso de contêineres em plataformas de avaliação automática é indispensável. Autores como Peveler et al. [6] e Boettiger [7] corroboram que soluções baseadas em Docker resolvem deficiências de alocação de recursos e segurança, garantindo reprodutibilidade e a execução de códigos de terceiros sem comprometer a estabilidade do hospedeiro.

Na prática, a implementação da plataforma `poker_nias` elimina os custos atrelados ao licenciamento de *software* e automatiza a lógica do afunilamento competitivo das mesas. Ao reduzir o tempo de processamento de 2,7 horas para aproximadamente 1 minuto e eliminar a necessidade de intervenção manual do corpo docente nas avaliações, a transição tecnológica resolve as vulnerabilidades de engenharia da disciplina. Consequentemente, o projeto devolve o foco da ELT 451 aos seus objetivos curriculares principais, o ensino da Inteligência Artificial e o contínuo aprimoramento da lógica de programação estruturada.

II. OBJETIVOS

O objetivo geral deste trabalho é desenvolver uma plataforma *web* automatizada e de código aberto (`poker_nias`) para a orquestração e avaliação de torneios de agentes autônomos (D_{En}TS), com o intuito de substituir a infraestrutura antiga por um ecossistema de simulação containerizado e reprodutível. Para alcançar o propósito geral planejado, estipularam-se os seguintes objetivos específicos:

- traduzir e refatorar a lógica matemática do mo-

tor de avaliação original, migrando do ambiente MATLAB para a linguagem Python, com foco na redução do tempo de processamento para cerca de 1 minuto por meio da integração da biblioteca `treys`;

- desenvolver a camada de orquestração *web* utilizando o *framework* Django, projetada para o gerenciamento centralizado de submissões de algoritmos, controle de usuários e estruturação dinâmica das chaves de torneio;
- implementar um ambiente de execução isolado (*sandbox*) baseado em contêineres Docker, garantindo a proteção e integridade do servidor contra códigos defeituosos ou instáveis e viabilizando a rotina de avaliação automatizada;
- validar o tempo de processamento e o funcionamento da nova arquitetura computacional por meio da simulação de um torneio real, submetendo ao orquestrador os algoritmos antigos (*bots*) desenvolvidos por turmas anteriores da disciplina ELT 451.

III. PROCEDIMENTOS METODOLÓGICOS

A. Visão Geral e Fluxo

Esta seção descreve a arquitetura de software, a modelagem de dados e a lógica algorítmica que sustentam a plataforma do projeto `poker_nias`. O foco principal é garantir a execução das simulações computacionais em um ambiente distribuído.

1) Visão Geral e Fluxo da Arquitetura

A plataforma coordena a execução das partidas através de uma arquitetura modular que utiliza o *framework* web Django para o gerenciamento do sistema e a interface com o usuário. A execução das partidas ocorre de forma assíncrona, onde a aplicação web (Django) aciona processos paralelos para ativar o motor de simulação desenvolvido em Python.

Como camada de infraestrutura web, o Django foi selecionado por fornecer proteções nativas contra vulnerabilidades, como *Cross-Site Request Forgery* (CSRF), *Cross-Site Scripting* (XSS) e injeções de SQL, auxiliando na diminuição de riscos de segurança durante o gerenciamento de requisições e persistência de dados [8].

Tendo em vista que o sistema executa códigos (algoritmos dos *bots*) submetidos por alunos, o isolamento computacional é um princípio crítico. Para isso, estruturou-se o ambiente via `Dockerfile` e `docker-compose`. De acordo com Peveler et al. [6] e Bui [9], em sistemas de avaliação automatizada o

Tabela I: Comparativo técnico entre o avaliador antigo e o novo motor de regras

Crítério	Avaliador Antigo (oPoker.mexw64)	Novo Motor (treys em Python)
Licenciamento	Proprietário (MATLAB).	<i>Open-source</i> (Licença MIT).
Portabilidade	Restrito à arquitetura x64.	Multiplataforma (Docker).
Integração	Binário opaco; bloqueia paralelismo.	Nativa no <i>runtime</i> do orquestrador.
Manutenibilidade	Código fechado e inacessível.	Aberto, permitindo rastreabilidade.

Fonte: Autoria própria.

Tabela II: Extrato detalhado da saída do *log* de telemetria da mesa (*log_acoes_mesa_1.txt*)

Torneio	Mesa	Jogador (ID)	Pote Atual	Valor da Ação	Stacks dos Jogadores (J1, J2, J3...)
1	1	1	10.000000	0.000000	1000.000000 1000.000000 1000.000000 1000.000000 ...
1	1	2	10.000000	0.000000	1000.000000 1000.000000 1000.000000 1000.000000 ...
1	1	3	10.000000	-1561.104617	1000.000000 1000.000000 2561.104617 1000.00 ...
1	1	4	10.000000	-1561.104617	1000.000000 1000.000000 2561.104617 1000.00 ...
1	1	5	10.000000	-1483.326839	1000.000000 1000.000000 2561.104617 1000.00 ...

Fonte: Autoria própria.

uso de contêineres oferece maior isolamento quando comparado às máquinas virtuais completas, evitando o comprometimento do sistema operacional *host* por códigos maliciosos ou *loops* infinitos. Ademais, o encapsulamento em Docker, conforme postulado por Bottiger [7], estabelece um paradigma de reprodutibilidade científica, garante-se que os códigos submetidos sejam avaliados no mesmo ambiente de bibliotecas e dependências para todos os competidores. Derval et al. [10] reiteram que a automação na correção exige *graders* isolados, seguros e genéricos para manter a confiabilidade.

2) Modelagem das Tabelas

A persistência de dados e a arquitetura das entidades estruturadas no framework Django foram modeladas para garantir a integridade referencial e a telemetria do torneio, conforme ilustrado no diagrama relacional da Figura 1. O armazenamento de dados é gerenciado por um banco de dados relacional, cujo mapeamento objeto-relacional (ORM) é definido principalmente nos arquivos `models_mesa.py` e `models_codigos.py`. A topologia de dados foi projetada para garantir a integridade referencial entre os códigos submetidos e as instâncias de jogo. As entidades estão mapeadas da seguinte forma:

Submissões de Scripts: Refletem os códigos algorítmicos desenvolvidos e enviados pelos usuários, versionados e armazenados para chamada posterior pelo motor. Ressalta-se que os códigos-fonte fornecidos pelos usuários não são armazenados diretamente no banco de dados. Com o objetivo de otimizar o desempenho do sistema e adotar boas práticas arquiteturais,

os arquivos são preservados em um volume persistente do Docker. Desse modo, o banco de dados armazena unicamente o caminho (path) de referência. Durante o acionamento do motor de avaliação, o sistema consulta essa referência para localizar e realizar a montagem do arquivo no respectivo contêiner de execução.

Torneio: Entidade que representa a instância macro do campeonato, agrupando os metadados de uma edição, como data, status e quantidade de inscritos.

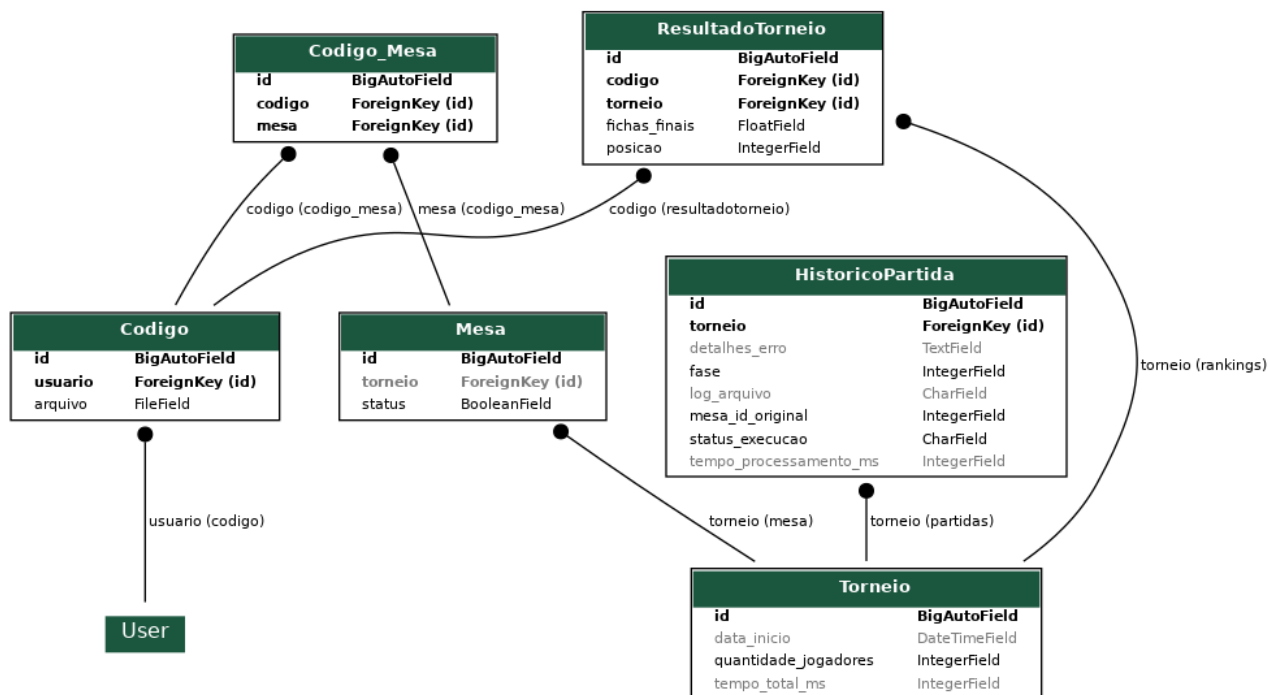
Mesa: Entidade que ramifica o torneio. Representa agrupamentos de partidas dentro de uma rodada ou fase específica.

Codigo_Mesa: Tabela associativa (*many-to-many* resolvida) fundamental para rastrear quais instâncias de scripts (arquivos dos competidores) estão alocadas em quais instâncias de mesas, permitindo o chaveamento e a distribuição dinâmica dos jogadores virtuais.

HistoricoPartida e ResultadoTorneio: Entidades voltadas ao armazenamento da telemetria e das métricas pós-jogo. O `HistoricoPartida` documenta transações, ações dos *bots* e *logs* de execução em nível de partida, enquanto o `ResultadoTorneio` consolida a classificação final de cada participante, fornecendo os dados brutos que alimentam os *dashboards* de liderança.

Para fornecer controle e monitoramento dessas estruturas sem a necessidade de intervenção direta no banco, os arquivos `admin_mesa.py` e `admin_codigo.py` implementam personalizações no

Figura 1: Diagrama entidade-relacionamento do banco de dados do sistema orquestrador. A modelagem ilustra a estrutura de persistência e os vínculos lógicos da plataforma. A entidade central **Torneio** agrupa as métricas gerais do campeonato e ramifica-se em instâncias de **Mesa** e **HistoricoPartida** para o controle das rodadas e telemetria. Os algoritmos submetidos pelos usuários (**User**) são mapeados na tabela **Codigo** e distribuídos dinamicamente nos confrontos através da tabela associativa **Codigo_Mesa**. Ao final das execuções, a entidade **ResultadoTorneio** consolida o desempenho, registrando as fichas finais e a posição de cada bot na respectiva edição da competição.



Fonte: Autoria própria.

painel administrativo nativo do Django. Neles, configuram-se as exibições contextuais (como visualizações *inline* de **Codigo_Mesa** dentro da gestão de **Mesa**) e o acesso direto aos *logs* locais, conferindo ao administrador uma interface gráfica centralizada para o monitoramento das métricas e chaveamento do campeonato.

3) Lógica de Chaveamento e Funilamento

A dinâmica de progressão do **poker_nias** obedece a critérios matemáticos de afunilamento que encontram-se ilustrados no fluxograma da Figura 2. O diagrama descreve o percurso do torneio da esquerda para a direita, iniciando-se no bloco cinza central que representa o número total de inscritos. Deste ponto, ramificam-se setas direcionais que distribuem os competidores para as caixas da “Fase Preliminar” (Mesas 1 a N). A transição entre as etapas do campeonato (Preliminar, Semifinais e Mesa Final) é intermediada por conectores lógicos denominados “Critérios de Avanço”, que atuam como os filtros de eliminação da simulação. O afunilamento visual finaliza no lado direito do diagrama, onde a “Mesa Única” converge em uma última seta para consagrar o “Vencedor”.

Essa inteligência de chaveamento e progressão visualizada na figura é gerenciada nos módulos analíticos

read.py e **analise.py**. O algoritmo obedece a três estágios principais de execução:

Fase Preliminar: Inicialmente, o sistema processa as requisições de entrada, coletando e validando todos os IDs (scripts) disponíveis e habilitados para a competição. Em seguida, executa-se o sorteio inicial, distribuindo o total de competidores de forma uniforme em múltiplas mesas. Esta distribuição balanceada evita sobrecarga no carregamento em memória e garante simetria no chaveamento inicial.

Fases Intermediárias e Semifinais: À medida que as baterias são submetidas, o orquestrador invoca módulos de **multiprocessing** nativos da linguagem Python para paralelizar os confrontos. Durante estas simulações, ocorrem as avaliações de integridade do código. Cenários de falha técnica, como tempo limite excedido (*flags* de *timeout*) ou erros de compilação/sintaxe desclassificam o respectivo *bot*. O módulo de análise processa a telemetria ao final da rodada e extrai os sobreviventes com melhores pontuações, a metade superior da classificação de cada mesa ($X/2$). Estes classificados são reagrupados para formar as novas

mesas da rodada seguinte.

Mesa Final: O laço contínuo mantém sua retroalimentação iterativa, reduzindo o quantitativo de mesas a cada ciclo de reestruturação. O processo se repete até atingir a condição de parada do sistema: a convergência para uma única mesa, configurada para comportar entre 3 e 10 jogadores remanescentes. Neste estágio, o afunilamento ocorre até determinar o vencedor, gerando os *logs* de encerramento do torneio e registrando as pontuações definitivas no banco de dados.

B. Motor de Avaliação

1) Transição Tecnológica e Substituição do *oPoker*

O desenvolvimento do novo motor de simulação exigiu a substituição do avaliador desenvolvido em MATLAB, previamente encapsulado no arquivo binário de código fechado *oPoker.mexw64*, pela biblioteca *treys*, construída de forma nativa e *open-source* na linguagem Python. A necessidade técnica para essa migração fundamenta-se em princípios estruturais de engenharia de software e arquitetura de sistemas. Em primeiro lugar, a transição elimina a dependência de licenças de software proprietário, promovendo um ecossistema de código aberto e facilitando a implantação em servidores remotos. As principais diferenças arquiteturais e operacionais que motivaram essa reestruturação do motor estão sumarizadas na tabela I.

Conforme evidenciado pelos dados da Tabela I, a adoção da biblioteca em Python diminui a restrição de portabilidade imposta pelo formato *.mexw64*, que atrelava o processamento a arquiteturas específicas de hardware e ao sistema operacional Windows. A adoção da biblioteca em Python elimina essa dependência estrutural, viabilizando a execução nativa e a redução do consumo de tempo da lógica de avaliação no interior dos contêineres Docker do orquestrador.

Essa padronização de ambiente elimina o custo de sobrecarga gerado pela comunicação interprocessos entre plataformas distintas. Como resultado, nota-se uma redução de 2,7 horas para 64 segundos no tempo total de execução da bateria de partidas.

2) Integridade Matemática da Simulação (D_{En}TS)

A integridade matemática da simulação D_{En}TS é baseada nos algoritmos da biblioteca *treys*. Para lidar com complexidade combinatória do poker, a arquitetura associa cada carta a um número primo distinto, substituindo comparações lógicas por tabelas de busca e operações de manipulação de bits (*bitwise*). Isso reduz a complexidade computacional, garantindo a conformidade com as regras no poker na avaliação matemática no ranqueamento das mãos

e permitindo simulações com múltiplos agentes simultâneos. Com isso, o desempenho medido deriva do mérito estratégico dos agentes, eliminando vieses causados por inconsistências na simulação.

Em testes práticos, o orquestrador demonstrou eficácia na mitigação de gargalos. Processou simulações de 3 rodadas de torneio (com 17 agentes) em um tempo médio de 62 segundos, com um desvio padrão de 3,5 segundos. Esse resultado supera o avaliador MATLAB utilizado até então (que leva mais de 10.000 segundos para o mesmo volume), comprovando a capacidade da arquitetura de sustentar cenários competitivos.

A avaliação *bitwise* determina o vencedor no *show-down*, mas define apenas o resultado isolado de uma mão. Para avaliar o comportamento estratégico de forma global, a métrica adotada é o saldo financeiro acumulado (*stack* final). Essa métrica é adotada porque penaliza a volatilidade e premia a consistência, refletindo a capacidade de sobrevivência e adaptação dos agentes em múltiplos cenários.

A plataforma conta com um módulo de telemetria que gera logs estruturados. Cada linha do arquivo corresponde a uma transação de jogo, contendo o identificador do torneio, o índice da mesa, o ID do agente atuante, o valor do pote e o saldo atualizado (*stack*) de cada competidor.

A capacidade de reconstruir e replicar o estado exato do jogo a partir desses registros mostra o determinismo do orquestrador. A Tabela II ilustra um extrato dessa formatação.

IV. RESULTADOS

A. Validação do Orquestrador e Dinâmica do Torneio

Para validar o tempo de execução da arquitetura computacional proposta e o isolamento do motor de avaliação nativo em Python, estruturou-se um ambiente de simulação submetido a um conjunto de dados reais de operação. A amostragem deste ensaio empírico foi constituída por 17 algoritmos (*bots*) independentes, desenvolvidos por estudantes em semestres pgressos no escopo da disciplina ELT 451. Cabe ressaltar que estes códigos antigos, originalmente concebidos no ambiente proprietário MATLAB, foram previamente transpostos para a linguagem Python com o suporte de Inteligência Artificial Generativa (modelo Gemini 1.5 Pro). Este procedimento metodológico não apenas garantiu a compatibilidade sintática estrita dos *scripts* com o novo orquestrador, mas também alinhou-se às práticas de modernização de *software* e tradução de

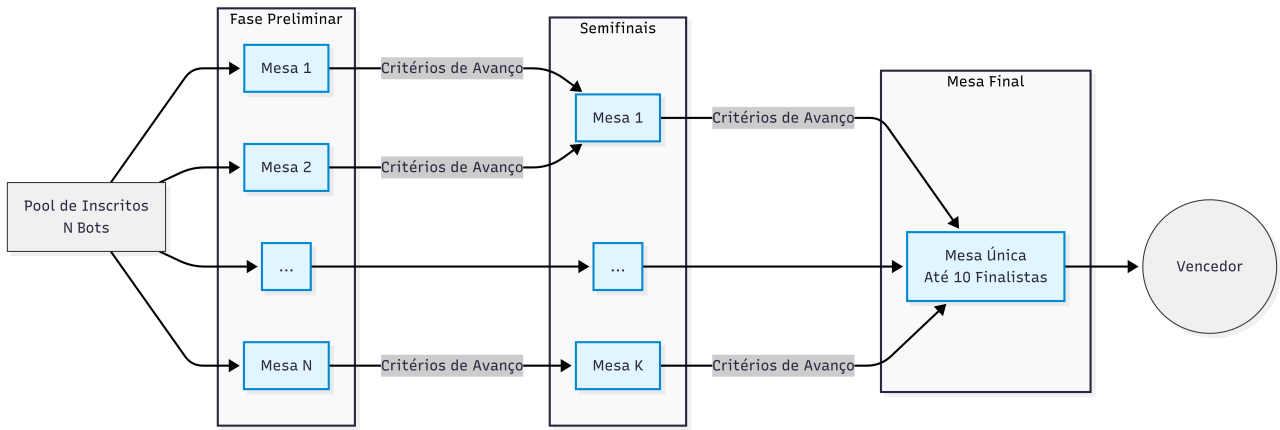


Figura 2: Fluxograma geral do processo de afunilamento e chaveamento das mesas do torneio.

Fonte: Autoria própria.

Tabela III: Desempenho geral e classificação dos algoritmos no torneio de validação.

Posição	Usuário (Dono)	Nome do Algoritmo	Estágio de Eliminação	Pontuação (Stack)
1º Lugar	daniel.silva	bot_daniel.py	Vencedor (Mesa Final)	1654
2º Lugar	torugo.peluzio	bot_torugo_peluzio.py	Mesa Final	1489
3º Lugar	amanda.patricia	bot_Amanda_Patricia.py	Mesa Final	1252
4º Lugar	werikson.frederiko	bot_Werikson_Frederiko.py	Mesa Final	1230
5º Lugar	daniel.costa	bot_daniel_2.py	Mesa Final	1192
6º Lugar	luana.ribeiro	bot_luana_rna.py	Mesa Final	1182
7º Lugar	pedro.alves	bot_pedro.py	Mesa Final	0
8º Lugar	pedro.carvalho	bot_pedro_rna.py	Mesa Final	0
–	rodrigo.gomes	bot_rodrigo_rna.py	Eliminado (Fase Preliminar - Mesa 1)	0
–	jose.pereira	bot_jose.py	Eliminado (Fase Preliminar - Mesa 1)	0
–	jose.lima	bot_jose_rna.py	Eliminado (Fase Preliminar - Mesa 1)	0
–	luana.almeida	bot_luana.py	Eliminado (Fase Preliminar - Mesa 1)	0
–	daniel.santos	bot_daniel_rna.py	Eliminado (Fase Preliminar - Mesa 1)	0
–	diego.martins	bot_diego_rna.py	Eliminado (Fase Preliminar - Mesa 2)	285
–	diego.souza	bot_diego.py	Eliminado (Fase Preliminar - Mesa 2)	0
–	daniel.oliveira	bot_daniel_2_rna.py	Eliminado (Fase Preliminar - Mesa 2)	0
–	rodrigo.ferreira	bot_rodrigo.py	Eliminado (Fase Preliminar - Mesa 2)	0

Fonte: Autoria própria.

código auxiliada por IA, eliminando a dependência de licenciamento comercial.

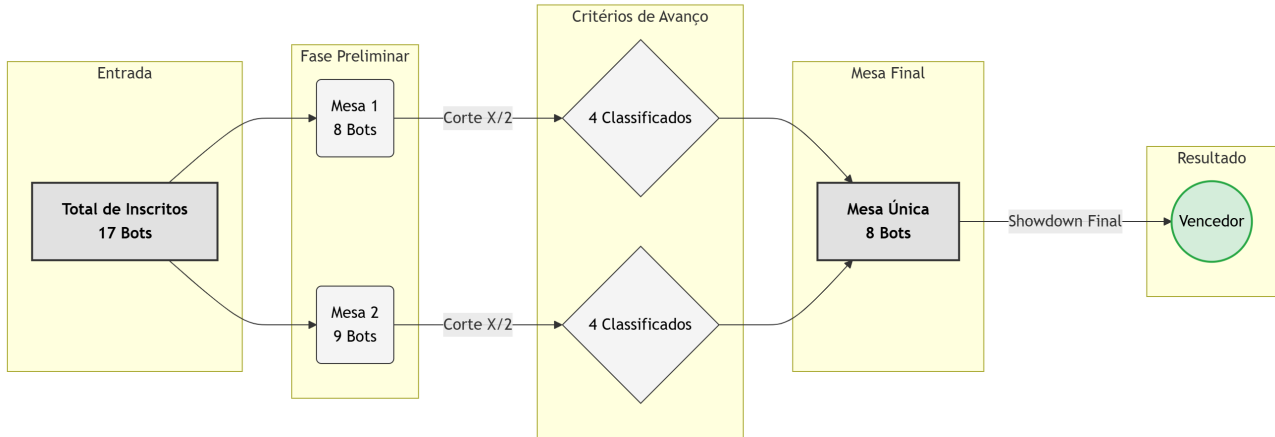
Uma vez integrados à plataforma, o sistema procedeu com a orquestração autônoma do torneio. O particionamento inicial da Fase Preliminar operou conforme os parâmetros sistêmicos previstos, dividindo o conjunto total de 17 competidores e distribuindo-os em duas mesas de simulação simultâneas. A conclusão das partidas sem falhas de execução concorrente demonstrou que o orquestrador consegue paralelizar a carga de trabalho, corroborando a literatura previamente discutida sobre as vantagens do encapsulamento de processos em contêineres Docker para garantir não apenas a reprodutibilidade dos ensaios em Engenharia de Software, mas também a segurança e o isolamento das submissões durante a execução paralela.

Após a conclusão das simulações da etapa inicial, o

módulo analítico aplicou a regra de corte ($X/2$). A Figura 3 ilustra a execução prática dessa dinâmica para o grupo de 17 bots, mapeando o número de agentes alocados, eliminados e classificados em cada rodada até a formação da mesa final. O mecanismo extraiu os quatro bots com as maiores pontuações de cada mesa preliminar, promovendo-os para a próxima fase. Com isso, a condição de parada foi atingida logo na Fase 2, momento em que os oito sobreviventes formaram a Mesa Final. Nela, ocorreu a rodada decisiva do torneio em condições de igualdade.

A capacidade do Python na manipulação de bits em cálculos combinatórios refletiu-se na conclusão da classificação final em 64,84 segundos, validando a arquitetura do orquestrador. A execução de todas as fases do torneio foi concluída em exatos 64,84 segundos. Esse tempo abrangeu toda a rotina do sistema, a leitura e

Figura 3: Fluxograma instanciado com os dados reais de chaveamento para o ensaio de 17 bots.



Fonte: Autoria própria.

validação dos 17 arquivos submetidos, o chaveamento dinâmico das mesas e o processamento de todas as combinações de mãos pelo motor *treys*. Na prática, esse resultado comprova o sucesso da migração do MATLAB para o Python. Ao remover o antigo motor compilado, `oPoker.mexw64`, e centralizar a execução no mesmo *runtime* do orquestrador, eliminou-se o problema da comunicação interprocessos, viabilizando o processamento dos 17 algoritmos testados.

O desempenho consolidado dos jogadores e a respectiva pontuação obtida no *showdown* da bateria de encerramento estão sumarizados na Tabela III.

B. Aplicação Pedagógica e Impacto na Disciplina

A avaliação dos algoritmos desenvolvidos na ELT 451 sempre enfrentou limitações na operação e na logística. Em semestres anteriores, a execução das partidas e a consequente avaliação dos discentes dependiam da execução local dos *scripts* em ambiente MATLAB. Esse modelo demandava um tempo de execução superior a 2,7 horas (10.000 segundos) por torneio, além de exigir intervenção manual do corpo docente a cada nova rodada para gerenciar o andamento das simulações. A ausência de um mecanismo de isolamento de falhas permitia que submissões contendo anomalias lógicas, como *loops* infinitos ou violações de memória, travassem a execução global do torneio. Como consequência direta, a carga horária semanal da disciplina, composta por 4 aulas de 50 minutos, era frequentemente consumida na íntegra por rodadas de teste consecutivas, em um ciclo repetitivo onde os alunos tentavam corrigir os erros e submetiam os códigos novamente até que todas as implementações fossem validadas com sucesso. Dessa forma, o tempo de classe que deveria ser destinado à análise das es-

tratégias algorítmicas e às discussões acadêmicas era desviadas para a triagem manual e a depuração de falhas de *software*.

A introdução do sistema orquestrador DEnTS em Python, projetado como uma plataforma *web* automatizada e independente, soluciona as restrições arquitetônicas do modelo antigo. A integração de um *framework web* (Django) à containerização de processos (Docker) estabelece um ambiente de avaliação automatizada de código isolado e de execução independente. O algoritmo é encapsulado em um contêiner, garantindo que a execução ocorra em uma *sandbox* controlada. Essa abordagem isola falhas individuais a nível de processo, impedindo que instabilidades no código de um aluno interfiram no desempenho dos demais competidores ou no orquestrador principal. Conforme fundamentado por Peveler et al. [6], e corroborado pelas diretrizes de Derval et al. [10] para sistemas de avaliação em engenharia de *software*, a transição para ambientes de correção containerizados otimiza a alocação de recursos e assegura a reprodutibilidade exata das condições de teste.

Ao remover os desafios infraestruturais, garante-se que a disciplina retome o direcionamento para suas metas curriculares primárias, a aplicação prática de Inteligência Artificial e o desenvolvimento da lógica de programação. Para o discente, a plataforma garante um ambiente de submissão padronizado, onde todos os algoritmos são interpretados sob restrições idênticas de processamento e tempo. Para o docente, ocorre a eliminação da intervenção manual, visto que o orquestrador permite a condução autônoma do torneio final em segundos, ao contrário do motor anterior, que exigia horas. Os resultados práticos atestam que o sistema resolve as chaves da competição em aproximadamente

64,84 segundos, consolidando os *logs* da partida e gerando a classificação definitiva de forma automatizada. Essa eficiência valida a arquitetura não apenas como uma solução viável para a disciplina, mas como um facilitador pedagógico que converte a avaliação final da disciplina em uma experiência padronizada e focada na aprendizagem.

V. CONCLUSÃO

Este trabalho teve como objetivo central desenvolver, implementar e validar uma plataforma *web* automatizada e de código aberto, denominada **poker_nias**, projetada para a orquestração e avaliação de torneios de agentes autônomos. A concepção deste sistema fundamentou-se na necessidade técnica de substituir a infraestrutura antiga e proprietária baseada no motor em MATLAB, fundamentada no binário proprietário **oPoker.mexw64**. Dessa forma, almejou-se a transição para um ecossistema conteinerizado e *open-source* que superasse as limitações de arquitetura do modelo preexistente e viabilizasse um ambiente de simulação reproduzível, seguro e alinhado aos padrões atuais de engenharia de *software*.

Os resultados obtidos demonstram a viabilidade da nova arquitetura computacional por meio da redução do tempo de execução e do isolamento de falhas. A tradução e refatoração da lógica matemática do motor de avaliação para a linguagem Python, potencializada pela integração da biblioteca **treys**, garantiu a otimização do processamento das combinações numéricas necessárias. Simultaneamente, a utilização do *framework* Django facilitou a orquestração para o gerenciamento de submissões e o chaveamento dinâmico do torneio. Sobretudo, destaca-se a estabilidade do sistema promovida pela implementação de ambientes de execução isolados (*sandboxes*) mediante a utilização de contêineres Docker. Essa abordagem containerizada contribuiu para diminuir falhas de execução introduzidas por códigos estudantis, como *loops* infinitos e violações de memória, garantindo a proteção do servidor e viabilizando um processo de avaliação automatizada (*autograding*) contínuo e ininterrupto.

Sob o ponto de vista prático e operacional, a eficácia do orquestrador foi validada por meio de uma simulação abrangendo um torneio com 17 algoritmos (*bots*) antigos. O processamento integral desses 17 algoritmos em 64,84 segundos atesta a escalabilidade da arquitetura e a otimização computacional do sistema. Por fim, o impacto pedagógico decorrente desta implementação apresenta melhorias para a disciplina. Ao eliminar a dependência e os custos associados a

licenciamentos de *software* proprietário, além de eliminar a necessidade de intervenção manual docente a cada execução, a plataforma democratiza o acesso à tecnologia computacional empregada. Consequentemente, a infraestrutura automatizada devolve o foco central das aulas para o ensino de Inteligência Artificial e lógica de programação, consolidando-se como uma ferramenta que padroniza o ambiente de teste e reduz o tempo do processo avaliativo.

REFERÊNCIAS

- [1] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron, "The challenge of poker," *Artificial Intelligence*, vol. 134, pp. 201–240, 01 2002.
- [2] M. Schmid, M. Moravčík, N. Burch, R. Kadlec, J. Davidson, K. Waugh, N. Bard, W. Finbarr, M. Lanctot, G. Z. Holland, E. Davoodi, A. Christianson, and M. Bowling, "Student of games: A unified learning algorithm for both perfect and imperfect information games," *Science Advances*, vol. 9, no. 2, 2023.
- [3] M. R. Scheessele and T. Schrieffer, "Poker as a group project for artificial intelligence," *SIGCSE '06*, pp. 548–552, 2006.
- [4] P. Jurica and C. van Leeuwen, "Ompc: an open-source matlab-to-python compiler," *Frontiers in Neuroinformatics*, vol. 3, p. 5, 2009.
- [5] P. F. S. Guedes and E. G. Nepomuceno, "Some remarks on the performance of matlab, python and octave in simulating dynamical systems," in *14 Simpósio Brasileiro de Automação Inteligente - SBAI*, pp. 554–560, 2019.
- [6] M. Peveler, E. Maicus, and B. Cutler, "Comparing jailed sandboxes vs containers within an autograding system," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pp. 139–145, ACM, 2019.
- [7] C. Boettiger, "An introduction to docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015.
- [8] K. Duisebekova, R. Khabirov, and A. Zholzhan, "Django as secure web-framework in practice," *The Bulletin of Kazakh Academy of Transport and Communications named after M. Tynyshpayev*, vol. 116, no. 1, pp. 275–281, 2021.
- [9] T. Bui, "Analysis of docker security," 2015.
- [10] G. Derval, A. Gego, P. Reinbold, B. Frantzen, and P. Van Roy, "Automatic grading of programming exercises in a mooc using the ingenious platform," in *Proceedings of the European MOOC Stakeholder Summit 2015*, (Mons, Belgium), pp. 86–91, 2015.