Algoritmo Genético Guiado por LLMs para Otimização de Modelos de Previsão de Séries Temporais

Projeto Orientado em Computação II

Alexis Duarte Guimarães Mariz
Depto. de Ciência da Computação
UFMG
Belo Horizonte, MG
alexismariz@dcc.ufmg.br

Orientado por Frederico Gadelha Guimarães
Depto. de Ciência da Computação
UFMG
Belo Horizonte, MG
fredericoguimaraes@ufmg.br

***Abstract*—A seleção e otimização de modelos para previsão de séries temporais é uma tarefa complexa que tradicionalmente exige conhecimento especializado e esforço manual. Este trabalho propõe e avalia o uso de algoritmo genético evolucionário em conjunto com Large Language Models(LLMs). Para isso, três abordagens foram desenvolvidas e comparadas: (i) um algoritmo genético clássico, que utiliza operadores estocásticos de seleção, crossover e mutação; (ii) um algoritmo genético com evolução guiada por LLM com uma estratégia definida no prompt, onde o LLM é responsável por gerar a população cada geração; e (iii) algoritmo genético com evolução guiada por LLM com estratégia livre. Também foram usados LLMs para gerar e corrigir o código dos modelos de maneira eficaz. Os experimentos indicam a capacidade dos LLMs de atuar como agente de evolução, obtendo resultados próximos ao estado da arte para o dataset analisado.**

I. Introdução

A modelagem de séries temporais é um dos problemas centrais de diversas áreas, como economia, meteorologia, transportes, tráfego, entre outras [1], [2]. Tradicionalmente, são usadas técnicas de estatística como ARIMA, baseado em médias móveis autoregressivas, TBATS, com modelagem de tendência, Holt-Winters, com suavização exponencial, modelos vetoriais, entre outros [3]–[6]. Técnicas mais modernas de aprendizado de máquina emergiram nas últimas décadas com o avanço do poder computacional e das grandes bases de dados, como XGBoost, LightGBM, Random Forest, LSTM, TCN, GRU, além de outras abordagens recentes [7]–[10].

Escolher qual modelo utilizar dentre tantos disponíveis envolve alguns desafios, como o trade-off viés-variância, underfitting e overfitting, a disponibilidade de dados e poder computacional, além da complexa tarefa de seleção de features, hiperparâmetros e a definição de esquemas de validação cruzada temporal. Esse processo é de extrema importância, uma vez que afeta diretamente a acurácia das previsões que impactam na tomada de decisão. Com isso, pequenos aumentos na acurácia de um modelo podem trazer ganhos a diversos domínios [11]–[13].

Esse trabalho propõe um algoritmo genético evolucionário em conjunto com Large Language Models (LLMs) para gerar modelos de previsão de séries temporais. O problema central é determinar se a abordagem evolucionária é eficaz para obter bons modelos por meio de código gerado por LLMs. Além disso, também avalia a eficácia dos LLMs como guia para a evolução em comparação com a evolução puramente estocástica.

II. Referencial Teórico

A. Previsão de Séries Temporais

A previsão de séries temporais envolve modelar sequências de dados ordenados no tempo para estimar valores futuros. Métodos clássicos incluem modelos estatísticos como ARIMA e Exponential Smoothing. Com o avanço do aprendizado de máquina, surgiram abordagens baseadas em árvores de decisão, redes neurais recorrentes e arquiteturas de atenção temporal. Essas técnicas requerem seleção de features, ajuste de hiperparâmetros e definição de esquemas de validação cruzada temporal (time-series cross-validation). [3], [11]

Paliari et al. [14] demonstraram que modelos de aprendizado de máquina, especificamente LSTM e XGBoost, tendem a superar o método estatístico tradicional ARIMA na previsão de séries temporais financeiras de curto prazo. Os resultados indicaram a superioridade geral do LSTM na redução de métricas de erro, com o ARIMA mostrando-se mais eficaz apenas em cenários específicos de valores nominais muito baixos.

B. Validação Cruzada em Séries Temporais

Em séries temporais, os dados não são independentes: o valor atual depende de passos anteriores. Por isso, o conjunto de treino deve sempre preceder o de validação/teste no tempo, evitando vazamento de informação futura para o modelo. Assim, não é possível utilizar a validação cruzada tradicional(k-fold), pois não é permitido embaralhar os dados e a validação cruzada tradicional alteraria a ordem e a dependência temporal

dos dados. A solução adequada é a validação cruzada temporal [15], utilizando estratégias de janela deslizante(rolling window) e janela expansiva(expanding window), que garante que o modelo sempre seja avaliado em dados futuros em relação ao período de treinamento.

C. Geração de Código via LLMs

Large Language Models(LLMs), como GPT-4 [16] e GPT-5, se mostraram efetivos em tarefas de processamento de linguagem natural, planejamento e geração de código. Os modelos de linguagem são capazes de gerar trechos de código-fonte em linguagens como Python, importar bibliotecas, definir funções e entender o contexto. Além disso, os modelos podem ser utilizados para limpeza dos dados, seleção de features e divisão treino/teste. Instruções detalhadas podem guiar o modelo e melhorar significativamente os resultados. [17]–[20]

D. Algoritmos Evolucionários

Algoritmos genéticos são usados para otimização. Uma população de soluções candidatas(indivíduos) participa de um processo de evolução baseado na teoria da evolução de Darwin. A população passa por um processo de seleção, recombinação e mutação e cada indivíduo é avaliado por uma função de avaliação(fitness) que quantifica o quão bem um indivíduo resolve um determinado problema. A cada geração, os indivíduos mais aptos(com melhor fitness) têm maior probabilidade de transmitir os seus genes(partes da solução) para a próxima geração. [21]

A combinação entre algoritmos evolucionários e LLMs pode ser extremamente poderosa. Por um lado, os algoritmos evolutivos podem aprimorar os resultados gerados pelo LLM através da engenharia de prompt. Por outro lado, o LLM pode utilizar seu grande conhecimento dos mais variados domínios para conduzir buscas mais inteligentes nos algoritmos genéticos, atuando como operadores evolutivos [22].

E. Algoritmos Evolucionários e LLMs

A combinação de algoritmos evolucionários e LLMs se mostrou eficaz para geração de funções de recompensa para aprendizado por reforço. Em [23] foi proposto o algoritmo EUREKA para design de recompensas que utiliza parte do código-fonte e a descrição da tarefa como contexto para um LLM(GPT-4). Cada função de recompensa é utilizada para o treinamento de um modelo de aprendizado por reforço e as métricas do treinamento são utilizadas para guiar um processo de busca evolucionária. Foram obtidos resultados de nível humano no design de funções de recompensa.

III. Metodologia

Este trabalho propõe e avalia uso de algoritmos evolucionários em conjunto LLMs para otimizar a geração de modelos de previsão de séries temporais, usando código gerado por LLMs. Foi definido o esquema genético, onde os genes de cada indivíduo são: o modelo, os hiperparâmetros e as features. Após a definição de uma população de indivíduos de acordo com o algoritmo genético, um prompt é construído através de um template fixo para o LLM gerar uma função `train_and_predict` em Python que implementa exatamente o modelo, as features e os hiperparâmetros definidos pelos genes de cada indivíduo e retorna o valor da função de fitness definida. A avaliação de cada modelo de previsão, definida pela medida de fitness do indivíduo, é utilizada para definir as próximas gerações.

A arquitetura do sistema está ilustrada na Fig. 1, composta por um Gerador de População(GPT-5.1), que define os indivíduos de cada população, LLMs Geradores de Código(GPT-5.1-Codex) e LLMs Corretores de Código(GPT-5.1-Codex). O prompt utilizado para geração de código está exibido na Fig. 2 e caso o código gerado pelo LLM Gerador de Código apresente algum erro, o LLM Corretor de Código gera novo código, usando o prompt exibido na Fig. 3, recebendo o código que apresentou erro e a mensagem de erro gerada.

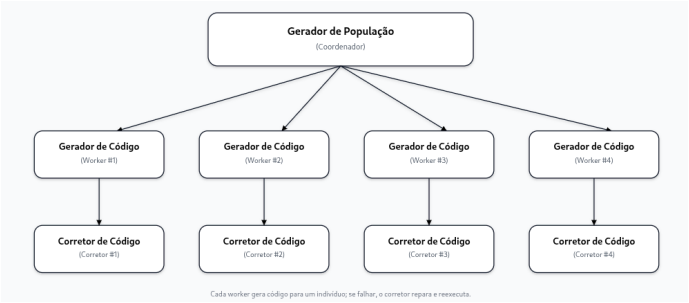

Fig. 1. Arquitetura do sistema

```
CODE_PROMPT = """You are an expert data scientist.
Given the training and testing data in pandas DataFrame, write Python code that:
 1. Receives a dictionary named `data_dict` with keys 'train_X', 'train_y', and
 'test_X'.
 2. Builds and fits the specified model on (train_X, train_y).
 3. Returns a dict with key 'preds' containing the NumPy array of predictions
 for test_X.

You receive two lists:
  - orig_features  = {orig_features}
  - new_features   = {new_features}
- All features in `orig_features + new_features` ALREADY EXIST in `train_X` and
`test_X`.
- You DO NOT need to create lags, rollings or time features. Just use the
columns.

 Additional requirements:
 - Target column is '{target}'
 - Use only the following feature columns: {orig_features} + {new_features}
 - Access the target series directly:
    train_y = data_dict['train_y']
  (do not index again por `[{target}]`).
 - Standardize the feature columns using StandardScaler fitted on train_X.
 - Model type: {model}
 - Hyperparameters (use exactly these): {hyperparams}
 - Use random_state={seed} where applicable.
 - Do not allow shape mismatches or NaN values in the predictions.
 - The code must return only a NumPy array of type float, not a list or pandas
 Series. For example: preds = np.array(preds, dtype=float)
 - Ensure there are no NaN or None values in `preds` before returning.
 - All indices are datetime-based and aligned; assume continuity.
 - Do not use `.reset_index()` or mix index types.
 - Do not print anything.
 - CUDA is available

 Provide only the Python code of a function called
 `train_and_predict(data_dict)` (no explanations, just code).**Do not include**
 any markdown fences, comments or extra text — respond ONLY with the raw
 function code.
 """
```

Fig. 2. Prompt para geração de código

```
ERROR_CORRECTION_PROMPT = """You are an expert Python developer.
The following code failed with an error.
Your task is to fix the code to make it work correctly.

## ORIGINAL CODE:
```python
{original_code}
```

ERROR ENCOUNTERED:
{error_message}

REQUIREMENTS:
- The function must be named `train_and_predict(data_dict)`
- It must return a dict with key 'preds' containing predictions as a NumPy array
- Use the same model type: {model}
- Use the same hyperparameters: {hyperparams}
- Access data as: train_X = data_dict['train_X'],
train_y = data_dict['train_y'], test_X = data_dict['test_X']
- Ensure predictions are float64 NumPy array with no NaN values
- Handle any potential data shape or type issues
- Add proper error handling and data validation

COMMON FIXES:
- Make sure the library is installed by adding "!pip install" before the code
- Check data shapes and types before fitting
- Handle missing values or infinite values
- Ensure model parameters are valid for the data size
- Add try-catch blocks for model fitting/prediction
- Validate prediction output format

Provide ONLY the corrected Python function code, no explanations or markdown."""
```

Fig. 3. Prompt para geração de código

Para validar a proposta, foram desenvolvidas e exploradas três abordagens distintas: (i) algoritmo genético clássico, onde o processo evolucionário é guiado pela aleatoriedade; (ii) algoritmo de evolução guiada por LLM, com estratégia de evolução definida no prompt, onde o LLM define a população de cada geração; e (iii) algoritmo de evolução guiada por LLM, com estratégia de evolução livre para o LLM definir a população de cada geração.

Em todas as abordagens, cada indivíduo é testado de acordo com um esquema de validação cruzada para séries temporais de Expanding Window(`TimeSeriesSplit`) e o fitness final do indivíduo é a média da métrica de todos folds.

A. Abordagem (i): Algoritmo Genético Clássico

No algoritmo genético clássico, a população é definida de acordo com a aleatoriedade. A população inicial é gerada através de sorteios de modelo e features, enquanto os hiperparâmetros de cada indivíduo são sorteados dentro de um espaço de busca pré-definido para cada hiperparâmetro de cada modelo. Os indivíduos são selecionados para a próxima geração através de torneio, onde dois indivíduos são escolhidos aleatoriamente e o de melhor fitness é selecionado. Selecionados os indivíduos, é realizada a etapa de crossover de features, onde é definido um ponto de corte aleatório, como ilustrado no exemplo a seguir:

```
ind1["features"] = [1,1,1,1]
ind2["features"] = [0,0,0,0]
ponto de corte: 2
resultado:
ind3["features"] = [1,1]+[0,0]=[1,1,0,0]
ind4["features"] = [0,0]+[1,1]=[0,0,1,1]
```

Além da seleção e crossover, também podem ocorrer mutações de duas formas: (i) ativação ou desativação de uma feature aleatória ou (ii) escolha de um novo valor aleatório para um hiperparâmetro, dentro do intervalo permitido.

B. Abordagem (ii): LLM como guia para evolução, com estratégia definida no prompt

No algoritmo genético guiado por LLM, a população não é definida de forma estocástica, mas pelo LLM. Ao invés de realizar seleção e definir probabilidades de crossover e mutação via código, a cada geração, o prompt da Fig. 4 é criado a partir de um template para que o LLM gere toda a população de indivíduos, atuando como um guia para a evolução. Esse prompt contém a lista de modelos e features disponíveis, o indivíduo de melhor fitness obtido até o momento e os melhores indivíduos da geração anterior com seus respectivos genes. Como ilustrado na Fig. 5, na primeira geração é pedido ao LLM para explorar e diversificar o espaço de busca, enquanto nas próximas gerações é pedido para utilizar uma estratégia que combine elitismo, crossover e exploração, sem restringir valores ou probabilidades para cada um, dando mais autonomia e liberdade para o LLM guiar o processo evolutivo. Assim como nas outras abordagens, assim que a população é definida, um outro prompt é construído e enviado para um outro LLM gerar a função `train_and_predict` que retorna os valores preditos para serem usados na validação cruzada temporal.

C. Abordagem (iii): LLM como guia para evolução, com estratégia definida pelo LLM de forma livre

Nessa abordagem, o LLM atua como guia para evolução, definindo os indivíduos de cada população de forma livre. A Fig. 6 ilustra como o LLM é instruído a escolher de forma livre elitismo, crossover e exploração. Assim como na Abordagem (ii), o LLM também recebe a lista de modelos e features disponíveis, o indivíduo de melhor fitness até o momento e os melhores indivíduos da geração anterior com seus respectivos genes.

IV. Experimentos

A. Dataset e pré-processamento

Para validação e implementação, foi utilizado o dataset Appliances Energy Prediction [24], com 19735 linhas de leituras de consumo de energia a cada 10 minutos com condições meteorológicas externas de uma estação próxima. Para preparar os dados para a modelagem, foi realizado um pré-processamento com engenharia de features para fornecer mais informação aos modelos. Foram criadas novas features de forma automatizada:

- Features de Lag: valores defasados da variável alvo criados para diversos intervalos(1, 3, 7, 24, 168 períodos), visando capturar a autocorrelação da série.
- Features de Janela Móvel: estatísticas como média, desvio padrão e valor máximo foram calculadas sobre janelas deslizantes de diferentes tamanhos(3, 7, 24 períodos).
- Features Temporais: foram extraídas a hora do dia e o dia da semana a partir da data. O mês foi decomposto em

```
POP_PROMPT = textwrap.dedent("""
You are an expert ML engineer specializing in genetic algorithms for
time series forecasting.

## CONTEXT
- Dataset: Time series forecasting with target '{target}'
- Population size: {size} individuals for GENERATION {gen}
- Current baseline RMSE: {baseline_rmse} (goal: minimize)
- Available models: {models}
- Original features: {orig_features}
- Engineered features: {new_features}

## TASK
Generate EXACTLY {size} diverse individuals as a JSON array.
Each individual has:
- "model": string from available models
- "orig_features": subset of original features
- "new_features": subset of engineered features
- "hyper": dictionary of hyperparameters specific to the model

## STRATEGY FOR GENERATION {gen}
{strategy_section}


## MODEL-SPECIFIC HYPERPARAMETER GUIDELINES
- **Tree-based (XGBoost, RandomForest, LightGBM)**:
Focus on n_estimators (50-500), max_depth (3-10), learning_rate (0.01-0.3)
- **Neural Networks (MLPRegressor, LSTM, GRU)**:
Hidden layer sizes, activation functions, learning rates
- **Time Series (ARIMA, SARIMAX, Prophet)**: Seasonal parameters,
trend components, order parameters
- **Regression (SVR, ElasticNet, BayesianRidge)**: Regularization parameters,
kernel choices, alpha values

## CONSTRAINTS
- Use ONLY existing features - DO NOT invent new ones
- NEVER include '{target}' in features (data leakage)
- Ensure all individuals are unique(different model OR features OR hyperparams)
- All hyperparameters must be valid for the chosen model
- Use random_state/seed values for reproducibility where applicable

=== TOP-5 PERFORMERS FROM PREVIOUS GENERATION ===
{top5_history}
=== END TOP-5 ===

Respond with ONLY a JSON array of {size} objects, no additional text.
""").strip()
```

Fig. 4. Prompt de geração de população

componentes seno e cosseno para capturar a ciclicidade sazonal de forma contínua.

Para avaliar o desempenho e a eficácia das duas abordagens propostas, foi definido o Root Mean Squared Error(RMSE) como métrica de fitness. O artigo que publicou o dataset [24], [25] obteve RMSE 66.65 utilizando validação cruzada de 10 folds. Assadian, Cameron Francis, and Francis Assadian [26] obtiveram RMSE 59.61 utilizando Extra Trees Regressor, 62.96 para Random Forest e 63.86 para XGBoost também utilizando validação cruzada de 10 folds.

Foi definido um conjunto de modelos disponíveis: *Arima, Holt-winters, DecisionTreeRegressor, XGBoost, KNRegressor, SARIMAX, RandomForestRegressor, LightGBM, SVR, CatBoost, MLPRegressor, Prophet, TBATS, NBEATS, ExtraTreesRegressor, LGBMRegressor, CatBoostRegressor, XGBRegressor, ElasticNet, TCN, RNN, LSTM, GRU, TemporalFusionTransformer, VAR, BayesianRidge, StackingRegressor, KalmanFilter, ETS, DeepAR.*

Durante a implementação, houve suspeita de que o LLM estaria utilizando de informação prévia do Dataset utilizado, uma vez que está disponível publicamente na internet. Com isso, foi implementado um processo de anonimização das features, através da criação de um mapa que converte cada

```
def get_strategy_section(gen: int, pop_size: int) -> str:
    """Gera seção de estratégia dinamicamente baseada na geração"""
    if gen == 0:
        return """**FIRST GENERATION - Random Initialization:**
- Generate {pop_size} diverse individuals with random model and feature
combinations
- Explore the full model space: try at least 10 different model types
- Use varied feature combinations: some with few features, others with more
- Set reasonable hyperparameter ranges for each model type
- Ensure maximum diversity to establish a strong foundation
""".format(pop_size=pop_size)

    else:
        # Calcula distribuição para gerações posteriores
        elitism = int(pop_size * 0.3)
        crossover = int(pop_size * 0.3)
        exploration = pop_size - elitism - crossover

        return f"""**GENERATION {gen} - Evolutionary Strategy:**
1. **Elitism ({elitism} individuals):**
   - Take the TOP performers and apply small mutations (add/remove 1 feature
   and/or tweak 1 hyperparameter)

2. **Crossover ({crossover} individuals):**
   - Combine features from one top performer with hyperparameters from another
   - Mix model types: use successful features with different algorithms
   - Blend hyperparameter ranges from successful candidates

3. **Exploration ({exploration} individuals):**
   - Try underexplored models or feature combinations
   - Experiment with hyperparameter ranges not yet tested
   - Focus on models/features that showed promise but need refinement
   - Balance exploitation of known good patterns with exploration of new
   possibilities"""
```

Fig. 5. Estratégia de geração de população definida no prompt

```
def get_strategy_section(gen: int, pop_size: int) -> str:
    if gen == 0:
        return f"""**FIRST GENERATION - Free Random Initialization**
- Generate {pop_size} diverse individuals with broad coverage of models and
features
- Prefer high diversity across models, feature subsets, and hyperparameters
- Do not assume narrow ranges; include both conservative and extreme (but valid)
values
""".format(pop_size=pop_size)
    else:
        return f"""**GENERATION {gen} - Free-form Evolution (LLM-decided)**
- You are free to choose any balance between elitism, crossover, and exploration
 (including assigning 0 to any of them)
- Feel free to introduce other operators: mutation-only, random injections,
novelty search, model swap with same features, feature swap with same model,etc.
- Leverage patterns from previous TOP-5 performers but keep a meaningful portion
 for wildcard ideas
- Avoid near-duplicates (change at least model OR features OR hyperparameters)
- Explore wide hyperparameter ranges (uniform/log-uniform where appropriate)
without sticking to narrow bands
- Produce exactly {pop_size} valid and unique individuals
"""
```

Fig. 6. Estratégia livre de geração de população

nome de feature original para um rótulo genérico, como `F01`, `F02`. Entretanto, após a anonimização dos nomes das colunas não foi constatada mudança no comportamento do LLM.

B. Resultados

Foram executados experimentos usando as 3 abordagens, com 20 indivíduos por geração e utilizando validação cruzada de 10 folds. A Fig. 7 mostra o gráfico de convergência do abordagem (i), algoritmo genético clássico estocástico, que obteve RMSE 64.51 na 5ª geração com o algoritmo CatBoostRegressor. Já as Figs. 8 e 9 retratam os gráficos de convergência das abordagens (ii) e (iii), com RMSE 63.38 na 7ª geração e RMSE 63.58 na 9ª geração, respectivamente, ambos com XGBoost.

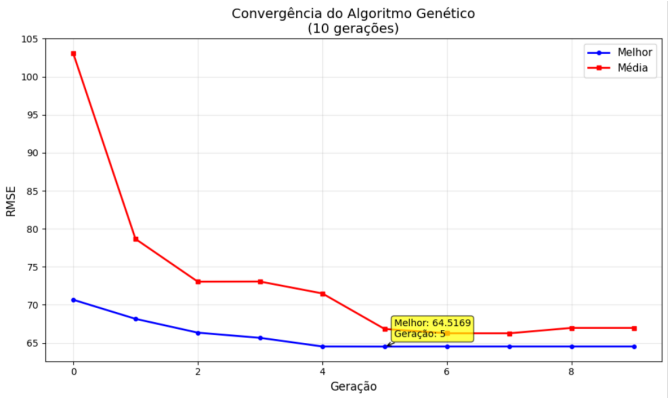

Fig. 7. Abordagem (i): Convergência do algoritmo genético clássico

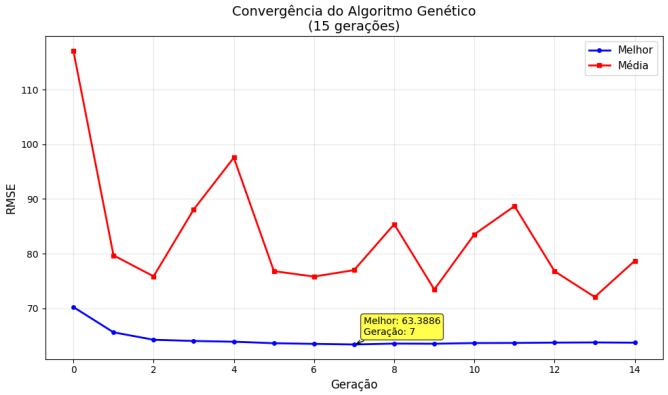

Fig. 8. Abordagem (ii): Convergência do algoritmo genético com LLM como guia para evolucão, com estratégia definida no prompt

C. Explicabilidade

Foi realizada uma análise dos indivíduos gerados nas populações de cada experimento. Analisando os indivíduos e suas métricas de cada abordagem, nota-se que a abordagem (i) converge cedo e a média de fitness dos invidíduos diminui e também converge para próximo do mínimo, enquanto as abordagens (ii) e (iii) também convergem cedo, mas a média de cada geração oscila, indicando que o LLM está tentando explorar o espaço de busca.

Na abordagem (i), estocástica, como não há nenhuma inteligência guiando a escolha de features ou a calibração dos hiperparâmetros, a evolução tende a recombinar cegamente partes de soluções que não tem fitness tão bom, fazendo com que certos padrões de features e modelos se tornem dominantes não por serem os melhores, mas por serem herdados com frequência. Isso leva a uma convergência prematura: a população evolui em poucos modelos e muitas variações são apenas clones com pequenas mudanças numéricas, podendo levar o algoritmo a ficar preso em um mínimo local. Essa convergência prematura pode ser observada na Fig. 7, onde a média de fitness de cada geração(linha vermelha) se aproxima cada vez mais do ótimo da geração(linha azul).

Com a estratégia de evolução definida no prompt da abordagem (ii) (30% elitismo, 30% crossover, 40% exploração), o LLM transformou uma população inicial muito diversa em um conjunto bem concentrado de árvores/GBMs e alguns modelos sequenciais apoiados em um conjunto estável de features(lights + RH_2 ou RH_6 + T3 + lags + rolling stats + tempo), com pouca exploração nas gerações finais, indicando que a evolução encontrou um padrão ótimo que o LLM passou a explorar quase exclusivamente. Diferente da abordagem (i), o LLM priorizou crossover entre modelos da mesma classe.

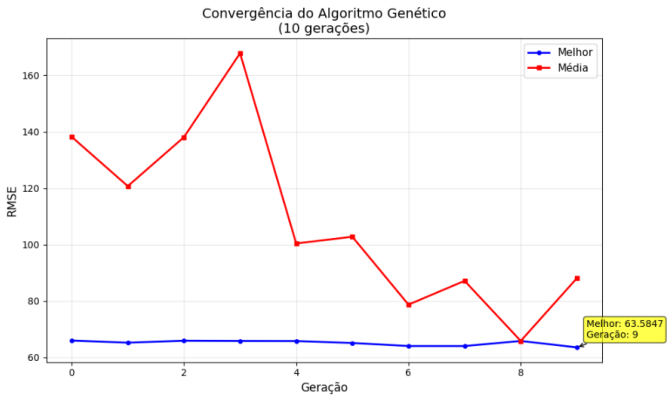

Fig. 9. Abordagem (iii): Convergência do algoritmo genético com LLM como guia para evolução, com estratégia definida pelo LLM de forma livre

Por fim, na abordagem (iii) com estratégia livre, o LLM também começou a evolução com uma população inicial muito diversa e evoluiu para o mesmo conjunto de modelos de ensembles de árvore (ExtraTrees, RandomForest, LGBM, XGBoost) e modelos lineares regularizados(ElasticNet e BayesianRidge), sempre usando o conjunto de features: lags, estatísticas de janelas e sazonalidade temporal. Em todas as gerações há algum LSTM/GRU/TCN/NBEATS/DeepAR/TFT, mas poucos por geração, gerados como exploração do espaço de busca. Também aparecem em todas as gerações modelos clássicos de séries, como ARIMA, SARIMAX e Holt-Winters, começando com poucas variáveis e expandindo o número de variáveis ao longo das gerações. O LLM aprendeu que os melhores modelos tem lag 1 e frequentemente utiliza outros lags, como 3,7,24. Também utiliza estatísticas rolling_mean, hora do dia, dia da semana e seno e cosseno do mês. É possível perceber claramente que o LLM utilizou elitismo, mutação de hiperparâmetros, crossover de features e alguns modelos novos aleatórios.

V. Conclusão

Este trabalho investigou a integração de LLMs com algoritmo genético evolucionário para a otimização de modelos de previsão de séries temporais. Os experimentos realizados compararam uma abordagem de algoritmo genético clássico, puramente estocástica, com duas abordagens guiadas por LLM, uma com estratégia de evolução definida no prompt e outra com estratégia livre, utilizando o dataset *Appliances Energy Prediction*.

A análise qualitativa dos indivíduos de cada geração indica que os LLMs podem ter alguma vantagem em relação à abordagem puramente estocástica. A abordagem estocástica pode convergir precocemente a um mínimo local, pois a aleatoriedade pode levar alguns padrões de modelos e features a se tornarem dominantes por serem herdados com frequência. Enquanto na abordagem com estratégia definida no prompt, o LLM foi instruído com valores percentuais de elitismo, crossover e exploração e ainda assim não atuou como um operador de evolutivo cego: o LLM identificou e preservou um conjunto de features(como lags e estatísticas de janela móvel) e realizou crossovers e mutações que faziam mais sentido do ponto de vista de arquitetura dos modelos, mantendo também uma taxa de mutação(exploração) de modelos. Por fim, na abordagem onde o LLM teve liberdade para definir sua própria estratégia de evolução, também foi identificado um conjunto de features e o LLM utilizou elitismo, crossover de hiperparâmetros e mutação de hiperparâmetros e modelos.

Foi realizada a anonimização das features, que não alterou o comportamento do LLM, sugerindo que o modelo conseguiu orientar as decisões evolutivas com base na estrutura dos dados e nas métricas de desempenho, e não por conhecimento prévio sobre o dataset.

A combinação entre algoritmo genético evolucionário e LLM foi capaz de obter resultados próximos ao estado da arte para o dataset analisado. Além disso, os resultados indicam que os LLMs atuais possuem capacidade suficiente para gerenciar o trade-off entre exploração(exploration) e aproveitamento(exploitation) de forma autônoma, sem a necessidade de definir taxas de mutação ou crossover. Por fim, a arquitetura proposta de geração e correção de código pelos LLMs se mostrou eficaz para modelos de previsão de séries temporais em Python.

REFERENCES

[1] Shumway RH, Stoffer DS. Time series analysis and its applications. New York: springer; 2000 Mar.

[2] Shaygan, Maryam and Meese, Collin and Li, Wanxin and Zhao, Xiaoliang (George) and Nejad, Mark. Traffic prediction using artificial intelligence: Review of recent advances and emerging opportunities. 2022

[3] Shumway, R.H., Stoffer, D.S.. ARIMA Models. In: Time Series Analysis and Its Applications. Springer Texts in Statistics. Springer, Cham. https://doi.org/10.1007/978-3-319-52452-8_3. 2017.

[4] G.Peter Zhang. Time series forecasting using a hybrid ARIMA and neural network model. Neurocomputing, vol. 50, pp. 159–175, 2003

[5] Alysha M. de Livera, Rob J. Hyndman, and Ralph D. Snyder. 2011. Forecasting time series with complex seasonal patterns using exponential smoothing. J. Amer. Statist. Assoc. 106 (12 2011), 1513–1527. Issue 496.

[6] Chatfield C. The Holt-winters forecasting procedure. Journal of the Royal Statistical Society: Series C (Applied Statistics). 1978 Nov;27(3):264-79.

[7] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[8] Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q, Liu TY. Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems. 2017;30.

[9] Breiman L. Random forests. Machine learning. 2001 Oct;45:5-32.

[10] Hearst MA, Dumais ST, Osuna E, Platt J, Scholkopf B. Support vector machines. IEEE Intelligent Systems and their applications. 1998 Jul;13(4):18-28.

[11] Hamilton, James D. Time series analysis. Princeton university press, 2020.

[12] Chan NH. Time series: applications to finance. John Wiley & Sons; 2004 Mar 22.

[13] Shengchao Chen and Guodong Long and Jing Jiang and Dikai Liu and Chengqi Zhang Foundation Models for Weather and Climate Data Understanding: A Comprehensive Survey. 2023.

[14] I. Paliari, A. Karanikola and S. Kotsiantis, "A comparison of the optimized LSTM, XGBOOST and ARIMA in Time Series forecasting," 2021 12th International Conference on Information, Intelligence, Systems & Applications (IISA), Chania Crete, Greece, 2021, pp. 1-7, doi: 10.1109/IISA52424.2021.9555520.

[15] Bergmeir C, Benítez JM. On the use of cross-validation for time series predictor evaluation. Information Sciences, vol. 191, pp. 192-213, 2012. https://doi.org/10.1016/j.ins.2011.12.028

[16] OpenAI. GPT-4 Technical Report. arXiv:2303.08774. 2023.

[17] Gautier Dagan and Frank Keller and Alex Lascarides. arXiv:2308.06391 Dynamic Planning with a LLM. 2023.

[18] Coignion, Tristan and Quinton, Clément and Rouvoy, Romain. A Performance Study of LLM-Generated Code on Leetcode. Association for Computing Machinery. 2024.

[19] Jiawei Gu et al. A Survey on LLM-as-a-Judge. arXiv:2411.15594. 2024.

[20] Brown et al. Language Models are Few-Shot Learners. Advances in Neural Information Processing Systems. 2020.

[21] Lambora, Annu and Gupta, Kunal and Chopra, Kriti. Genetic Algorithm-A Literature Review. 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon). 2019.

[22] Wu X, Wu SH, Wu J, Feng L, Tan KC. Evolutionary computation in the era of large language model: Survey and roadmap. IEEE Transactions on Evolutionary Computation. 2024 Nov 27.

[23] Yecheng Jason Ma et al. Eureka: Human-Level Reward Design via Coding Large Language Models. arXiv:2310.12931. 2023.

[24] Luis M. Ibarra Candanedo and Veronique Feldheim and Dominique Deramaix. "Appliance Energy Prediction". Kaggle. DOI 10.24432/C5VC8G. 2017.

[25] Luis M. Ibarra Candanedo and Veronique Feldheim and Dominique Deramaix. Data driven prediction models of energy use of appliances in a low-energy house. Energy and Buildings. 2017.

[26] Assadian, Cameron Francis, and Francis Assadian. 2023. "Data-Driven Modeling of Appliance Energy Usage" Energies 16, no. 22: 7536. https://doi.org/10.3390/en16227536