



UNIVERSIDADE ESTADUAL PAULISTA
“JÚLIO DE MESQUITA FILHO”
Instituto de Ciência e Tecnologia de Sorocaba

CAMILLA RODRIGUES CARVALHO

Previsão de Consumo Energético com Redes Neurais

Sorocaba

2025

CAMILLA RODRIGUES CARVALHO

PREVISÃO DE CONSUMO ENERGÉTICO COM REDES NEURAIAS

Trabalho de Conclusão de Curso apresentado à
Universidade Estadual Paulista (UNESP),
Instituto de Ciência e Tecnologia, Sorocaba,
como parte dos requisitos para obtenção do grau
de Bacharela em Engenharia de Controle e
Automação.

Orientador: Prof. Dr. Leopoldo André Dutra
Lusquino Filho

Sorocaba

2025

C331p

Carvalho, Camilla Rodrigues

Previsão de consumo energético com redes neurais / Camilla Rodrigues Carvalho. -- Sorocaba, 2025

60 p. : il., tabs.

Trabalho de conclusão de curso (Bacharelado - Engenharia de Controle e Automação) - Universidade Estadual Paulista (UNESP), Instituto de Ciência e Tecnologia, Sorocaba

Orientador: Leopoldo André Dutra Lusquino Filho

1. Aprendizado do computador. 2. Redes neurais (Computação). 3. Análise energética. I. Título.

CAMILLA RODRIGUES CARVALHO

PREVISÃO DE CONSUMO ENERGÉTICO COM REDES NEURAI

Trabalho de Conclusão de Curso apresentado ao Instituto de Ciência e Tecnologia de Sorocaba, Universidade Estadual Paulista (UNESP), como parte dos requisitos para obtenção do grau de Bacharel(a) em Engenharia de Controle e Automação.

Data da defesa: 17 / 06 / 2025

BANCA EXAMINADORA:

Prof. Dr. Leopoldo André Dutra Lusquino Filho
UNESP – Instituto de Ciência e Tecnologia – Campus de Sorocaba

Profa. Dra. Luiza Amália Pinto Cantão
UNESP – Instituto de Ciência e Tecnologia – Campus de Sorocaba

Prof. Dr. Luis Armando De Oro Arenas
UNESP – Instituto de Ciência e Tecnologia – Campus de Sorocaba

Dedico este trabalho a todos que estiveram ao meu lado nesta jornada, nos momentos de alegria e nas adversidades. Agradeço, em especial, a quem me deu forças para seguir adiante, mesmo nos momentos mais difíceis, e a quem compartilhou comigo risadas e desabafos durante os estudos e trabalhos. Por fim, minha gratidão e dedicação à minha família, por todo o apoio e suporte essenciais.

Agradecimentos

Meus agradecimentos iniciam-se com a minha família, que me deu todo o suporte até este ponto. Em particular, sou grato ao meu pai, pelo incentivo e pelos conselhos firmes que foram cruciais. Às minhas irmãs, por serem o porto seguro onde pude compartilhar minhas dificuldades e frustrações. Estendo minha gratidão aos profissionais de saúde que foram essenciais para minha saúde mental e para manter o foco necessário para não desistir. Agradeço, também, ao meu orientador, Leopoldo, pela valiosa orientação, paciência e incentivo que foram fundamentais para a realização deste trabalho. A todos os professores e técnicos, por compartilhar não apenas conhecimento acadêmico, mas também importantes lições de vida. E, por fim, agradeço ao meu artista favorito (BTS), cujas músicas inspiradoras foram fonte de paz, meditação e conforto nos momentos mais árduos.

CARVALHO, Camilla Rodrigues. **Previsão de consumo energético com redes neurais**. 2025. 60 p. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Controle e Automação) - Instituto de Ciência e Tecnologia, Universidade Estadual Paulista, Sorocaba, 2025.

RESUMO

Por meio de técnicas avançadas de Machine Learning (ML), busca-se maximizar o aproveitamento dos recursos energéticos disponíveis em cada localidade, garantindo o fornecimento de energia de forma sustentável e eficiente. Essa abordagem permite prever padrões de consumo, otimizar a geração de energia a partir de fontes renováveis e não renováveis, e reduzir desperdícios, alinhando-se às metas globais de descarbonização e à promoção de um futuro energético mais resiliente e sustentável. Tendo isso em vista, o presente trabalho propõe um estudo abrangente que investiga a eficácia do uso de redes neurais, em particular a arquitetura LSTM (Long Short-Term Memory), GRU (Gated Recurrent Unit) e LSTM-Bidirecional, GRU-Bidirecional para a previsão de consumo residencial. Foram realizados experimentos em cenários distintos, utilizando-se de duas submedições, e entradas variadas a fim de avaliar a robustez e performance dos métodos, comparando-os com métricas já consolidadas, encontrando assim melhores resultados para LSTM e BiGRU.

Palavras-chave: modelo preditivo; LSTM; BiLSTM; GRU; BiGRU; consumo energético; métrica; RNN.

CARVALHO, Camilla Rodrigues. **Energy consumption forecasting with neural networks.** 2025. 60 p. Final Paper (Bachelor's degree in Control and Automation Engineering) – Institute of Science and Technology, São Paulo State University, Sorocaba, 2025.

ABSTRACT

Through advanced Machine Learning (ML) techniques, the goal is to maximize the utilization of available energy resources in each location, ensuring a sustainable and efficient energy supply. This approach enables the prediction of consumption patterns, the optimization of energy generation from both renewable and non-renewable sources, and the reduction of waste, aligning with global decarbonization goals and promoting a more resilient and sustainable energy future. The present work proposes a comprehensive study that investigates the effectiveness of using neural networks particularly the LSTM (Long Short-Term Memory), GRU (Gated Recurrent Unit), Bidirectional LSTM, and Bidirectional GRU architectures for residential consumption forecasting. Experiments were conducted in different scenarios using two submetering configurations and varied input features to assess the robustness and performance of the methods. The results were compared to using established metrics, with LSTM and BiGRU showing the best performance.

Keywords: predictive model; LSTM; BiLSTM; GRU; BiGRU; energy consumption; metric; RNN.

Lista de Figuras

Figura 1 - Representação esquemática de um neurônio artificial, mostrando suas entradas, pesos, função de soma e função de ativação que gera a saída.	15
Figura 2 - Arquitetura simplificada de uma Rede Neural Recorrente (RNN), evidenciando o fluxo de informação sequencial e o loop de feedback.	18
Figura 3 - Diagrama interno de uma célula Long Short-Term Memory (LSTM).	19
Figura 4 - Diagrama interno de uma célula Gated Recurrent Unit Memory (LSTM).	20
Figura 5 - Fluxograma do experimento	23
Figura 6 - Amostra do consumo [Wh] Real vs. Previsto para input 10 [Sub2 LSTM].	27
Figura 7 - Perda de treino e teste em Wh_2 para input 30 [Sub2 LSTM].	28
Figura 8 - Perda de treino e teste em Wh_2 [Sub2 LSTM].	28
Figura 9 - Amostra do consumo [Wh] Real vs. Previsto para input 30 [Sub3 LSTM].	30
Figura 10 - Perda de treino e teste em Wh_2 para input 30 [Sub3 LSTM].	30
Figura 11 - Perda de treino e teste em Wh_2 [Sub3 LSTM].	31
Figura 12 - Amostra do consumo [Wh] Real vs. Previsto para input 20 [Sub2 BiLSTM].	32
Figura 13 - Perda de treino e teste em Wh_2 para input 10 [Sub2 BiLSTM].	33
Figura 14 - Perda de treino e teste em Wh_2 [Sub2 BiLSTM].	34
Figura 15 - Amostra do consumo [Wh] Real vs. Previsto para input 30 [Sub3 BiLSTM].	35
Figura 16 - Perda de treino e teste em Wh_2 para input 10 [Sub3 BiLSTM].	36
Figura 17 - Perda de treino e teste em Wh_2 [Sub3 BiLSTM].	36
Figura 18 - Amostra do consumo [Wh] Real vs. Previsto para input 15 [Sub2 GRU].	38
Figura 19 - Perda de treino e teste em Wh_2 para input 15 [Sub2 GRU].	38
Figura 20 - Perda de treino e teste em Wh_2 [Sub2 GRU].	39
Figura 21 - Amostra do consumo [Wh] Real vs. Previsto para input 25 [Sub3 GRU].	40
Figura 22 - Perda de treino e teste em Wh_2 para input 10 [Sub3 GRU].	41
Figura 23 - Perda de treino e teste em Wh_2 [Sub3 GRU].	41
Figura 24 - Amostra do consumo [Wh] Real vs. Previsto para input 25 [Sub2 BiGRU].	43
Figura 25 - Perda de treino e teste em Wh_2 para input 10 [Sub2 BiGRU].	43
Figura 26 - Perda de treino e teste em Wh_2 [Sub2 BiGRU].	44
Figura 27 - Amostra do consumo [Wh] Real vs. Previsto para input 20 [Sub3 BiGRU].	45
Figura 28 - Perda de treino e teste em Wh_2 para input 25 [Sub3 BiGRU].	46
Figura 29 - Perda de treino e teste em Wh_2 [Sub3 BiGRU].	46

Lista de Tabelas

Tabela 1 - Métricas de Teste LSTM Sub2.	26
Tabela 2 - Métricas de Treino LSTM Sub2.	26
Tabela 3 - Métricas de Teste LSTM Sub3.	29
Tabela 4 - Métricas de Treino LSTM Sub3.	29
Tabela 5 - Métricas de Teste BiLSTM Sub2.....	32
Tabela 6 - Métricas de Treino BiLSTM Sub2.....	32
Tabela 7 - Métricas de Teste BiLSTM Sub3.....	34
Tabela 8 - Métricas de Treino BiLSTM Sub3.....	35
Tabela 9 - Métricas de Teste GRU Sub2.....	37
Tabela 10 - Métricas de Treino GRU Sub2.....	37
Tabela 11 - Métricas de Teste GRU Sub3.....	40
Tabela 12 - Métricas de Treino GRU Sub3.....	40
Tabela 13 - Métricas de Teste BiGRU Sub2.....	42
Tabela 14 - Métricas de Treino BiGRU Sub2.....	42
Tabela 15 - Métricas de Teste BiGRU Sub3.....	45
Tabela 16 - Métricas de Treino BiGRU Sub3.....	45

SUMÁRIO

1. INTRODUÇÃO.....	12
2. REVISÃO BIBLIOGRÁFICA.....	14
2.1 Neurônio Artificial: O Bloco Construtor das Redes Neurais.....	14
2.2 Machine Learning na Predição de Energia	15
2.2.1 Predição da Demanda de Energia	15
2.2.2 Predição da Geração de Energia Renovável	16
2.3 Redes Neurais Recorrentes (RNNs) e suas Variantes para Séries Temporais.....	17
2.3.1 Redes Neurais Recorrentes (RNNs)	18
2.3.2 <i>Long Short-Term Memory</i> (LSTM)	18
2.3.3 Gated Recurrent Unit (GRU).....	19
2.4 Métricas de Avaliação em Regressão	20
2.4.1 Erro Médio Absoluto (MAE - <i>Mean Absolute Error</i>)	20
2.4.2 Erro Quadrático Médio (MSE - <i>Mean Squared Error</i>)	21
2.4.3 Raiz do Erro Quadrático Médio (RMSE - <i>Root Mean Squared Error</i>).....	21
2.4.4 Erro Percentual Absoluto Médio Simétrico (SMAPE - <i>Symmetric Mean Absolute Percentage Error</i>).....	22
3. Abordagem Proposta	23
3.1 Ambiente de Desenvolvimento.....	24
3.2 Dependências do Projeto	24
3.3 Importação das fontes dos dados	24
3.4 Limpeza dos dados e Funções.....	24
3.5 Execução e geração dos resultados	25
4. RESULTADOS E DISCUSSÕES.....	26
4.1 Modelo <i>Long Short-Term Memory</i> (LSTM).....	26
4.1.1 Sub-medição 2	26

4.1.2 Sub-medição 3	29
4.2 Modelo <i>Bidirectional Long Short-Term Memory</i> (BiLSTM).....	31
4.2.1 Sub-medição 2	31
4.2.2 Sub-medição 3	34
4.3 Modelo <i>Gated Recurrent Unit</i> (GRU)	37
4.3.1 Sub-medição 2	37
4.3.2 Sub-medição 3	39
4.4 Modelo <i>Bidirectional Gated Recurrent Unit</i> (BiGRU)	42
4.4.1 Sub-medição 2	42
4.4.2 Sub-medição 3	44
5. CONCLUSÃO.....	47
REFERÊNCIAS	49
APÊNDICE A – CÉLULAS DE IMPORTAÇÃO DE BIBLIOTECAS	51
APÊNDICE B - CÉLULA DE IMPORTAÇÃO E TRANSFORMAÇÃO.....	52
APÊNDICE C - CÉLULA DE TRATAMENTO DE VALORES FALTANTES	53
APÊNDICE D - CÉLULA DE TRATAMENTO DE OUTLIERS.....	54
APÊNDICE E - CÉLULA CRIAÇÃO DE FUNÇÕES	55
APÊNDICE F – FUNÇÃO DE EXECUÇÃO.....	56
APÊNDICE G – EXECUÇÃO DOS RESULTADOS.....	57
APÊNDICE H – PLOTAGEM E ARMAZENAMENTO DOS GRÁFICOS DE PERDA	58
APÊNDICE I – PLOTAGEM E ARMAZENAMENTO DOS GRÁFICOS DE REAL VS. PREVISTO	59
APÊNDICE J – ARMAZENAMENTO DAS MÉTRICAS UTILIZADAS	60

1. INTRODUÇÃO

O objetivo do trabalho é analisar a performance de modelos preditivos aplicados ao setor de consumo energético residencial, avaliando sua capacidade de prever demandas e otimizar a distribuição de energia de maneira eficaz.

A elaboração do trabalho de conclusão de curso visa fortalecer competências técnicas e científicas, expandir a perspectiva na área científica e, potencialmente, criar um recurso fundamental para o início de uma trajetória acadêmica ou no mercado de trabalho. Desta forma, a realização desta investigação reveste-se de grande importância, assim como a seleção do domínio de estudo. A temática abordada foi selecionada com o propósito de integrar os saberes assimilados às demandas sociais atuais, buscando, com isso, otimizar a aplicação prática do aprendizado.

A eficiência energética vem se consolidando como uma estratégia central para mitigar mudanças climáticas, conservar recursos naturais e reduzir a dependência de fontes externas de energia, sobretudo em países ainda vinculados a combustíveis fósseis importados. Contudo, persistem desafios econômicos, como os altos investimentos iniciais em tecnologias limpas, custos operacionais e a viabilidade financeira para o consumidor. Essa dinâmica, no entanto, tem sido reavaliada à luz de transformações recentes, como a emergência climática, a transição energética global e a escassez de recursos, que redefiniram a concepção econômica da eficiência energética (GIRAUDET; MISSEMER, 2023).

O setor residencial é um dos principais responsáveis pelo consumo global de eletricidade, representando cerca de 26,8% do consumo final mundial em 2022. Esse percentual refere-se à energia efetivamente utilizada em residências para iluminação, eletrodomésticos, climatização e outros fins domésticos, excluindo perdas associadas à geração e distribuição. A demanda residencial por eletricidade mantém-se elevada devido ao aumento da urbanização, expansão do acesso à energia e adoção de tecnologias intensivas em consumo (INTERNATIONAL ENERGY AGENCY, 2023).

No setor energético, o *Machine Learning* (aprendizado de máquina - ML) tem se mostrado uma ferramenta estratégica para enfrentar desafios críticos, incluindo a transição para fontes renováveis, a integração de sistemas de energia distribuída e a otimização da eficiência operacional. Entre as aplicações mais relevantes destacam-se a previsão de preços de commodities energéticas (petróleo e gás natural) e a previsão de demanda energética - esta última particularmente crucial para o planejamento e operação de redes elétricas inteligentes (*smart grids*). Tais aplicações apresentam elevada complexidade devido à influência conjunta

de fatores econômicos, ambientais e climáticos, que introduzem significativas incertezas e variabilidades nos sistemas energéticos (ZHANG; WANG, 2021).

No Brasil, algumas empresas já estão à frente no desenvolvimento de técnicas para a eficiência energética residencial, podemos citar como exemplo IoGrid que utiliza inteligência artificial (IA) para otimização do consumo residencial, Lummi com automação e eficiência de iluminação, a Lemon Energia que por meio de sua tecnologia, faz a medição e predição do consumo de estabelecimentos, que também podem ser condomínios residenciais, e a partir da análise do consumo os conectam a fornecedores de energia sustentável em todo o país, além das tradicionais Siemens Brasil e Enel.

Após considerações, utilizou-se do *dataset* de medições do consumo de energia elétrica em uma residência com amostragem ao longo de um período de quase 47 meses, para a análise de modelos com foco em área residencial, considerando apenas o consumo como variável. Utilizou-se para estudar o comportamento, a submedição 2 sendo ela em watts-hora de energia ativa, que indica a lavanderia (contendo uma máquina de lavar roupa, uma secadora, uma geladeira e uma luz) e a submedição 3 também em watts-hora de energia ativa que corresponde a um aquecedor de água elétrico e a um ar-condicionado.

Os principais objetivos deste trabalho foram:

- Realizar uma análise comparativa do desempenho dos algoritmos de aprendizado de máquina selecionados (LSTM, GRU, BiLSTM e BiGRU) para prever o consumo de energia em um conjunto de dados residenciais;
- Utilizando a literatura, descrever quais os desempenhos ideais para cada um dos modelos e qual deveria ser mais performático;
- Discutir possíveis erros realizados na predição e como podem afetar os resultados.

O restante do relatório está organizado com os tópicos: seção 2 nos traz alguns trabalhos relacionados e revisão de literatura. A seção 3 apresenta a abordagem proposta. A seção 4 apresenta os experimentos, resultados e discussão e por fim a seção 5 nos traz a conclusão.

2. REVISÃO BIBLIOGRÁFICA

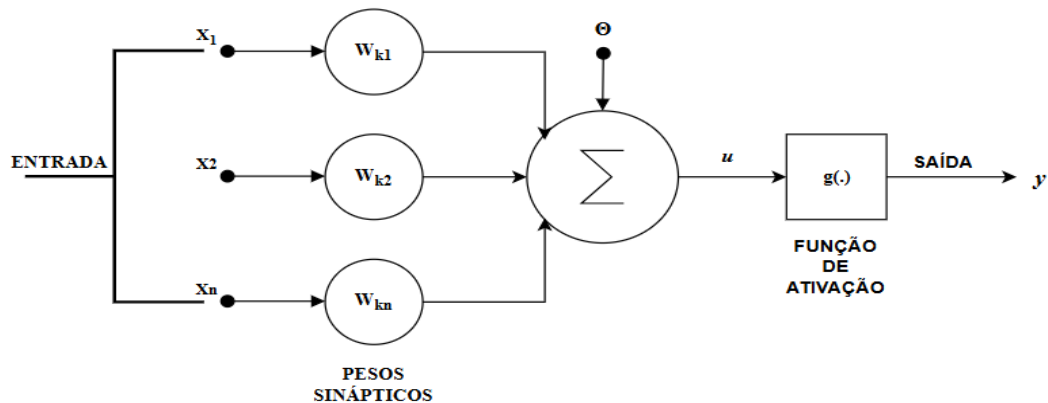
A escolha do algoritmo de *Machine Learning* (ML) é intrinsecamente ligada à natureza do problema e às características dos dados. No domínio da predição de energia, as abordagens de aprendizado mais relevantes incluem:

- **Aprendizado Supervisionado:** Envolve o treinamento de modelos com dados "rotulados", onde os pares de entrada e saída são conhecidos, permitindo que o algoritmo estabeleça um mapeamento preditivo, suas aplicações incluem regressão (para prever valores contínuos como demanda ou geração) e classificação.
- **Aprendizado Não Supervisionado:** Empregado para descobrir estruturas ou agrupamentos intrínsecos em dados sem rótulos predefinidos, sendo útil na detecção de anomalias ou na segmentação de perfis de consumo.
- **Aprendizado por Reforço:** Embora menos direto na previsão pura, possui grande potencial na otimização de sistemas de energia, como na gestão de armazenamento, onde o algoritmo aprende a melhor sequência de ações para maximizar uma recompensa como economia de custos ou eficiência energética.

2.1 Neurônio Artificial: O Bloco Construtor das Redes Neurais

No cerne de muitas arquiteturas de ML, especialmente as redes neurais, encontra-se o conceito de neurônio artificial, que é inspirado no neurônio biológico. Para entender o funcionamento básico das redes neurais, é fundamental compreender a estrutura de um neurônio artificial, ilustrada na Figura 1. Resumidamente, um neurônio artificial recebe múltiplos sinais de entrada ($X_1, X_2, \dots, X_n$) cada um associado a um peso ($Wk_1, Wk_2, \dots, Wk_n$) que determina sua importância. Estes sinais ponderados são somados, e o resultado limiar de ativação (Θ) que especifica qual será o “patamar” apropriado para que o resultado produzido pelo combinador linear possa gerar um valor de disparo de ativação (u) para assim passar por uma função de ativação ($g(\cdot)$). Essa função introduz não-linearidade no modelo, permitindo que a rede aprenda relações complexas e não-lineares nos dados, algo crucial para a modelagem de fenômenos energéticos que exibem alta complexidade. A saída (y) do neurônio é, então, propagada para outros neurônios ou constitui a saída final do modelo. Essa estrutura em camadas de neurônios, com suas conexões e pesos ajustáveis durante o treinamento, permite que as redes neurais aprendam representações hierárquicas dos dados, tornando-as extremamente poderosas para tarefas de predição em séries temporais (GOODFELLOW et al., 2016).

Figura 1 - Representação esquemática de um neurônio artificial, mostrando suas entradas, pesos, função de soma e função de ativação que gera a saída.



Fonte: Adaptado de Deep Learning Book Brasil, 2024.

2.2 Machine Learning na Predição de Energia

2.2.1 Predição da Demanda de Energia

A previsão precisa da demanda de energia é vital para o planejamento operacional das redes elétricas, mitigando sobrecargas e assegurando a estabilidade do suprimento. Modelos de Machine Learning analisam séries temporais complexas, incorporando dados históricos de consumo, fatores climáticos (temperatura, umidade), padrões sazonais (feriados, períodos de pico) e variáveis socioeconômicas. Para isso alguns algoritmos são comumente utilizados:

- **Modelos de Regressão Tradicionais:** Incluem Regressão Linear e Polinomial, servindo como linhas de base para comparação.
- **Máquinas de Vetores de Suporte (SVM):** Empregadas em tarefas de regressão para identificar padrões complexos.
- **Redes Neurais Artificiais (RNA) e Deep Learning:** Notavelmente eficazes na captura de não linearidades e interações complexas entre variáveis, conforme demonstrado em estudos sobre previsão de carga (GASPARIN, ALIPPI & LUKOVIC, 2019; MOCANU et al., 2018).
- **Modelos Baseados em Árvores de Decisão:** Como *Random Forest* e *Gradient Boosting Machines* (GBM), são valorizados pela sua robustez a ruídos e capacidade de lidar com diversos tipos de dados (AHMAD et al., 2022).

Ramos, Pousinho e Dias (2018) realizaram uma revisão abrangente sobre a aplicação de técnicas de aprendizado de máquina na previsão de demanda energética. Em seu estudo, os autores compararam sistematicamente o desempenho de diferentes abordagens, incluindo Redes Neurais Artificiais (RNAs), Máquinas de Vetor de Suporte (SVMs) e modelos

híbridos. A análise demonstrou que a eficácia de cada método varia significativamente conforme o horizonte temporal de previsão considerado, com os modelos híbridos apresentando particular vantagem em cenários complexos que exigem análise multiescala. O artigo ainda discute em profundidade as limitações práticas de cada técnica, oferecendo diretrizes valiosas para a seleção de algoritmos conforme as especificidades de cada aplicação em sistemas energéticos.

Lahouar e Ben Hadj Braiek (2015) investigaram a aplicação prática de algoritmos de aprendizado de máquina para previsão do consumo energético residencial diário. Em seu estudo, os autores demonstraram a eficácia de Redes Neurais Artificiais (RNAs) e Máquinas de Vetor de Suporte (SVMs) na identificação de padrões complexos de consumo elétrico em edificações. Os resultados evidenciaram a capacidade desses modelos em capturar tanto as dependências temporais quanto às variações sazonais no consumo energético, fator essencial para o gerenciamento preciso da demanda em nível local. A pesquisa destacou ainda o potencial dessas técnicas para otimizar operações em redes inteligentes, proporcionando maior eficiência energética em ambientes residenciais.

Ahmad et al. (2022) realiza um estudo comparativo abrangente sobre técnicas de aprendizado de máquina aplicadas à previsão de demanda energética residencial nos Estados Unidos. A pesquisa avaliou sistematicamente o desempenho de diversos algoritmos, com destaque para *Random Forest* e *Gradient Boosting Machines*, na modelagem do consumo doméstico de energia. Os resultados demonstraram que a acurácia das previsões está intrinsecamente relacionada à adequação do modelo escolhido às características específicas dos dados e aos objetivos da análise preditiva. O estudo enfatiza a importância de considerar fatores como sazonalidade, padrões de consumo e variáveis socioeconômicas na seleção da abordagem mais adequada para cada cenário de previsão energética.

2.2.2 Predição da Geração de Energia Renovável

A previsão da geração de energia renovável é crucial, considerando a natureza intermitente e variável de fontes como a solar e a eólica, que representam um desafio considerável para a estabilidade do sistema elétrico. Para superar essa questão, o aprendizado de máquina tem sido amplamente empregado na estimativa da produção dessas fontes, que por sua vez por serem previsões energéticas podem ser avaliadas para previsões residenciais. Essa abordagem se baseia na análise de dados meteorológicos, como a velocidade e direção do vento, irradiação solar, temperatura e nebulosidade, além de características geográficas e o histórico

de geração de energia que também podem ser abordados na área de consumo de energia residencial (Nielsen, Pinson e Madsen, 2017; Mocanu et al., 2018; Wang, Zhang e Ding, 2019). Diversos algoritmos de aprendizado de máquina são empregados para essa finalidade, destacando-se:

- **Redes Neurais Artificiais (RNAs) e Deep Learning:** Essas abordagens são notavelmente eficazes para modelar as complexas e não lineares relações entre os dados meteorológicos e a produção de energia.
- **Máquinas de Vetores de Suporte (SVM):** São aplicadas em tarefas de regressão para prever a potência gerada.
- **Modelos Baseados em Árvores (*Random Forest, Gradient Boosting*):** Demonstram bom desempenho na identificação de relações não lineares nos dados.
- **Modelos Híbridos:** Combinam técnicas de aprendizado de máquina com modelos estatísticos (como ARIMA) para capturar tanto os componentes de séries temporais quanto a influência de covariáveis na previsão.

No campo da previsão de energia eólica, Nielsen, Pinson e Madsen (2017) investigaram a aplicação de algoritmos de Machine Learning (ML). Em seu estudo, eles avaliaram a performance de diferentes modelos, como redes neurais e regressão, na capacidade de prever a geração de energia eólica. A pesquisa demonstrou como esses métodos avançados são eficazes para lidar com a variabilidade da produção eólica, utilizando dados de vento e outras variáveis atmosféricas, o que é fundamental para mitigar a intermitência dessa fonte de energia.

Complementarmente, Wang, Zhang e Ding (2019) ofereceram uma revisão abrangente sobre o uso de Machine Learning na previsão de diversas fontes de energia renovável, incluindo solar e eólica. O trabalho detalha os algoritmos de ML mais empregados, como redes neurais, Máquinas de Vetores de Suporte (SVMs) ao mesmo tempo em que aborda os desafios específicos de cada fonte. A revisão também enfatiza o impacto significativo da previsão precisa na integração bem-sucedida das energias renováveis na rede elétrica.

2.3 Redes Neurais Recorrentes (RNNs) e suas Variantes para Séries Temporais

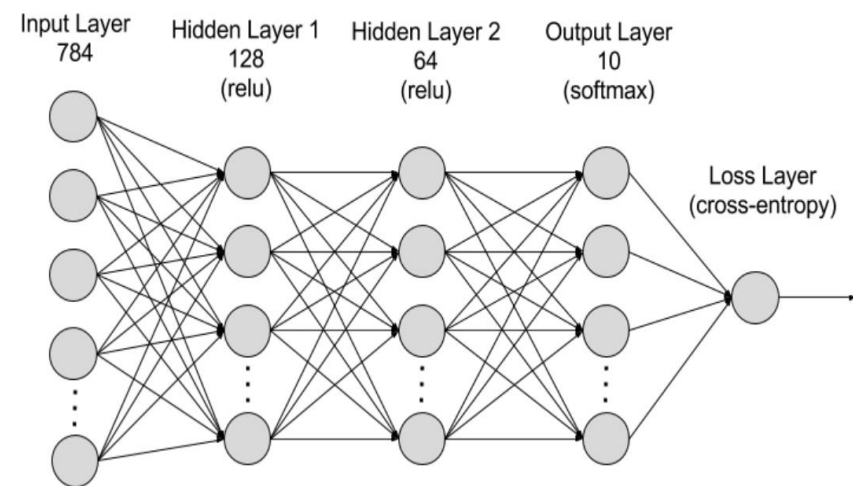
A natureza sequencial e temporal dos dados de consumo e geração de energia torna as Redes Neurais Recorrentes (RNNs) e suas variantes particularmente adequadas para tarefas de predição. Diferentemente das redes neurais *feedforward* tradicionais (MLPs), as RNNs possuem conexões que formam ciclos, permitindo que a informação persista ao longo do tempo. Isso lhes confere uma "memória", crucial para capturar dependências de longo prazo em séries

temporais.

2.3.1 Redes Neurais Recorrentes (RNNs)

Uma RNN processa sequências de dados, onde a saída em um determinado passo de tempo depende não apenas da entrada atual, mas também de uma "memória" do estado oculto anterior como demonstrado na Figura 2. Isso as torna ideais para tarefas como previsão de séries temporais, onde a ordem dos dados é crucial. As RNNs clássicas sofrem com o problema do "gradiente evanescente" (*vanishing gradient problem*) ou com problemas de explosão do gradiente (*exploding gradient problem*), que dificulta o aprendizado de dependências de longo prazo.

Figura 2 - Arquitetura simplificada de uma Rede Neural Recorrente (RNN), evidenciando o fluxo de informação sequencial e o loop de feedback.

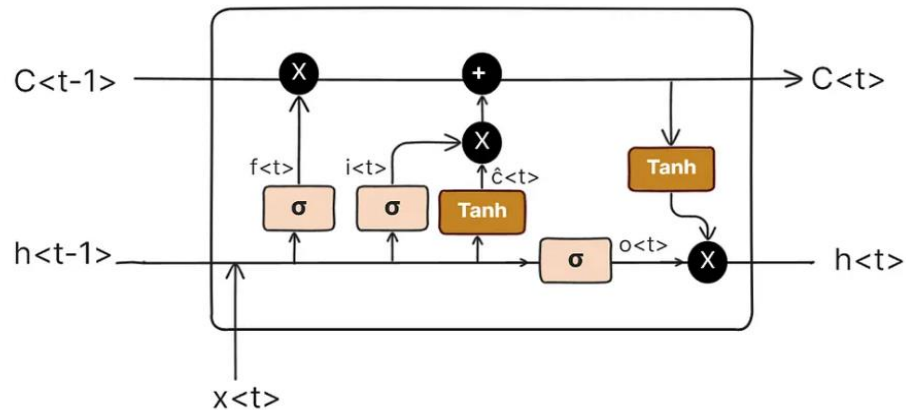


Fonte: Amazon Web Services, 2024.

2.3.2 Long Short-Term Memory (LSTM)

As LSTMs são uma arquitetura de RNN projetada para superar as limitações das RNNs tradicionais em aprender dependências de longo prazo. Elas introduzem o conceito de "células de memória" ($c_{<t>}$) e "portões" (*input* ($i_{<t>}$), *forget* ($f_{<t>}$) e *output gates* ($o_{<t>}$)) que controlam o fluxo de informações que podem ser vistos na Figura 3. Esses portões permitem que a LSTM selecione quais informações manter, quais descartar e quais passar adiante, protegendo o fluxo de gradiente e facilitando o aprendizado de padrões distantes no tempo (HOCHREITER; SCHMIDHUBER, 1997, citado por GASPARIN, ALIPPI & LUKOVIC, 2019). Elas são altamente eficazes na previsão de demanda e geração de energia, onde padrões sazonais e de longo prazo são comuns (ABDUL RAZAK et al., 2023).

Figura 3 - Diagrama interno de uma célula Long Short-Term Memory (LSTM).



Fonte: OMARZAI, Fraidoon, Medium, 2024.

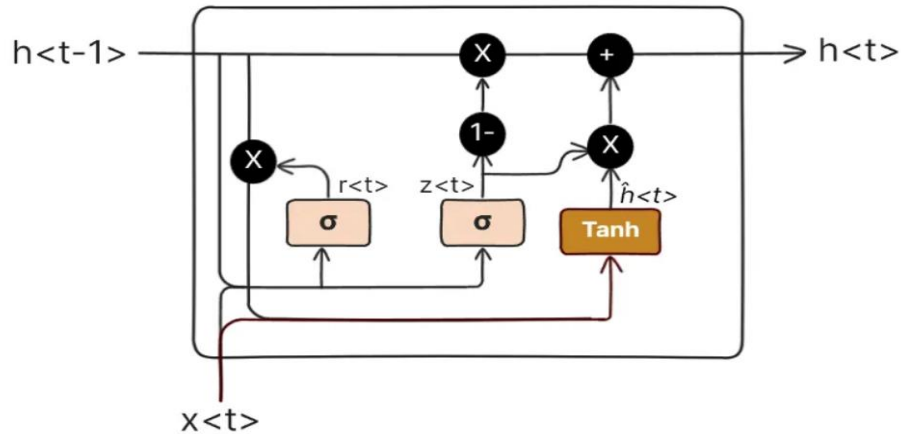
- **Bidirectional Long Short-Term Memory (BiLSTM)**

BiLSTM é uma variante da LSTM que processa a sequência de entrada em duas direções: uma para a frente (do passado para o futuro) e outra para trás (do futuro para o passado). Isso permite que o modelo capture informações de contexto de ambas as direções, o que pode ser benéfico em situações onde a previsão de um ponto no tempo pode se beneficiar do conhecimento de eventos futuros (no conjunto de dados de treinamento). Embora não seja um modelo preditivo que possa ser utilizado em *real time*, em um contexto de *offline training* e avaliação, a BiLSTM pode fornecer previsões mais robustas ao capturar nuances que dependem do contexto completo da série temporal analisada.

2.3.3 Gated Recurrent Unit (GRU)

GRUs são uma alternativa mais simples às LSTMs, com menos portões (apenas reset e update gates) como visto na Figura 4. Elas mantêm a capacidade de capturar dependências de longo prazo, mas com menos parâmetros, o que pode levar a um treinamento mais rápido e menor risco de overfitting em alguns casos. As GRUs oferecem um bom equilíbrio entre desempenho e complexidade computacional para a previsão de séries temporais de energia (ABDUL RAZAK et al., 2023).

Figura 4 - Diagrama interno de uma célula Gated Recurrent Unit (GRU).



Fonte: OMARZAI, Fraidoon, Medium, 2024.

- **Bidirectional Gated Recurrent Unit (BiGRU)**

Semelhante à BiLSTM, a BiGRU processa a sequência em ambas as direções (frente e atrás), mas utilizando a arquitetura simplificada da GRU. Ela combina a eficiência da GRU com a capacidade de capturar contexto bidirecional, útil para análises mais aprofundadas da série temporal.

2.4 Métricas de Avaliação em Regressão

Em problemas de regressão, onde o objetivo é prever um valor numérico contínuo, a avaliação se concentra em medir a diferença entre os valores previstos e os valores reais. Abaixo, detalhamos algumas das métricas mais comuns e suas nuances. Para todas as métricas, considere y_i como o valor real e $\hat{y}_i$ como o valor previsto para a i -ésima observação, e N como o número total de observações.

2.4.1 Erro Médio Absoluto (MAE - *Mean Absolute Error*)

O Erro Médio Absoluto é a média das diferenças absolutas entre as previsões e os valores reais, sendo sua fórmula:

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i| \quad (1)$$

O MAE representa a magnitude média dos erros sem considerar sua direção (se a previsão foi maior ou menor que o real) (WILLMOTT; MATSUURA, 2005). Ele é expresso na mesma unidade da variável de saída, o que o torna intuitivo e fácil de entender. Portanto, para os experimentos realizados, sua saída está em Wh.

O MAE é menos sensível a outliers do que o RMSE, pois não eleva ao quadrado os erros, o que significa que erros grandes não são penalizados desproporcionalmente. Porém a função de erro não é diferenciável em zero, o que pode ser um problema para alguns algoritmos de otimização baseados em gradiente, embora isso raramente seja uma limitação prática em bibliotecas modernas.

2.4.2 Erro Quadrático Médio (MSE - *Mean Squared Error*)

O Erro Quadrático Médio (MSE) é uma das métricas de perda e avaliação fundamentais em problemas de regressão. Ele calcula a média dos quadrados das diferenças entre os valores que o modelo previu ($\hat{y}_i$) e os valores reais ou observados (y_i) (Goodfellow, Bengio & Courville, 2016) como vemos na fórmula:

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (2)$$

Um valor de MSE menor indica um modelo com melhor desempenho, porém com a elevação ao quadrado dos erros temos que erros maiores são penalizados desproporcionalmente mais do que erros menores. Se o modelo cometer um erro significativo, o termo ao quadrado amplifica seu impacto na perda total, o tornando útil em cenários onde isto não ocorra.

2.4.3 Raiz do Erro Quadrático Médio (RMSE - *Root Mean Squared Error*)

O RMSE é a raiz quadrada da média dos quadrados dos erros entre as previsões e os valores reais, representada na fórmula:

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2} \quad (3)$$

Assim como o MAE, o RMSE é expresso na mesma unidade da variável de saída, facilitando a interpretação. Contudo, devido à operação de elevação ao quadrado, o RMSE penaliza erros maiores de forma mais significativa do que o MAE. Isso significa que um modelo com RMSE baixo tende a evitar grandes erros, mesmo que tenha muitos pequenos erros. A principal característica do RMSE é sua sensibilidade a erros grandes, portanto se o modelo cometer um ou dois erros muito grandes, o RMSE será consideravelmente maior do que o MAE, refletindo essa penalidade. Isso o torna útil em cenários onde grandes erros são particularmente indesejáveis (WILLMOTT; MATSUURA, 2005).

2.4.4 Erro Percentual Absoluto Médio Simétrico (SMAPE - *Symmetric Mean Absolute Percentage Error*)

O SMAPE é uma métrica de erro percentual que lida com o problema de divisão por zero comum em outras métricas percentuais (como o MAPE) e tenta ser simétrica em relação à subestimativa ou superestimativa mostrado pela fórmula:

$$SMAPE = \frac{100\%}{N} \sum_{i=1}^N \frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2} \quad (4)$$

O SMAPE retorna um erro em termos de porcentagem, o que é extremamente útil quando se deseja comparar o desempenho do modelo em diferentes conjuntos de dados ou séries temporais com escalas variadas. A divisão pela média dos valores reais e previstos na sua forma simétrica busca ser mais justa quando há zeros ou valores muito pequenos em y_i ou $\hat{y}_i$.

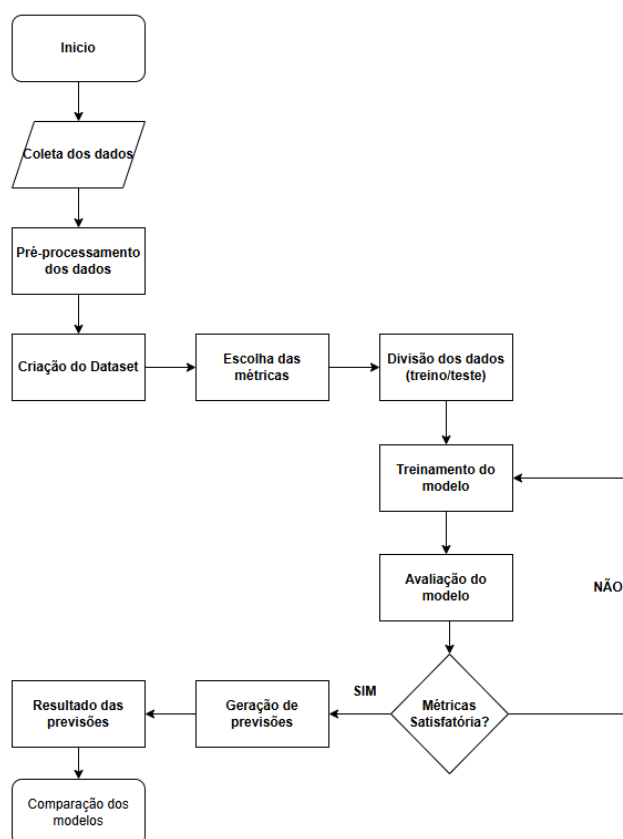
Por ser uma métrica percentual, o SMAPE é independente da escala dos dados, permitindo comparações significativas entre diferentes conjuntos de dados. Embora o SMAPE seja projetado para lidar melhor com zeros do que o MAPE, ele ainda pode apresentar problemas se tanto y_i quanto $\hat{y}_i$ forem zero, resultando em uma divisão por zero. Nesses casos, a contribuição para o erro é frequentemente definida como 0. Mesmo não sendo perfeitamente simétrico em todos os casos de erros (Chen; Yang, 2004), ele é geralmente preferível ao MAPE em muitos contextos de previsão devido à sua robustez a divisões por zero, ele é comumente utilizado em previsão de demanda e gestão de estoque, onde o erro percentual é uma métrica chave para a tomada de decisão.

Para problemas de regressão, métricas como MAE, MSE, RMSE e SMAPE oferecem perspectivas distintas sobre a precisão do modelo, com MAE focando na magnitude média do erro, RMSE penalizando erros maiores, e SMAPE fornecendo uma visão percentual e independente da escala. Ao empregar uma estratégia de avaliação robusta, os cientistas de dados e engenheiros podem garantir que os modelos desenvolvidos sejam não apenas tecnicamente sofisticados, mas também eficazes, confiáveis e capazes de gerar valor real.

3. Abordagem Proposta

O principal objetivo deste trabalho é analisar a performance de modelos preditivos aplicados ao setor de consumo energético residencial, dessa forma foi utilizado o *Dataset* de consumo de energia elétrica de domicílios individuais (HEBRAIL; BERARD, 2006) para treinar o modelo de consumo individual de energia elétrica residencial. Para a escolha dos modelos, foi pensado em sua aplicação de forma geral, embora o *dataset* apresentado não possua mais de uma variável, o LSTM é um modelo que captura dependências a longo prazo (padrões sazonais, feriados) e sua capacidade de aprender representações complexas e de longo prazo a torna mais robusta a imperfeições nos dados, resultando em previsões mais estáveis e precisas, já os outros modelos (BiLSTM, GRU, BiGRU) foram escolhidos por serem uma variação da LSTM ou modelos alternativos. Para a realização dos testes para os modelos foi utilizado um *notebook* e para o tratamento dos dados, foi utilizado o *google colab*, plataforma open source que permite a criação de notebooks utilizando-se da linguagem de programação *Python* e as bibliotecas necessárias. A Figura 5 nos mostra de forma resumida os processos que serão citados a frente.

Figura 5 - Fluxograma do experimento



Fonte: Autoria própria.

3.1 Ambiente de Desenvolvimento

Foi utilizada uma máquina de modelo Latitude 3410, produzido pela *Dell*, com processador *Intel(R) Core(TM) i7-10510U*, 16GB de memória RAM, com sistema operacional de 64 bits, edição Windows 11 Pro, versão 22H2 para utilização da ferramenta *open source Google Colab* não sendo necessário a instalação de ambiente virtual para a execução do código.

3.2 Dependências do Projeto

As três células iniciais do programa importam as bibliotecas necessárias para o desenvolvimento do projeto. Além das bibliotecas principais, existem mais as dependências, que são módulos e pacotes adicionais necessários para o funcionamento adequado das bibliotecas e destaca-se também a integração do google drive como repositório para os arquivos gerados.

3.3 Importação das fontes dos dados

Após a instalações realizadas, utiliza-se a biblioteca Pandas e zipfile, para importar e descompacta o arquivo da fonte selecionando apenas as colunas necessárias e as transformando em uma tabela. Portanto, o programa converte os dados da coluna “*Datetime*” que são amostragens por minutos as agrupando as transformando em amostragens horárias, assim podemos obter informações agregando os dados no tempo.

3.4 Limpeza dos dados e Funções

Nesta seção, são descritos os passos de limpeza e adequação dos dados, que é fundamental para melhor *performance* dos modelos escolhidos. Para o *dataset* em específico não foi necessário o tratamento de valores faltantes, porém, como passo fundamental e caso necessário aplicou-se três etapas para o tratamento desses valores no código: 1. A inserção de valores sazonais utilizando o dado da semana anterior; 2. interpolação temporal, caso houver valores faltantes nas primeiras 168 horas ou se o valor da semana anterior também ser faltante e 3. Verificação se ainda há valores faltantes, para assim removê-los manualmente ou optar pela remoção da coluna. Como se trata de medições de um ambiente que apresenta equipamentos eletrônicos, foram mantidos os valores de 0.00, pois contam como medição dos aparelhos desligados, ou seja, a não utilização do ambiente ou falta de energia.

Após isso, foi feito o tratamento de outliers inserindo os valores de borda (limite inferior e superior) e caso não fossem inseridos seria feito pelo método de Intervalo Interquartil (IQR). Para os dados em questão utilizou-se valores inseridos manualmente após avaliação dos gráficos de dispersão das colunas.

Feito o pré-processamento dos dados foram criadas as funções de métrica e criação do *dataset*. A última função criada é a de execução do modelo escolhido, criando a faixa de treinamento e teste, definindo as entradas, o valor da época, número de camadas, número de neurônios por camada, função de ativação, bem como o cálculo das métricas e de perda do modelo.

3.5 Execução e geração dos resultados

Nesta etapa, temos a execução do código para as duas variáveis de predição, utilizando a proporção de 80% de dados para treino e 20% para teste (ROSHAN et al., 2022), nos trazendo os resultados gerados e armazenando-os em variáveis e com essas informações foram gerados os gráficos e tabelas que serão discutidos na próxima seção.

4. RESULTADOS E DISCUSSÕES

Nesta seção são apresentados os resultados da execução do programa desenvolvido.

Foram executados os modelos de forma separada para cada *input_size* (10, 15, 20, 25 e 30). Para uma análise mais detalhada serão tratados em tópicos cada modelo e ao fim as considerações finais. Como as plotagens gráficas não trazem grades discrepâncias para as diferentes entradas, será focado nas análises métricas.

4.1 Modelo *Long Short-Term Memory* (LSTM)

4.1.1 Sub-medição 2

- **Métricas de Desempenho:**

Com base nas métricas geradas pelos inputs notamos uma certa consistência nos resultados, demonstrando ligeiramente um maior desempenho com o input = 10 como definido abaixo.

Tabela 1 - Métricas de Teste LSTM Sub2.

Input	MAE [Wh]	RMSE [Wh]	SMAPE [%]
10	1,26	3,09	125,72
15	1,42	3,11	127,67
20	1,37	3,11	131,55
25	1,31	3,1	126,48
30	1,49	3,71	119,29

Tabela 2 - Métricas de Treino LSTM Sub2.

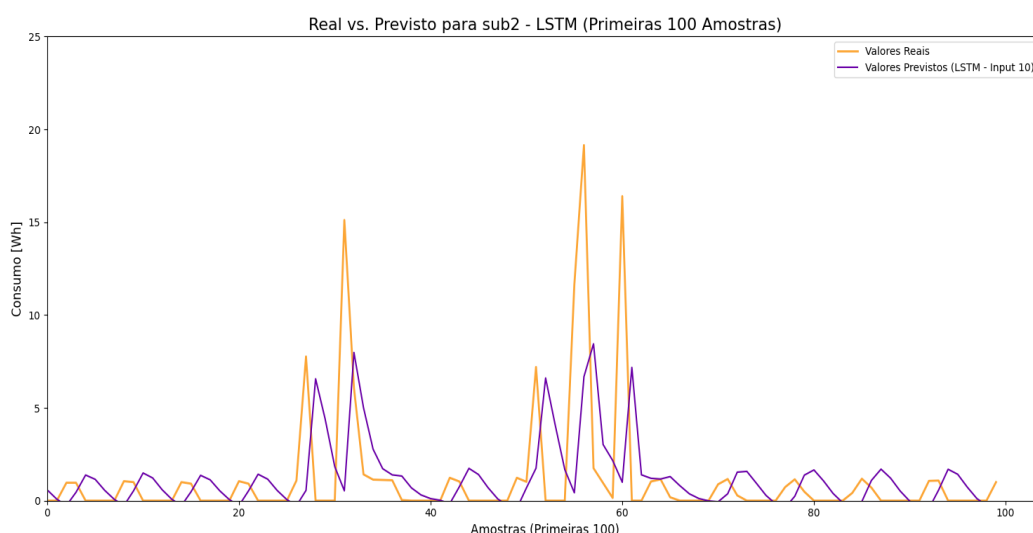
Input	MAE [Wh]	RMSE [Wh]	SMAPE
10	1,41	3,75	117,36
15	1,49	3,70	117,19
20	1,44	3,70	119,30
25	1,41	3,74	117,57
30	1,49	3,71	119,29

Mesmo com diferentes tamanhos de janela de entrada (10 a 30), o modelo LSTM teve um desempenho consistente no teste para a *feature* analisada, que as dependências mais relevantes para a previsão já são capturadas mesmo com janelas menores e janelas maiores não adicionavam contexto significativo.

Curiosamente, o modelo apresentou melhores resultados de MAE e RMSE no conjunto de teste do que no de treino, mas piores para SMAPE. dados ligeiramente diferentes buição de

dados ligeiramente diferente entre treino e teste, onde o teste era "mais fácil" para métricas absolutas, ou que o conjunto de treino tinha mais *outliers* que impactavam o RMSE. O SMAPE (erro percentual) maior no teste é mais esperado e sugere que, embora os erros absolutos (MAE, RMSE) fossem baixos, o modelo teve dificuldade em prever a magnitude percentual correta, especialmente por conta da presença de muitos zeros nos dados (cerca de um terço do *dataset*), o que pode inflacionar o erro percentual. Essa discrepância é visível nos gráficos de consumo Real vs. Previsto da Figura 6. Como os resultados foram similares entre os diferentes tamanhos de janela, apenas um gráfico foi apresentado.

Figura 6 - Amostra do consumo [Wh] Real vs. Previsto para input 10 [Sub2 LSTM].

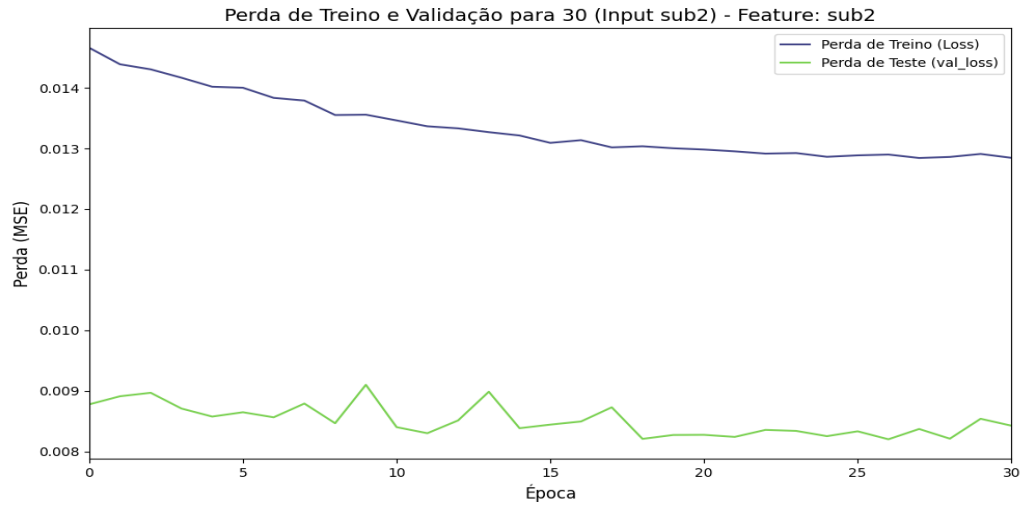


Fonte: Autoria própria.

- **Perdas do Modelo:**

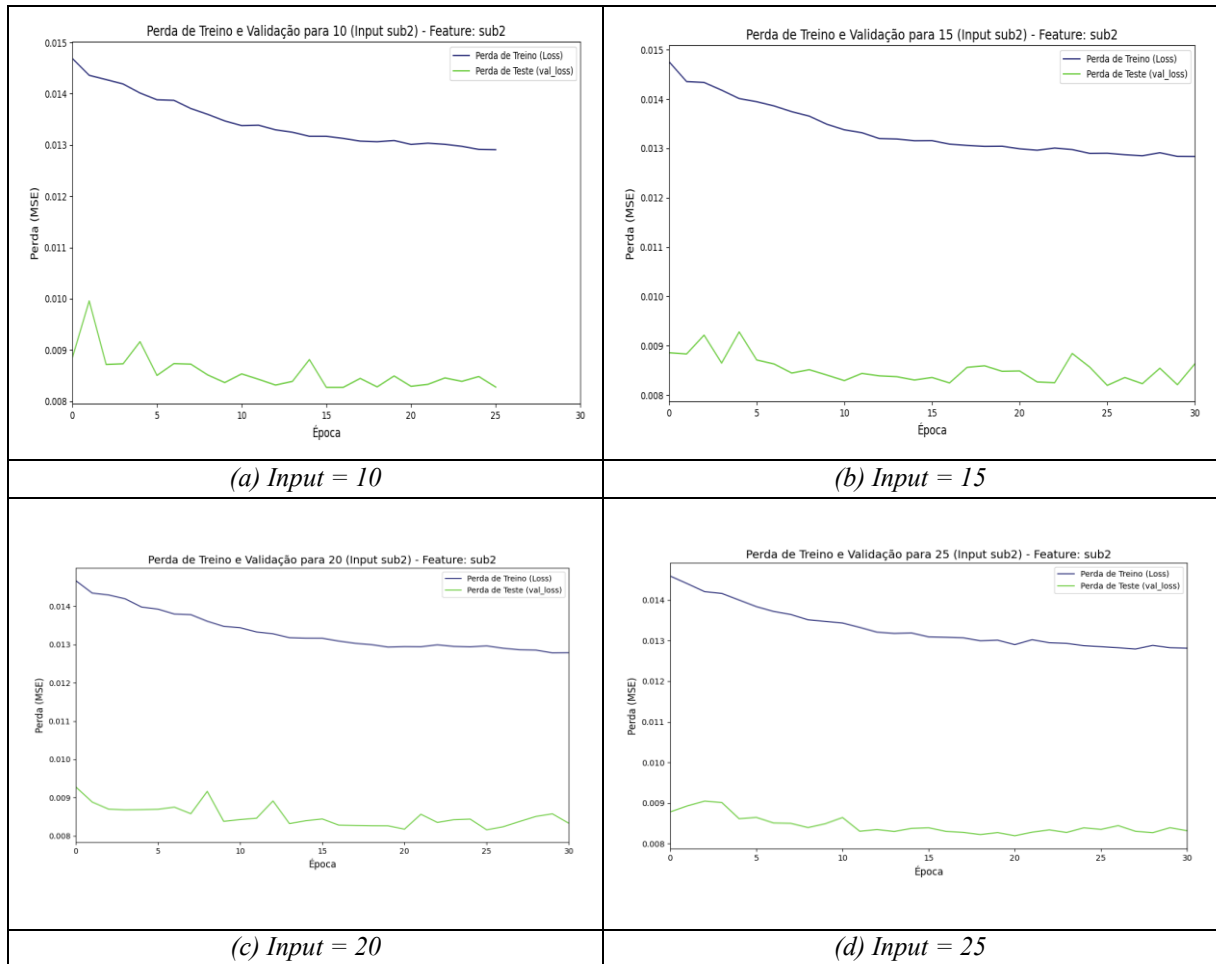
Olhando para as perdas do modelo para os inputs vemos em geral os valores de treino mostram uma tendencia a convergência demonstrando que o modelo está aprendendo os padrões nos dados de treinamento. Enquanto no teste temos uma tendência geral de queda, e apesar das oscilações, não há um aumento sustentado e significativo na perda. Portanto o modelo parece estar aprendendo bem e generalizando razoavelmente para os dados de treino, sem sinais claros de *overfitting*. Podemos observar melhor essas nuances na Figura 7, trazendo também para comparação as outras entradas nas Figura 8.

Figura 7 - Perda de treino e teste em $(Wh)^2$ para input 30 [Sub2 LSTM].



Fonte: Autoria própria.

Figura 8 - Perda de treino e teste em $(Wh)^2$ [Sub2 LSTM].



Fonte: Autoria própria.

4.1.2 Sub-medição 3

- **Métricas de Desempenho:**

Com base nas métricas geradas pelos inputs notamos uma clara tendência de melhoria no desempenho do modelo no conjunto de teste à medida que o tamanho da janela de *input* aumenta, como vemos no *input*=25 e *input*=30. Isso mostra que “sub3” possui dependências de longo prazo ou padrões complexos que se beneficiam de um contexto histórico mais amplo. O modelo LSTM, sendo eficaz na captura dessas dependências, consegue extrair informações mais valiosas quando recebe sequências de entrada mais longas. Isso é um forte indício de que a informação contida em um histórico maior é relevante para a sua previsão.

Tabela 3 - Métricas de Teste LSTM Sub3.

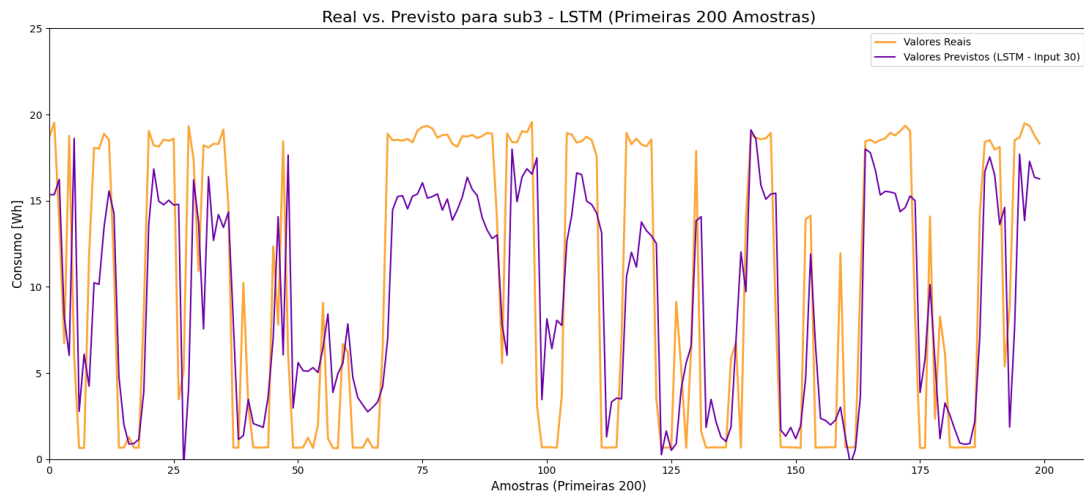
Input	MAE [Wh]	RMSE [Wh]	SMAPE [%]
10	4,10	5,49	79,83
15	4,10	5,43	78,43
20	4,00	5,38	77,52
25	3,89	5,19	75,05
30	3,74	5,06	75,43

Tabela 4 - Métricas de Treino LSTM Sub3.

Input	MAE [Wh]	RMSE [Wh]	SMAPE [%]
10	4,27	5,46	107,73
15	4,13	5,34	106,16
20	4,01	5,28	104,41
25	3,87	5,09	105,07
30	3,48	4,77	102,76

Ao contrário da análise anterior da *feature* "sub2", para a *feature* "sub3" observamos uma relação mais esperada e saudável entre as métricas de treino e teste em que as métricas de teste são consistentemente maiores do que as métricas de treino para todas as configurações de input. Aqui o SMAPE de teste é menor que o de treino, o que é o comportamento esperado para o *overfitting*. O desempenho geral é razoável, mas ainda há espaço para otimização, principalmente na redução da lacuna de generalização observada entre o treino e o teste. A Figura 9 nos traz a visão do gráfico de consumo Real vs. Previsto que não possui grandes variações entre as entradas.

Figura 9 - Amostra do consumo [Wh] Real vs. Previsto para input 30 [Sub;3 LSTM].

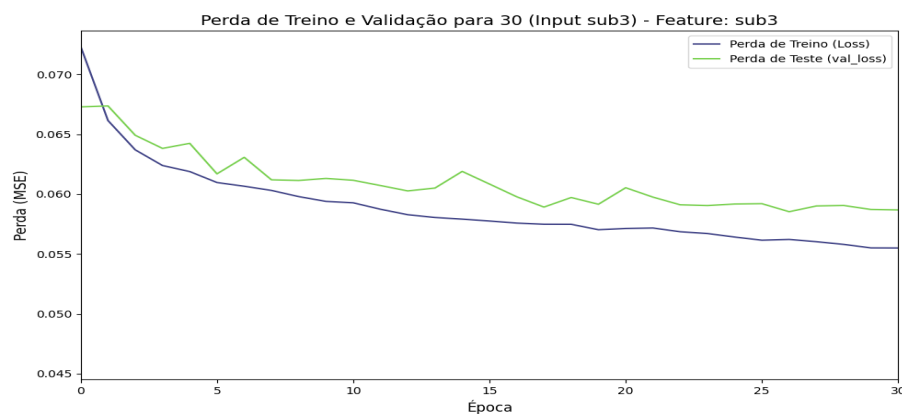


Fonte: Autoria própria.

- **Perdas do Modelo:**

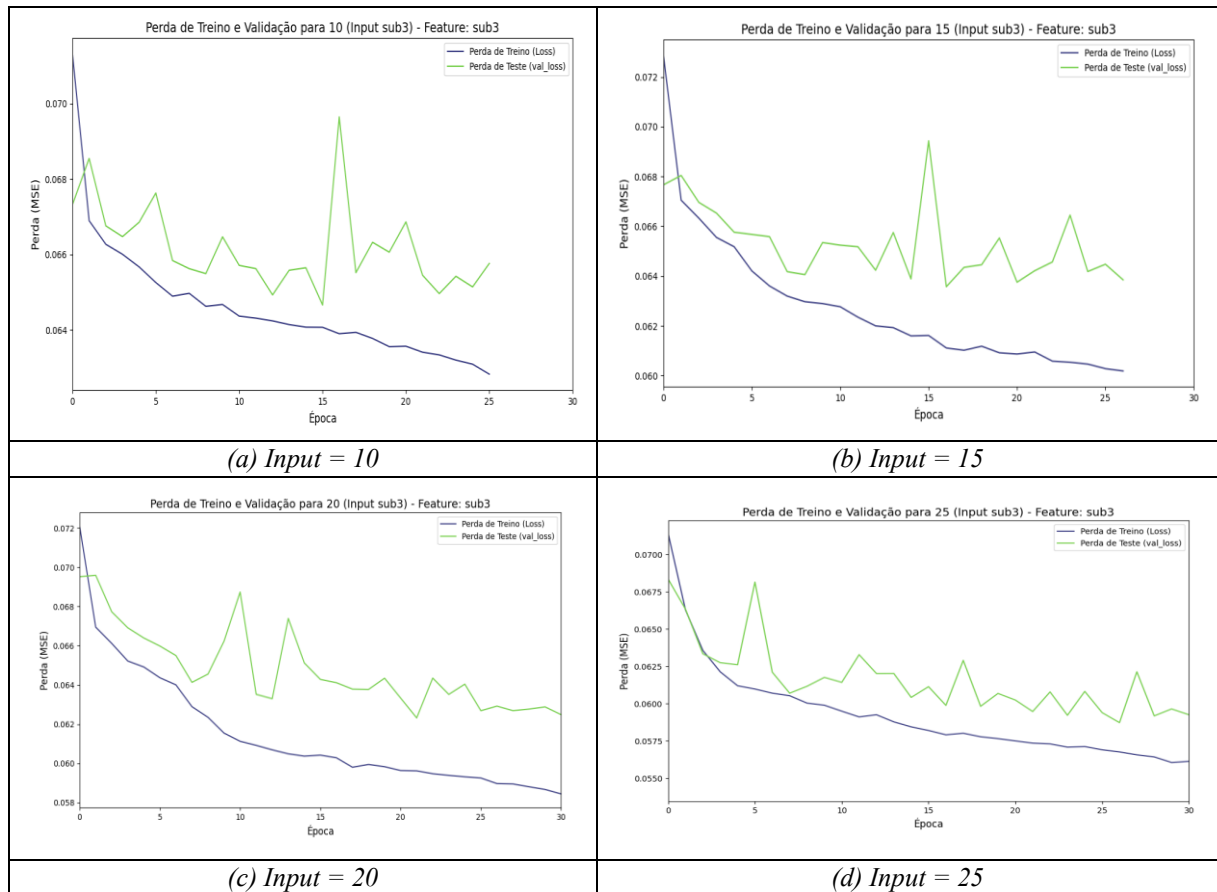
Para os diferentes tamanhos de janela de entrada (10, 15, 20, 25 e 30), observamos indícios de *overfitting* a partir de um certo ponto no treinamento. A perda de treinamento geralmente continuava a diminuir, enquanto a perda de teste indica que o modelo alcançou sua melhor capacidade de generalização e que o treinamento adicional não traria benefícios significativos, podendo até prejudicar o desempenho em dados novos, ocorrendo em pontos diferentes para cada entrada. Em geral, apesar das variações no ponto de início do *overfitting*, a observação da saturação da perda de teste para diferentes inputs sugere que o modelo se ajusta bem aos padrões dos dados, e a otimização do número de épocas de treinamento é crucial para maximizar a generalização. Podemos observar melhor essas nuances na Figura 10, trazendo também para comparação as outras entradas nas Figura 11.

Figura 10 - Perda de treino e teste em $(Wh)^2$ para input 30 [Sub3 LSTM].



Fonte: Autoria própria.

Figura 11 - Perda de treino e teste em $(Wh)^2$ [Sub3 LSTM].



Fonte: Autoria própria.

4.2 Modelo *Bidirectional Long Short-Term Memory* (BiLSTM)

4.2.1 Sub-medição 2

- **Métricas de Desempenho:**

O desempenho do modelo BiLSTM, assim como no LSTM, no conjunto de teste não é altamente sensível ao tamanho da janela de entrada dentro do intervalo testado (10 a 30). Isso pode indicar que as dependências mais relevantes para a previsão da *feature* são capturadas mesmo com janelas menores, ou que aumentar a janela não adiciona informações contextuais significativamente mais úteis para essa feature específica.

Tabela 5 - Métricas de Teste BiLSTM Sub2.

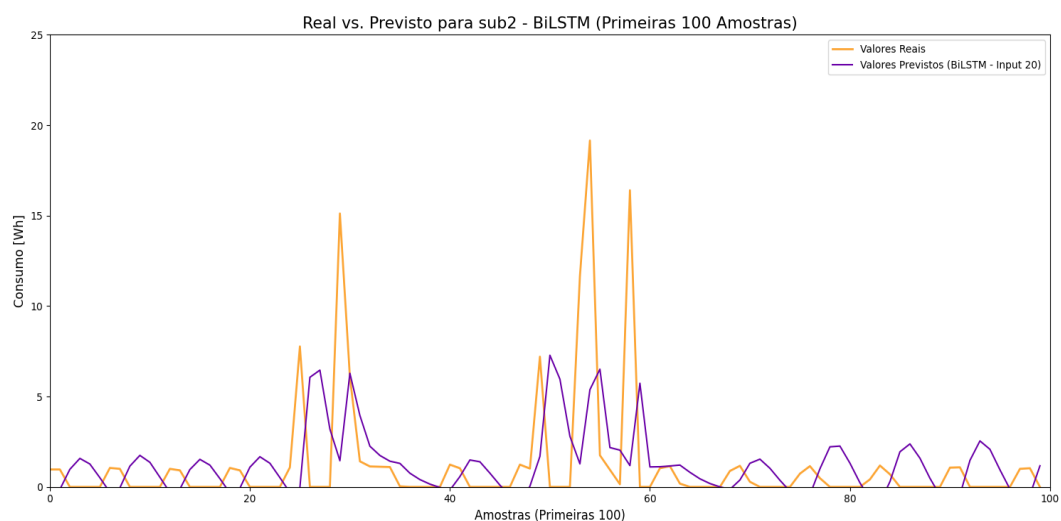
Input	MAE [Wh]	RMSE [Wh]	SMAPE
10	1,30	3,10	126,16
15	1,38	3,09	127,17
20	1,33	3,11	127,19
25	1,41	3,11	134,88
30	1,31	3,10	134,74

Tabela 6 - Métricas de Treino BiLSTM Sub2.

Input	MAE [Wh]	RMSE [Wh]	SMAPE
10	1,39	3,75	118,74
15	1,49	3,71	119,67
20	1,42	3,74	116,93
25	1,44	3,70	121,13
30	1,43	3,73	130,73

Os resultados do modelo apresentaram um padrão incomum nas métricas de desempenho. Observamos um desempenho melhor nas métricas de teste (MAE e RMSE) em comparação com as métricas de treino, enquanto o SMAPE foi pior no teste indicando uma menor capacidade de generalização do modelo para dados novos sugerindo que o modelo tem dificuldade em prever a magnitude percentual correta mostrado na Figura 12. A aparente discrepância para MAE e RMSE pode ser atribuída a algumas possibilidades como diferenças na distribuição de dados ou ruído no conjunto de treino caso o conjunto de treino tiver mais *outliers* ou *outliers* de maior impacto, isso justificaria um RMSE mais alto no treino.

Figura 12 - Amostra do consumo [Wh] Real vs. Previsto para input 20 [Sub2 BiLSTM].



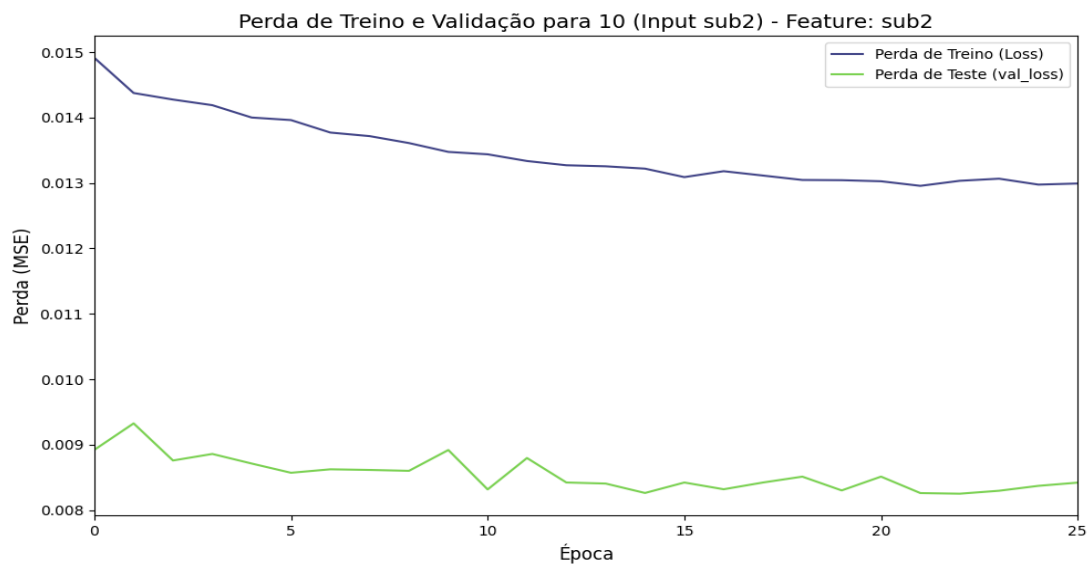
Fonte: Autoria própria.

- **Perdas do Modelo:**

Em todos os casos de input, a perda de treinamento diminui de forma contínua, indicando que o modelo está efetivamente aprendendo os padrões dos dados. Paralelamente, a perda de teste mantém-se consistentemente menor que a perda de treinamento ao longo de todas as etapas e épocas. Embora a perda de teste possa apresentar algumas oscilações, não há um aumento sustentado ou significativo, nem sinais claros de *overfitting* (sobreajuste do modelo aos dados de treino). Em etapas posteriores na Figura 14 (c) e (d) as oscilações na perda de teste se mostram mais suavizadas, e a perda de treinamento demonstra uma tendência gradual à convergência.

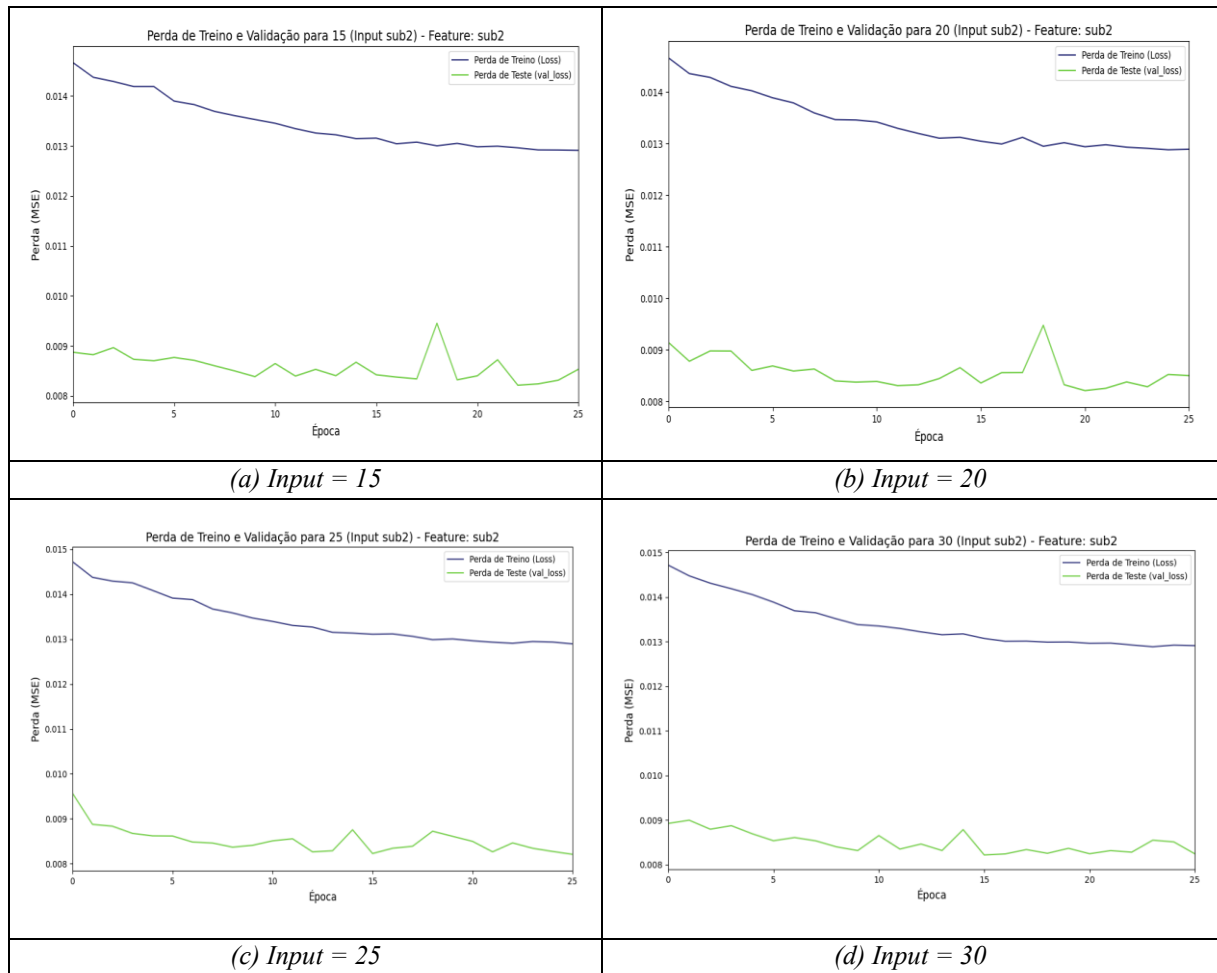
Em suma, o modelo demonstra um bom aprendizado sem sobreajuste evidente, com a performance no conjunto de teste sendo consistentemente superior podendo ser constatadas na Figura 13 e Figura 14.

Figura 13 - Perda de treino e teste em $(Wh)^2$ para input 10 [Sub2 BiLSTM].



Fonte: Autoria própria.

Figura 14 - Perda de treino e teste em $(Wh)^2$ [Sub2 BiLSTM].



Fonte: Autoria própria.

4.2.2 Sub-medição 3

- **Métricas de Desempenho:**

O desempenho do modelo no conjunto de teste melhora diretamente com o aumento do tamanho da janela de input. Isso sugere que há dependências de longo prazo ou padrões complexos que são melhor capturados por um histórico mais amplo, o que o modelo se mostrou eficaz em identificar. Ou seja, um contexto histórico maior é crucial para prever "sub3".

Tabela 7 - Métricas de Teste BiLSTM Sub3.

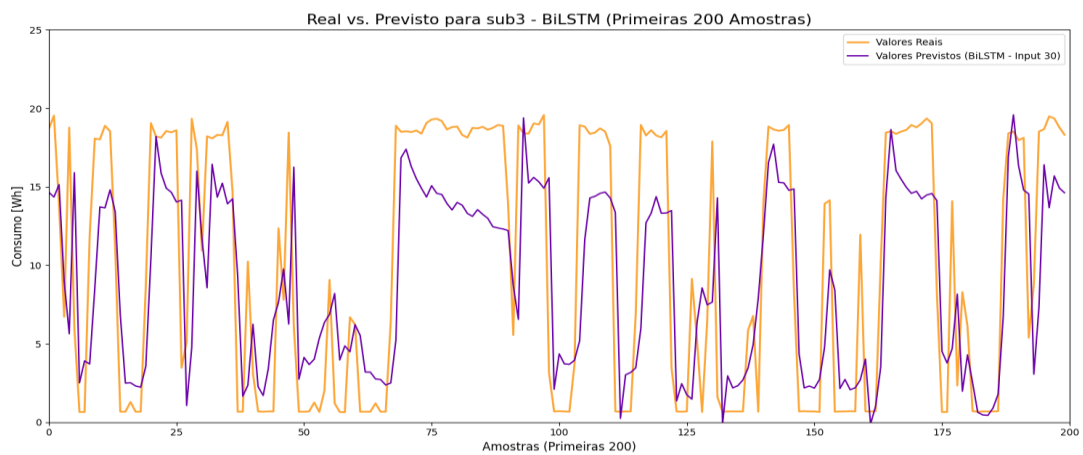
Input	MAE [Wh]	RMSE [Wh]	SMAPE
10	4,05	5,17	82,69
15	4,03	5,42	78,33
20	3,83	5,27	74,78
25	3,85	5,18	75,55
30	3,82	5,18	74,57

Tabela 8 - Métricas de Treino BiLSTM Sub3.

Input	MAE [Wh]	RMSE [Wh]	SMAPE
10	3,77	4,97	108,69
15	4,06	5,33	106,19
20	3,68	4,99	103,08
25	3,81	5,06	103,25
30	3,82	5,08	103,47

Diferentemente da *feature* "sub2", para "sub3" observamos uma relação inversa entre as métricas de treino e teste: as métricas de teste (MAE e RMSE) são consistentemente maiores que as de treino, enquanto o SMAPE é relativamente menor. Esse comportamento indica um possível *overfitting*. Embora o desempenho geral seja razoável, há espaço para otimização, focando na redução da lacuna de generalização entre os conjuntos de treino e teste. Podemos ver uma amostra na Figura 15.

Figura 15 - Amostra do consumo [Wh] Real vs. Previsto para input 30 [Sub3 BiLSTM].



Fonte: Autoria própria.

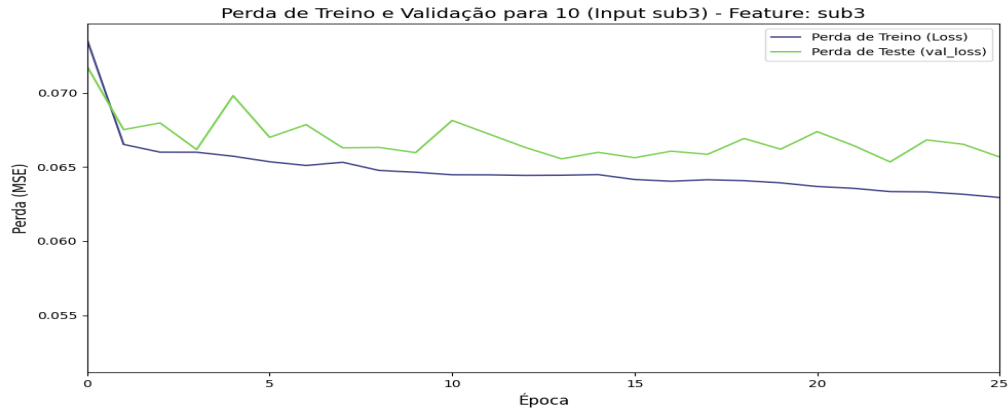
• Perdas do Modelo

A análise da perda do modelo em diferentes *inputs* (10, 15, 20, 25 e 30) revela um padrão claro: a perda de teste é consistentemente maior que a perda de treinamento em todas as etapas, o que é um comportamento esperado em modelos de *Machine Learning*.

No entanto, à medida que o input aumenta, surgem sinais crescentes de *overfitting*. No input 10 como visto na Figura 16, o modelo ainda não mostra indícios significativos de sobreajuste. Mas a partir do input 15 e mais acentuadamente nos inputs 20 e 30 na Figura 17, a perda de validação começa a oscilar e, por vezes, aumentar, enquanto a perda de treino continua

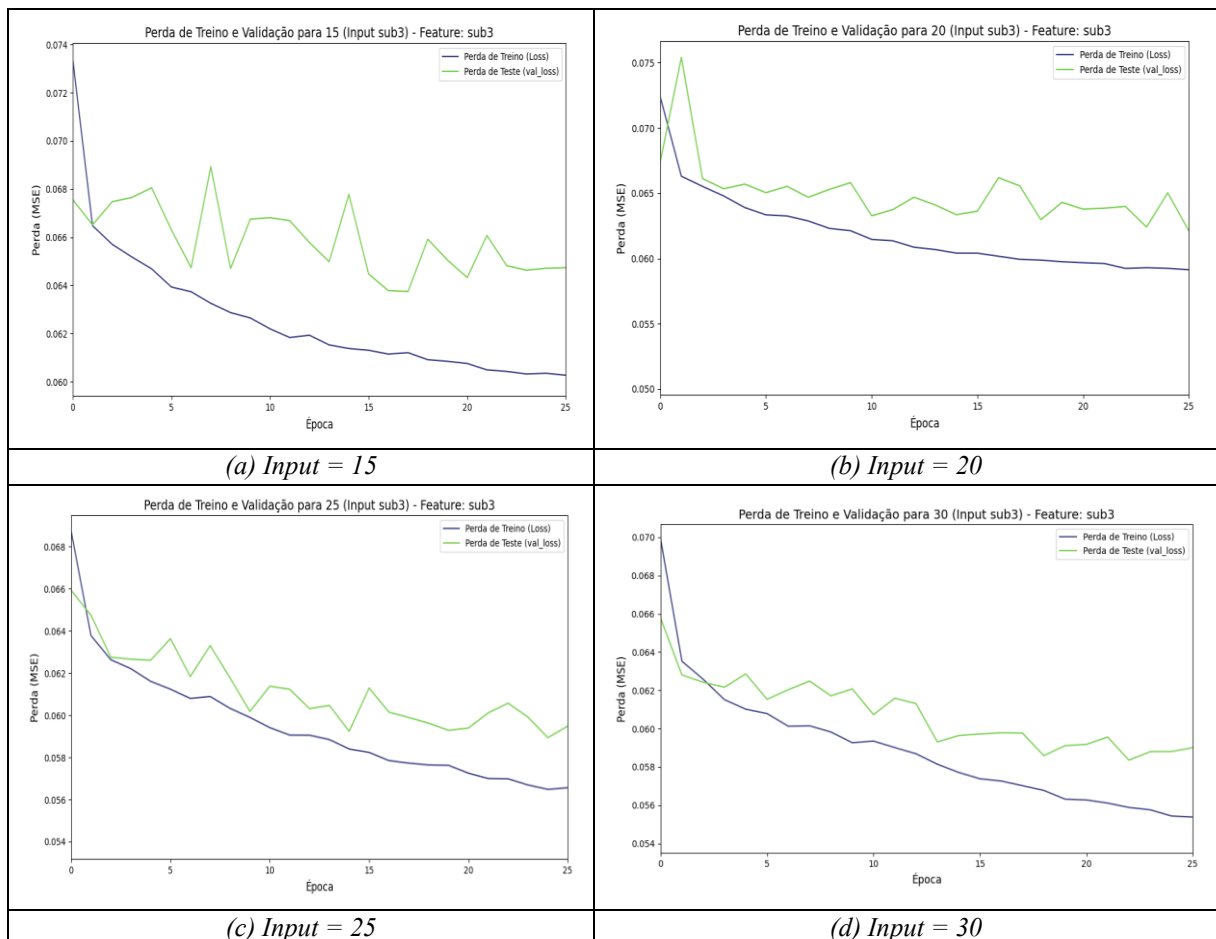
a diminuir. Isso indica que o modelo começa a memorizar os dados de treino em detrimento da generalização, indicando que o melhor seria uma parada antecipada do treinamento, a fim de evitar o *overfitting*.

Figura 16 - Perda de treino e teste em $(Wh)^2$ para input 10 [Sub3 BiLSTM].



Fonte: Autoria própria

Figura 17 - Perda de treino e teste em $(Wh)^2$ [Sub3 BiLSTM].



Fonte: Autoria própria.

4.3 Modelo *Gated Recurrent Unit* (GRU)

4.3.1 Sub-medição 2

- **Métricas de Desempenho:**

Com base nas métricas geradas pelos *inputs* notamos uma certa consistência nos resultados, demonstrando ligeiramente um maior desempenho com o *input* = 15 e para *input* = 25 como definido abaixo.

Tabela 9 - Métricas de Teste GRU Sub2.

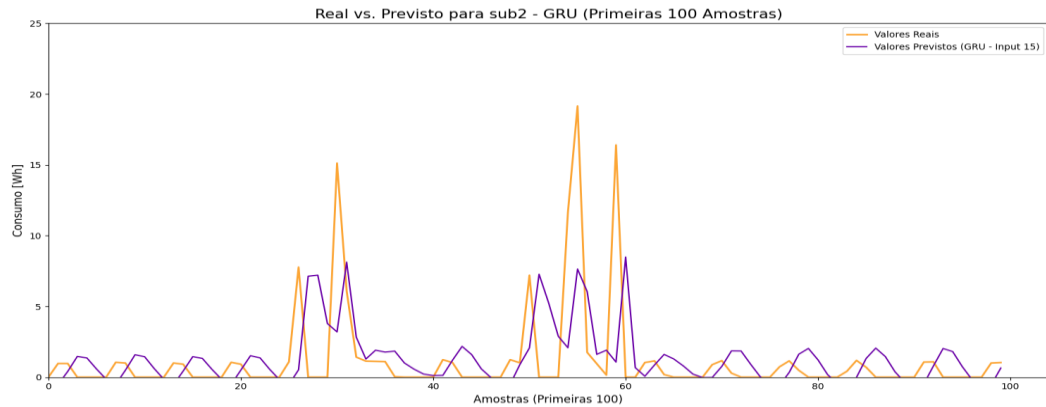
Input	MAE [Wh]	RMSE [Wh]	SMAPE
10	1,45	3,16	131,96
15	1,32	3,08	131,46
20	1,32	3,09	130,02
25	1,34	3,09	126,92
30	1,34	3,10	133,86

Tabela 10 - Métricas de Treino GRU Sub2.

Input	MAE [Wh]	RMSE [Wh]	SMAPE
10	1,41	3,71	117,12
15	1,42	3,67	122,38
20	1,43	3,74	120,63
25	1,42	3,72	115,52
30	1,39	3,69	121,75

O modelo GRU para a *feature* mostrou que seu desempenho no teste não muda muito com o tamanho da janela de entrada (de 10 a 30) trazendo uma visão gráfica na Figura 18, não demonstrando grandes mudanças em relação aos outros modelos. Demonstrando um padrão incomum: MAE e RMSE foram melhores no teste, mas o SMAPE foi pior. Isso indica que o modelo tem dificuldade em prever a magnitude exata dos dados, mesmo que as previsões absolutas (MAE e RMSE) pareçam melhores no teste.

Figura 18 - Amostra do consumo [Wh] Real vs. Previsto para input 15 [Sub2 GRU].

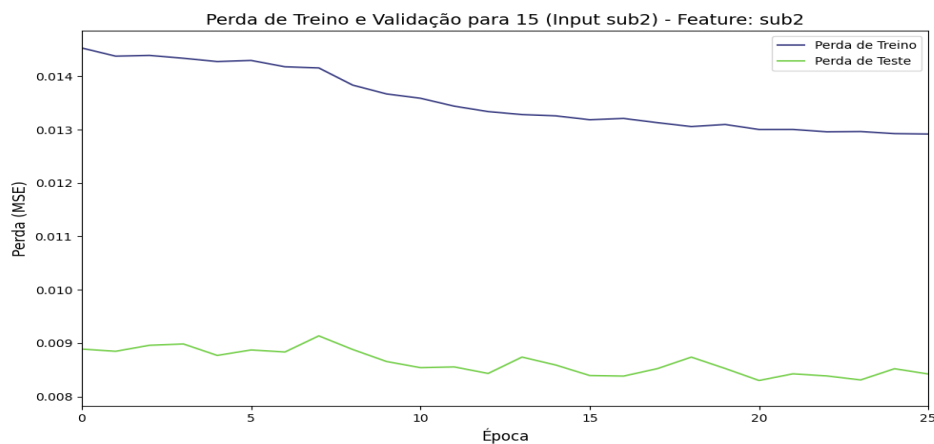


Fonte: Autoria própria.

- **Perdas do Modelo:**

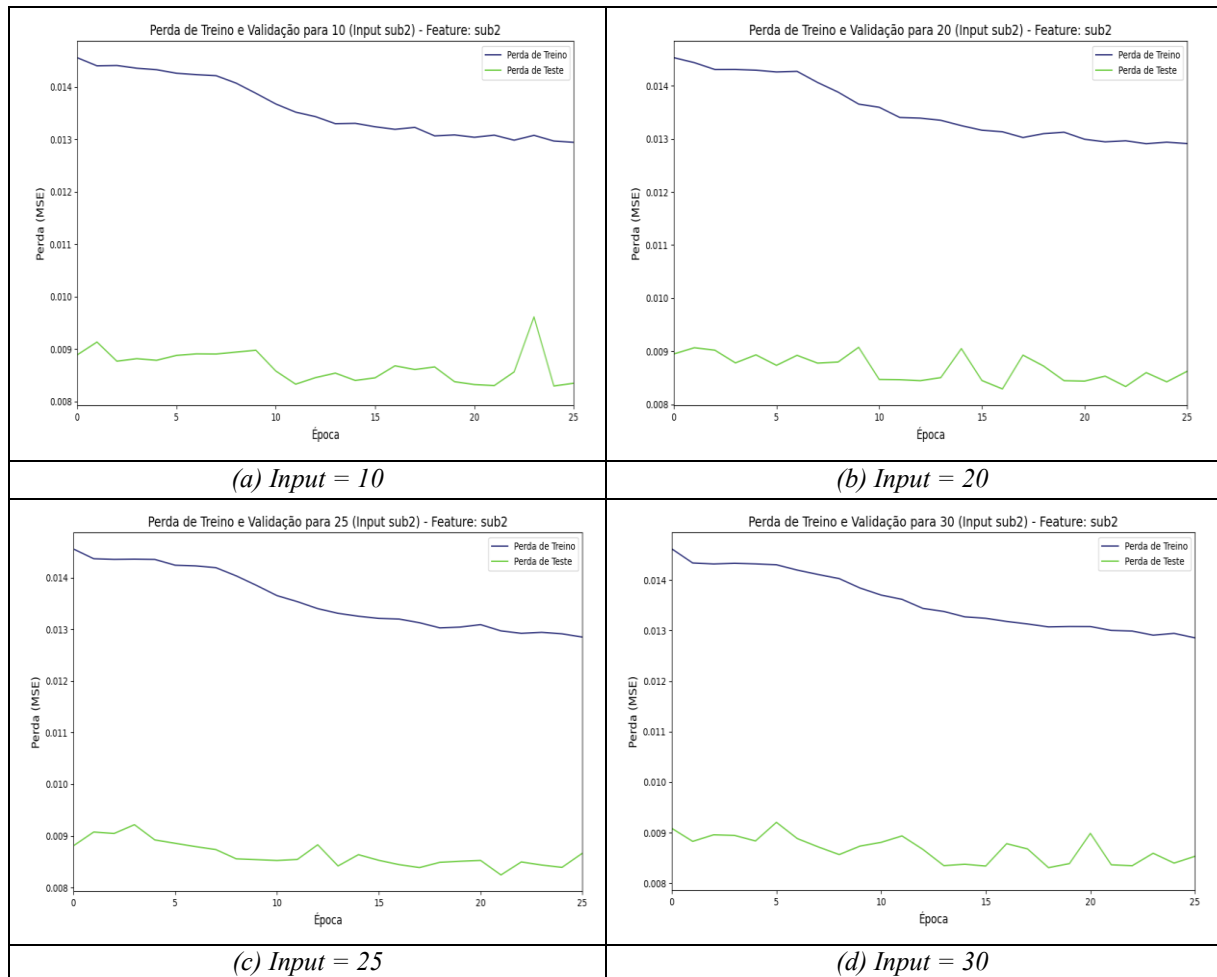
A análise da perda do modelo para os diferentes tamanhos de *input* revela um padrão consistente de aprendizado eficaz e boa generalização. Em todos os cenários, a perda de treinamento diminui de forma consistente, o que indica que o modelo está aprendendo os padrões nos dados de treino. A perda de teste também mostra uma tendência geral de queda e apesar de algumas oscilações, mantém-se consistentemente menor que a perda de treinamento à medida que o input aumenta. Em inputs maiores (25 e 30), as oscilações na perda de teste são mais suaves, e a perda de treinamento continua a diminuir gradualmente, tendendo à convergência, acompanhamos os gráficos na Figura 19 e Figura 20.

Figura 19 - Perda de treino e teste em $(Wh)^2$ para input 15 [Sub2 GRU].



Fonte: Autoria própria.

Figura 20 - Perda de treino e teste em $(Wh)^2$ [Sub2 GRU].



Fonte: Autoria própria.

4.3.2 Sub-medição 3

- **Métricas de Desempenho:**

As métricas do modelo GRU para a feature "sub3" revelam padrões distintos entre treino e teste. O SMAPE de treinamento é consistentemente maior que o de teste em todas as configurações de input. Já para MAE e RMSE, as métricas de treino e teste são mais próximas, com a perda de treino variando entre ser marginalmente maior ou menor em comparação com a perda de teste. Podemos acompanhar na Figura 21 o consumo [Wh] Real vs. Previsto que se mostra semelhante aos anteriores.

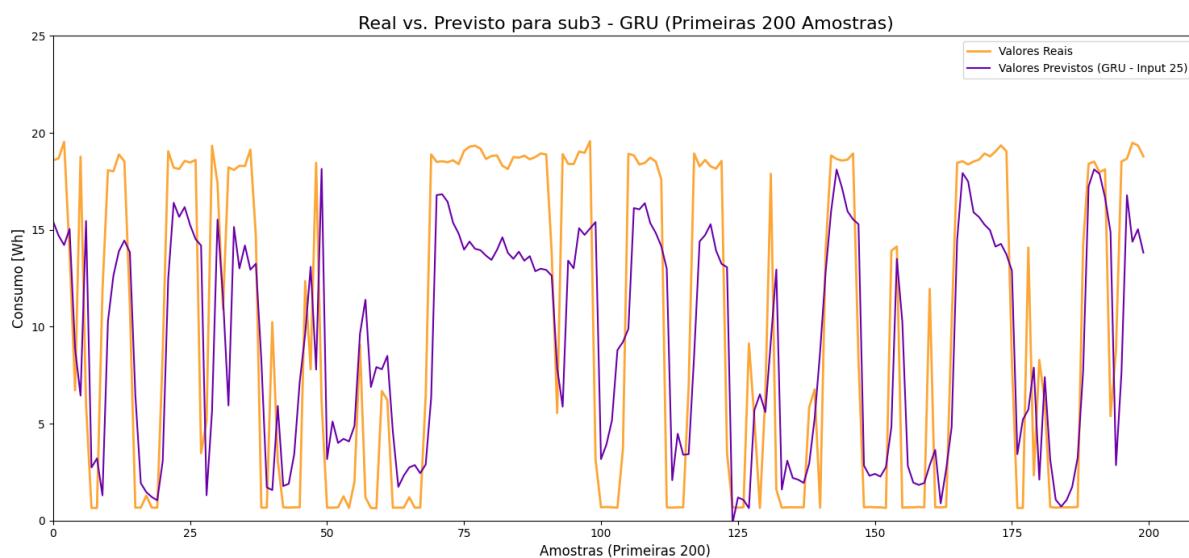
Tabela 11 - Métricas de Teste GRU Sub3.

Input	MAE [Wh]	RMSE [Wh]	SMAPE
10	4,15	5,49	79,85
15	4,03	5,32	78,76
20	3,93	5,29	75,90
25	3,79	5,10	74,63
30	3,78	5,11	74,31

Tabela 12 - Métricas de Treino GRU Sub3.

Input	MAE [Wh]	RMSE [Wh]	SMAPE
10	4,27	5,44	108,11
15	4,05	5,22	105,46
20	3,95	5,25	104,57
25	3,80	5,00	102,67
30	3,89	5,10	103,86

Figura 21 - Amostra do consumo [Wh] Real vs. Previsto para input 25 [Sub3 GRU].



Fonte: Autoria própria.

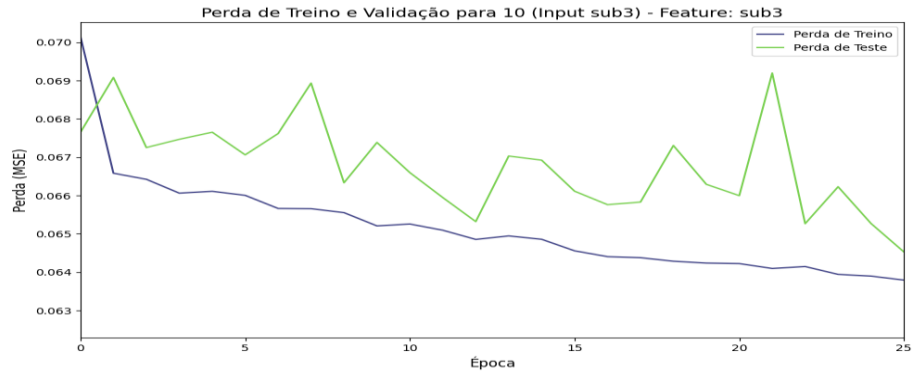
- **Perdas do Modelo:**

Os dados analisados consistentemente demonstram que a perda de teste é maior que a perda de treinamento em todas as épocas observadas. Para o *input* 10, a perda de treinamento exibe sinais de convergência, com uma desaceleração em sua diminuição nas épocas finais, e a perda de teste também parece estar se aproximando de um ponto de saturação.

Nos *inputs* 15, 20 e 25, não há indícios de *overfitting* durante os períodos analisados. Finalmente, para o *input* 30, ambos os valores de perda (treinamento e teste) mostram sinais de

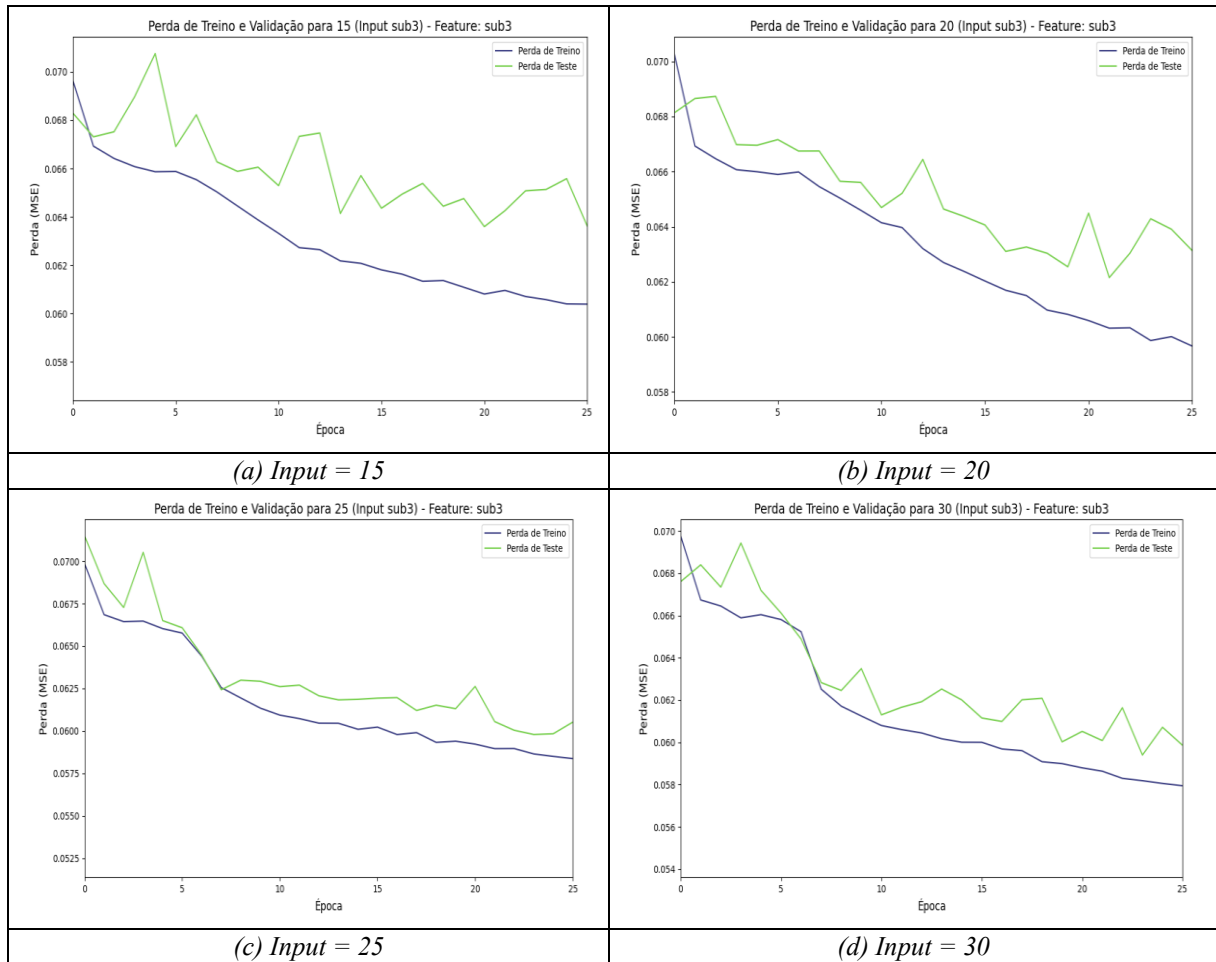
convergência, e, de forma consistente com os inputs anteriores, não há sinais de que o modelo esteja começando a apresentar problemas de overfitting para as épocas avaliadas, como podemos ver nas Figura 22 e Figura 23.

Figura 22 - Perda de treino e teste em $(Wh)^2$ para input 10 [Sub3 GRU].



Fonte: Autoria própria.

Figura 23 - Perda de treino e teste em $(Wh)^2$ [Sub3 GRU].



Fonte: Autoria própria.

4.4 Modelo *Bidirectional Gated Recurrent Unit* (BiGRU)

4.4.1 Sub-medição 2

- Métricas de desempenho

Com base nas métricas geradas pelos inputs notamos uma certa consistência nos resultados, demonstrando ligeiramente um maior desempenho com o *input* = 25 e para *input* = 30 como definido abaixo, tendo o gráfico de 25 expresso na Figura 24.

Tabela 13 - Métricas de Teste BiGRU Sub2.

Input	MAE [Wh]	RMSE [Wh]	SMAPE
10	1,36	3,10	131,25
15	1,49	3,13	128,84
20	1,37	3,10	127,00
25	1,35	3,07	125,31
30	1,34	3,08	133,72

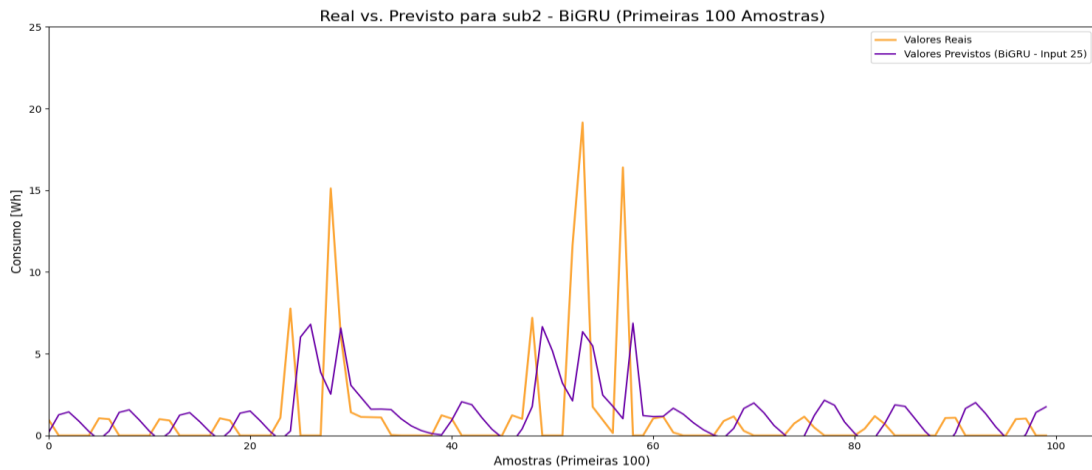
Tabela 14 - Métricas de Treino BiGRU Sub2.

Input	MAE [Wh]	RMSE [Wh]	SMAPE
10	1,45	3,67	122,70
15	1,52	3,69	117,71
20	1,43	3,70	116,34
25	1,48	3,71	116,63
30	1,42	3,69	122,98

O desempenho do modelo BiGRU, se assemelha aos anteriores citados, no conjunto de teste não é altamente sensível ao tamanho da janela de entrada dentro do intervalo testado (10 a 30). Isso pode indicar que as dependências mais relevantes para a previsão da *feature* são capturadas mesmo com janelas menores, ou que aumentar a janela não adiciona informações contextuais significativamente mais úteis para essa *feature* específica.

Também se nota uma relação entre as métricas de treino e teste incomum em que obtemos melhores resultados no teste para MAE e RMSE, mas piores para SMAPE, demonstrando sua dificuldade em igualar a magnitude dos dados estudados.

Figura 24 - Amostra do consumo [Wh] Real vs. Previsto para input 25 [Sub2 BiGRU].

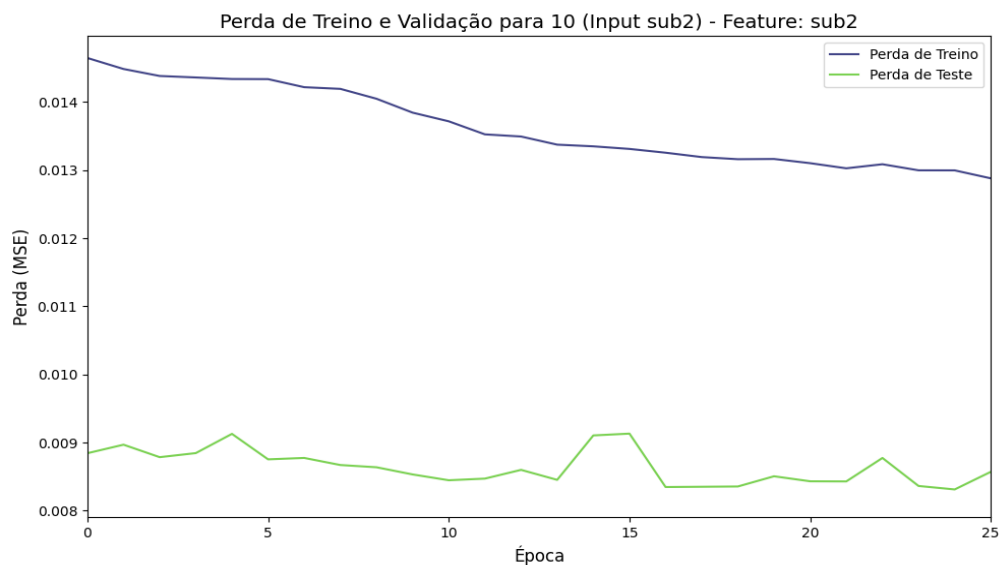


Fonte: Autoria própria.

- **Perdas do Modelo:**

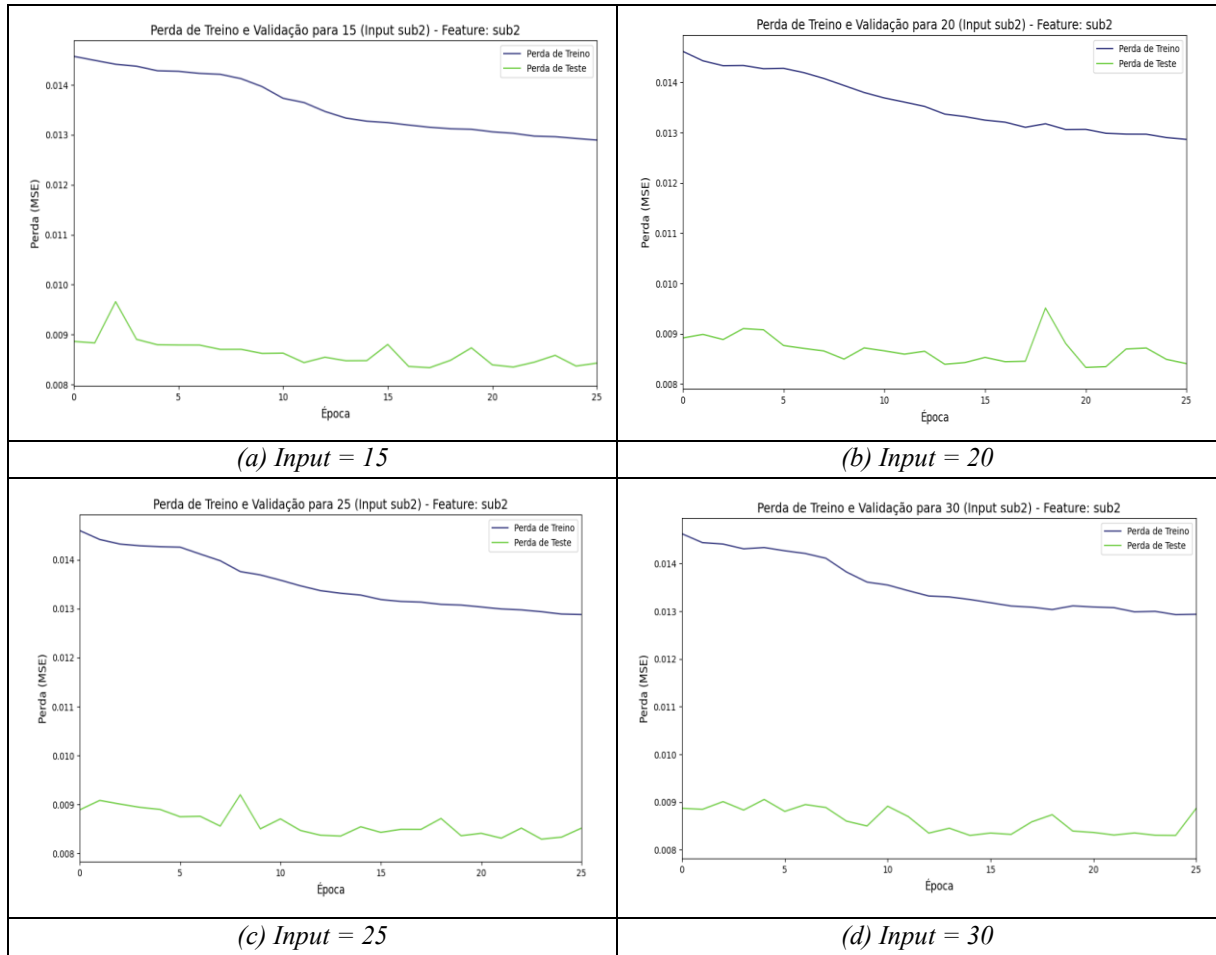
A análise dos dados de treino e teste em todos os inputs demonstra um padrão consistente de convergência do modelo e ausência de overfitting. Nos *inputs* 10, 15, 20, 25 e 30, mostrados na Figura 25 e Figura 26 observa-se que a perda de teste é consistentemente menor que a perda de treinamento. Embora a perda de teste possa apresentar algumas oscilações, não há um aumento sustentado e significativo que indique overfitting. A perda de treinamento, por sua vez, continua a diminuir de forma gradual, tendendo à convergência.

Figura 25 - Perda de treino e teste em $(Wh)^2$ para input 10 [Sub2 BiGRU].



Fonte: Autoria própria.

Figura 26 - Perda de treino e teste em $(Wh)^2$ [Sub2 BiGRU].



Fonte: Autoria própria.

4.4.2 Sub-medição 3

- **Métricas de Desempenho:**

Para a maioria das configurações, as métricas de teste (MAE, RMSE) são ligeiramente maiores do que as respectivas métricas de treino, o que é um comportamento esperado e indica que o modelo se ajusta melhor aos dados em que foi treinado, porém a relação entre SMAPE de treino e teste é incomum, com o teste apresentando valores menores, o que pode justificar uma investigação mais aprofundada dos conjuntos de dados ou da métrica em si, tendo o gráfico de $input = 20$ apresentado na Figura 27.

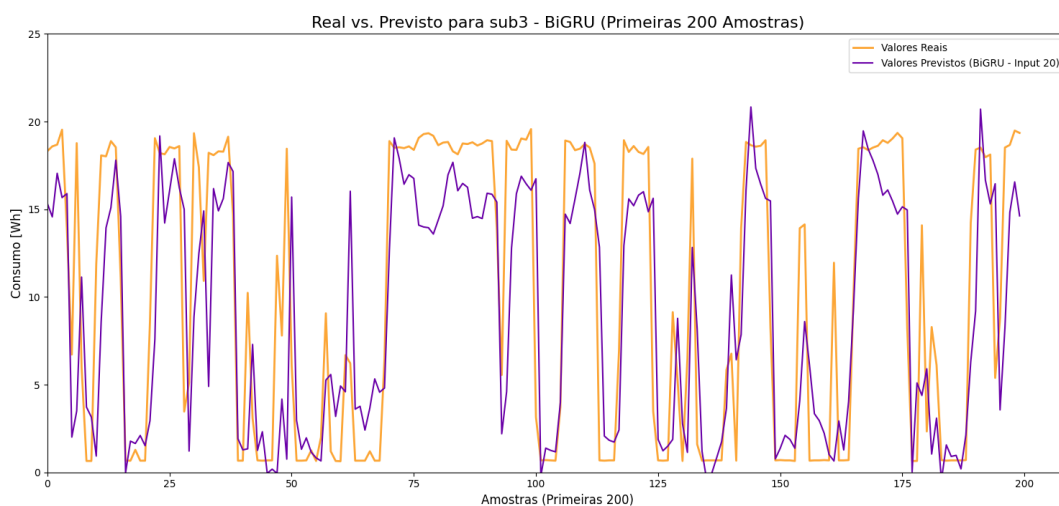
Tabela 15 - Métricas de Teste BiGRU Sub3.

Input	MAE [Wh]	RMSE [Wh]	SMAPE
10	3,90	5,17	79,73
15	4,03	5,36	77,72
20	3,57	5,00	73,16
25	3,59	4,87	73,71
30	3,80	5,12	73,76

Tabela 16 - Métricas de Treino BiGRU Sub3.

Input	MAE [Wh]	RMSE [Wh]	SMAPE
10	3,66	4,96	106,29
15	4,08	5,30	106,50
20	3,39	4,68	101,58
25	3,55	4,83	103,17
30	3,67	4,96	102,87

Figura 27 - Amostra do consumo [Wh] Real vs. Previsto para input 20 [Sub3 BiGRU].



Fonte: Autoria própria.

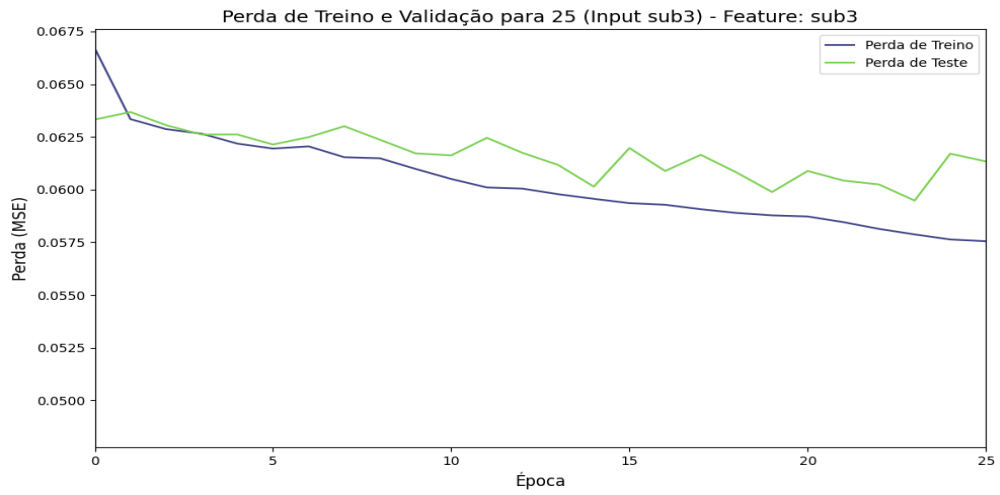
- **Perdas do Modelo:**

A análise dos dados de perda (treinamento e teste) indica, na maioria dos cenários, um aprendizado saudável dos modelos, ou seja, onde a perda de teste é consistentemente maior que a perda de treinamento sendo um indicativo de que não estão apresentando *overfitting*. Observa-se que a perda de treinamento geralmente mostra sinais de convergência, com sua diminuição desacelerando nas épocas finais, o que é um comportamento esperado.

Em um dos casos (*input 25*), visto na Figura 28, apesar da perda de teste ainda ser maior que a de treinamento, houve um ligeiro aumento na perda de teste na última época analisada. Embora não seja um *overfitting* claro, é um sinal precoce de que o modelo pode estar começando a se ajustar demais aos dados de treinamento e generalizar um pouco menos para

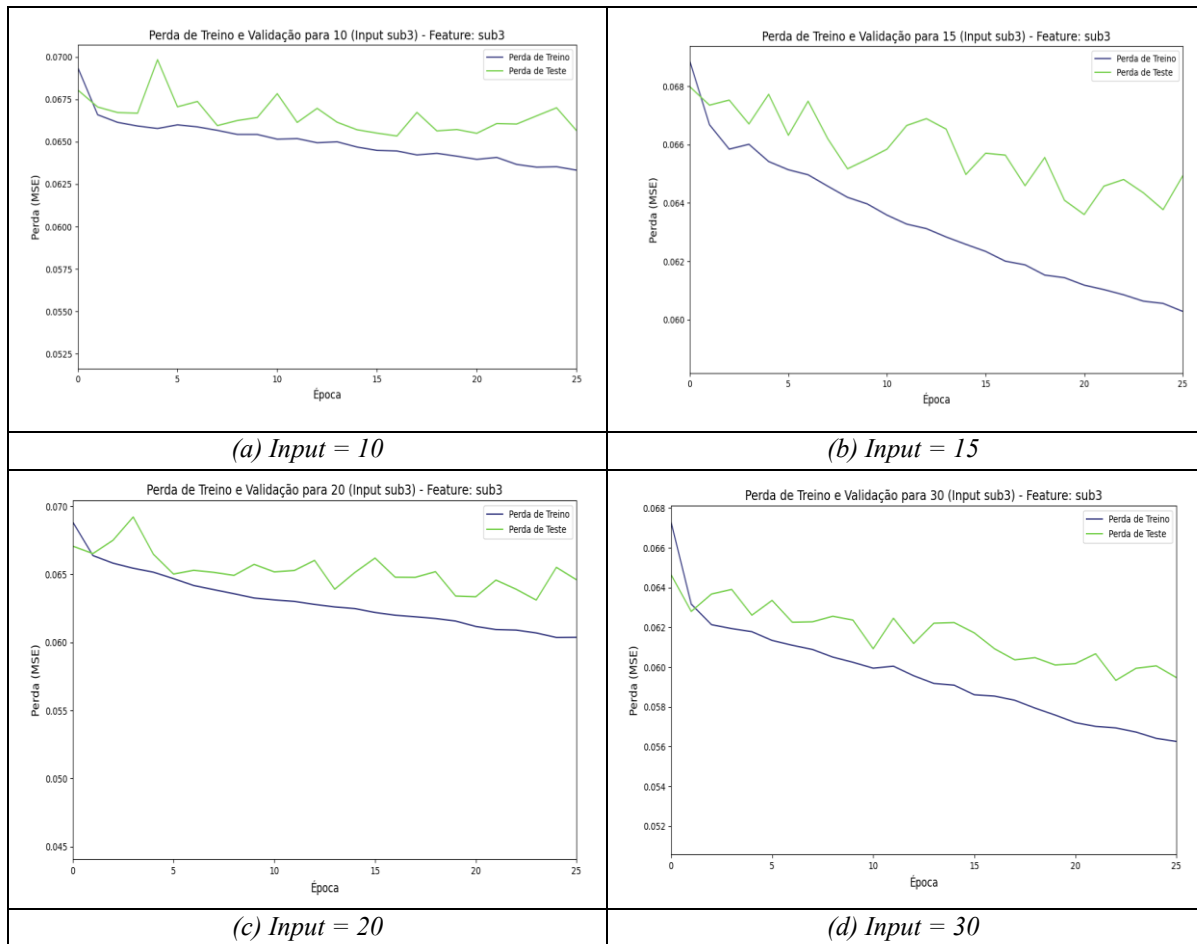
novos dados, diferente dos outros *inputs* na Figura 29.

Figura 28 - Perda de treino e teste em $(Wh)^2$ para input 25 [Sub3 BiGRU].



Fonte: Autoria própria.

Figura 29 - Perda de treino e teste em $(Wh)^2$ [Sub3 BiGRU].



Fonte: Autoria própria.

5. CONCLUSÃO

A predição de energia já se tornou essencial para a sociedade atual, podendo ser feita de diversas formas, olhando não só para a predição de geração e demanda de energia em larga escala mas também para predições residenciais, os modelos utilizados para este trabalho representam apenas alguns dos métodos que podem ser utilizados. Ao longo deste projeto, exploramos e aplicamos diversas ferramentas e etapas cruciais para o desenvolvimento e avaliação destes modelos. Ficou evidente que a seleção e o pré-processamento dos dados são fatores determinantes para o desempenho dos modelos, embora possa haver picos de consumo e também valores zero que indicam não utilização o que é comum para o tipo de dados, já que uma residência possui variantes como tempo de uso de eletrodomésticos, utilização do cômodo, sazonalidade, entre outros. O desempenho dos modelos não teve grandes variações de resultados, mostrando que todos atendem ao objetivo para a predição residencial.

Para a *feature* sub2, o modelo LSTM com *input* = 10 parece ser o mais adequado. Ele apresenta o menor MAE (1,26 Wh) nas métricas de teste, uma forte indicação de seu desempenho superior na média dos erros absolutos. O RMSE (3,09 Wh) e o SMAPE (125,72%) também foram muito competitivos. Apesar do BiGRU com *input* = 25 ter um RMSE ligeiramente menor, o MAE do LSTM com *input* = 10 foi superior. A consistência entre as métricas de treino e teste do LSTM sugere uma boa capacidade de generalização e baixa incidência de *overfitting*, indicando um bom desempenho em dados não vistos. É importante notar que valores de SMAPE acima de 100% são altos, refletindo a dificuldade do modelo em prever a magnitude percentual correta, especialmente em cenários com valores próximos de zero ou grandes flutuações.

Já para a *feature* sub3, o modelo BiGRU com *input* = 20 destacou-se como o mais apropriado. Ele obteve o menor MAE (3,57 Wh), o menor RMSE (5,00 Wh) e o menor SMAPE (73,16%) nas métricas de teste. Isso significa que o BiGRU foi o modelo que cometeu os menores erros médios (absolutos e quadráticos) e demonstrou a melhor precisão percentual em dados não vistos para essa *feature*. A performance em dados de teste, embora ligeiramente inferior à de treino, confirma que o modelo aprendeu os padrões de forma eficaz e generalizou bem, sem sinais significativos de *overfitting*.

No geral, os modelos não mostraram uma grande discrepância de resultados entre si, olhando como o modelo se “ajusta” aos dados. Naturalmente, BiGRU e BiLSTM tendem a ter um desempenho superior pois suas estruturas analisam os dados tanto no passado quanto no futuro. Por outro lado, GRU e LSTM oferecem uma performance mais rápida, o que pode ser

preferível em muitas situações.

Este estudo representou um primeiro passo, explorando uma pequena gama de soluções para a eficiência energética, demonstrando o treinamento e teste de modelos com base em dados residenciais, trazendo a clareza de que os modelos podem ser utilizados para modelagem e projetos relacionados a consumo residencial. Em projetos futuros, expandiremos a pesquisa para incluir novos modelos, como redes neurais sem peso e outros modelos híbridos, e sua integração em *smart grids*, investigando a fundo suas *performances* e aplicações.

REFERÊNCIAS

- ABDUL RAZAK, Mohd Zulhelmi et al. **A Comparative Study of LSTM/GRU Models for Energy Long-Term Forecasting in IoT Networks**. In: 2023 2nd International Conference on Wireless Communications, Networking and Applications (ICWCNA). [S. l.]: IEEE, p. 1-6, 2023. Disponível em: <https://ieeexplore.ieee.org/abstract/document/10210257>. Acesso em: 1 jun. 2025.
- AHMAD, T. et al. **How do machines predict energy use? Comparing machine learning approaches for modeling household energy demand in the United States**. Energy and Climate Change, [s. l.], v. 3, p. 100067, 2022. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S2214629622002195>. Acesso em: 1 jun. 2025.
- ALHARBI, Khalid G. et al. **MACHINE LEARNING FOR INTELLIGENT ENERGY CONSUMPTION IN SMART HOMES**. International Journal of Computing and Information Management, [s. l.], v. 28, n. 1, p. 57-69, 2024. Disponível em: <https://journals.gaftim.com/index.php/ijcim/article/view/75>. Acesso em: 1 jun. 2025.
- AMAZON WEB SERVICES. **O que é RNN? — Explicação sobre redes neurais recorrentes**. Seattle, WA, 2024. Disponível em: <https://aws.amazon.com/pt/what-is/recurrent-neural-network/>. Acesso em: 1 jun. 2025.
- BISHOP, Christopher M. **Pattern Recognition and Machine Learning**. New York: Springer, 2006.
- CHEN, J. S.; YANG, B. **A symmetric ARIMA forecasting system**. Journal of Forecasting, v. 23, n. 7, p. 543-552, 2004.
- DEEP LEARNING BOOK BRASIL. **O neurônio biológico e matemático**. 2024. Disponível em: <https://www.deeplearningbook.com.br/o-neuronio-biologico-e-matematico/>. Acesso em: 1 jun. 2025.
- GASPARIN, Alberto; ALIPPI, Cesare; LUKOVIC, Slobodan. **DEEP LEARNING FOR TIME SERIES FORECASTING: THE ELECTRIC LOAD CASE**. arXiv, 2019. Disponível em: <https://www.semanticscholar.org/reader/003a11f401e286ccfd3a699d8f55db5cf81fd540>. Acesso em: 1 jun. 2025.
- GÉRON, A. **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems**. 2. ed. Boston: O'Reilly Media, 2019.
- GIRAUDET, Louis-Gaëtan; MISSEMER, Antoine. **The history of energy efficiency in economics: Breakpoints and regularities**. ScienceDirect, 2023. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2214629623000336#s0010>. Acesso em 15 fev 2025.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. Cambridge, MA: MIT Press, 2016.

HEBRIL, G.; BERARD, A. **Individual Household Electric Power Consumption**. Irvine, CA: UCI Machine Learning Repository, 2006. Disponível em: <<https://archive.ics.uci.edu/dataset/235/individual+household+electric+power+consumption>>. Acesso em: 27 maio 2025.

HYNDMAN, R. J.; ATHANASOPOULOS, G. **Forecasting: Principles and Practice**. 3. ed. Melbourne: OTexts, 2018. Disponível em: <<https://otexts.com/fpp3/>>. Acesso em: 1 jun. 2025.

INTERNATIONAL ENERGY AGENCY. **Electricity Market Report 2023**. Paris: IEA, 2023. Disponível em: <<https://www.iea.org/reports/electricity-market-report-2023>>. Acesso em: 31 maio 2025.

LAHAOUAR, S.; BEN HADJ BRAIEK, I. A hybrid ARIMA-ANN model for forecasting the daily peak electricity demand in Tunisia. **Renewable and Sustainable Energy Reviews**, Amsterdam, v. 41, p. 574-580, jan. 2015.

MOCANU, E. et al. **Deep learning for forecasting photovoltaic power production**. Solar Energy, v. 165, p. 297-309, 2018.

NIELSEN, H. A.; PINSON, P.; MADSEN, H. **Statistical forecasting of hourly solar power production: A comparison of models and methods**. Solar Energy, v. 143, p. 257-271, 2017.

OMARZAI, Fraidoon. **LSTM And GRU In Depth**. Medium, 16 jul. 2024. Disponível em: <<https://medium.com/@fraidoonomarzai99/lstm-and-gru-in-depth-40aba24bfe53>>. Acesso em: 1 jun. 2025.

RAMOS, Diogo S.; POUSINHO, Helena M.; DIAS, Vera. **Machine Learning in Energy Demand Forecasting: A Review**. Energy Systems, v. 9, n. 3, p. 523-549, 2018.

ROSHAN, V.; STEWART, J. H. M.; JOSEPH, R.; STEWART, H. M. Optimal ratio for data splitting. **Statistical Analysis and Data Mining: The ASA Data Science Journal**, Hoboken, v. 15, n. 4, p. 531-538, ago. 2022.

WANG, J.; ZHANG, W.; DING, M. **Short-term wind power prediction using deep learning and a novel hybrid forecasting model**. Energy Conversion and Management, v. 198, p. 111977, 2019.

WILLMOTT, C. J.; MATSUURA, K. **Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing model performance**. Climate Research, v. 30, n. 1, p. 79-82, 2005.

ZHANG, X.; WANG, Y. Applications of Machine Learning in Energy Systems: A Review. **Renewable and Sustainable Energy Reviews**, Amsterdam, v. 136, p. 110468, fev. 2021.

APENDICE A – CÉLULAS DE IMPORTAÇÃO DE BIBLIOTECAS

```
[ ] !pip install tabulate -q  
    !pip install fpdf -q
```

```
[ ] from google.colab import drive  
    drive.mount('/content/drive')
```

```
[ ] import numpy as np  
    import pandas as pd  
    import matplotlib.pyplot as plt  
    import seaborn as sns  
    from sklearn.preprocessing import MinMaxScaler  
    from sklearn.metrics import mean_absolute_error, mean_squared_error  
    from keras.models import Sequential  
    from keras.layers import LSTM, GRU, Bidirectional, Dense  
    from keras.callbacks import EarlyStopping  
    import math  
    import requests  
    import zipfile  
    from io import BytesIO  
    import os
```

APENDICE B - CÉLULA DE IMPORTAÇÃO E TRANSFORMAÇÃO

```
[ ] # Baixar e carregar o dataset
url = "https://archive.ics.uci.edu/static/public/235/individual+household+electric+power+consumption.zip"
r = requests.get(url)
z = zipfile.ZipFile(BytesIO(r.content))
df = pd.read_csv(z.open("household_power_consumption.txt"), sep=';',
                 parse_dates={'Datetime': ['Date', 'Time']},
                 infer_datetime_format=True, na_values='?',
                 low_memory=False)

# Manter apenas as colunas necessárias
df = df[['Datetime', 'Sub_metering_2', 'Sub_metering_3']].dropna()
df.columns = ['Datetime', 'sub2', 'sub3']
df.set_index('Datetime', inplace=True)
df = df.astype(float)
df = df.resample('h').mean().dropna() # Amostragem horária
df.head()
```

APENDICE C - CÉLULA DE TRATAMENTO DE VALORES FALTANTES

```
[ ] print("Iniciando pré-processamento adicional...")

metering_columns = ['sub2', 'sub3']

# --- Tratamento de NaN (sem alterar 0.00) e Imputação Sazonal ---
# df_after_imputation será o DataFrame após a imputação de NaN's (0.00 são mantidos)
df_after_imputation = df.copy()

for col in metering_columns:
    print(f"Processando coluna para NaNs e Imputação Sazonal: {col}")

    # --- Passo 1: Tentar imputar com o valor da semana anterior (168 horas = 7 dias * 24 horas) ---
    # Shift para pegar o valor da semana anterior.
    # O `fillna` tenta preencher os NaN no df_after_imputation com o valor da semana anterior.
    df_after_imputation[col].fillna(df_after_imputation[col].shift(periods=168), inplace=True)
    print(f" NaN's preenchidos com o valor da semana anterior em {col}.")

    # --- Passo 2: Fallback para Interpolação Temporal para NaN's restantes ---
    # Isso é importante se houver NaN's nas primeiras 168 horas ou se o valor da semana anterior também for NaN.
    df_after_imputation[col] = df_after_imputation[col].interpolate(method='time').copy()
    print(f" NaN's restantes preenchidos com interpolação temporal em {col}.")

    # --- Passo 3: Fallback final para NaNs nos extremos (se houver) ---
    if df_after_imputation[col].isna().sum() > 0:
        print(f" Ainda há NaN's em {col} após a interpolação (provavelmente no início/fim da série). Preenchendo com ffill/bfill.")
        df_after_imputation[col].fillna(method='ffill', inplace=True)
        df_after_imputation[col].fillna(method='bfill', inplace=True)
        if df_after_imputation[col].isna().sum() > 0:
            print(f" Aviso: Coluna {col} ainda contém NaN's. Pode ser necessário tratamento manual ou remoção da coluna/linhas.")
    else:
        print(f" Todos os NaN's preenchidos em {col} (se houver).")
```

APENDICE D - CÉLULA DE TRATAMENTO DE OUTLIERS

```
[ ] # --- Tratamento de Outliers (Capping/Flooring usando IQR) ---
df_final_processed = df_after_imputation.copy()

print("\n--- Tratamento de Outliers (Capping/Flooring usando IQR) ---")
custom_outlier_bounds = {

    'sub2': {'lower': 0.0, 'upper': 34.0},
    'sub3': {'lower': 0.0, 'upper': 30.0}
}

for col in metering_columns:
    print(f"Processando outliers para coluna: {col}")

    if col in custom_outlier_bounds:
        lower_bound = custom_outlier_bounds[col]['lower']
        upper_bound = custom_outlier_bounds[col]['upper']
        print(f" Usando limites definidos manualmente: Inferior={lower_bound:.2f}, Superior={upper_bound:.2f} para {col}.")
    else:
        Q1 = df_final_processed[col].quantile(0.25)
        Q3 = df_final_processed[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        print(f" Calculando limites IQR: Inferior={lower_bound:.2f}, Superior={upper_bound:.2f} para {col}.")

    outliers_low_count = (df_final_processed[col] < lower_bound).sum()
    outliers_high_count = (df_final_processed[col] > upper_bound).sum()
    print(f" Outliers detectados: Abaixo do limite: {outliers_low_count}, Acima do limite: {outliers_high_count}.")
```

APENDICE E - CÉLULA CRIAÇÃO DE FUNÇÕES

```
[ ] def create_dataset(series, input_size):
    X, y = [], []
    for i in range(len(series) - input_size):
        X.append(series[i:i+input_size])
        y.append(series[i+input_size])
    return np.array(X), np.array(y)

def smape(y_true, y_pred):
    numerator = np.abs(y_pred - y_true)
    denominator = (np.abs(y_true) + np.abs(y_pred)) / 2
    return 100 * np.mean(numerator[denominator != 0] / denominator[denominator != 0])

def evaluate_forecast(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = math.sqrt(mean_squared_error(y_true, y_pred))
    s = smape(y_true, y_pred)
    return mae, rmse, s
```

APENDICE F – FUNÇÃO DE EXECUÇÃO

```
def run_models(data, feature_name, input_sizes=[10, 15, 20, 25, 30]):
    results = {}
    results_t = {}
    plot_data = {}
    loss_history = {} # Dictionary to store loss history

    scaler = MinMaxScaler()
    scaled = scaler.fit_transform(data[[feature_name]])

    for input_size in input_sizes:
        X, y = create_dataset(scaled, input_size)
        split = int(0.8 * len(X))
        X_train, X_test = X[:split], X[split:]
        y_train, y_test = y[:split], y[split:]

        X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
        X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

        #for model_type in ['LSTM', 'GRU', 'BiLSTM', 'BiGRU']:
        for model_type in ['BiGRU']:
            model = Sequential()
            if model_type == 'LSTM':
                model.add(LSTM(50, input_shape=(input_size,1)))
            elif model_type == 'GRU':
                model.add(GRU(50, input_shape=(input_size,1)))
            elif model_type == 'BiLSTM':
                model.add(Bidirectional(LSTM(50), input_shape=(input_size,1)))
            elif model_type == 'BiGRU':
                model.add(Bidirectional(GRU(50), input_shape=(input_size,1)))

            model.add(Dense(64))
            model.add(Dense(32))
            model.add(Dense(16))
            model.add(Dense(1))
            model.compile(optimizer='adam', loss='mse')

            history = model.fit(X_train, y_train, epochs=100, batch_size=64, verbose=1,
                                validation_split=0.2,
                                callbacks=[EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)])

            y_pred = model.predict(X_test)
            y_pred_rescaled = scaler.inverse_transform(y_pred)
            y_pred_train = model.predict(X_train)
            y_pred_train_rescaled = scaler.inverse_transform(y_pred_train)
            y_test_rescaled = scaler.inverse_transform(y_test.reshape(-1,1))
            y_train_rescaled = scaler.inverse_transform(y_train.reshape(-1,1))

            mae, rmse, smape_val = evaluate_forecast(y_test_rescaled, y_pred_rescaled)
            mae_train, rmse_train, smape_train = evaluate_forecast(y_train_rescaled, y_pred_train_rescaled)
            label = f"{model_type} | input={input_size} | MAE={mae:.2f}, RMSE={rmse:.2f}, SMAPE={smape_val:.2f}"
            label_train = f"{model_type} | input={input_size} | MAE={mae_train:.2f}, RMSE={rmse_train:.2f}, SMAPE={smape_train:.2f}"
            results[label] = (y_test_rescaled.flatten(), y_pred_rescaled.flatten())
            results_t[label_train] = (y_train_rescaled.flatten(), y_pred_train_rescaled.flatten())

            if input_size not in plot_data:
                plot_data[input_size] = {'true': y_test_rescaled.flatten(), 'predictions': {}}
            plot_data[input_size]['predictions'][model_type] = (y_pred_rescaled.flatten(), label)

            if feature_name not in loss_history:
                loss_history[feature_name] = {}
            if input_size not in loss_history[feature_name]:
                loss_history[feature_name][input_size] = {}
            loss_history[feature_name][input_size][model_type] = history.history

    return results, results_t, plot_data, loss_history
```

APENDICE G – EXECUÇÃO DOS RESULTADOS

```
[ ] all_results = {}
    all_plot_data = {}
    all_loss_history = {} # Dictionary to store loss history for all features
    for feature in ['sub2', 'sub3']:
        print(f"Rodando modelos para: {feature}")
        results, results_t, plot_data, loss_history = run_models(df, feature)
        all_results[feature] = (results, results_t)
        all_plot_data[feature] = plot_data
        all_loss_history[feature] = loss_history # Store loss history per feature

    print("\nTreinamento de modelos concluído.")
```

APENDICE H – PLOTAGEM E ARMAZENAMENTO DOS GRÁFICOS DE PERDA

```
# --- Diretório para salvar os gráficos ---
output_dir_plots = 'diretorio designado'
os.makedirs(output_dir_plots, exist_ok=True)
print(f"Diretório '{output_dir_plots}' garantido para salvamento dos gráficos.")

# --- Plotagem de Gráficos Individuais (Viridis) ---
print("\n--- Plotagem e salvamento de Gráficos Individuais de Perda (Treino vs Validação) ---")

for feature, feature_loss_history in all_loss_history.items():
    print(f"\nProcessando feature: {feature}")
    for input_size, input_size_history in feature_loss_history.items():
        for model_type, history_data_per_model in input_size_history.items():
            print(f"Gerando gráfico para: Feature={feature}, Input={input_size}, Modelo={model_type}")

            plt.figure(figsize=(10, 6))

            # Obter cores do colormap viridis para perda de treino e validação
            cmap_individual = plt.get_cmap('viridis')
            color_train_loss = cmap_individual(0.2) # Cor para perda de treino
            color_val_loss = cmap_individual(0.8)   # Cor para perda de validação

            # Garante que o modelo exista e tenha as chaves 'loss' e 'val_loss'
            if ('LSTM', 'GRU', 'BiLSTM', 'BiGRU') in history_data_per_model and 'loss' in history_data_per_model['BiGRU'] and 'val_loss' in history_data_per_model['LSTM', 'GRU', 'BiLSTM', 'BiGRU']:
                plt.plot(history_data_per_model['LSTM', 'GRU', 'BiLSTM', 'BiGRU']['loss'], label='Perda de Treino', color=color_train_loss)
                plt.plot(history_data_per_model['LSTM', 'GRU', 'BiLSTM', 'BiGRU']['val_loss'], label='Perda de Teste', color=color_val_loss, linestyle='--')
            else:
                print(f"    Aviso: Dados 'loss' ou 'val_loss' não encontrados para xxx em {feature}/{input_size}/{model_type}. Pulando gráfico.")
                plt.close() # Fecha a figura vazia
                continue # Pula para a próxima iteração

            plt.title(f'Perda de Treino e Validação para {model_type} (Input {input_size}) - Feature: {feature}', fontsize=14)
            plt.xlabel('Época', fontsize=12)
            plt.ylabel('Perda (MSE)', fontsize=12)

            # --- NOVO: Definir limites dos eixos X e Y para iniciar em 0 ---
            plt.xlim(0,25) # Garante que o eixo X comece em 0
            #plt.ylim(bottom=0) # Garante que o eixo Y comece em 0
            # --- FIM NOVO ---

            # --- NOVO: Legenda no canto superior direito ---
            plt.legend(loc='upper right', fontsize=10)
            # --- FIM NOVO ---

            plt.grid(False) # Mantido False conforme seu código original
            plt.tight_layout()

            # Salvar o gráfico individual
            filename_individual = f"perda_{feature}_{model_type}_input_{input_size}.png"
            filepath_individual = os.path.join(output_dir_plots, filename_individual)
            plt.savefig(filepath_individual)
            plt.close() # Fechar a figura para liberar memória
            print(f"    Gráfico individual salvo em: {filepath_individual}")
```

APENDICE I – PLOTAGEM E ARMAZENAMENTO DOS GRÁFICOS DE REAL VS. PREVISTO

```
# --- Plotagem dos Gráficos 'Real vs. Previsto' ---
print("\n--- Gerando gráficos 'Real vs. Previsto' (Normal e com Área Preenchida) ---")

output_dir_predictions = '/content/drive/MyDrive/ISF/Bigou/RealPre'
os.makedirs(output_dir_predictions, exist_ok=True) # Criar o diretório se não existir

cmap_plasma = plt.get_cmap('plasma')

for feature, (results_test, results_train) in all_results.items():
    print(f"\nGerando gráficos para a Feature: {feature}")

    # Determinar num_samples_to_plot e xlim_max_value com base na feature
    if feature == 'sub2':
        num_samples_to_plot = 100
        xlim_max_value = 100
    elif feature == 'sub3':
        num_samples_to_plot = 200
        xlim_max_value = 200
    else: # Para outras features (ex: 'sub1')
        num_samples_to_plot = 200 # Valor padrão
        xlim_max_value = None # Deixa o Matplotlib determinar o xlim máximo automaticamente

    for label, (true_vals, pred_vals) in results_test.items():
        parts = label.split(' | ')
        model_type = parts[0]
        input_size_str = parts[1].split('=')[1]

        # Cria uma figura com DOIS subplots (1 linha, 2 colunas)
        fig, axes = plt.subplots(1, 2, figsize=(20, 7)) # Aumenta a largura para acomodar dois gráficos

        # --- Configurações de Cores para o Plasma ---
        color_pred = cmap_plasma(0.2) # Uma cor do início do plasma para previsão
        color_true = cmap_plasma(0.8) # Uma cor do final do plasma para valores reais

        # --- Subplot 1: Gráfico Normal 'Real vs. Previsto' ---
        ax1 = axes[0] # Primeiro subplot
        ax1.plot(true_vals[:num_samples_to_plot], label='Valores Reais', color=color_true, linewidth=2)
        ax1.plot(pred_vals[:num_samples_to_plot], label=f'Valores Previstos ({model_type} - Input {input_size_str})', color=color_pred, linestyle='-.')

        ax1.set_title(f'Real vs. Previsto para {feature}\n(model_type) (Primeiras {num_samples_to_plot} Amostras)', fontsize=14)
        ax1.set_xlabel(f'Amostras (Primeiras {num_samples_to_plot})', fontsize=12)
        ax1.set_ylabel('Consumo [Wh]', fontsize=12)

        # Definir limites do eixo X e Y para o subplot 1
        if xlim_max_value is not None:
            ax1.set_xlim(0, xlim_max_value)
        else:
            ax1.set_xlim(left=0) # Apenas garante que comece em 0 para outras features

        ax1.set_ylim(0, 25) # Limite Y de 0 a 25 para todas as features
        ax1.legend(loc='upper right', fontsize=10) # Legenda no canto superior direito
        ax1.grid(False) # Sem grid

        # --- Subplot 2: Gráfico com Área Preenchida ---
        ax2 = axes[1] # Segundo subplot

        # Plota os valores reais e previstos novamente para referência
        ax2.plot(true_vals[:num_samples_to_plot], label='Valores Reais', color='black', linewidth=2)
        ax2.plot(pred_vals[:num_samples_to_plot], label=f'Valores Previstos ({model_type} - Input {input_size_str})', color='black', linestyle='-.')

        # Preenche a área entre os valores reais e previstos em preto
        x_values = np.arange(num_samples_to_plot)
        ax2.fill_between(x_values, true_vals[:num_samples_to_plot], pred_vals[:num_samples_to_plot],
                        color='black', alpha=0.3, label='Diferença (Real vs. Previsto)')

        ax2.set_title(f'Diferença (Real vs. Previsto) para {feature}\n(model_type) (Primeiras {num_samples_to_plot} Amostras)', fontsize=14)
        ax2.set_xlabel(f'Amostras (Primeiras {num_samples_to_plot})', fontsize=12)
        ax2.set_ylabel('Consumo [Wh]', fontsize=12)

        # Definir os mesmos limites do eixo X e Y que o primeiro subplot
        if xlim_max_value is not None:
            ax2.set_xlim(0, xlim_max_value)
        else:
            ax2.set_xlim(left=0)
        ax2.set_ylim(0, 25)

        ax2.legend(loc='upper right', fontsize=10) # Legenda também no canto superior direito
        ax2.grid(False) # Sem grid

        # --- Finalização da Figura Completa ---
        plt.tight_layout() # Ajusta automaticamente os parâmetros da subtrama para evitar sobreposição

        # Salva a figura que contém AMBOS os subplots
        plot_filename = os.path.join(output_dir_predictions, f'real_vs_pred_double_plot_{feature}_{model_type}_input{input_size_str}.png')
        plt.savefig(plot_filename)
        plt.close(fig) # Fecha a figura inteira para liberar memória
        print(f" Gráficos 'Real vs. Previsto' (Normal e com Área Preenchida) salvo para {feature} - {model_type} (Input {input_size_str}): {plot_filename}")

print("\nTodos os gráficos duplos 'Real vs. Previsto' foram gerados e salvos.")
```

APENDICE J – ARMAZENAMENTO DAS MÉTRICAS UTILIZADAS

```
[ ] output_directory = 'metricas_tabelas'
os.makedirs(output_directory, exist_ok=True)
print(f"\nDiretório '{output_directory}' garantido para salvamento das tabelas de métricas.")

# Crie o diretório se ele não existir
os.makedirs(output_directory, exist_ok=True)

# Combine o caminho do diretório com o nome do arquivo
output_filename = os.path.join(output_directory, "metrics_table.txt")

with open(output_filename, "w") as f:
    f.write("Tabela de Métricas de Treino e Teste\n")
    f.write("=" * 60 + "\n\n")

    for feature, (test_results, train_results) in all_results.items():
        f.write(f"Feature: {feature}\n")
        f.write("-" * (len(f"Feature: {feature}") + 5) + "\n")

        f.write("Métricas de Teste:\n")
        for label, (y_true, y_pred) in test_results.items():
            parts = label.split(" | ")
            model_info = parts[0]
            input_info = parts[1]
            metrics_info = parts[2]
            f.write(f" {model_info}, {input_info}: {metrics_info}\n")
        f.write("\n")

        f.write("Métricas de Treino:\n")
        for label_train, (y_true_train, y_pred_train) in train_results.items():
            parts = label_train.split(" | ")
            model_info = parts[0]
            input_info = parts[1]
            metrics_info = parts[2]
            f.write(f" {model_info}, {input_info}: {metrics_info}\n")
        f.write("\n" + "=" * 60 + "\n\n")

    print(f"\nTabela de métricas salva em: {output_filename}")
```