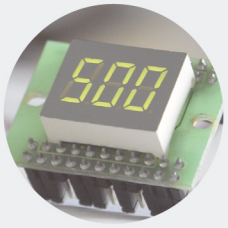
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

MESTRADO EM ENGENHARIA ELECTROTÉCNICA - SISTEMAS ELÉCTRICOS DE ENERGIA






Integração de machine learning em subestações: Estudos de previsão

PEDRO JOÃO PEREIRA ALVES

outubro de 2024

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Integração de machine learning em subestações: Estudos de previsão

Pedro João Pereira Alves

Dissertação para obtenção do Grau de Mestre em Engenharia Eletrotécnica – Sistemas Elétricos de Energia

Orientador: Professora Doutora Teresa Alexandra Ferreira Mourão Pinto Nogueira

Júri:

Presidente:

Professor Doutor Fernando Maurício Teixeira de Sousa Dias, Professor Adjunto, ISEP

Vogais:

Professora Doutora Teresa Alexandra Ferreira Mourão Pinto Nogueira, Professora Adjunta, ISEP

Professor Doutor Nuno Filipe da Fonseca Bastos Gomes, Professor Adjunto, ISEP

Porto, Setembro 2024

Dedicatória

Dedico esta dissertação para a conclusão de uma tão importante etapa á minha família e amigos que sempre me apoiaram e exerceram uma função basilar ao longo do meu percurso académico.

Resumo

Todo o setor energético está a passar por um período de mudanças drásticas em várias valências e isso exige que, cada vez mais, a rede seja fiável, estável e eficiente.

Dado isto, o presente trabalho está intrinsecamente relacionado com as temáticas mencionados. Sendo o foco principal o estudo da implementação de *machine learning* em subestações de forma a aumentar a sua fiabilidade e, consequentemente, a sua estabilidade.

As subestações atualmente contam com um avançado sistema de monitorização baseado no SCADA (*Supervisory control and data acquisition*), no entanto esta implementação apenas permite melhorias na ótica de monitorização e controlo da rede elétrica. A integração de *machine learning* irá permitir que se façam previsões de eventuais defeitos na rede.

Primeiramente será feita uma introdução às subestações, a sua importância e respetivos equipamentos que a integram. Abordando concisamente os principais componentes de uma subestação e como os mesmos podem ser integrados no sistema de previsão.

Segundamente será feita uma introdução ao *machine learning*, incluindo o seu funcionamento e as diferentes tipologias existentes. Concluindo esta parte com uma breve comparação entre os diferentes tipos.

Após ser feita uma breve introdução das subestações e de *machine learning* separadamente, é feita uma análise das vantagens que poderão surgir da integração.

Estando feita a introdução teórica, é elaborado um código em *python* que permite, com recurso a redes neuronais, fazer previsões e extrair dado das mesmas. São criados diferentes cenários para perceber quais os melhores parâmetros nesta situação de integração. Os dados desses cenários são apresentados e analisados.

Com a análise feita, conclui-se com considerações importantes relativamente á utilidade do algoritmo e a eficiência demostrada.

Palavras-chave: Subestação, *Machine Learning*

Abstract

The entire energy sector is going through a period of drastic change in various areas, and this requires the grid to be increasingly reliable, stable and efficient.

Given this, this work is intrinsically related to those themes. The focus is to study the implementation of machine learning in substations to increase their reliability and, consequently, their stability.

Substations currently have an advanced monitoring system based on SCADA (Supervisory control and data acquisition), but this implementation only allows for improvements in terms of monitoring and controlling the electrical network. The integration of machine learning will allow predictions to be made of possible faults in the network.

First, an introduction will be given to substations, their importance and the equipment they contain. The main components of a substation and how they can be integrated into the forecasting system will be concisely covered.

Secondly, there will be an introduction to ML, including how it works and the different types, this part concludes with a brief comparison between the different types.

After a brief introduction of substations and ML separately, an analysis is made of the advantages that could arise from integration.

Once the theoretical introduction has been made, a python code is created which allows predictions to be made using neural networks and to extract data from them. Different scenarios are created to see which parameters are best in this integration situation. The data from these scenarios is presented and analysed.

The analysis concludes with important considerations regarding the usefulness of the algorithm and its demonstrated efficiency.

Keywords: Electrical Substations, Machine Learning

Índice

Lista de Figuras

Acrónimos e Símbolos

Lista de Acrónimos

REN	Redes Energéticas Nacionais

Lista de Siglas

IA	Inteligência Artificial
ML	*Machine Learning*
RN	Redes Neuronais
LSTM	*Long Short-Term Memory*
MAE	*Mean Absolute Error*
MSE	*Mean Squared Error*
RMSE	*Root Mean Squared Error*

Lista de Símbolos

V	Volts (Tensão)
I	Ampere (Corrente)

1 Introdução

1.1 Enquadramento e Objetivos

As subestações são fundamentais para a rede elétrica, desempenhando um papel central no transporte e distribuição eficientes da energia. Facilitando a alteração da tensão, o controlo do fluxo, a modificação dos desfasamentos e o fornecimento estratégico de eletricidade. Ao servirem como centros de transição e interligação, as subestações desempenham uma função fundamental na garantia da estabilidade e fiabilidade da rede elétrica, facilitando a entrega segura da energia gerada nos centros de geração até aos consumidores finais. Gradualmente, as subestações evoluíram para acompanhar o ritmo da crescente demanda. A complexidade destas instalações aumentou drasticamente, incorporando tecnologias para melhorar a eficiência do ponti de vista operacional, a monitorização em tempo real e a capacidade de reagir a variações nas necessidades ou a incidentes inesperados. Face a esta evolução, o objetivo deste trabalho centra-se essencialmente no aumento da eficácia e fiabilidade das linhas de transmissão através da implementação de *machine learning* em subestações recorrendo a redes neuronais.

1.2 Problemas a resolver

O trabalho que se segue visa resolver problemáticas na ótica da prevenção de defeitos e otimização das redes de transmissão. Com o cerne da pesquisa a assentar nas redes neurais e respetivas potencialidades, será feita uma análise de diferentes cenários cujos resultados ditam os híper parâmetros a utilizar. Os parâmetros do algoritmo terão de ser ajustados de forma a otimizar a precisão das previsões. Essencialmente pretende-se, com a maior precisão possível, prever defeitos na rede. A necessidade de manutenção está também relacionada com a ocorrência de falhas na rede, é também nessa perspetiva que se desenvolve o seguinte trabalho.

1.3 Organização do relatório

Inicialmente é feito um levantamento dos constituintes de uma subestação e tudo o que a envolve, são analisados alguns algoritmos de ML e respetivas vantagens e desvantagens. Terminando a introdução com a conjugação de ambos os temas e qual é a tipologia de ML que mais se adequa ao pretendido.

Após o algoritmo estar definido, é criada a metodologia para a sua implementação nas subestações. Ou seja, exatamente o que vai prever e como vai prever.

Tendo definido qual algoritmo e metodologia, são feitos cenários com o objetivo de otimizar os parâmetros. Sempre considerando que qualquer teste assenta na criação de dados simulados recorrendo a funções de natureza aleatória embora com algumas restrições.

Com os resultados obtidos fazem-se breves considerações sobre o que foi testado e possibilidades futuras, concluindo o trabalho após isso.

2 As subestações e integração de ML

2.1 Características e equipamentos das subestações

As subestações elétricas fundamentais na infraestrutura do sistema elétrico. Desempenham o papel crucial de transformar a tensão da energia elétrica para adequá-la às necessidades de transmissão, distribuição e consumo. [15] Numa subestação elevadora, geralmente a energia proveniente da geração é recebida e a tensão é elevada, permitindo a transmissão eficiente em longas distâncias minimizando perdas. Próximo dos centros de consumo final, outra subestação reduz a tensão para níveis mais seguros e adequados para distribuição até ao posto de transformação que depois reduz para o consumidor finais. Existem também as subestações especiais, utilizadas em situações especificas. Por exemplo a subestação móvel.

Existem diferentes tipos de subestações, podendo ser classificadas segundo a alteração que fazem á tensão (elevar ou reduzir) ou segundo a sua localização na rede elétrica.

São instalações geralmente equipadas com transformadores e dispositivos de proteção que garantem a segurança e a eficiência do fornecimento da energia elétrica. Sem as subestações, seria impraticável transmitir eletricidade em longas distâncias mantendo, também, a capacidade de fornecer uma quantidade crescente de energia.

2.1.1 Classificação e tipos de subestações

As subestações estão divididas por níveis de tensão e a sua função na rede. No que toca aos níveis de tensão temos:

- Baixa tensão
- Média tensão
- Alta tensão
- Extremamente alta tensão

As tensões presentes nas subestações, de uma forma generalizada, dependem do sítio da rede onde se encontram. Se na produção, distribuição ou transporte de energia.

Sendo que as subestações perto da produção e no transporte lidam com alta e extremamente alta tensão enquanto as subestações na distribuição contam com baixa e média tensão.

A sua classificação é também feita pela função, podendo ser:

- Distribuição
- Transformação
- Elevadora
- Abaixadora
- Manobra e controle
- Móvel

2.1.1.1 Subestações de distribuição

As subestações de distribuição têm a função principal de modificar os níveis de tensão, reduzindo a tensão transmitida pelas linhas de transmissão para níveis seguros e adequados à distribuição nas redes locais.

2.1.1.2 Subestação elevadora

A função principal de uma subestação elevadora é aumentar a tensão da energia elétrica gerada para facilitar o transporte de longa distância minimizando as perdas. Isso é realizado através do uso de transformadores elevadores, que elevam os níveis de tensão para valores adequados para a transmissão.

2.1.1.3 Subestação abaixadora

Uma subestação abaixadora diminui os níveis de tensão da energia elétrica proveniente das linhas de transporte, tornando-a apropriada para a distribuição em média tensão ou alta tensão. Este processo permite que a energia seja posteriormente distribuída pelos postos de transformação. Baixar a tensão na distribuição intermédia (entre subestação e posto de transformação) é importante por questões de segurança e facilidade de construção das infraestruturas elétricas necessárias.

2.1.1.4 Subestação de manobra e controle

A subestação de manobra e controle realiza operações de manobra, como abrir e fechar circuitos, e fornecer controle local sobre o fluxo de eletricidadena rede elétrica. Esta tipologia de subestações está localizada em sítios estratégicos onde poderá ser necessário efetuar manobras ou é conveniente fazê-lo. Tanto para distribuir energia por diferentes circuitos, como para manutenção. São essencialmente compostas por equipamentos como seccionadores e disjuntores.

2.1.1.5 Subestação móvel

As subestações móveis são unidades temporárias e transportáveis que desempenham funções semelhantes às subestações fixas, mas podem ser movidas conforme necessário. São projetadas para fornecer soluções flexíveis em situações de emergência, manutenção planeada, expansões temporárias ou quando é preciso restabelecer rapidamente o fornecimento de energia. Podem ser constituídas por apenas um atrelado ou vários, conforme o que for planeado.

Na Fig. 1 está representada uma subestação móvel onde é possível observar que estão presentes todos os elementos principais expectáveis de uma subestação fixa, transformador e equipamentos de proteção.

Figura 1 - Subestação movel EFACEC [19]

2.1.2 Importância das subestações nas energias renováveis

A variabilidade e um pouco instabilidade das energias renováveis no que toca á potencia produzida, a existência de uma rede bem planeada e equilibrada de subestações é essencial para o aproveitamento máximo de todo o potencial renovável.

Isto porque a capacidade de fazer a gestão energética de forma eficiente irá ditar o crescimento das energias renováveis. Apenas porque na grande maioria dos casos as localidades com a maior capacidade de gerar energia desta natureza são as que menos necessitam dela devido ao baixo consumo energético. Um exemplo claro disso seria a produção fotovoltaica no Alentejo. Se a infraestrutura elétrica não estiver devidamente preparada, é esse o ponto de estrangulamento ao crescimento da produção sustentável e verde de energia.

Na Fig. 2 pode-se observar um diagrama representativo da infraestrutura elétrica de uma central eólica *offshore* desde a produção eólica até à rede de transporte de energia a 400kV, onde estão inseridas duas subestações essenciais para o seu funcionamento. Neste caso concreto não so há alterações nas tensões, mas também na corrente pois, com recurso a um inversor, a corrente é transportada em corrente continua.

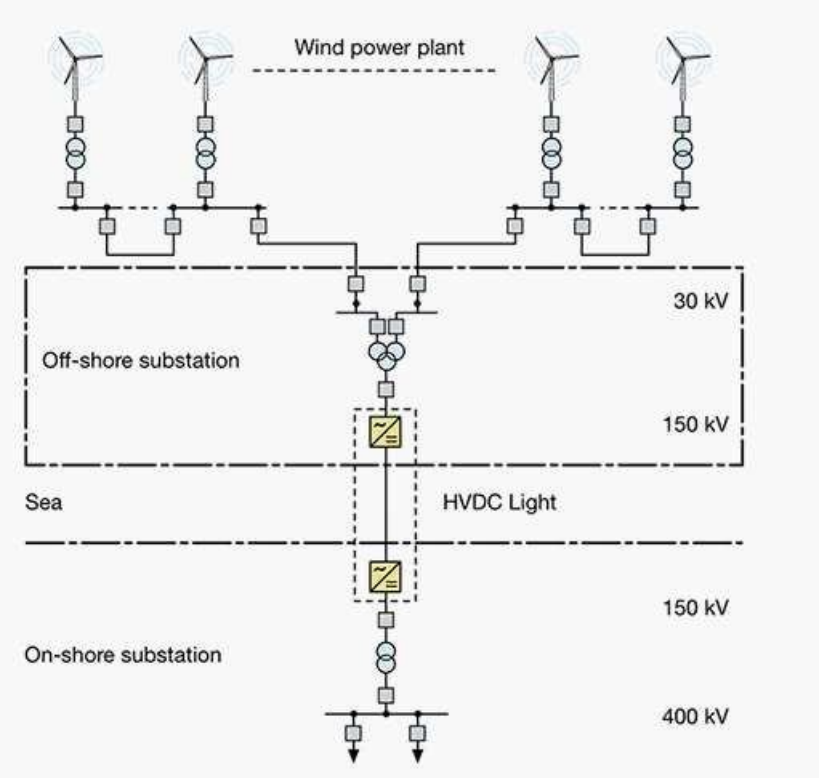


Figura 2 - Representação da rede elétrica desde a produção eólica até à rede de transporte [8]

2.1.3 Elementos que constituem uma subestação

As subestações podem ter uma diversidade de equipamentos e componentes, conforme o seu propósito.

Dado isto, os principais equipamentos e componentes a ter em consideração para o estudo da aplicação de ML em subestações são:

- Transformador (incluindo todos os seus elementos condutores de potência)
- Condutores (Cabos e barramentos)
- Disjuntor
- Seccionador

No entanto, qualquer outro ponto com possibilidade de aquecimento tem interesse para o estudo. Sendo que do ponto de vista da previsão de manutenção preventiva, todos os pontos de aquecimento são relevantes.

2.1.3.1 Transformador

O transformador é o equipamento principal de qualquer subestação exceto das subestações de manobra e controle onde o mesmo não consta.

É o equipamento responsável pelas alterações dos parâmetros da rede que ocorrem nas subestações. Podendo ser alterados os níveis de tensão, a frequência ou o desfasamento entre a tensão e a corrente. É, também, constituído por vários acessórios e componente tal como está descrito na Fig. 3.

Figura 3 - Transformador com respetivos componentes identificados [27]

2.1.3.2 Condutores

Nas subestações existem os elementos condutores que fazem a ligação entre os equipamentos. Neste âmbito temos como condutores os barramentos e os cabos presentesnas subestações. Na Fig. 4 estão a cor vermelha representados os condutores da subestação.

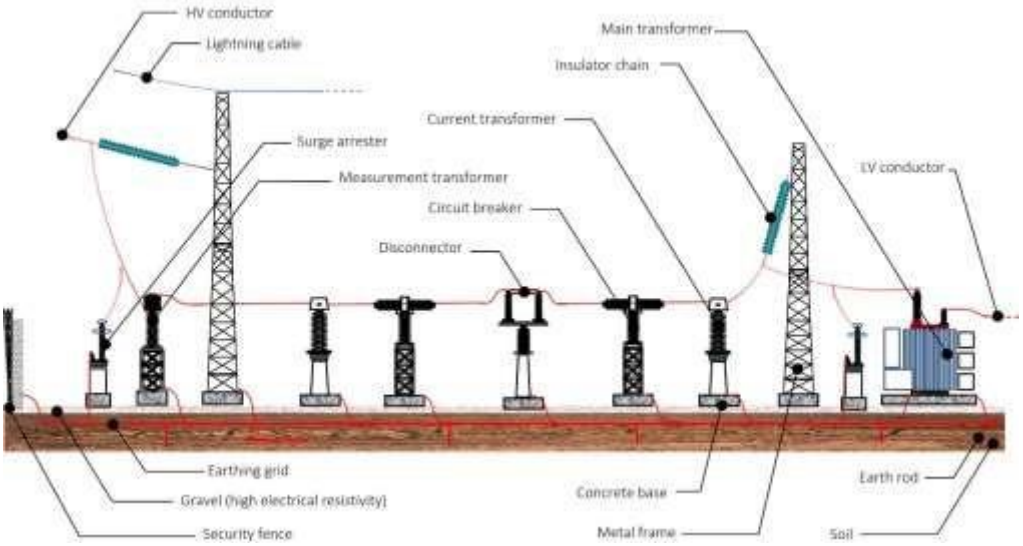


Figura 4 - Representação unifilar de uma subestação [5]

2.1.3.3 Disjuntor

Os disjuntores são equipamentos fulcrais para garantir a proteção das linhas em caso de defeito. Neste âmbito, os disjuntores com relevância são os disjuntores utilizados em situações em que a tensão é maior ou igual que 15kV. Mais concretamente os disjuntores de média tensão (neste caso isolados com SF6) representado na Fig. 6 e os disjuntores de alta tensão (neste caso *dead tank* mas poderia ser live *tank*) representado na Fig. 5.

Figura 6 - Disjuntor media tensão (Cela MT) [4]

Figura 5 - Disjuntor alta tensão (Dead tank) [3]

2.1.3.4 Seccionador

O seccionador é um equipamento que não tem acionamento automático contrariamente ao disjuntor. Com o seccionador o circuito é aberto quando é dada uma instrução (no caso de ser motorizado) ou quando é feito um deslastre mecânico do mesmo.

É importante no que toca à gestão das cargas e também na ótica da proteção (sendo que não atua autonomamente, mas quando auxiliado por um relé de proteção pode também proteger o sistema).

Numa subestação convencional (que não seja uma subestação integralmente isolada com gás SF6), os seccionadores são isolados a ar com semelhanças ao seccionador na Fig. 7. Dependendo da necessidade e da forma como a subestação foi concebida, podem existir seccionadores ligeiramente diferentes ao representado na Fig 7, mas o princípio de funcionamento mantém-se. Existem um ou dois pontos de contactos que quando afastados deslastram o sistema e extinguem o arco elétrico recorrendo ao isolamento por ar.

Figura 7 - Seccionador com abertura vertical de alta tensão [22]

2.1.4 Subestações da rede elétrica nacional

A REN é a entidade responsável por garantir o fornecimento ininterrupto de eletricidade e gásem Portugal Continental, sendo, portanto, a entidade exploradora das subestações. Dado isto,é possível ver na Fig. 8 a localização de cada uma das subestações de muito alta tensão controladas pela REN. Pode-se observar que estão operacionais 70 subestações ao controlo da REN.

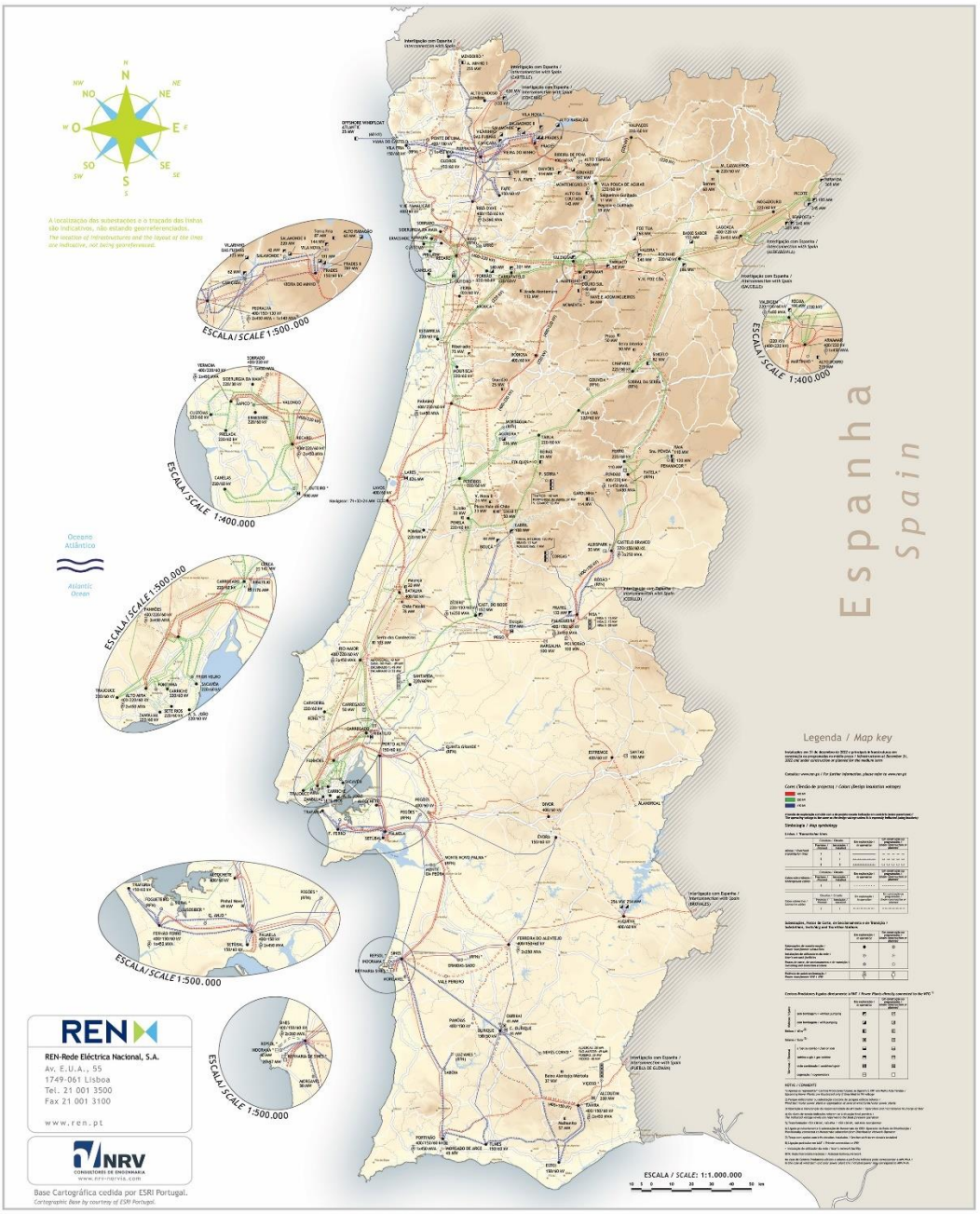


Figura 8 - Mapa da rede nacional de 150kV, 220kV e 400kV [21]

2.1.5 Manutenção das subestações

Devido á natureza modular, as subestações carecem de manutenção periódica preventiva e corretiva caso seja necessário. As manutenções devem ser realizadas por profissionais qualificadas para o efeito, seguindo escrupulosamente as normas de segurança. Podem ser operações demoradas, o que implica que a rede pode ficar debilitada durante um período alongado. Isto porque, mesmo havendo redundância através de um transformador suplente, nesse período da manutenção deixa de haver redundância dado que o transformador principal está em manutenção. Em caso de defeito no transformador de reserva, o sistema fica em baixo. Este tipo de eventos são muito custosos para os operadores sendo que, no caso das alimentações de media, alta e muito alta tensão diretamente para clientes finais, os períodos sem alimentação são indemnizados com valores que chegam aos milhares de euros por cada minuto em falha.

Existem dois tipos de manutenções:

Corretiva- As manutenções corretivas acontecem após ser detetado um problema que precisa de ser resolvido. Por exemplo, um trabalhador da subestação repara que existe um terminal mal apertado. Esse ponto poderá dar origem a um aquecimento e consequentemente a outros problemas. Após esta deteção, é agendada manutenção.

Preventiva- As manutenções preventivas são agendadas periodicamente e, como o próprio nome indica, são de caracter preventivo. Com o objetivo de garantir o funcionamento sem imprevistos da subestação.

2.2 Defeitos na rede

Os defeitos nas redes elétricas de transmissão de energia são classificados, geralmente, em curtos-circuitos, sobrecargas ou defeitos de isolamento. Curto-circuitos ocorrem quando há contacto inadequada entre fases ou entre fase e terra (Fig. 9), causando uma corrente muito acima do dimensionado. As sobrecargas dão-se quando a corrente que passa no circuito é acima do esperado, levando ao aquecimento e à degradação dos componentes. Já os defeitos de isolamento ocorrem quando o material isolante, que isola condutores, se deteriora, permitindo a passagem de corrente. Dado isto, a manutenção preventiva e a rápida deteção de falhas são essenciais para garantir a fiabilidade da rede.

2.2.1 Sobrecargas

A sobrecarga de energia na linha de transmissão dá-se quando a corrente elétrica excede o limite de funcionamento dimensionado para o sistema. Podem ser causadas por uma série de razões, tais como falhas no equipamento, aumento inesperado do consumo ou condições climáticas desfavoráveis. A sobrecarga contínua pode reduzir a eficiência do sistema elétrico devido ao aumento de temperatura do condutor e o consequente aumento na resistência elétrica.

Reduz a vida útil do equipamento e aumenta o risco de falha. As sobrecargas são monitorizadas com sistemas de proteção, tais como relés de sobreintensidade e dispositivos de desligamento automático concebidos para isolar o sistema em caso de discrepância.

É importante utilizar tecnologia de monitorização e controlo em tempo real para reduzir o impacto das sobrecargas. Estes sistemas detetam rapidamente o aumento fora do normal da corrente. Permitindo uma intervenção automática ou manual antes que ocorram danos severos. Dado isto, as práticas de manutenção preventiva e o reforço das estruturas de transmissão, como a utilização de condutores sobredimensionados, são importantes para reduzir a probabilidade de sobrecargas atualmente e também no futuro.

As sobrecargas têm diferentes causas:

- **Demanda excessiva:** Quando a demanda por eletricidade num local ultrapassa a capacidade projetada da linha de transmissão, ocorre uma sobrecarga. Acontece em momentos de pico de consumo, como durante o Inverno ou situações especificas que causam aumento da carga.

- **Falhas ou desligamentos noutras linhas:** Se uma ou mais linhas de transmissão falharem ou forem desligadas para manutenção, a carga é redistribuída para outras linhas. Caso as linhas não estejam preparadas para suportar o aumento de carga, podem entrar em sobrecarga.

- **Problemas na geração de energia:** Se uma unidade de geração parar de fornecer energia ou reduzir a capacidade, as linhas conectadas a outras unidades de geração podem ser sobrecarregadas ao tentar compensar a falta causada pela redução.

- **Crescimento desordenado da demanda:** O crescimento populacional ou a expansão de indústrias sem o devido planeamento e expansão da infraestrutura de transmissão pode sobrecarregar o sistema.

- **Perdas técnicas:** As perdas de energia ao longo das linhas de transmissão aumentam com o aumento da corrente elétrica. Se a corrente for maior que o projetado, pode haver uma sobrecarga por causa do aquecimento excessivo dos condutores.

- **Problemas climáticos:** Temperaturas altas podem diminuir a capacidade das linhas de transmissão, enquanto ventos fortes ou tempestades podem causar falhas ou curtos-circuitos, aumentando o risco de sobrecargas noutras linhas.

- **Subdimensionamento da rede:** Se a rede de transmissão não for corretamente dimensionada para suportar o crescimento da demanda ou se houver erros de planeamento, as linhas podem operar constantemente próximas ou além de sua capacidade nominal.

2.2.2 Assimetrias

As assimetrias nas redes elétricas de transmissão de energia ocorrem quando há um desequilíbrio entre as fases de um sistema trifásico, causando variações nas tensões e correntes de cada fase. Este pode ser resultado de cargas que não estejam balanceadas, falhas em equipamentos (como transformadores e linhas) ou até mesmo de defeitos, como curto-circuitos monofásicos. As assimetrias provocam perdas de energia, aquecimento excessivo em máquinas rotativas (como motores e geradores) e diminuição da eficiência dos equipamentos. Além disso, podem gerar alterações na qualidade da energia e impactos na estabilidade do sistema.

2.2.3 Curto-circuitos

Os curto-circuitos nas linhas de transmissão de energia são falhas que ocorrem quando há uma ligação de baixa resistência entre dois ou mais condutores de fases ou entre fases e terra, o que causa um pico de corrente. Resultando numa sobrecarga no sistema, que poderá causar danos aos equipamentos, interrupções no fornecimento de energia e, em casos extremos, riscos à segurança publica.

Os curto-circuitos podem ser classificados em diferentes tipos, dependendo das fases envolvidas e da natureza do contacto:

- **Curto-circuito fase-fase** *(Line to Line fault)***:** Ocorre entre duas fases da linha de transmissão. É um tipo de falha que pode provocar correntes elevadas e uma redução de tensão significativa nas fases envolvidas.

- **Curto-circuito fase-terra** *(Single line to ground fault):* Envolve o contacto de uma fase com o solo (terra). Esse é o tipo mais comum de curto-circuito em sistemas de transmissão, muitas vezes causado por descargas atmosféricas (raios) ou falhas de isoladores.

- **Curto-circuito trifásico** *(Three phase fault):* Acontece entre as três fases simultaneamente e, embora seja menos comum, é a falha mais grave. A corrente neste tipo de curto-circuito tende a ser extremamente alta, exigindo ação rápida dos sistemas de proteção.

- **Curto-circuito fase-fase-terra** *(Single line to ground fault):* Envolve o contacto de duas fases com o solo, o que pode causar altos níveis de corrente de falha e, consequentemente, danos aos equipamentos.

Na Fig. 9 estão representações de todos os tipos de curto-circuitos descritos, os diagramas mostram o percurso da corrente em cada cenário de defeito.

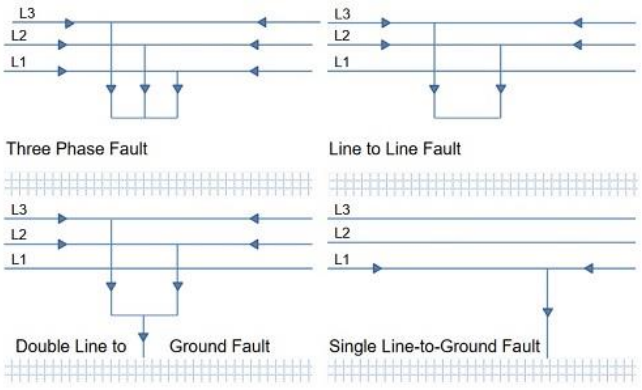


Figura 9 - Diferentes cenários de CC [23]

Origem dos CC

Os curto-circuitos podem ter várias origens:

- **Descargas atmosféricas (raios):** Estas ocorrem quando os raios atingem diretamente os condutores de uma linha de transmissão. Neste caso, desenvolve-se um arco e, consequentemente, um curto-circuito.

- **Contacto com árvores ou objetos:** Os ramos de árvores ou objetos que permitem a condução da corrente podem também ser a causa de um curto-circuito.

- **Deterioração do isolamento:** O material isolante que isola os cabos deteriora-se ao longo do tempo, uma vez que está continuamente exposto aos raios UV, à humidade, à poluição ou a temperaturas extremas. A perda de qualidade do isolamento facilita a ocorrência de curto-circuitos entre os condutores ou entre um condutor e a terra.

- **Fauna (animais):** Os animais, como as aves ou pequenos mamíferos, podem tocar em diferentes condutores ou terras ao mesmo tempo e viabilizar um caminho de menos resistência para a corrente elétrica.

- **Fatores climáticos (tempestades, ventos fortes e gelo):** Os ventos fortes fazem com que os cabos balancem e toquem uns nos outros, provocando curto-circuitos entre fases. A acumulação de gelo pode provocar o contacto dos condutores ou com estruturas ligadas à terra.

- **Falhas mecânicas ou estruturais:** Os elementos mecânicos de suporte podem partir-se devido a deterioração, sobrecargas ou acidentes, permitindo assim que os condutores entrem em contacto. A corrosão dos suportes ou das ligações também pode provocar avarias.

2.3 Machine learning

2.3.1 Enquadramento

É um campo da inteligência artificial (IA) que se concentra no desenvolvimento de algoritmos e modelos que permitem aos sistemas computacionais aprenderem e melhorarem a partir de dados fornecidos. Em vez de seguir algo explicitamente programado, um sistema de ML utiliza dados para treinar e aprimorar as capacidades de previsão ao longo do tempo.

2.3.2 Tipologias de Machine Learning

Certos métodos de ML têm vantagens e desvantagens específicas, e a escolha do método depende do problema em questão, do tamanho e complexidade do conjunto de dados e de outros fatores.

Os métodos assentes em ML têm vindo a ser cada vez mais integrados em todo o tipo detecnologias e serviços. Na Fig. 10 é possível ver o histórico e a previsão do rendimento proveniente de sistemas em sistemas integrados com inteligência artificial.

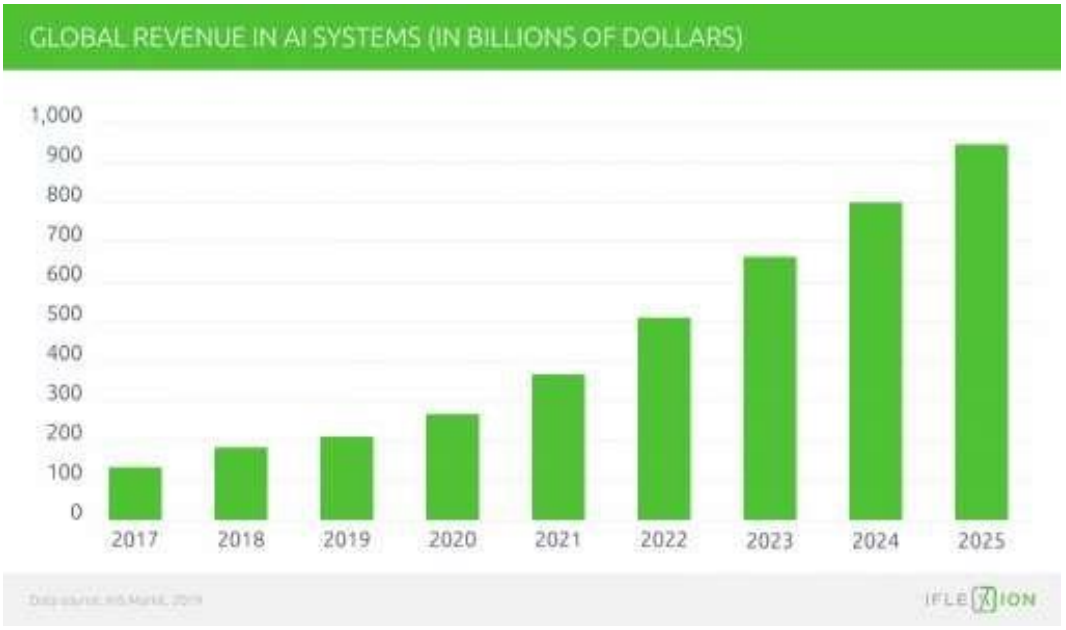


Figura 10 - Rendimentos em sistemas de inteligência artificial (em biliões de dólares) [14]

A tabela 1 faz uma breve comparação dos algoritmos de ML, esta informação será importante na escolha do algoritmo que será implementado.

Tabela 1 - Comparação de algoritmos de *machine learning*

Método	Vantagens	Desvantagens
Regressão Linear	Simplicidade e facilidade de interpretação; Bons resultados quando a relação entre variáveis é aproximadamente linear;	Limitações quando a relação entre variáveis é complexa ou não linear;
Árvores de Decisão	Fácil interpretação de visualização; Lida bem com dados não lineares e não requer normalização;	Tendência a *overfitting*, especialmente em árvores profundas; Sensíveis a pequenas variações nos dados;
Máquinas de Vetores de Suporte	Eficientes em espaços de alta dimensão; Boa performance em conjuntos de dados pequenos;	Pode ser computacionalmente intensivo; Sensível à escolha dos *híper* parâmetros;
Redes Neurais	Capacidade de aprender padrões complexos; Pode lidar com grandes conjuntos de dados e tarefas complexas;	Requer grande quantidade de dados para evitar *overfitting*; Pode ser computacionalmente intensivo e demorado para treinar;
K-Nearest Neighbors	Simplicidade e facilidade de implementação; Funciona bem em dados com estrutura local clara;	Sensível a *outliers* e ruído nos dados; Requer armazenamento detodo o conjunto de dados;
Clustering	Descobre padrões em dados não rotulados; Útil para identificar grupos ou estruturas nos dados;	Interpretação subjetiva dos resultados; Dependência da qualidade inicialização;
Aprendizagem por reforço	Pode aprender comportamentos sequenciais em ambientes dinâmicos;	Pode ser suscetível a problemas de exploração versus explotação;

2.3.3 O presente e o futuro de Machine Learning

Atualmente já é uma tecnologia amplamente utilizada, no entanto todas as previsões indicama sua expansão exponencial. Como se pode observar na Fig. 11, num espaço de apenas 7 anos o valor do mercado passou de 924.39 milhões de dólares para mais de 10 vezes esse valor totalizando 11,078.32 milhões de dólares como previsão para 2024.

Todos os indicadores apontam para o mesmo sentido, o crescimento. Como não há qualquer indicativo de abrandamento, é esperado que pelo ano de 2050 o valor deste mercado atinga 47,342.36 milhões de dólares. Valor muito expressivo e que claramente indica uma tecnologiabem estabelecida.

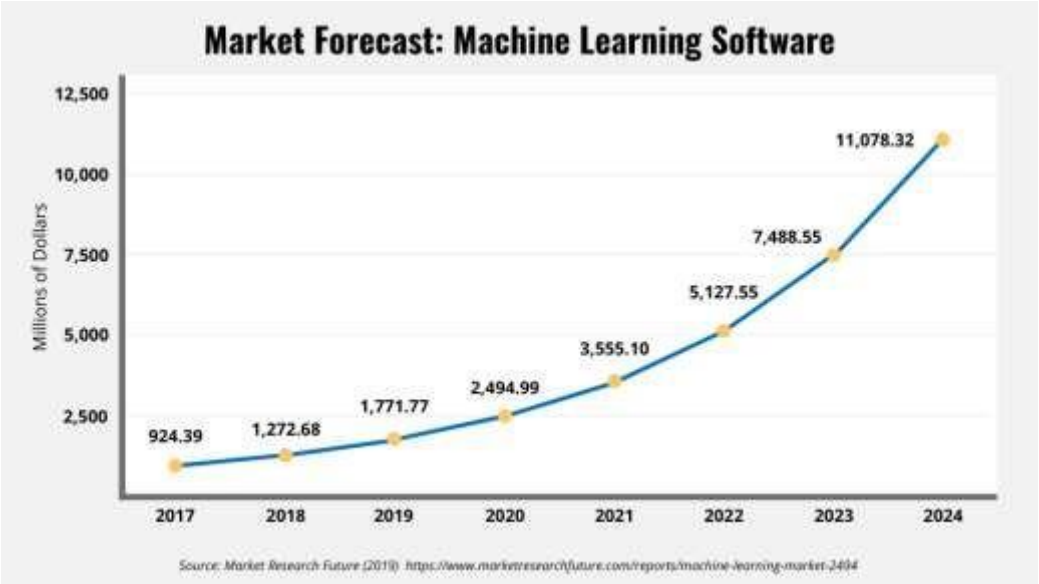


Figura 11 - Evolução do valor do mercado do software de machine learning [14]

A implementação do ML nas subestações não é uma possibilidade, mas sim uma certeza, o caminho para a eficiência é a integração de algoritmos que permitem analisar dados e tirar conclusões que apenas são possíveis através de cálculos computorizados avançados.

2.4 Integração de *Machine Learning* em subestações

2.4.1 Os problemas e a solução

A ML pode desempenhar um papel crucial na resolução de vários desafios e na melhoria da eficiência, fiabilidade e segurança das subestações. Alguns problemas que ML pode ajudar aresolver no contexto das subestações incluem:

- Deteção de falhas
- Previsão de carga
- Estabilidade e controlo da tensão
- Otimização das operações da rede

É expectável que a integração de ML consiga abonar a favor de todas as temáticas referidas em cima.

2.4.2 Deteção de falhas e manutenção preditiva

Os modelos de ML identificam padrões em dados antigos que podem ser indicativos de falhas no equipamento ou de possíveis falhas na subestação. Podem, também, indicar a possibilidade de falhas entre os principais componentes, como transformadores e disjuntores, para permitir uma manutenção proactiva e evitar interrupções não planeadas.

Oferecem novas capacidades para a prevenção proactiva de problemas, redução do tempo de inatividade não programado e otimização das operações de manutenção.

A recolha de dados é o primeiro passo na deteção de avarias e da manutenção preditiva. Os sensores instalados no equipamento das subestações recolhem dados operacionais (tensões e correntes), condições ambientais e de desempenho. Os dados recolhidos são enviados para processamento em tempo real nos algoritmos de ML. Os modelos são treinados com dados previamente recolhidos para aprender padrões normais de funcionamento e para detetar desvios desses padrões, que podem indicar possíveis falhas.

Permite o diagnóstico de vários tipos de avarias em equipamentos, desde transformadores a disjuntores e outros dispositivos essenciais, tendo em consideração múltiplas e complexas variáveis que levam à deteção da causa raiz das avarias. Com a análise de dados contínuos, os modelos ML produzem alertas preditivas que mostram quando o equipamento irá falhar. Esta abordagem permite que a manutenção seja efetuada antes da ocorrência de problemas iminentes e dá margem de manobra ao operador da rede, evitando assim períodos de inatividade não programados. Por todas estas razões, a deteção de falhas e a manutenção preditiva reduzem efetivamente os custos operacionais, uma vez que as intervenções de manutenção podem ser otimizadas para minimizar o tempo de inatividade.

A integração de modelos ML com SCADA permitirá uma resposta mais rápida a eventos anómalos. isto melhora as capacidades de gestão em tempo real. Assim, a deteção de avarias e a manutenção preditiva permitem uma abordagem mais proactiva, em que a manutenção é programada com base no estado real do equipamento e não em intervalos de tempo fixos.

2.4.3 Previsão de carga

Ao analisar grandes conjuntos de dados complexos e ao ter em conta os fenómenos meteorológicos com variações, os algoritmos de ML podem prever as necessidades futuras de eletricidade ao nível da distribuição. Isto ajuda os operadores de rede no que toca á afetação de recursos, a programar os intervalos de pico de necessidades e a melhorar a administração, de um modo geral, da rede. [19]

O diagnóstico prematuro de anomalias permite realizar intervenções de manutenção antes de surgirem falhas graves. Ao analisar continuamente as estatísticas em tempo real, os modelos são capazes de gerar alertas preditivos, indicando o momento ideal para efetuar a intervenção com base nas circunstâncias reais de funcionamento do aparelho. Esta execução não só contribui para reduzir os preços operacionais, como também melhora significativamente a fiabilidade e a segurança das subestações.

Com toda a revolução relacionada com as *smart-grids* e a criação em massa de centrais de geração de energia renovável, supondo um futuro sustentável, a capacidade de previsão será fundamental para o bom funcionamento de toda a infraestrutura elétrica. Isto não só a nível das ligações internas, mas também nas interligações internacionais. Para que toda a produção seja bem aproveitada é fulcral a existência de comunicação internacional de dados reais e de previsão.

Além disso, a integração de modelos de ML com estruturas de *Supervisory Control and Data Acquisition* (SCADA) permite uma resposta mais rápida a eventos anómalos, garantindo-a em tempo real e melhorando ainda mais a eficiência operacional das subestações.

2.4.4 Estabilidade e controlo da tensão

Relativamente ao controlo da tensão, acaba por estar também relacionado com a introdução da variabilidade das energias renováveis. Quando é possível saber com alta precisão qual vai ser a necessidade da rede, permite que haja estabilidade na tensão pois é possível contrariar os efeitos negativos que a instabilidade destas tecnologias causa.

Uma outra questão muito importante relativamente á estabilidade e controlo da tensão é o crescimento em massa da mobilidade elétrica. Os carregadores de veículos elétricos, mais concretamente carregadores públicos, estão equipados com elevadíssimas potências de carregamento. Ou seja, do ponto de vista do controlo da rede existem cada vez mais picos de potência que ocorrem de forma imprevisível na interpretação humana. Sendo possível que apenas os modelos de ML encontrem padrões.

Os sistemas de controlo automatizados, guiados pela ML, podem ajustar os parâmetros em tempo real para manter os níveis de tensão ideal e evitar interrupções no fornecimento de energia.

2.4.5 Otimização das operações da rede

Recorrendo a ML é possível otimizar o funcionamento geral da rede elétrica através da análise de dados. Isto inclui a previsão de caminhos otimizados para corrente, a gestão de sistemas de armazenamento de energia e o ajuste dinâmico dos fluxos de energia para melhorar a eficiência e a estabilidade da rede.

Os modelos ML podem classificar diferentes tipos de eventos que ocorrem na subestação, como falhas, perturbações ou avarias de equipamento. As respostas automáticas podem ser acionadas com base nestas classificações, simplificando o processo de tomada de decisões e reduzir os tempos de resposta.

Na ótica da otimização há várias rotas pelas quais podemos envergar de forma a criar uma operação o mais eficiente possível. No entanto assenta essencialmente na otimização via previsão de cargas, defeitos e condições, viabilizando a operação sem atritos da infraestrutura elétrica.

2.4.6 Implementações reais

Apesar de que no ponto de vista teórico esta integração pareça ser simples, quando falamos de fazer um projeto numa subestação real já não seria realista considerar uma implementação sem atritos e facilitada. Um dos principais desafios seria a recolha de dados, dado que mesmo na subestação mais avançada não há sensores em todos os pontos que necessitamos. [8] Em muitos dos casos apenas são feitas duas leituras de temperaturas, temperatura ambiente e do transformador. Apenas estes valores são manifestamente insuficientes para uma previsão e análise com a precisão pretendida. Numa situação ideal teríamos leitura de temperatura em todos os pontos que são alvo de reaperto quando há manutenções. No entanto, e mesmo com uma recolha de dados não ideal, é possível fazer uma previsão. Apenas não será com a precisão desejada.

Idealmente a integração de ML nas subestações e no seu funcionamento tem de ser pensadana altura do planeamento da própria subestação anteriormente à sua construção. Isto para serem incluídos todos os equipamentos que serão necessários como também as respetivas linhas de comunicação para o sistema central.

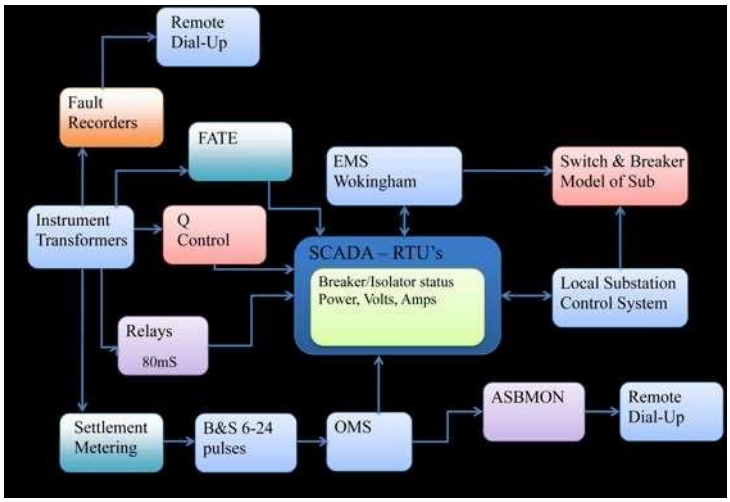


Figura 12 - Exemplo de um sistema SCADA [24]

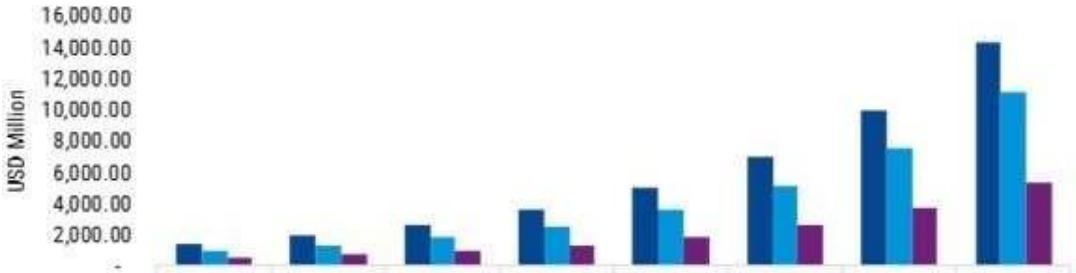
Na Fig 12 está representado um digrama de um sistema SCADA, para a implementação pretendida teríamos mais uma seta apontar para o RTU vinda dos sensores necessários à obtenção precisa das leituras de valores de grandezas de temperatura e até vibração. Depoisteríamos uma comunicação bidirecional do RTU para um computador local onde esteja um software de *machine learning* instalado com o algoritmo mais favorável para a situação.

2.4.7 Viabilidade económica

Embora todas as vantagens do ponto de vista operacional são favoráveis, qualquer projeto tem de ter viabilidade económica. No entanto, nesta implementação concretamente, não sãoesperados custos altos. Isto deve-se ao facto de apenas estarmos a fazer uma interpretação computorizada dos dados que já foram recolhidos e que estão a ser recolhidos no momento. A única alteração do ponto de vista estrutural é a eventual adição de sensores, no entanto os custos não serão significativos pois são equipamentos de pouco valor.

Então é possível concluir que o projeto terá viabilidade económica, pois devido ao seu baixoinvestimento, basta ter resultados ligeiramente positivos para ter rendimento positivo.

Outro indicativo de que a integração desta tecnologia tem sido feita com muito sucesso é a evolução na valorização que tem sofrido nos últimos anos. Na Fig. 13 podemos facilmentever um crescimento acentuado tanto no *hardware, software* e serviços. Isto reflete projetosimplementados com sucesso com resultados operacionais positivos e que levam à implementação noutras situações.



	2017	2018	2019	2020	2021	2022	2023	2024
Hardware	1,408.54	1,896.55	2,580.82	3,550.51	4,939.76	6,952.60	9,902.87	14,278.94
Software	924.39	1,272.68	1,771.77	2,494.99	3,555.10	5,127.55	7,488.55	11,078.32
Services	516.74	695.69	947.37	1,305.35	1,820.53	2,570.89	3,677.39	5,330.00

Figura 13 - Valor do *machine learning* em diferentes mercados [18]

2.4.8 Escolha do algoritmo

As redes neurais são adequadas para a previsão de defeitos em subestações devido à sua capacidade de modelar padrões complexos e não lineares presentes nos dados gerados por sistemas elétricos. [1] Subestações produzem uma grande quantidade de dados multidimensionais, como medições de corrente, tensão e temperatura, que podem apresentar interações difíceis de identificar com técnicas tradicionais de análise. As redes neurais conseguem aprender e generalizar a partir desses dados, permitindo que detetem anomalias e tendências que podem indicar falhas iminentes. [8] Além disso, adaptam-se bem a variações e ruídos nos dados, tornando-as mais robustas. As redes neuronais têm várias vantagens tais como:

- Capacidade de Modelar Relações Não Lineares

A capacidade de aprender e modelar relações altamente não lineares entre as variáveis de entrada e saída. É crucial em muitos problemas com aplicações reais, como a previsão de grandezas elétricas, onde as relações entre variáveis raramente são lineares. Algoritmos tradicionais, como regressão linear ou árvores de decisão simples, podem falhar. [22]

- Flexibilidade e Generalização

Podem ser configuradas com várias camadas e neurónios, permitindo modelar uma variedade quase infinita de funções complexas. Tornando-se extremamente flexíveis em termos de arquitetura, e permite que sejam aplicadas a diferentes tipos de dados e problemas.

Esta flexibilidade também se traduz na capacidade de generalização, ou seja, aprender padrões de dados antigos e aplicá-los a dados novos, mesmo quando esses dados novos diferem ligeiramente dos dados de treino.

- Aprendizagem com base na experiência

A aprendizagem em redes neurais ocorre através de um processo conhecido como treino supervisionado, onde a rede ajusta pesos internos com base em exemplos de dados anteriores (entradas) e saídas correspondentes (valores reais). Inicialmente os pesos são atribuídos de forma aleatória. Permitindo que a rede aprenda diretamente com os dados e melhore as previsões ao longo do tempo, conseguindo identificar padrões. A capacidade de ajustar continuamente os parâmetros utilizando técnicas como *backpropagation* permite que as redes neurais lidem bem com dados poluídos (dados que contenham variações erráticas)

- Redes neuronais profundas (*Deep Learning*)

As redes neuronais profundas, também conhecidas como *Deep Learning*, são redes com várias camadas ocultas (*deep networks*). Elas são especialmente capazes a modelar dados complexos, como imagens, áudio e grandes conjuntos de dados temporais. Camadas mais profundas permitem que a rede capture hierarquias de padrões, identificando características simples nas camadas iniciais e características mais complexas nas camadas posteriores.

- Trabalhar com Grandes Quantidades de Dados

Redes neurais são particularmente eficazes em lidar com grandes volumes de dados. Nos problemas de previsão complexos, onde o volume de dados anteriores é muito grande, as redes neurais podem processar e extrair padrões de forma eficiente, algo que algoritmos mais simples podem não conseguir devido à falta de capacidade computacional ou limitações na modelação de dados complexos.

- Capacidade de "Aprender" Características Importantes

Têm a capacidade de aprender automaticamente quais características dos dados são mais importantes para a tarefa de previsão. Diferente de muitos algoritmos tradicionais, onde as características precisam ser selecionadas e processadas manualmente, as redes neurais ajustam os parâmetros internos para destacar os atributos mais relevantes, eliminando a necessidade de selecionar manualmente variáveis.

- Treino Paralelo e Aceleração por Hardware

Redes neurais, especialmente as profundas, podem tirar proveito da GPU (*Graphics Processing Unit*) e outras tecnologias de hardware para acelerar significativamente o processo de treino. Permitindo que modelos de grande dimensão sejam treinados em prazos viáveis, algo que seria impossível com muitos outros algoritmos.

2.4.9 LSTMs e híper parâmetros

As redes neurais do tipo LSTM são mais adequadas para este tipo de previsão porque são projetadas para trabalhar com dados sequenciais e temporais, conseguindo assim detetar dependências de longo prazo entre eventos. As LSTMs conseguem manter informações de estados anteriores recorrendo a células de memória, o que é crucial para sistemas de energia, onde os defeitos podem resultar de mudanças graduais ao longo do tempo. [28]

As LSTMs funcionam por portas que funcionam segundo o diagrama da Fig. 14 e são identificadas como:
$f_t : porta\ de\ esquecimento$
$i_t : porta\ de\ entrada$
$o_t : porta\ de\ saida$

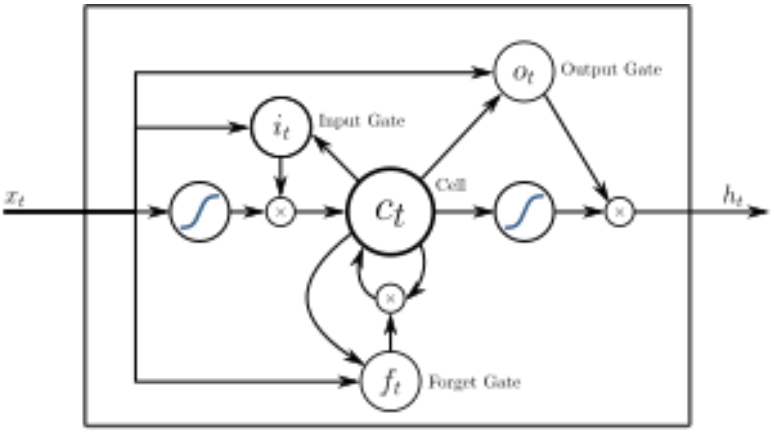


Figura 14 – Esquema funcionamento rede neuronal LSTM [13]

2.4.10 Métricas e avaliação de redes neuronais

Enquanto modelo matemático com alta complexidade, a sua avaliação não é simples e linear. Há muitas variáveis a considerar e possíveis resultados a avaliar.

Sendo assim tem-se:

Taxa de sucesso (*Success Rate*)

Relevante para análise generalizada, mede a percentagem de previsões (verdadeiras ou falsas) certas com o cálculo:

$$Taxa\ de\ sucesso\ = \frac{VP+VN}{VP+VN+FP+FN} * 100 \quad (1)$$

Precisão (*Precision*)

Mede a percentagem de previsões certas dentro de uma classe:

$$Precisão\ = \frac{VP}{VP+FP} \quad (2)$$

Recordar (*Recall*)

Mede a capacidade de identificação correta de todas as amostras dentro de uma classe especifica:

$$Recordar\ = \frac{VP}{VP+FN} \quad (3)$$

Erro médio absoluto (MAE)

O MAE calcula a diferença média absoluta entre o previsto e o real.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |yi - \hat{y}i|\ ,\ n \in \mathbb{N} \quad (4)$$

Erro quadrático médio (MSE)

O MSE calcula a diferença média absoluta entre o previsto e o real ao quadrado, dando ênfase a discrepâncias maiores pois aumenta de forma quadrática:

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (yi - \hat{y}i)^2\ ,\ n \in \mathbb{N} \quad (5)$$

Raiz do erro quadrático médio (RMSE)

O RMSE, á semelhança do MAE e do MSE calcula um erro com base na diferença média absoluta entre o previsto e o real. No entanto, apos o cálculo faz a raiz quadrada do resultado. Fazendo com que seja possível ser feita uma interpretação mais intuitiva pois fica na mesma escala que os erros.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (yi - \hat{y}i)^2}\ ,\ n \in \mathbb{N} \quad (6)$$

Coeficiente de determinação (R^2)

O R^2 calcula a variabilidade dos dados, dando uma visão geral da *performance* do modelo com base nos erros.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(yi-\hat{y}i)^2}{\sum_{i=1}^{n}(yi-\bar{y}i)^2},\ n \in \mathbb{N} \quad (7)$$

Legenda das equações:

$VP - Verdadeiros\ positivos$
$VN - Verdadeiros\ negativos$
$FP - Falsos\ positivos$
$FN - Falsos\ negativos$
$yi - valor\ real$
$\hat{y}i - valor\ previsto$
$\bar{y}i - valor\ da\ média\ do\ real$
$n - numero\ total\ de\ amostras$

3 Metodologia

A metodologia assenta na construção de uma rede neuronal LSTM para prever tensões e correntes com dados simulados, munido com a capacidade de detetar defeitos como queda de tensão, sobrecarga de corrente e variações assimétricas entre fases. [25] A LSTM foi escolhida devido à capacidade de processar dados sequenciais, como séries temporais de correntes e tensões. [9]

Pretende-se encontrar os parâmetros ideais para rede sendo que a amostra tem 2 anos de duração.

Nas linhas aéreas de transmissão de energia, poderemos ter várias situações que levam á existência de defeitos, no entanto apenas serão analisados os defeitos possíveis de prever recorrendo a ML.

3.1 Organização

Nesta parte do trabalho, o principal objetivo é explicar detalhadamente qual o pensamento em que se baseiam as decisões tomadas para o desenvolvimento e o código desenvolvido. É feita inicialmente a interpretação de dados que permite perceber o porque das escolhas no desenvolvimento do código posteriormente explicado.

3.2 Interpretação de dados

É relevante analisar como é que os valores da tensão e da corrente variam imediatamente antes de acontecerem os diferentes cenários de defeito, de forma a se definirem limites e valores que são as delimitações entre defeitos e a normalidade

3.2.1 Interpretação de Curto-Circuitos

A seguinte tabela mostra os cenários de defeito e como os valores de tensão e de corrente variam, pode-se também observar na figura 10 os diferentes defeitos de CC.

Tabela 2 - Cenários Curto-circuito (Momento Imediatamente antes)

Cenários CC	Tensão (V)	Corrente (I)
Fase-fase sem rutura de condutor	Mantém	Mantém
Fase-fase com rutura de um dos condutores	Há alterações *	Há alterações – a corrente na fase cortada é zero
Fase-fase com rutura de ambos os condutores	Há alterações *	Há alterações – a corrente nas fases cortadas é zero
Fase-fase-fase sem rutura de condutores	Mantém	Mantém
Fase-fase-fase com rutura de um dos condutores	Há alterações *	Há alterações - A corrente na fase cortada é zero
Fase-fase-fase com rutura de dois dos condutores	Há alterações *	Há alterações - A corrente nas fases cortadas é zero
Fase-fase-fase com rutura dos três condutores	Há alterações *	Há alterações - A corrente é zero
Fase-terra	Mantém	Mantém
Fase-terra e fase-terra	Mantém	Mantém
Fase-terra e fase-terra e fase-terra	Mantém	Mantém
Fase-terra com rutura	Há alterações *	Há alterações – a corrente na fase cortada é zero
Fase-terra e fase-terra com ruturas	Há alterações *	Há alterações – a corrente nas fases cortadas é zero
Fase-terra e fase-terra e fase-terra com ruturas	Há alterações *	Há alterações – a corrente é zero

*- As alterações na tensão são apenas por uma questão de assimetrias quando o sistema deixa de ser equilibrado devido a roturas nos condutores, neste caso não será relevante para a prevenção de defeitos. No entanto como será utilizado ML, poderá ser encontrado um padrão relevante com base na tensão.

Dado isto, o algoritmo terá de conseguir interpretar os valores da seguinte forma:

Tabela 3 - Interpretação de possível defeito CC com ML

Tensões (kV)			Correntes (A)			Resultado
V1	V2	V3	I1	I2	I3	
60	60	60	10	10	0	Defeito
60	60	60	10	0	10	Defeito
60	60	60	0	10	10	Defeito
60	60	60	10	0	0	Defeito
60	60	60	0	0	10	Defeito
60	60	60	0	10	0	Defeito
60	60	60	0	0	0	Defeito

3.3 Divisão e explicação do código

Para facilitar interpretação e perceção do funcionamento do código em *Python* [29][20], divide-se o mesmo em 6 partes:

- Importação de bibliotecas
- Configuração de dados
- Gerar dados e inserir defeitos
- Tratamento de dados
- Criação e configuração rede neuronal LSTM
- Identificação de defeitos
- Processamento de resultados e exportação

3.3.1 Importação de bibliotecas

Escolher quais bibliotecas utilizar depende unicamente dos dados a serem processados e a forma como os mesmos são importados. [29]

Na criação dos dados com base aleatória recorre-se a *numpy*, biblioteca cuja criação assenta na computação de âmbito científico. Conjugando a *numpy* com *panda* [17], é possível facilitar a criação dos dados e a forma

como são organizados, pois com recurso a *panda* é possível a criação de *DataFrames* utilizados para guardar os dados das correntes e tensões.

Já para criar a rede neuronal recorre-se à *tensorflow,* de onde se importa a rede LSTM com a camada *Dense.* E da *scikit-learn* vem a função *train_test_split* que divide os dados entre treino e teste para ser possível fazer uma avaliação da rede. Para analisar os resultados da rede é importada *sklearn.metrics*.

Para o processamento dos resultados com gráficos usa-se *matplotlib* e a *os* para exportar diretamente os gráficos gerados e os ficheiros *excel* para o computador. A biblioteca *time* é meramente para facilitar a contagem do tempo de processamento total.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import time
import os from tensorflow.keras.models
import Sequential from tensorflow.keras.layers
import Dense, LSTM from sklearn.preprocessing
import MinMaxScaler from sklearn.model_selection
import train_test_split
import sklearn.metrics
```

3.3.2 Configuração de dados

Inicialmente definem-se as datas de inicio e fim da amostra e logo de seguida é gerado numero de amostras total, isto com base no tempo de amostragem *freq* definido também nesta fase. É também definida uma *seed* que faz com que, apesar de aleatória, a amostra gerada seja sempre a mesma.

```
# Definir frequencia de amostragem
freq='10T'

timestamp = time.strftime("%Y%m%d-%H%M%S")
output_folder = f'/content/drive/MyDrive/Tese/scenario_{timestamp}'
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Definir datas
start_date = pd.Timestamp('2022-01-01')
end_date = pd.Timestamp('2024-01-01')

# Função para calcular o número de amostras
def calculate_samples(start_date, end_date, freq=freq):
    return len(pd.date_range(start=start_date, end=end_date, freq=freq))

np.random.seed(42)
# Calcular o número de amostras e o intervalo de datas
n_samples = calculate_samples(start_date, end_date)
```

3.3.3 Gerar dados e inserir defeitos

Para criar dados que poderiam simular uma situação real, é necessário criar 3 valores de tensão (Va, Vb e Vc) e 3 valores de currentes (Ia, Ib e Ic). Na função *generate_smooth_currents* é possível definir qual a percentagem de existirem defeitos individualmente nos dados alterando os valores de *zero_prob, asymetry_prob* e *overload_pro*b.

```
# Gerar tensões simuladas
voltages = np.random.uniform(56, 60, (n_samples, 3))  # Tensões nas 3 fases

# Gerar correntes simuladas
def generate_smooth_currents(n_samples, zero_prob=0.01, asymmetry_prob=0.10,
overload_prob=0.02):
    base_current = 8 + np.random.normal(0, 0.8, n_samples)  # Valor base com média de
8A e desvio padrão maior de 0.8
    variation_percentage = np.random.uniform(0.05, 0.10, (n_samples, 3))  # Variação
mais acentuada entre 5% e 10%

    # Aplicar variação nas 3 fases com valores aleatórios ao longo do tempo
    currents = base_current[:, np.newaxis] * (1 + variation_percentage)  # Aplicar
variação nas 3 fases
       # Inserir defeitos aleatórios
    for i in range(n_samples):
        if np.random.rand() < zero_prob:  # Simular corrente zero
            phase = np.random.choice([0, 1, 2])
            currents[i, phase] = 0
        # Inserir assimetria entre as fases
        elif np.random.rand() < asymmetry_prob:
            phase = np.random.choice([0, 1, 2])
            factor = 1 + np.random.uniform(0.15, 0.3)  # Simular assimetria
            currents[i, phase] *= factor
        # Inserir sobrecarga aleatória
        elif np.random.rand() < overload_prob:
            phase = np.random.choice([0, 1, 2])
            currents[i, phase] = np.random.uniform(9, 11)  # Sobrecarga aleatória
entre 9A e 11A
    currents = np.clip(currents, 0, 11)  # Limitar os valores entre 0A e 10A
    return currents


currents = generate_smooth_currents(n_samples, zero_prob=0.01, asymmetry_prob=0.1,
overload_prob=0.02)
```

Após estarem os intervalos de tensões, de correntes e os defeitos, organizam-se os dados associando valores a variáveis e datas.

```
# Criar DataFrame com dados e datas
data = np.hstack((voltages, currents))
df = pd.DataFrame(data, columns=['V_a', 'V_b', 'V_c', 'I_a', 'I_b', 'I_c'])
date_range = pd.date_range(start=start_date, periods=n_samples, freq=freq)
df['DateTime'] = date_range
```

3.3.4 Tratamento de dados

A função *create_sequences* é responsável por gerar sequências temporais a partir dos dados de entrada. Prepara o conjunto de dados de modo que o modelo LSTM possa processar dados que têm uma estrutura de dependência temporal.

```
# Função para criar sequências temporais
def create_sequences(data, time_steps):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i:i + time_steps])
        y.append(data[i + time_steps])
    return np.array(X), np.array(y)

time_steps = 10
data = df[['V_a', 'V_b', 'V_c', 'I_a', 'I_b', 'I_c']].values
X, y = create_sequences(data, time_steps)
dates = df['DateTime'][time_steps:].reset_index(drop=True)  # Datas correspondentes às
amostras y
```

A normalização é crucial quando se trabalha com redes neurais, dá a garantia que os dados estejam numa escala adequada, o que acaba por melhorar o desempenho da rede.

```
# Normalizar os dados
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X.reshape(-1, X.shape[-1])).reshape(X.shape)
y_scaler = MinMaxScaler()
y_scaled = y_scaler.fit_transform(y)
```

3.3.5 Criação e configuração rede neuronal

Primeiro é necessário dividir os dados em treino e teste.

```
# Dividir os dados em treino e teste
X_train, X_test, y_train, y_test, dates_train, dates_test = train_test_split(
    X_scaled, y_scaled, dates, test_size=0.1, random_state=42
)
```

Seguidamente é a fase de criar e definir parâmetros da rede neuronal lstm, neste exemplo está configurada com 32 neurónios por camada, a função *relu,* uma camada *dense* de 6 neurónios, otimizador *adam* e *mse* que faz com que o erro quadrático seja minimizado. [12][26]

```
# Construir o modelo LSTM
model = Sequential()
model.add(LSTM(32, activation='relu', input_shape=(time_steps, X.shape[2])))
model.add(Dense(6))
model.compile(optimizer='adam', loss='mse')
```

No processo de treino da rede, definem-se os *epochs* e *batch size.* Após estar tudo devidamente configurado fazem-se as previsões.

```
# Treinar a rede
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test,
y_test))
# Fazer previsões
```

```
y_pred = model.predict(X_test)
y_pred_rescaled = y_scaler.inverse_transform(y_pred)
```

3.3.6 Identificação de defeitos

Os defeitos são definidos em função de três valores, tensão mínima, corrente máxima e assimetria máxima aceite entre correntes das fases.

```
thresholds = {
    'voltage_min': 57,  # Definir tensão mínima
    'current_max': 9    # Definir corrente máxima
}
```

A função *detect_defects* é criada para detetar os defeitos com base nos limites previamente estabelecidos. [11]

```
# Função para identificar defeitos com base nos dados
def detect_defects(data, thresholds):
    defects = []
    voltage_min = thresholds['voltage_min']
    current_max = thresholds['current_max']

    for sample in data:
        sample_defects = []
        # Verificar se há valores zero (corrente ou tensão)
        if np.any(sample == 0):
            sample_defects.append('Corrente ou tensão zero')

        # Verificar variações >10% entre correntes
        mean_sample = np.mean(sample[3:])
        if np.any(np.abs(sample[3:] - mean_sample) / mean_sample > 0.1):
            sample_defects.append('Variação >10% entre correntes')

        # Verificar sobrecarga de corrente
        if np.any(sample[3:] > current_max):
            sample_defects.append('Sobrecarga de corrente')

        # Verificar tensão minima
        if np.any(sample[:3] < voltage_min):
            sample_defects.append('Queda de tensão')

        defects.append(sample_defects)

    return defects
```

Os dados são também revertidos para os valores originais (previamente a escala foi alterada para o seu processamento) e de seguida faz-se a deteção dos defeitos nos dados simulados e nos previstos.

```
y_test_rescaled = y_scaler.inverse_transform(y_test)  # Desnormalizar os dados reais
defects_real = detect_defects(y_test_rescaled, thresholds)
defects_pred = detect_defects(y_pred_rescaled, thresholds)
```

3.3.7 Cálculo de erros e de indicadores

Os erros que regem o sucesso da implementação de uma rede neural anteriormente descritos, são calculados recorrendo á biblioteca *sklearn.metrics* importada ou através de calculo direto:

```
mae = mean_absolute_error(y_test_rescaled, y_pred_rescaled)
rmse = np.sqrt(mean_squared_error(y_test_rescaled, y_pred_rescaled))
r2 = r2_score(y_test_rescaled, y_pred_rescaled)
mse = mean_squared_error(y_test_rescaled, y_pred_rescaled)
precision = num_true_positives / (num_true_positives+num_false_positives)
recall = num_true_positives / (num_true_positives+num_false_negatives)
f1 = 2*(precision*recall)/(precision+recall)
success_rate = correct_detections / total_defects_real * 100 if total_defects_real > 0
else 0
```

3.3.8 Processamento de resultados e exportação

Organizar os defeitos detectados em dois *DataFrames* (*defects_real_df* e *defects_pred_df*) contendo as datas (*DateTime*) e os defeitos reais e previstos.

```
# Converter lista de defeitos para DataFrames
defects_real_df = pd.DataFrame({'DateTime': dates_test, 'Defeito_Real': defects_real})
defects_pred_df = pd.DataFrame({'DateTime': dates_test, 'Defeito_Pred': defects_pred})

# Usar explode para separar múltiplos defeitos em linhas diferentes
defects_real_df = defects_real_df.explode('Defeito_Real')
defects_pred_df = defects_pred_df.explode('Defeito_Pred')
```

Combina os *DataFrames* de defeitos reais e previstos com base nas datas (*DateTime*), permitindo comparações linha a linha.

```
# Mesclar os DataFrames para comparar defeitos reais e previstos
comparison_df = defects_real_df.merge(defects_pred_df, on='DateTime', how='outer')
comparison_df['Match'] = comparison_df['Defeito_Real'] ==
comparison_df['Defeito_Pred']
```

Soma os valores *True* da coluna *Match*, que representa quantos defeitos foram corretamente previstos.

success_rate: A taxa de sucesso geral é calculada como a proporção de defeitos corretamente detetados sobre o total de defeitos reais, multiplicada por 100 para obter percentagem.

```
# Contar defeitos corretamente detectados
correct_detections = comparison_df['Match'].sum()
total_defects_real = defects_real_df['Defeito_Real'].count()

# Calcular a taxa de sucesso (Success Rate)
success_rate = correct_detections / total_defects_real * 100 if total_defects_real > 0
else 0
```

total_defects: Conta o número de vezes que cada tipo de defeito apareceu nos dados reais.

correct_defects: Conta quantas vezes cada defeito foi corretamente detetado.

success_rate_per_defect: Calcula a taxa de sucesso para cada tipo de defeito, dividindo o número de deteções corretas pelo total do defeito.

```
# Contagem de defeitos por tipo
total_defects = defects_real_df['Defeito_Real'].value_counts().to_dict()
correct_defects = comparison_df[comparison_df['Match'] ==
True]['Defeito_Real'].value_counts().to_dict()

# Calcular a taxa de sucesso para cada defeito
success_rate_per_defect = {defect: (correct_defects.get(defect, 0) / total) * 100
                           for defect, total in total_defects.items()}

# Mostrar resultados
print(f"Total de defeitos reais: {total_defects_real}")
print(f"Defeitos corretamente detectados: {correct_detections}")
print(f"Taxa de sucesso geral: {success_rate:.2f}%")
```

Gera gráficos que comparam as variáveis simuladas e as previstas pela rede para cada uma das variáveis (tensões e correntes: Va, Vb, Vc, Ia, Ib, Ic).

Cada gráfico é guardado numa pasta (output_folder definida no início) com resolução de 300 DPI. [10][16]

```
for var, color_sim, color_pred in zip(variables, colors_simulated, colors_predicted):
    plt.plot(df_simulated_filtered['DateTime'], df_simulated_filtered[var],
label=f'Simulated {var}', color=color_sim)
    plt.plot(df_predicted_filtered['DateTime'], df_predicted_filtered[var],
label=f'Predicted {var}', color=color_pred)
    # Configuração do gráfico (eixo x, rótulos, título, etc.)
    plt.savefig(f'{output_folder}/{var}_comparison.png', dpi=300)
    plt.show()
```

false_positives: Previsões de defeitos onde não há um defeito real.

true_positives: Defeitos corretamente detetados (previsões corretas).

false_negatives: Defeitos reais que o modelo não conseguiu prever.

```
false_positives = comparison_df[comparison_df['Defeito_Real'].isna() &
comparison_df['Defeito_Pred'].notna()]
true_positives = comparison_df[comparison_df['Match'] == True]
false_negatives = comparison_df[comparison_df['Defeito_Real'].notna() &
comparison_df['Defeito_Pred'].isna()]
```

Os resultados das comparações, incluindo defeitos reais e previstos, falsos positivos/negativos e a taxa de sucesso por tipo de defeito, são exportados para dois arquivos Excel (Estatistica.xlsx e Real e previsao.xlsx).

```
with pd.ExcelWriter(f'{output_folder}/Estatistica.xlsx') as writer:
    results_df.to_excel(writer, sheet_name='Comparison', index=False)
    false_positives.to_excel(writer, sheet_name='False_Positives', index=False)
    true_positives.to_excel(writer, sheet_name='True_Positives', index=False)
    false_negatives.to_excel(writer, sheet_name='False_Negatives', index=False)

with pd.ExcelWriter(f'{output_folder}/Real e previsao.xlsx') as writer:
    df_simulated.to_excel(writer, sheet_name='Simulated Data', index=False)
    df_predicted.to_excel(writer, sheet_name='Predicted Data', index=False)
    summary_df.to_excel(writer, sheet_name='Summary', index=False)
```

4 Resultados

Após o código ser desenvolvido, de forma a se poderem tirar conclusões, são elaborados vários cenários que permitem perceber a melhor forma de usar o algoritmo. O código foi sempre executado no *Google Colab* [6]. Os seguintes parâmetros serão alterados até chegar aos resultados mais significantes:

- Frequência da amostra
- *Test size*
- *Batch size*
- Número de neurónios
- *Epochs*

É feita também uma análise generalizada dos gráficos resultantes dos cenários mais favoráveis e depois uma análise detalhada da configuração final otimizada.

No início dos testes a rede encontra-se configurada da seguinte forma:

```
Freq = “10T”
X_train, X_test, y_train, y_test, dates_train, dates_test = train_test_split(
    X_scaled, y_scaled, dates, test_size=0.15, random_state=42
)
model = Sequential()
model.add(LSTM(64, activation='relu', input_shape=(time_steps, X.shape[2])))
model.add(Dense(6))
model.compile(optimizer='adam', loss='mse')

model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test,
y_test))
```

Ou seja:

- Amostragem de 10 em 10 minutos
- 0.15 *test size*
- 32 *batch size*
- 64 neurónios
- 10 *epochs*

Conforme os resultados vão sendo analisados, os parâmetros vão mudando até chegar aos parâmetros finais mais favoráveis. Sendo que o objetivo é encontrar os parâmetros que otimizam a taxa de sucesso, mas cujo tempo de processamento não poderá superar o tempo de amostragem. Isto tendo também em consideração o número de falsos positivos.

É também importante notar que a função de gerar os defeitos está com as seguintes probabilidades:

- Probabilidade de 0 é 1%
- Probabilidade de uma assimetria é 10%
- Probabilidade de sobrecarga é 2%

4.1 Otimizar os híper-parâmetros

4.1.1 Cenário 1 – Variar a Frequência da amostra

É esperado que a frequência de amostragem influencie o comportamento da rede, foram feitos testes com intervalos de 10 minutos, 30 minutos, 1 hora. A quantidade de amostras que a rede neuronal usa para treinar dita a fiabilidade dos resultados.

Sendo que o que se pretende é a previsão de defeitos, não faz sentido aumentar o tempo mais que 1 hora. Pensando num cenário real onde é necessário saber se a próxima amostra tem defeito ou não.

Analisando os resultados das iterações, o tempo de amostragem de 10 minutos tem os melhores resultados com 84,22% de taxa de sucesso, 279,52 segundos de tempo de processamento e 3,07% de erro.

Os resultados dos cenários com o tempo de amostragem de 30 minutos e 1 hora estão no anexo B.

Tabela 4 – Comparação das iterações do cenário 1

Tempos	Taxa sucesso (%)	Falsos positivos	Verdadeiros positivos	ET (%)	Tempo processamento (s)
10 minutos	84,22	504	15917	3,07	279,52
30 minutos	84,92	278	5311	4,97	103,66
1 hora	82,35	133	2591	4,89	54,32

$$Erro\ total\ (ET) = \frac{Nº\ Previsões\ erradas}{Nº\ Previsões\ certas + Nº\ Previsões\ erradas} * 100 \quad (8)$$

4.1.2 Cenário 2 - Variar *test-size*

No que diz respeito ao *test-size* é necessário fazer a interpolação para um cenário real e que valores fazem sentido para além dos resultados. Não faz sentido estar a fazer um estudo baseado em menos de 5% das amostras, daí começar as iterações no *test-size = 0.05.* Na Fig. 15 estão representados dois gráficos de tensões simuladas (preto) e tensões previstas (verde), na iteração do gráfico à esquerda foi utilizado 0.10 de *test_size*, já no gráfico à direita foi utilizado 0.20 de *test_size*. Sendo que o tempo de amostra dos gráficos são os 2 anos definidos, é percetível a diferença entre os 0.10 e os 0.20 (10% de teste e 20% de teste)

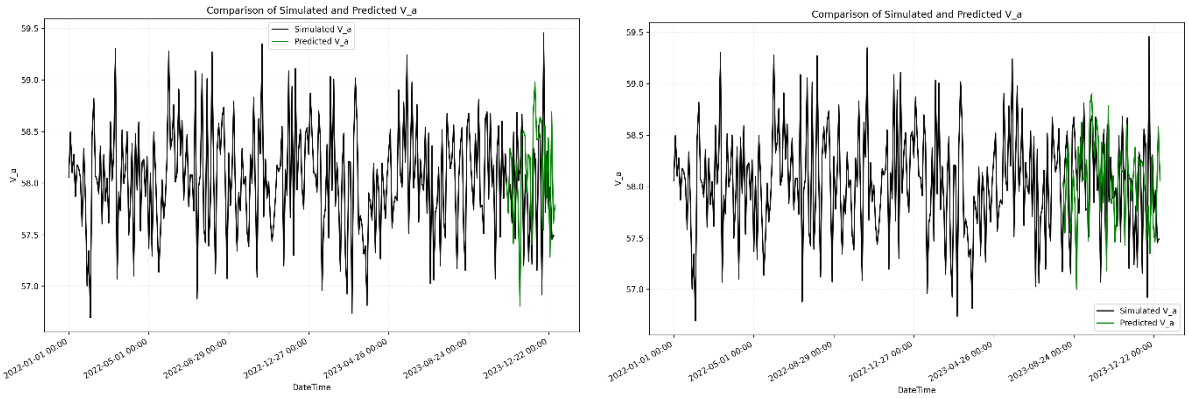


Figura 15 – Representação *test_size* 0.10 vs 0.20

Analisando a tabela 5 é possível concluir que o *test-size = 0.10* é o valor mais adequado, sendo a taxa de sucesso superior e com um erro total baixo. Apesar de o erro diminuir com o aumento do test_size, a taxa de sucesso baixa também e a um ritmo mais expressivo.

Tabela 5 – Comparação das iterações do cenário 2

test-size	Taxa sucesso (%)	Falsos positivos	Verdadeiros positivos	ET (%)	Tempo processamento (s)
0.05	85,72	250	5538	4,32	262,37
0.10	86,40	485	11235	4,14	274,76
0.15	84,14	695	16406	4,06	269,32
0.20	84,03	890	21890	3,91	273,59

4.1.3 Cenário 3 - Variar número de *batch size*

Quando se muda o valor de *batch size* é com o objetivo de melhorar os resultados em dados mais complexos. Mesmo neste caso a complexidade dos dados não ser grande, é possível ver um resultado melhor quando o valor é 8, no entanto ultrapassa o tempo máximo de processamento aceitável. Como o tempo de amostragem

são 10 minutos e o que se pretende é saber se a próxima amostra representa um defeito, o tempo de processamento máximo tolerável são 600 segundos (10 minutos).

Para o valor de *batch size* o mais favorável é 32. Os gráficos das Fig. [22-27] representa a melhor iteração.

Os gráficos das iterações com *batch-size* 1, 8, 16 e 64 estão no anexo D.

Tabela 6 – Comparação das iterações do cenário 3

Batch size	Taxa sucesso (%)	Falsos positivos	Verdadeiros positivos	ET (%)	Tempo processamento (s)
1	86,26	464	11216	3,97	3846,45
8	89,24	560	11604	4,60	840,03
16	84,50	450	10988	3,93	450,44
32	86,40	485	11235	4,14	274,76
64	86,31	516	11224	4,40	175,24

4.1.4 Cenário 4 - Variar número de neurónios

Os neurónios por camada são, á semelhança com o *batch_size,* escolhidos em função da complexidade dos dados. A execução de várias iterações é a única maneira de chegar ao número de neurónios ideal, tendo sempre em considerações as limitações do tempo de processamento. o cenário com 128 neurónios não tem o melhor resultado, mas mesmo supondo que sim, não seria possível escolher essa opção devido ao tempo de processamento apontado a amarelo.

O número de neurónios tem um impacto significativo no tempo de processamento, como em tudo ao longo do estudo, é sempre preciso equilibrar todos os parâmetros dentro das limitações impostas.

Nesta situação a taxa de sucesso mais alta é com 16 neurónios, o segundo tempo de processamento mais rápido. Os 16 neurónios reduzem um potencial risco de *overfitting* quando forem utilizados dados reais de correntes e tensões obtidos numa subestação.

Tabela 7 – Comparação das iterações do cenário 4

Neurónios	Taxa sucesso (%)	Falsos positivos	Verdadeiros positivos	ET (%)	Tempo processamento (s)

8	72,87	431	9475	4,35	263,03
16	86,97	533	11309	4,5	285,28
32	85,66	469	11138	4,04	254,10
128	84,86	397	11034	3,56	771,34

4.1.5 Cenário 5 - Variar número de *epochs*

Relativamente a variar epochs, num valor acima de 10 encontrou-se pouca vantagem a nível de resultados. Com a análise da Fig. 15, que representa um gráfico das perdas (*Training and Validation loss*) em função do número de *epochs*, é possível perceber que a partir de 10 *epochs* os valores, já baixos, de perda tendem a estagnar. A prioridade é sempre a taxa de sucesso e o tempo de processamento, e como tal, escolheu-se os 10 *epochs*. Os valores baixos de perdas não implicam obrigatoriamente um melhor resultado sendo que no modelo desenvolvido se faz a análise do seu sucesso pela deteção de defeitos e não pela aproximação a um valor real concreto.

Figura 16 – Gráfico *Training and Validation loss*

Apesar de na iteração com 10 *epochs* e na iteração de 15 *epochs* as taxas de sucesso serem muito semelhantes, dá-se sempre prioridade ao tempo de processamento mais baixo. Isto porque significa que necessita de menos poder de computação e porque quanto menos tempo houver de processamento mais rapidamente se deteta a possível falha.

Tabela 8 - Comparação das iterações do cenário 5

Epochs	Taxa sucesso (%)	Falsos positivos	Verdadeiros positivos	Erro (%)	Tempo processamento (s)
1	75,45	443	9811	4,32	94,77
10	86,97	533	11309	4,50	285,28
15	86,89	568	11472	4,72	326, 29
20	82,75	441	11298	3,76	446,76
30	86,93	473	11181	4,05	673,41

Os gráficos resultantes da iteração deste cenário mais favorável estão nas Fig. [28-34], no cenário anterior a iteração mais favorável acabou por ter os mesmos híper-parâmetros que a melhor iteração deste cenário. No anexo F é também possível analisar os gráficos resultantes de 1, 15 e 20 *epochs*.

4.2 Resultados obtidos

Todos os gráficos nos resultados obtidos deste subcapítulo representam as últimas 36h de valores simulados x previstos. Devido á grande quantidade de amostras, mais concretamente 105117 amostras, a representação gráfica percetível não é possível. Se fossem mostrados os gráficos dos 2 anos ficariam os dados todos sobrepostos ou, caso fosse feito um *resampling,* não corresponderiam exatamente ao que foi gerado.

A identificação de cada linha está no próprio gráfico, a preto o simulado e a cor o previsto. Cada *tick* do eixo x representa uma data, cada uma separada por 2 horas e começa no dia 30 de dezembro de 2023 até dia 1 de janeiro de 2024. No caso foi definido que a última amostra prevista seja à meia-noite do dia 1 de janeiro de 2024. Já os *ticks* do eixo y são automaticamente ajustados de forma que o gráfico caiba na figura e representam quilovolts (kV) referindo-se a tensões e a amperes (A) quando mostram correntes. Todos os gráficos em anexo seguem a mesma formatação descrita.

4.2.1.1 Resultados com tempo de amostragem 10 minutos

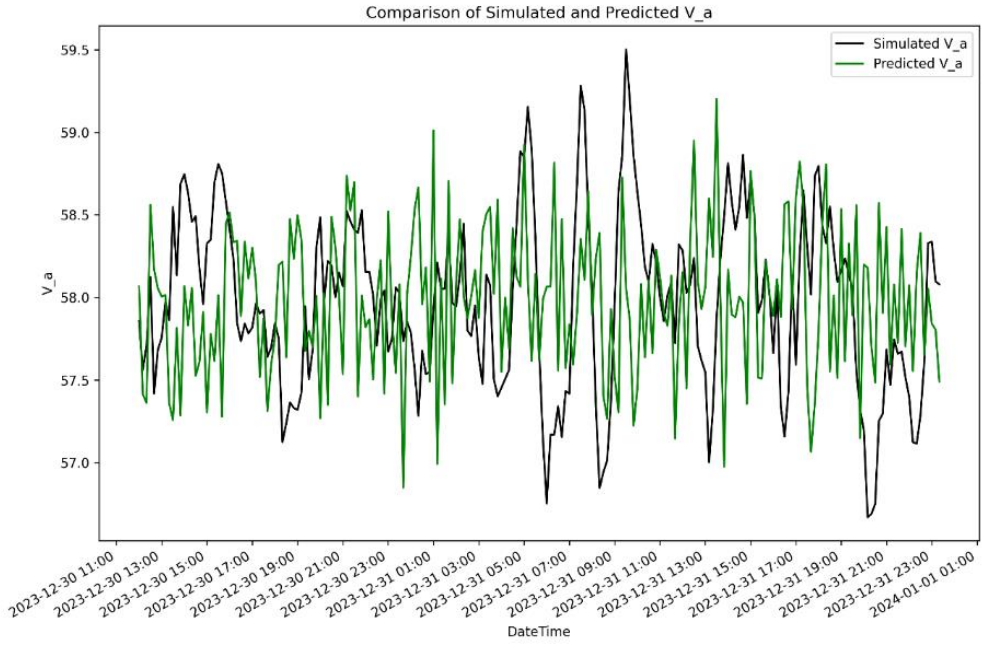


Figura 17 - Gráfico Va com 10 minuto de amostragem

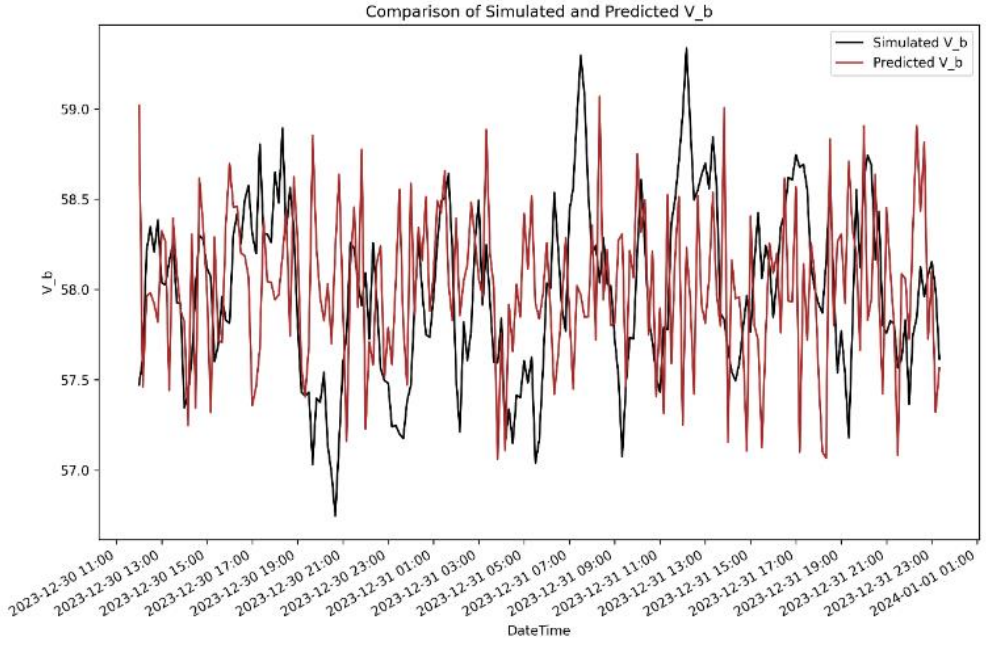


Figura 18 – Gráfico Vb com 10 minuto de amostragem

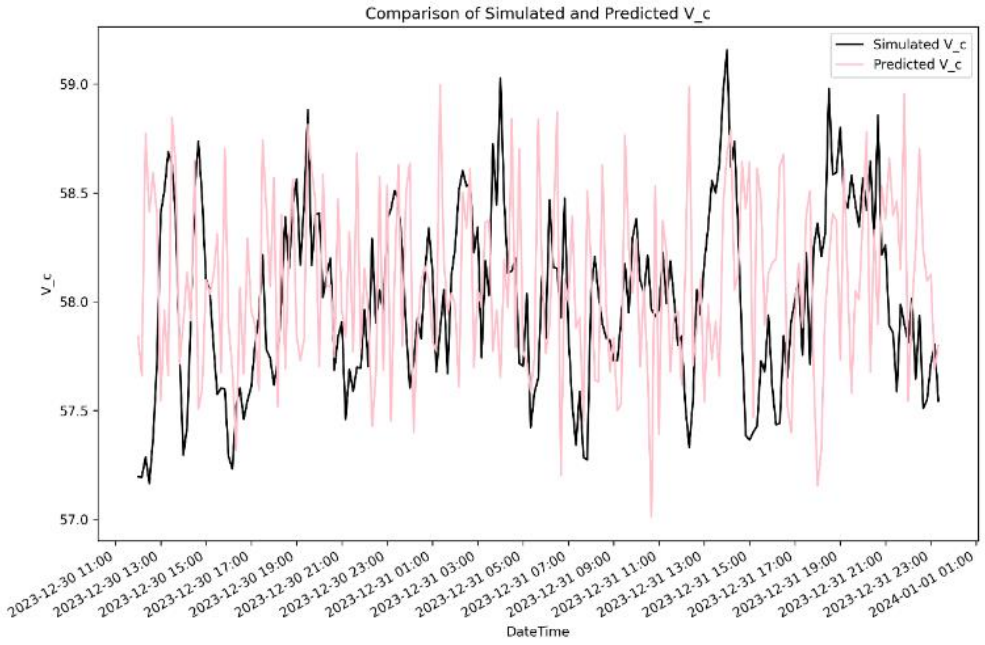


Figura 19 - Grafico Vc com 10 minutos de amostragem

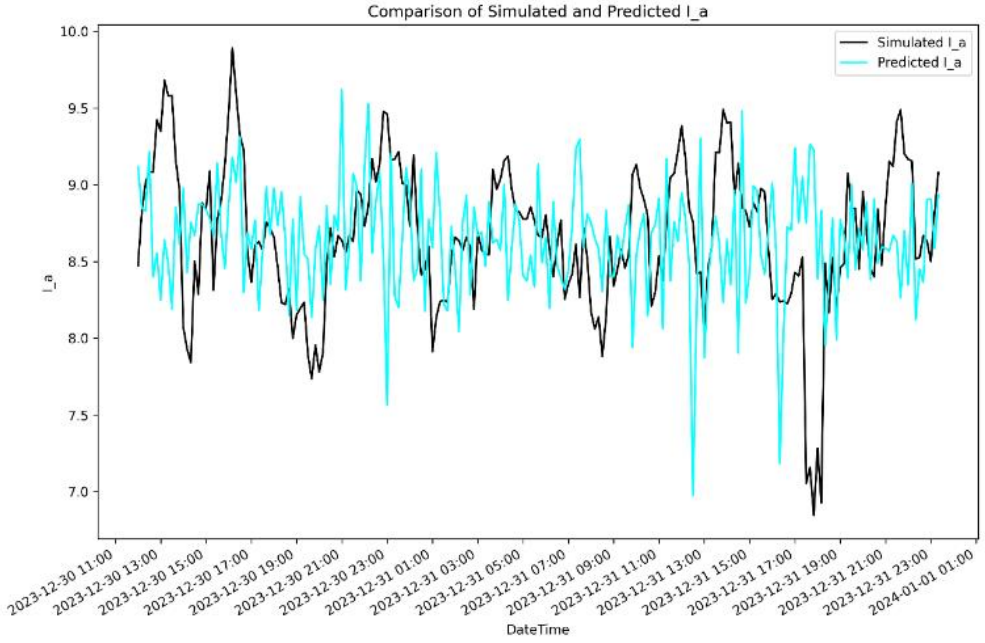


Figura 20 - Gráfico Ia com 10 minuto de amostragem

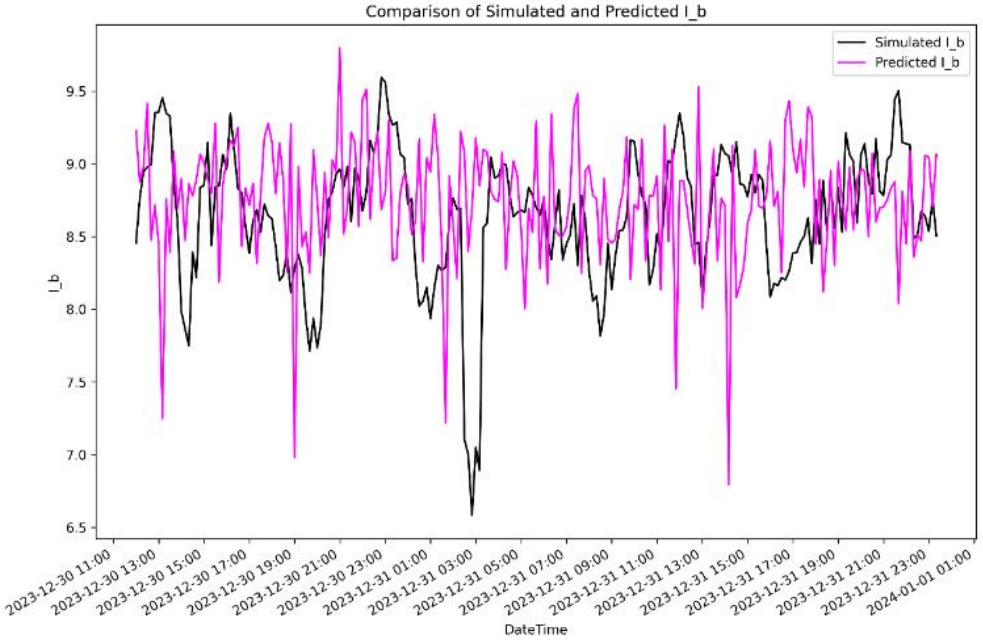


Figura 21 – Gráfico Ib com 10 minuto de amostragem

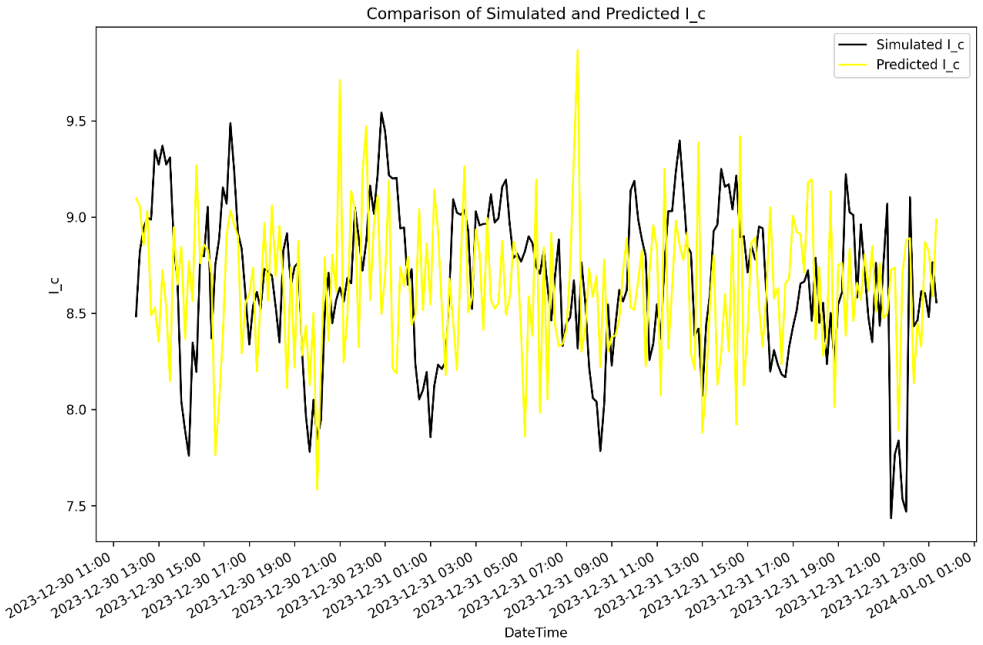


Figura 22 - Gráfico Ic com 10 minutos de amostragem

As Fig. [17-19] representam tensões, analisando esses gráficos dá para perceber que, devido a não haver introdução propositada de defeitos nas tensões, mas apenas serem geradas de forma aleatória, a rede consegue acompanhar com mais viabilidade as tensões simuladas em relação ao que é possível analisar nas

Fig. [20-22] que mostram corrente. Nos gráficos representativos das correntes, o modelo consegue acompanhar com alguma eficácia, mas percebe-se claramente uma dificuldade na deteção de picos.

Nestes gráficos, comparativamente aos gráficos das restantes iterações (Anexo B), é possível ver uma diferença significativa nos dados simulados e previstos. O tempo de amostragem de 10 minutos permite uma melhor perceção das curvas de demanda comparativamente às alternativas de 30 minutos e 1 hora.

4.2.1.2 Resultados com *test-size = 0.10*

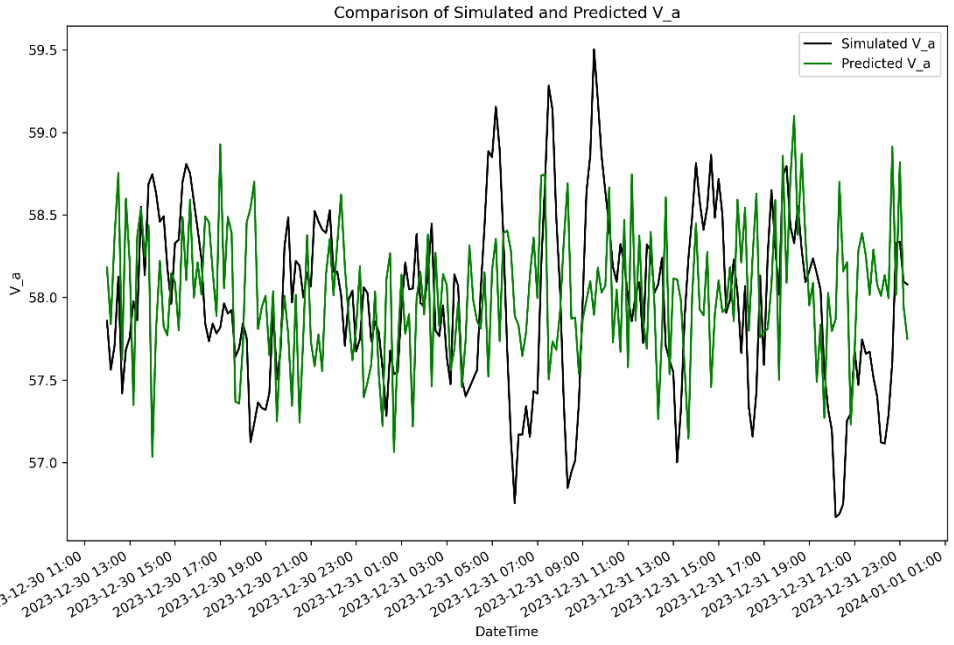


Figura 23 - Gráfico Va com *test-size = 0.10*

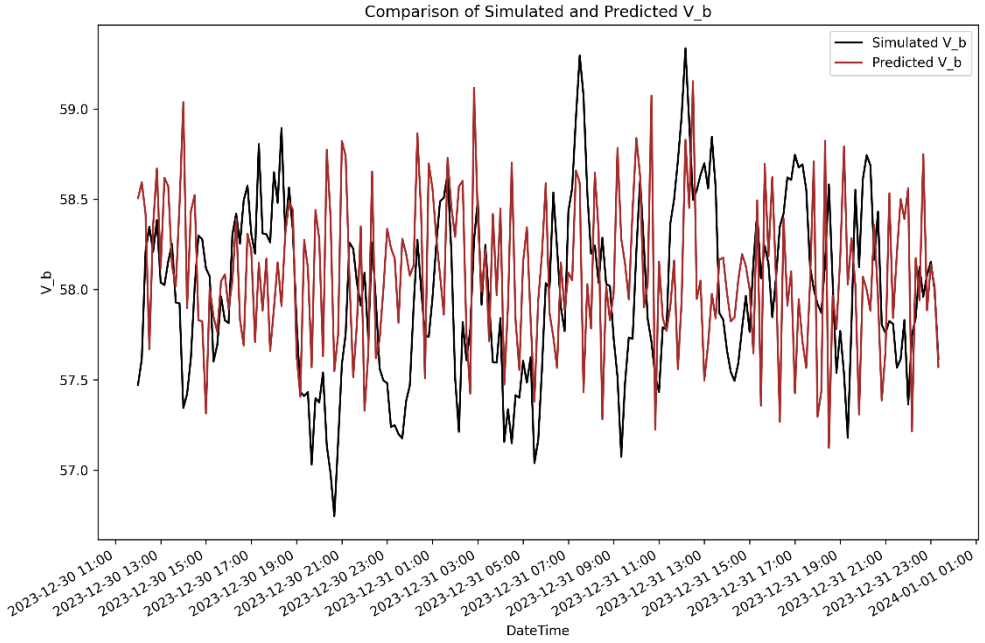


Figura 24 - Gráfico Vb com *test-size = 0.10*

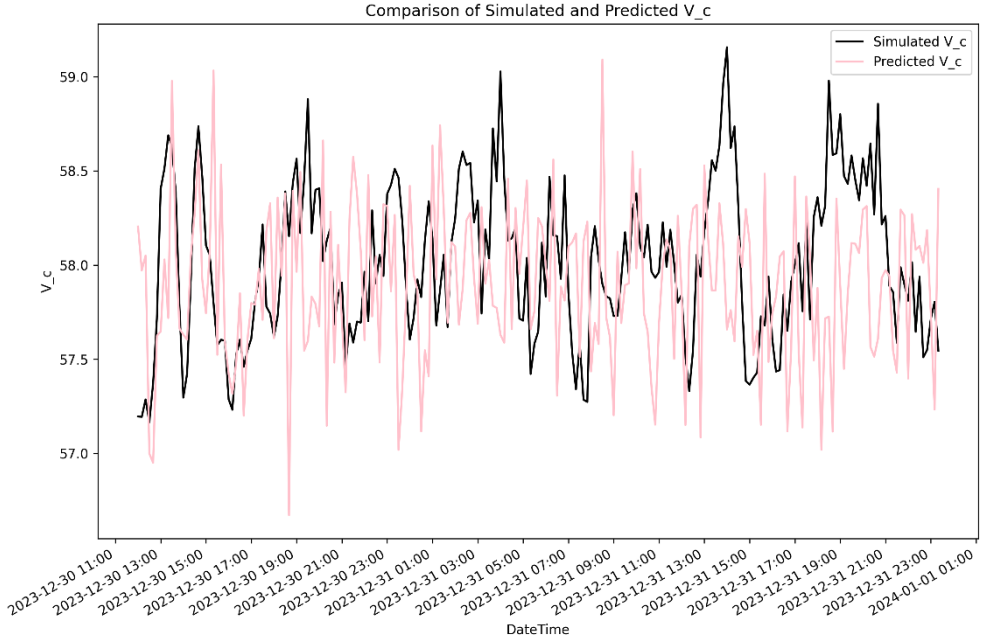


Figura 25 - Gráfico Vc com *test-size = 0.10*

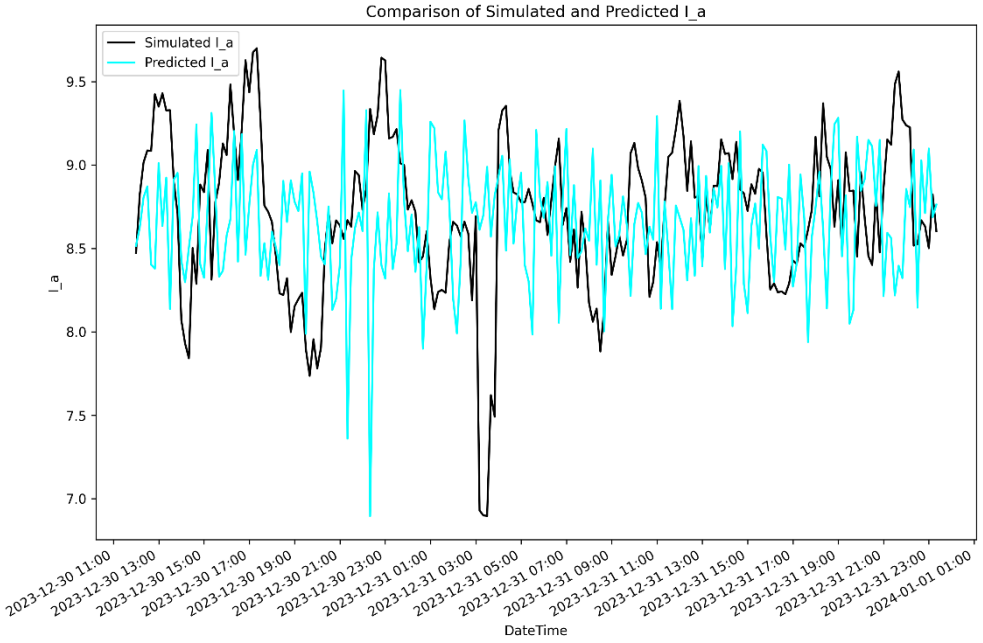


Figura 26 - Gráfico Ia com *test-size = 0.10*

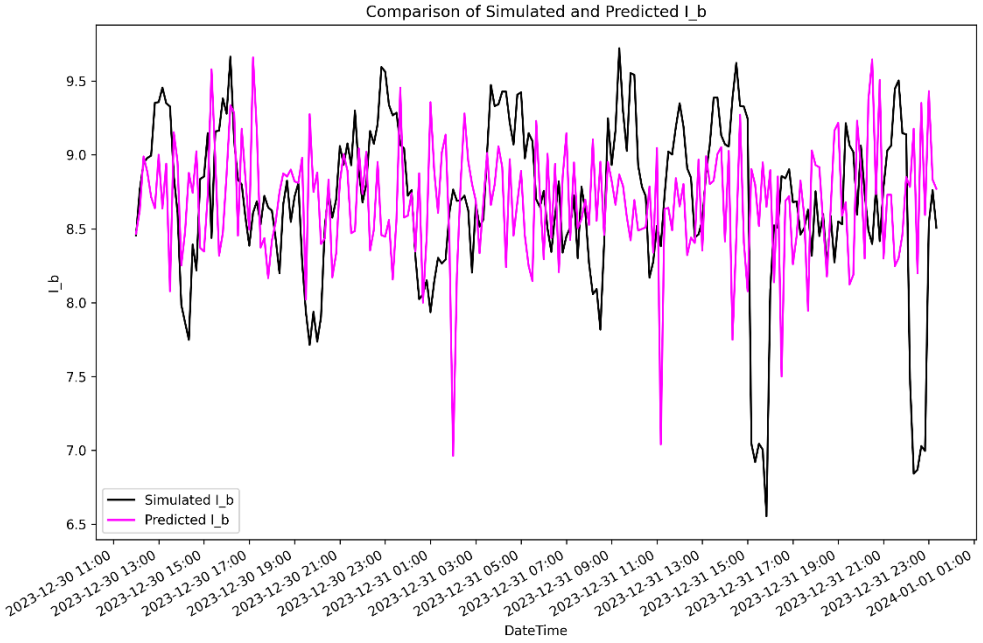


Figura 27 - Gráfico Ib com *test-size = 0.10*

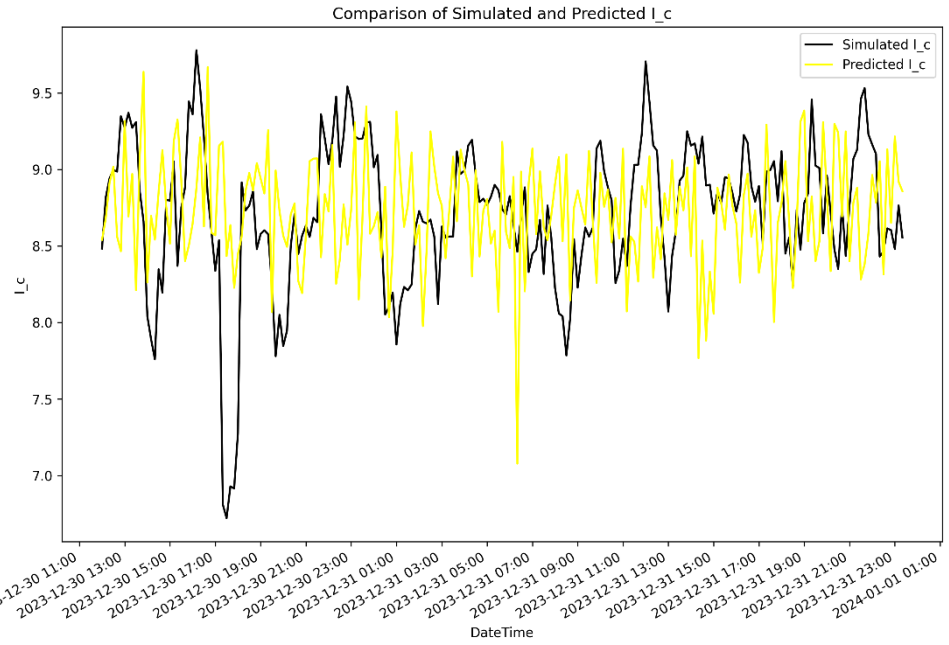


Figura 28 - Gráfico Ic com *test-size = 0.10*

Analisando os gráficos, é possível observar que a rede consegue prever com alguma precisão os dados, á semelhança do conjunto de gráficos previamente analisados cujo *test_size* é 0.15. Nas Fig. [23-25], em comparação com os restantes gráficos do anexo C e analisados no subcapítulo anterior, não são visíveis diferenças significativas. Já nas Fig. [26-28] é facilmente percetível uma diferente bastante significativa na capacidade de previsão. Nos *test_size* superiores a 0.10, consequência direta de menos treino, há várias situações de previsão de picos erradamente elaboradas. Mais concretamente comparando o gráfico da Fig. 21 com a Fig. 27 é possível contar quatro situações em que a corrente prevista é muito superior á corrente simulada, enquanto na Fig. 27 há apenas duas situações de engano significativo

Os resultados dos cenários com *test-size* 0.05, 0.15 e 0.20 estão no anexo C.

4.2.1.3 Resultados com 16 neurónios

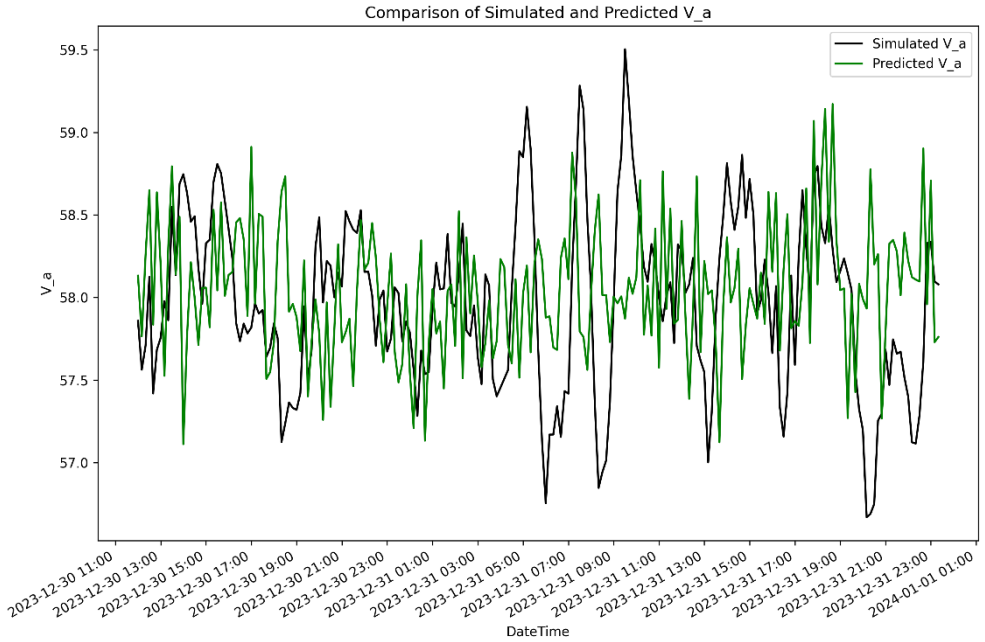


Figura 29 – Gráfico Va com 16 neurónios

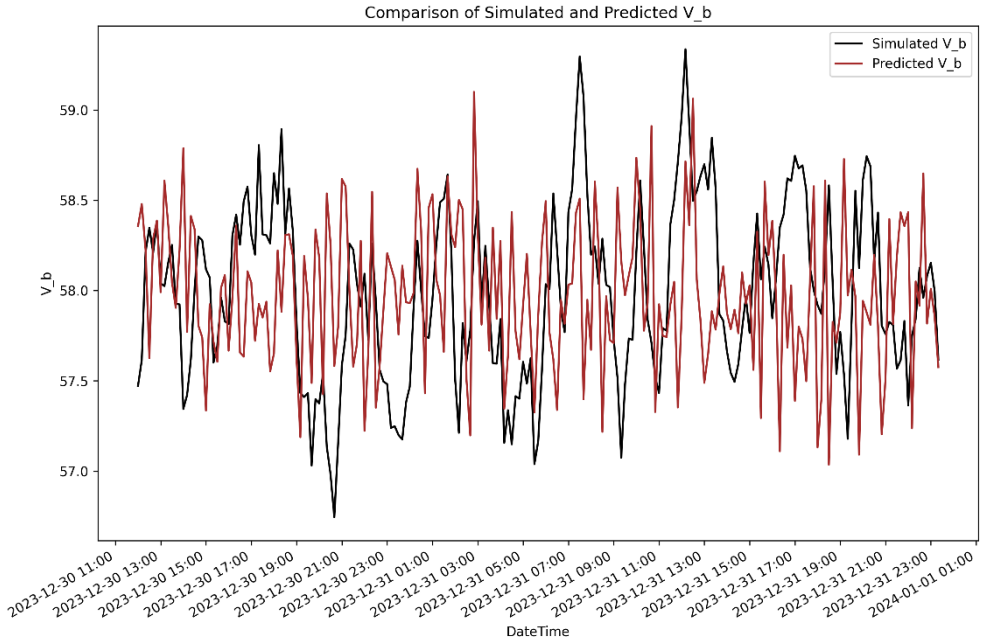


Figura 30 - Gráfico Vb com 16 neurónios

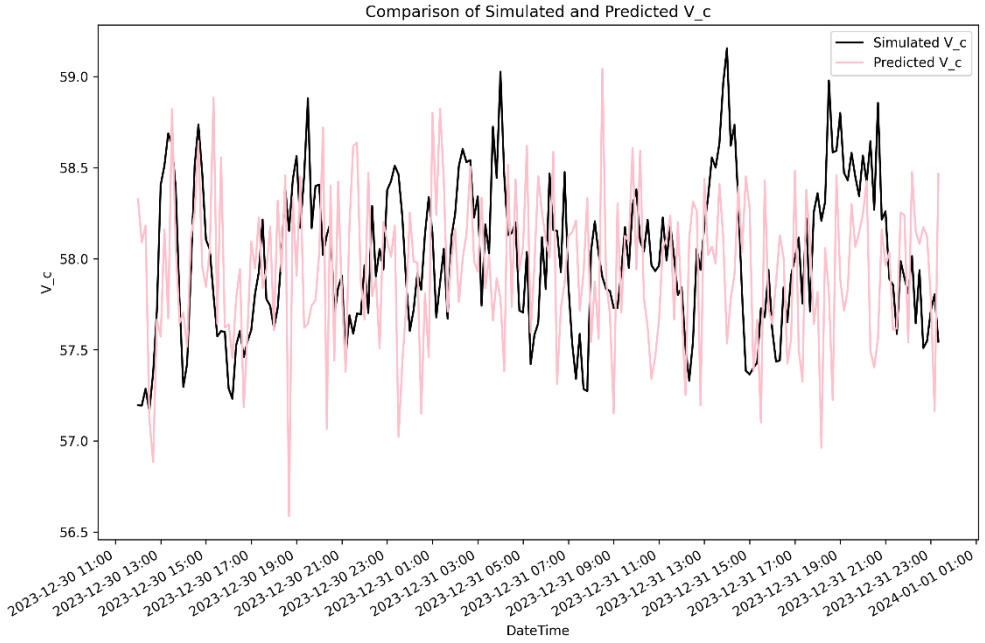


Figura 31 - Gráfico Vc com 16 neurónios

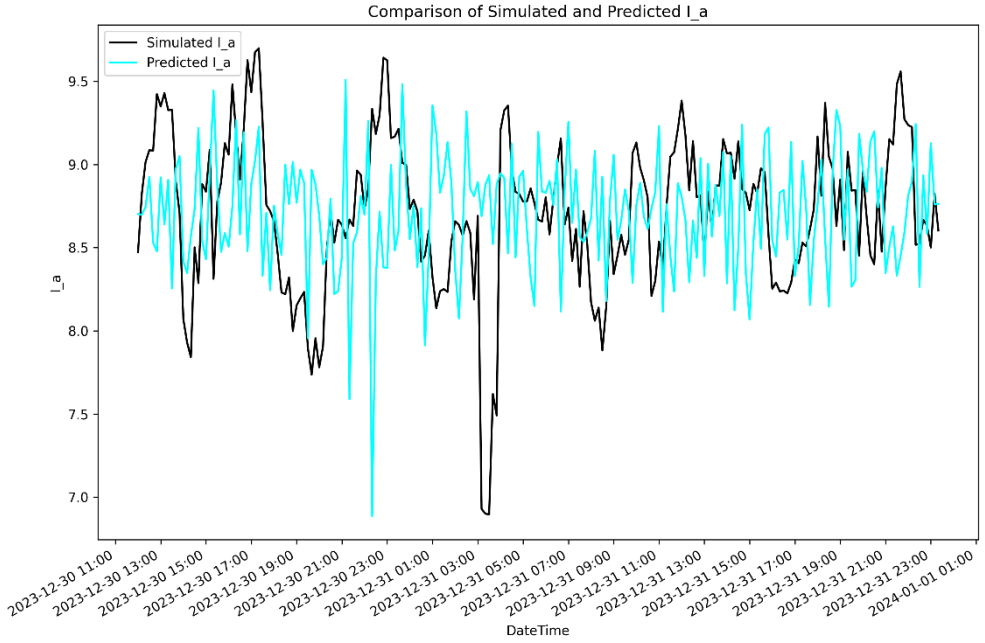


Figura 32 - Gráfico Ia com 16 neurónios

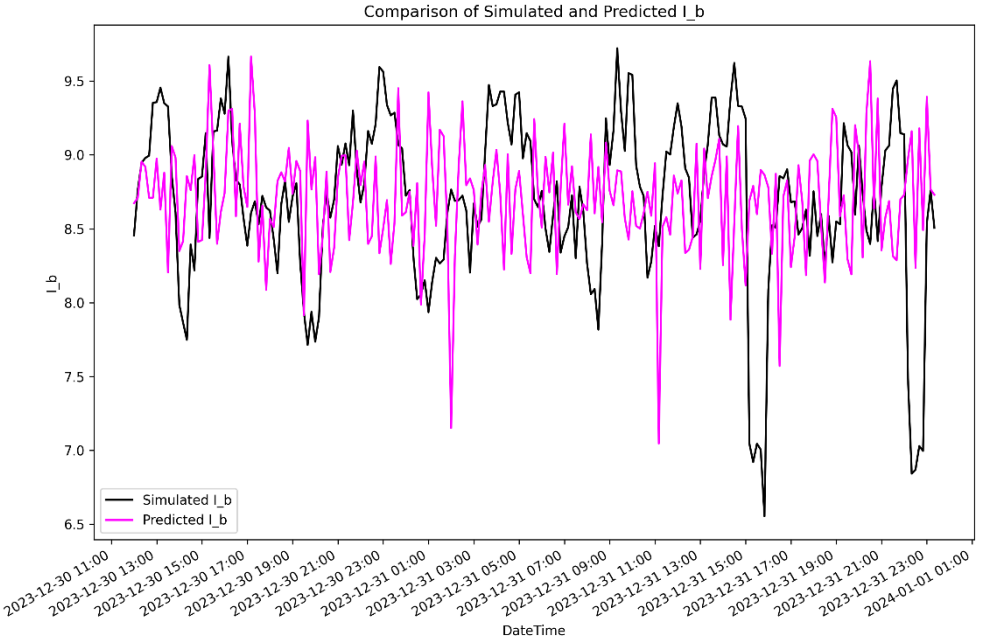


Figura 33 - Gráfico Ib com 16 neurónios

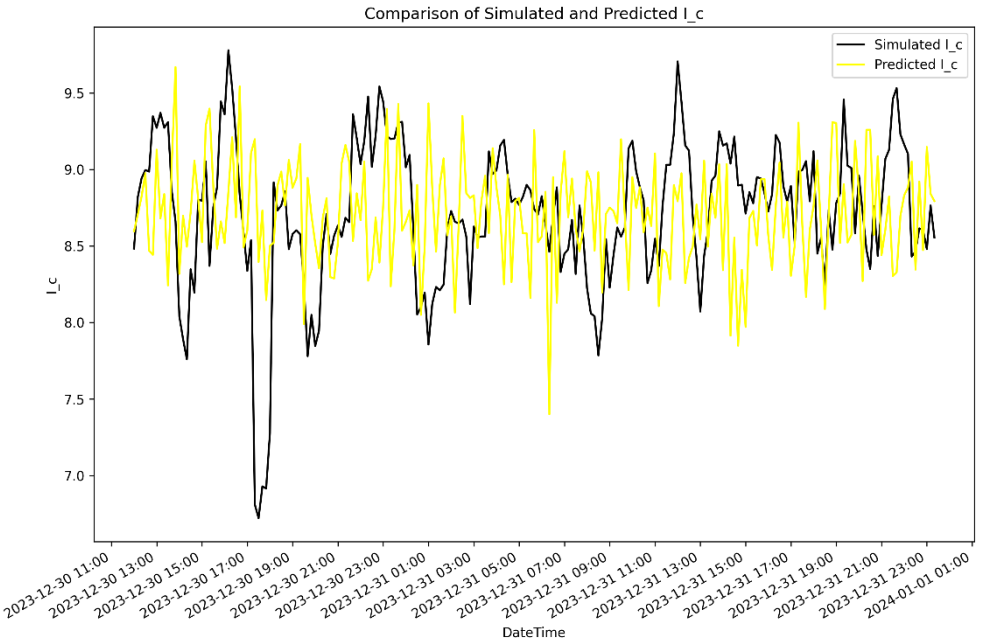


Figura 34 - Gráfico Ic com 16 neurónios

As iterações com 16 neurónios mostram que anteriormente com 64 neurónios estaria a haver *overfitting*. É possível verificar essa situação analisando tanto as tensões como as correntes. Na Fig. 23 vêem-se mais valores deslocados e picos mal previstos comparativamente á Fig. 29, sendo que ambas se referem a Va. Nas

restantes figuras, tanto nas tensões das Fig. [30-31] como nas correntes das Fig. [32-34] existe exatamente o mesmo fator diferenciador relativamente aos seus pares com 64 neurónios.

Estas diferenças e a conclusão de que o valor ideal são os 16 neurónios, implica que os dados mesmo sendo algo complexos não têm complexidade para tirar partido do um número de neurónios elevado. Normalmente quando se fazem previsões de natureza sequencial temporal, o que tem mais impacto é o tempo de amostragem e o número de *epochs.*

4.3 Comparação dos cenários

Analisando a tabela 9, por mero acaso, os parâmetros que foram definidos logo á partida estavam quase na sua forma otimizada. Pode-se chegar a esta conclusão com base no facto de que o cenário 2 e 3 partilham a melhor iteração e os cenários 4 e 5 também.

Os híper-parâmetros escolhidos dependem do que se pretende exatamente, se o objetivo é ter o menos de previsões erradas possível face ao número de previsões total ou se é conseguir prever o máximo de defeitos com a contrapartida de que há mais falsos positivos.

Tabela 9 – Comparação dos cenários

Cenários	Hiper parâmetros					Erro (%)	Taxa sucesso (%)
	Amostragem	*Test-size*	*Batch-size*	Neurónios	*Epochs*		
1	10	0.15	32	64	10	3,1	84,22
2	10	0.10	32	64	10	4,3	86,40
3	10	0.10	32	64	10	4,3	86,40
4	10	0.10	32	16	10	4,5	86,97
5	10	0.10	32	16	10	4,5	86,97

Se o pretendido for minimizar os falsos positivos, então os híper-parâmetros do cenário 1 serão os mais adequados pois é onde o menor erro foi calculado. Supondo uma situação real onde é essencial que não haja falsos positivos.

Se o objetivo for maximizar a taxa de sucesso já se adequam mais os híper-parâmetros dos cenários 4 e 5. Considerando parâmetros como os mais favoráveis com base na taxa de sucesso de 86,97%, temos os seguintes valores finais:

- 10 minutos de tempo de amostragem
- 0.10 *test-size*
- 32 *Batch-size*
- 16 neurónios

- 10 *epochs*

4.4 Análise detalhada do cenário mais favorável

Retirando o cenário com os parâmetros otimizados da tabela 9, foram calculados valores de erro e de indicadores utilizados para classificar redes neuronais.

Tabela 10 – Erros e indicadores do cenário mais favorável

MAE	MSE	RMSE	R^2	*Precision*	*Recall*	*F1-Score*
0.223	0.081	0.285	0.661	0.950	0.960	0.960

Na tabela 10 estão sete valores que permitem uma análise mais detalhada dos resultados extraídos dos que foram considerados os híper parâmetros mais favoráveis. Sendo que quatro calculam diferentes tipo de erros e três calculam indicadores.

MAE: Um valor de 0.223 para este erro significa que, em média, a previsão falha em 0.223 unidades. Indica que o modelo está bem ajustado, no entanto inviabiliza a informação da existência de *outliers.*

MSE: Neste erro os valores são elevados ao quadrado, o que consequentemente permite uma melhor deteção de *outliers*. O valor 0.081 significa que não existem muitos valores cuja previsão está significativamente deslocada do valor simulado.

RMSE:

R^2: O R^2 de 0.661 implica que 66,1% da variação nos dados é detetada e em contrapartida 33,9% das variações não são detetadas. Este valor quer dizer que ainda há margem para melhorar. Mas como existem as limitações de tempo, poderá não ser possível.

***Precision*:** A precisão de 0.95 é muito positivo e indica que 95% das previsões positivas estão corretas.

***Recall*:** Este indicador dá ênfase ao valor baixo de falsos negativos. Pois o 0.96 significa que dentro da amostra dos verdadeiros positivos, a rede conseguiu prever 96%.

***F1-Score*:** O indicador com 0.96 significa, principalmente, equilíbrio entre falsos positivos e falsos negativos.

Conclui-se que o modelo tem erros baixos e com pouca significância, calculados pelo MAE, MSE E RMSE apesar de ainda existir alguma dificuldade a detetar as variações indicado pelo R^2. No que toca ao desempenho, com base nos indicadores calculados de *precision, recall* e *F1-Score*, todos indicam para um funcionamento muito equilibrado e consequentemente que está bem configurado.

5 Conclusão

5.1 Considerações

Considerando que são valores simulados com defeitos introduzidos propositadamente, nas amostras haverá sempre defeitos. Situação que pode não representar a realidade. Idealmente teria de se correr o código com dados reais para ter uma taxa de sucesso real. É também importante notar, não só nos defeitos, mas na quantidade dos mesmos. Os resultados obtidos são fruto da existência considerável de defeitos, caso não ocorram com tanta frequência a rede neuronal terá mais dificuldade em encontrar padrões. Bastaria reduzir o limite da tensão mínima, aumentar a corrente de sobrecarga ou diminuir as probabilidades de defeito na função para os resultados da taxa de sucesso alterarem drasticamente. Quanto menos defeitos houver para serem detetados, mais difícil é a rede neuronal conseguir detetar padrões.

O resultado obtido com a taxa de sucesso mais alta (86,97%), apontam para uma rede ideal com uma configuração de 10 minutos de período de amostragem, com 16 neurónios, 0.10 de *test-size*, 32 de *batch-size* e 10 *epochs.* Isto com este conjunto de dados específico e limites definidos anteriormente, é importante frisar.

5.2 Limitações

O modelo LSTM é relativamente simples, com apenas 32 neurónios por camada. Embora possa ser funcional para problemas básicos, pode não ser suficientemente complexo para capturar padrões mais avançados em situações reais. E também por outro lado se o conjunto de dados a analisar for demasiado curto e simples, a rede neuronal poderá ter dificuldades.

5.3 Futuras possibilidades

O código desenvolvido (Anexo A), poderá facilmente ser adaptado para aceitar *inputs* por uma folha Excel invés de ser simulado. Com a existência de dados reais, todas as questões anteriormente abordadas sobre a variabilidade dos resultados, iriam desaparecer. Pois, com testes assentes em dados reais e numa futura mudança do ambiente de simulação (*Google Colab* [20]) será possível colocar automação na otimização dos hiper-parâmetros e, assim, agilizar todo o processo. Dependendo da forma como a implementação do algoritmo é feita, pode haver restrições no tempo de processamento, como também foi abordado nos resultados. Outra nota que é preciso ter em consideração são as possíveis limitações computacionais numa situação real, só fazendo simulações no equipamento específico é que se poderiam tirar conclusões.

Relativamente a eventuais melhorias, uma integração que certamente poderá beneficiar os resultados, é a adição de dados de temperatura e vento á analise. Formulando de forma que a rede neuronal inclua na previsão estas variáveis. Este acréscimo permite que se possam fazer correlações entre condições climáticas e eventos de defeito. Como referido na parte inicial do trabalho, em muitas situações os defeitos têm origens ambientais, certamente que as utilizações de dados desta natureza poderão permitir previsões mais precisas.

5.4 Balanço total

Mesmo considerando que fundamentalmente são resultados fictícios devido á sua origem simulada, o balanço é muito positivo. Com implementações em infraestruturas reais, a obtenção continua de dados e a atualização constante do modelo otimizando os resultados, certamente que é uma tecnologia com potencial para estar presente em todas as instalações com a capacidade de tirar proveito da mesma. Desde as subestações até mesmo ao posto de transformação ou até aplicações domésticas (mais difícil de justificar financeiramente).

As potencialidades do *machine learning* são infinitas e certamente que cada vez mais estará implementado nas mais diversas situações e ambientes.

Referências

(IEEE 2020) [1] O. Yu. Maryasin and A. I. Lukashov, “A Python Application for Hourly Electricity Prices Forecasting Using Neural Networks,” 2020 International Russian Automation Conference (RusAutoCon), Sep. 2020, doi: 10.1109/rusautocon49822.2020.9208035. (Viewed 12 September 2024)

(IEEE 2021) [2] N. B. Vanting, Z. Ma, and B. N. Jørgensen, “A scoping review of deep neural networks for electric load forecasting,” Energy Informatics, vol. 4, no. S2, Sep. 2021, doi: 10.1186/s42162-021-00148-6. (Viewed 5 September 2024)

(ABB 2010) [3] “ABB Delivers 25,000th High Voltage Type PM Dead Tank Breaker.” https://www.electricnet.com/doc/abb-delivers-25000-high-voltage-type-0001 (Viewed 15 January 2024)

(ABB) [4] “ABB Medium voltage.” http://new.abb.com/medium-voltage/apparatus/circuit-breakers/indoor-cb/iec- primary-gas-cb-hd4 (Viewed 15 January 2024)

(Kingsmill Industries 2011) [5] Dev, “AC Substations and Earthing System Fundamentals,” Kingsmill Industries, Mar. 02, 2022. https://kingsmillindustries.com/ac-substations-and-earthing-system-fundamentals/ (Viewed 12 January 2024)

(GOOGLE 2024) [6] Google Colab, “colab.google,” *colab.google*. https://colab.google/ (Viewed 1 September 2024)

(IEEE 2020) [7] P. Sykora, M. Sinko, R. Vrskova, P. Kamencay, and R. Hudec, “Comparison of convolutional neural network in Python environment on CPU and GPU,” 2020 ELEKTRO, May 2020, doi: 10.1109/elektro49696.2020.9130321. (Viewed 12 September 2024)

(EEP 2015) [8] “Complete Guide To Wind Power Plants.” https://electrical-engineering-portal.com/download-center/books-and-guides/power-%20substations/wind-power-plants (Viewed 4 January 2024)

(IEEE 2023) [9] I. Kutsevol and O. Buhrii, “Continuous Time Neural Network,” IEEE, Sep. 2023, doi: 10.1109/elit61488.2023.10310940. (Viewed 30 August 2024)

(GFG 2024) [10] GeeksforGeeks, “Graph Plotting in Python | Set 1,” GeeksforGeeks, Jul. 26, 2024. https://www.geeksforgeeks.org/graph-plotting-in-python-set-1/ (Viewed 3 September 2024)

(IEEE 2021) [11] S. Kumar, A. K. Singh, A. Kalam, and D. Singh, “Improved Fault Prediction using Hybrid Machine Learning Techniques,” 2021 2nd Global

(TensorFlow 2024) [12] “Introdução ao TensorFlow,” TensorFlow. https://www.tensorflow.org/learn?hl=pt (Viewed 10 September 2024)

(GFG 2024) [13] GeeksforGeeks, “Long shortterm memory (LSTM) RNN in Tensorflow,” GeeksforGeeks, Jan. 10, 2023. https://www.geeksforgeeks.org/long-short-term-memory-lstm-rnn-in-tensorflow/ (Viewed 3 September 2024)

(MRF 2020) [14] Market Research Future, “Machine Learning Market Size & Growth Forecast 2032,” Market Research Future. https://www.marketresearchfuture.com/reports/machine-learning-market-2494 (Viewed 21 January 2024)

(IFLEXION 2020) [15] M. Anderson, “Machine Learning Overview: Understanding The ‘Gold Rush,’” Iflexion, Jan. 12, 2023. https://www.iflexion.com/blog/machine-learning-overview (Viewed 22 January 2024)

(MATPLOTLIB 2024) [16] “Matplotlib — Visualization with Python.” https://matplotlib.org/ (Viewed 13 September 2024)

(Pandas 2024) [17] “pandas - Python Data Analysis Library.” https://pandas.pydata.org/ (Viewed 7 September 2024)

(ICBAIE 2022) [18] Y. Wu, Q. Tan, J. Xia, B. Li, and Y. Yu, “Prediction of Defect Occurrence of Substation Equipment Based on Time Series and Neural Network,” 2022 3rd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE), Jul. 2022, doi: 10.1109/icbaie56435.2022.9985803. (Viewed 18 January 2024)

(EFACEC 2019) [19] “Press kit | Efacec,” Efacec, May 27, 2019. https://www.efacec.pt/press-kit-comunicados-imprensa/ (Viewed 4 January 2024)

(Real Python 2020) [20] D. Mesquita, “Python AI: How to Build a Neural Network & Make Predictions,” Jul. 24, 2023. https://realpython.com/python-ai-neural-network/ (Viewed 5 September 2024)

(Research Gate) [21] “Rede nacional alta tensão.” https://www.researchgate.net/figure/Mapa-da-rede-nacional-de-transporte-de- energia-electrica-da-REN-em Portugal_fig1_290797699 (Viewed 6 January 2024)

(WEG 2024) [22] “Representadas – R3 Representações.” https://r3representacoes.com/representadas/ (Viewed 7 January 2024)

(Omazaki) [23] Admin and Admin, “Short Circuit Study & Analysis - Omazaki Group,” *Omazaki Group - Official Website Omazaki Group*, Sep. 24, 2024. https://www.omazaki.co.id/en/short-circuit-study-analysis/ Viewed 11 September 2024)

(Research Gate) [24] “Substation SCADA systems.” https://www.researchgate.net/figure/Substation-SCADA systems_fig1_261451861 (Viewed 2 January 2024)

(IEEE 2023) [25] O. Abdelqader and F. Mancilla-David, “The Potential Role of Machine Learning in Improving Protective Relaying of Substations,” IEEE, Oct. 2023, doi: 10.1109/naps58826.2023.10318807. (Viewed 8 September 2024)

(Machine Learning Mastery 2022) [26] “Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras.” https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/ (Viewed 5 September 2024)

(Electricalsite 2016) [27] Electricalsite, “Transformer Main Parts,” Electricalsite, May 24, 2016. https://electrical-site.blogspot.com/2016/05/transformer-main-parts.html (Viewed 19 January 2024)

(Medium 2023) [28] P. Mishra, “Understanding and Implementing LSTM Networks - Palash Mishra - Medium,” Medium, Dec. 29, 2023. [Online]. Available: https://medium.com/@palashm0002/understanding-and-implementing-lstm-networks-41ca52495108 (Viewed 10 September 2024)

(Python 2024) [29] “Welcome to Python.org,” Python.org, Sep. 24, 2024. https://www.python.org/ (Viewed 27 August 2024)

Anexos

Índice de anexos

Anexo A : Código completo utilizado

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import time
import os
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score,
precision_score, recall_score, f1_score

start_time = time.time()

freq='10T'

timestamp = time.strftime("%Y%m%d-%H%M%S")
output_folder = f'/content/drive/MyDrive/Tese/scenario_{timestamp}'
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

start_date = pd.Timestamp('2022-01-01')
end_date = pd.Timestamp('2024-01-01')

def calculate_samples(start_date, end_date, freq=freq):
    return len(pd.date_range(start=start_date, end=end_date, freq=freq))

np.random.seed(42)

n_samples = calculate_samples(start_date, end_date)


voltages = np.random.uniform(56, 60, (n_samples, 3))

def generate_smooth_currents(n_samples, zero_prob=0.01, asymmetry_prob=0.10,
overload_prob=0.02):
    base_current = 8 + np.random.normal(0, 0.8, n_samples)
    variation_percentage = np.random.uniform(0.05, 0.10, (n_samples, 3))


    currents = base_current[:, np.newaxis] * (1 + variation_percentage)

    for i in range(n_samples):
        if np.random.rand() < zero_prob:
            phase = np.random.choice([0, 1, 2])
            currents[i, phase] = 0

        elif np.random.rand() < asymmetry_prob:
            phase = np.random.choice([0, 1, 2])
```

```
            factor = 1 + np.random.uniform(0.15, 0.3)
            currents[i, phase] *= factor

        elif np.random.rand() < overload_prob:
            phase = np.random.choice([0, 1, 2])
            currents[i, phase] = np.random.uniform(9, 11)
    currents = np.clip(currents, 0, 11)
    return currents

currents = generate_smooth_currents(n_samples, zero_prob=0.01, asymmetry_prob=0.1,
overload_prob=0.02)

data = np.hstack((voltages, currents))
df = pd.DataFrame(data, columns=['V_a', 'V_b', 'V_c', 'I_a', 'I_b', 'I_c'])
date_range = pd.date_range(start=start_date, periods=n_samples, freq=freq)
df['DateTime'] = date_range

def apply_moving_average(data, window_size):
  return np.convolve(data, np.ones(window_size), 'valid') / window_size

window_size = 5
voltages_smooth = np.apply_along_axis(apply_moving_average, 0, voltages,
window_size=window_size)
currents_smooth = np.apply_along_axis(apply_moving_average, 0, currents,
window_size=window_size)

n_samples_smooth = len(voltages_smooth)

data_smooth = np.hstack((voltages_smooth, currents_smooth))
df = pd.DataFrame(data_smooth, columns=['V_a', 'V_b', 'V_c', 'I_a', 'I_b', 'I_c'])

date_range = pd.date_range(start=start_date, periods=n_samples_smooth, freq=freq)
df['DateTime'] = date_range

def create_sequences(data, time_steps):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i:i + time_steps])
        y.append(data[i + time_steps])
    return np.array(X), np.array(y)

time_steps = 10
data = df[['V_a', 'V_b', 'V_c', 'I_a', 'I_b', 'I_c']].values
X, y = create_sequences(data, time_steps)
dates = df['DateTime'][time_steps:].reset_index(drop=True)

scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X.reshape(-1, X.shape[-1])).reshape(X.shape)
y_scaler = MinMaxScaler()
y_scaled = y_scaler.fit_transform(y)

X_train, X_test, y_train, y_test, dates_train, dates_test = train_test_split(
    X_scaled, y_scaled, dates, test_size=0.10, random_state=42
)

model = Sequential()
model.add(LSTM(16, activation='relu', input_shape=(time_steps, X.shape[2])))
model.add(Dense(6))
model.compile(optimizer='adam', loss='mse')

history = model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_data=(X_test, y_test))
```

```
print(f"Training Loss: {history.history['loss'][-1]:.4f}")
print(f"Validation Loss: {history.history['val_loss'][-1]:.4f}")

y_pred = model.predict(X_test)
y_pred_rescaled = y_scaler.inverse_transform(y_pred)

thresholds = {
    'voltage_min': 58,
    'current_max': 9
}

def detect_defects(data, thresholds):
    defects = []
    voltage_min = thresholds['voltage_min']
    current_max = thresholds['current_max']

    for sample in data:
        sample_defects = []
        if np.any(sample == 0):
            sample_defects.append('Corrente ou tensão zero')

        mean_sample = np.mean(sample[3:])
        if np.any(np.abs(sample[3:] - mean_sample) / mean_sample > 0.1):
            sample_defects.append('Variação >10% entre correntes')

        if np.any(sample[3:] > current_max):
            sample_defects.append('Sobrecarga de corrente')

        if np.any(sample[:3] < voltage_min):
            sample_defects.append('Queda de tensão')

        defects.append(sample_defects)

    return defects

y_test_rescaled = y_scaler.inverse_transform(y_test)
defects_real = detect_defects(y_test_rescaled, thresholds)
defects_pred = detect_defects(y_pred_rescaled, thresholds)

defects_real_df = pd.DataFrame({'DateTime': dates_test, 'Defeito_Real': defects_real})
defects_pred_df = pd.DataFrame({'DateTime': dates_test, 'Defeito_Pred': defects_pred})

aligned_df = pd.merge(defects_real_df, defects_pred_df, on='DateTime', how='outer')

defects_real_df = defects_real_df.explode('Defeito_Real')
defects_pred_df = defects_pred_df.explode('Defeito_Pred')

aligned_df = defects_real_df.merge(defects_pred_df, on='DateTime', how='outer')
aligned_df['Match'] = aligned_df['Defeito_Real'] == aligned_df['Defeito_Pred']

correct_detections = aligned_df['Match'].sum()
total_defects_real = defects_real_df['Defeito_Real'].count()

success_rate = correct_detections / total_defects_real * 100 if total_defects_real > 0
else 0

total_defects = defects_real_df['Defeito_Real'].value_counts().to_dict()
correct_defects = aligned_df[aligned_df['Match'] ==
True]['Defeito_Real'].value_counts().to_dict()

success_rate_per_defect = {defect: (correct_defects.get(defect, 0) / total) * 100
                           for defect, total in total_defects.items()}
```

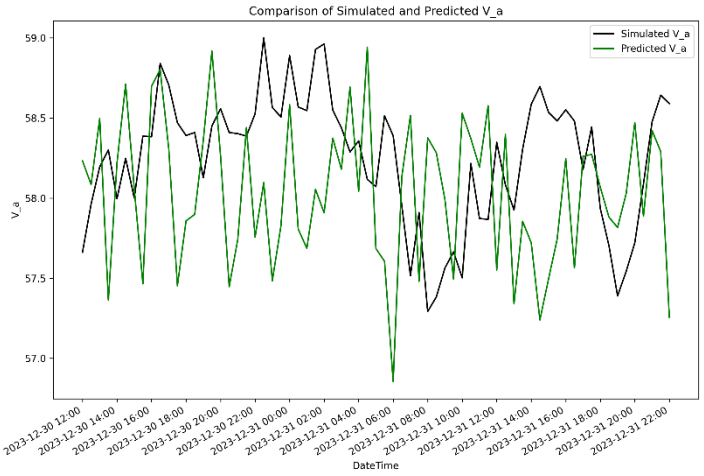
```
print(f"Total de defeitos reais: {total_defects_real}")
print(f"Defeitos corretamente detectados: {correct_detections}")
print(f"Taxa de sucesso geral: {success_rate:.2f}%")

start_plot_date = pd.Timestamp(start_date)
end_plot_date = pd.Timestamp(end_date)

df_simulated = df[['DateTime', 'V_a', 'V_b', 'V_c', 'I_a', 'I_b', 'I_c']].copy()
df_predicted = pd.DataFrame(y_pred_rescaled, columns=['V_a', 'V_b', 'V_c', 'I_a',
'I_b', 'I_c'])
df_predicted['DateTime'] = df['DateTime'].iloc[-len(y_pred_rescaled):].values

df_simulated_filtered = df_simulated[
    (df_simulated['DateTime'] >= start_plot_date) & (df_simulated['DateTime'] <=
end_plot_date)
]

df_predicted_filtered = df_predicted[
    (df_predicted['DateTime'] >= start_plot_date) & (df_predicted['DateTime'] <=
end_plot_date)
]

df_simulated_filtered = df_simulated[
    (df_simulated['DateTime'] >= start_plot_date) & (df_simulated['DateTime'] <=
end_plot_date)
]
df_predicted_filtered = df_predicted[
    (df_predicted['DateTime'] >= start_plot_date) & (df_predicted['DateTime'] <=
end_plot_date)
]

comparison_df = defects_real_df.merge(defects_pred_df, on='DateTime', how='outer')
comparison_df['Match'] = comparison_df['Defeito_Real'] ==
comparison_df['Defeito_Pred']

false_positives = comparison_df[comparison_df['Defeito_Real'].isna() &
comparison_df['Defeito_Pred'].notna()]

true_positives = comparison_df[comparison_df['Match'] == True]

false_negatives = comparison_df[comparison_df['Defeito_Real'].notna() &
comparison_df['Defeito_Pred'].isna()]

num_false_positives = len(false_positives)
num_true_positives = len(true_positives)
num_false_negatives = len(false_negatives)

print(f"Número de falsos positivos: {num_false_positives}")
print(f"Número de verdadeiros positivos: {num_true_positives}")

results_df = pd.DataFrame({
    'DateTime': comparison_df['DateTime'],
    'Defeito_Real': comparison_df['Defeito_Real'],
    'Defeito_Pred': comparison_df['Defeito_Pred'],
    'Match': comparison_df['Match']
})

end_time = time.time()
processing_time = end_time - start_time

mae = mean_absolute_error(y_test_rescaled, y_pred_rescaled)
rmse = np.sqrt(mean_squared_error(y_test_rescaled, y_pred_rescaled))
r2 = r2_score(y_test_rescaled, y_pred_rescaled)
```

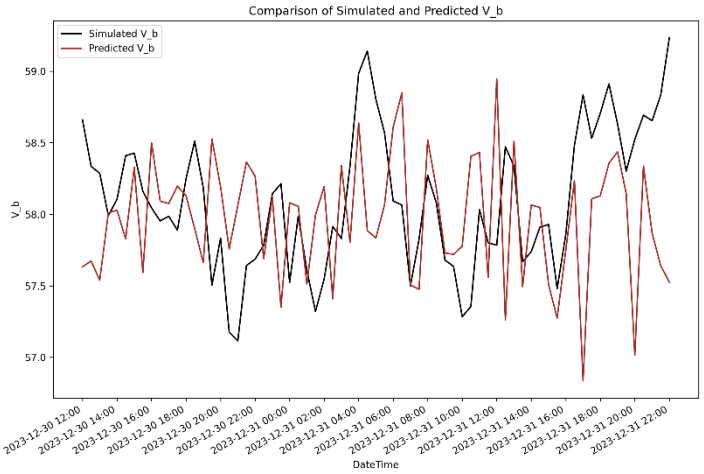
```
mse = mean_squared_error(y_test_rescaled, y_pred_rescaled)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-Squared (R²): {r2}")

aligned_df['Defeito_Real'] = pd.to_numeric(aligned_df['Defeito_Real'],
errors='coerce')
aligned_df['Defeito_Pred'] = pd.to_numeric(aligned_df['Defeito_Pred'],
errors='coerce')

aligned_df = aligned_df.dropna(subset=['Defeito_Real', 'Defeito_Pred'])

print(aligned_df.dtypes)

precision = num_true_positives / (num_true_positives+num_false_positives)
recall = num_true_positives / (num_true_positives+num_false_negatives)
f1 = 2*(precision*recall)/(precision+recall)

success_rate = correct_detections / total_defects_real * 100 if total_defects_real > 0
else 0

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")

variables = ['I_a', 'I_b', 'I_c', 'V_a', 'V_b', 'V_c']
colors_simulated = ['black', 'black', 'black', 'black', 'black', 'black']
colors_predicted = ['cyan', 'magenta', 'yellow', 'green', 'brown', 'pink']

sampling_interval_days = 360

for var, color_sim, color_pred in zip(variables, colors_simulated, colors_predicted):
    plt.figure(figsize=(12, 8))


    df_simulated_sampled = df_simulated_filtered.iloc[::sampling_interval_days, :]
    df_predicted_sampled = df_predicted_filtered.iloc[::sampling_interval_days, :]


    plt.plot(df_simulated_sampled['DateTime'], df_simulated_sampled[var],
             label=f'Simulated {var}', color=color_sim)


    plt.plot(df_predicted_sampled['DateTime'], df_predicted_sampled[var],
             label=f'Predicted {var}', color=color_pred)

    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d %H:%M'))
    plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=120))
    plt.gcf().autofmt_xdate()


    plt.xlabel('DateTime')
    plt.ylabel(f'{var}')
    plt.title(f'Comparison of Simulated and Predicted {var}')
    plt.legend()
    plt.grid(True, which='both', linestyle='--', linewidth=0.1)


    plt.savefig(f'{output_folder}/{var}_comparison.png', dpi=300)
    plt.show()
```

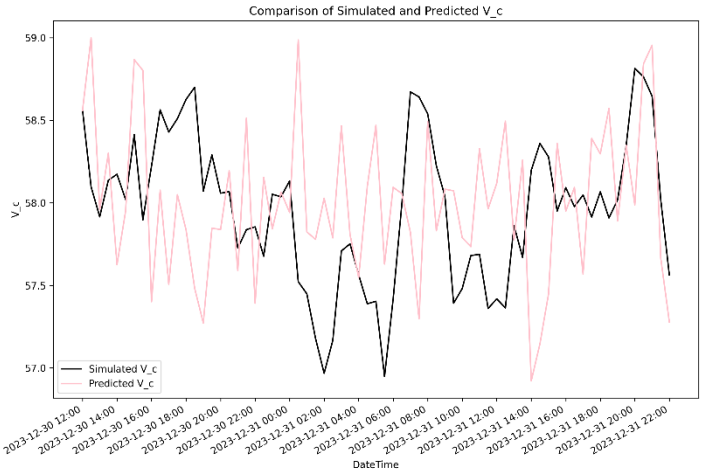
```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.savefig(f'{output_folder}/{var}_losscurve.png', dpi=300)
plt.show()

print(df_simulated.tail())
print(df_predicted.tail())

summary_df = pd.DataFrame({
    'Defeito': list(total_defects.keys()),
    'Total em Dados Reais': list(total_defects.values()),
    'Detectados Corretamente': [correct_defects.get(defect, 0) for defect in
total_defects.keys()],
    'Precisão': [success_rate_per_defect.get(defect, 0) for defect in
total_defects.keys()],
    'Taxa de Sucesso Geral (%)': [success_rate] * len(total_defects),
    'Tempo de Processamento (s)': [processing_time] * len(total_defects)
})

metrics_df = pd.DataFrame({
    'Métrica': ['MAE', 'RMSE', 'R²', 'Precisão', 'Recall', 'F1-Score'],
    'Valor': [mae, rmse, r2, precision, recall, f1]
})

with pd.ExcelWriter(f'{output_folder}/Estatistica.xlsx') as writer:
    results_df.to_excel(writer, sheet_name='Comparison', index=False)
    false_positives.to_excel(writer, sheet_name='False_Positives', index=False)
    true_positives.to_excel(writer, sheet_name='True_Positives', index=False)
    false_negatives.to_excel(writer, sheet_name='False_Negatives', index=False)
    metrics_df.to_excel(writer, sheet_name='Metrics', index=False)

print("Dados exportados para Estatistica.xlsx")

with pd.ExcelWriter(f'{output_folder}/Real e previsao.xlsx') as writer:
    df_simulated.to_excel(writer, sheet_name='Simulated Data', index=False)
    df_predicted.to_excel(writer, sheet_name='Predicted Data', index=False)
    summary_df.to_excel(writer, sheet_name='Summary', index=False)

print("Dados exportados para Real e previsao.xlsx")
```

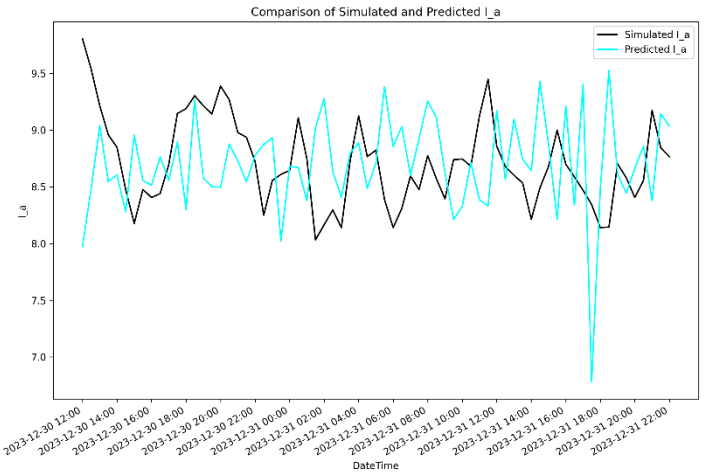
Anexo B : Gráficos gerados cenário 1 (30 minutos e 1 hora)

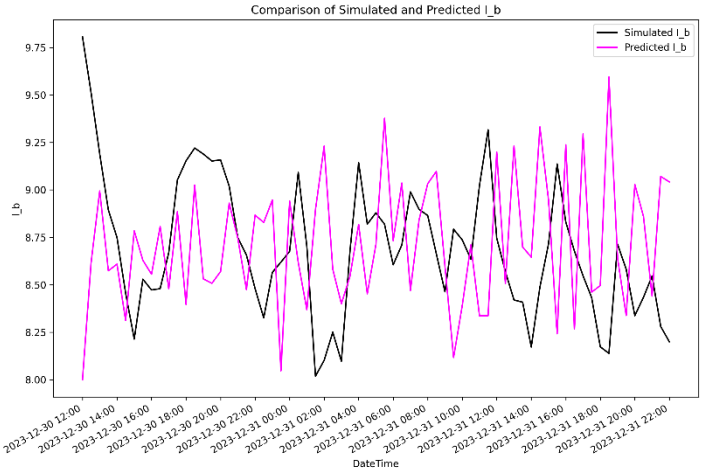
30 minutos

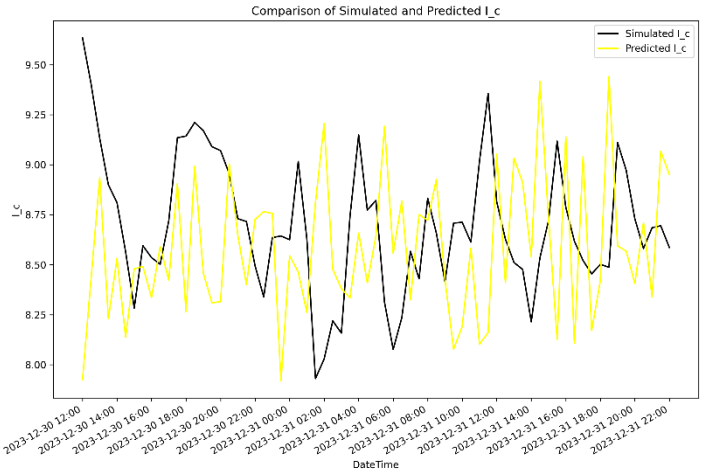

Comparison of Simulated and Predicted V_a
Simulated V_a
Predicted V_a
V_a
DateTime


Comparison of Simulated and Predicted V_b
Simulated V_b
Predicted V_b
V_b
DateTime

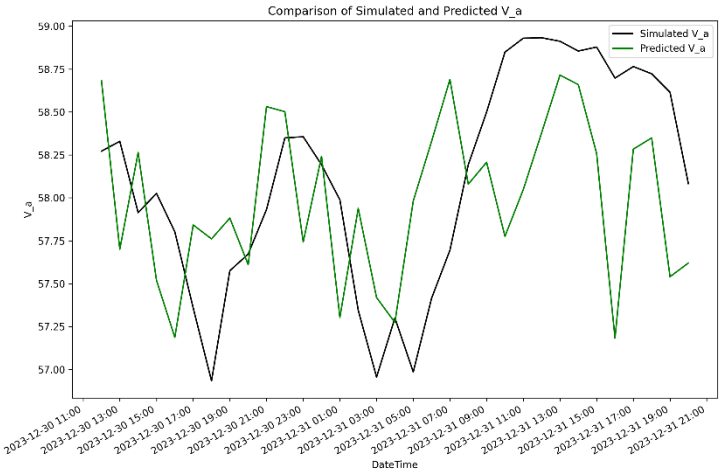

Comparison of Simulated and Predicted V_c
V_c
DateTime
Simulated V_c
Predicted V_c

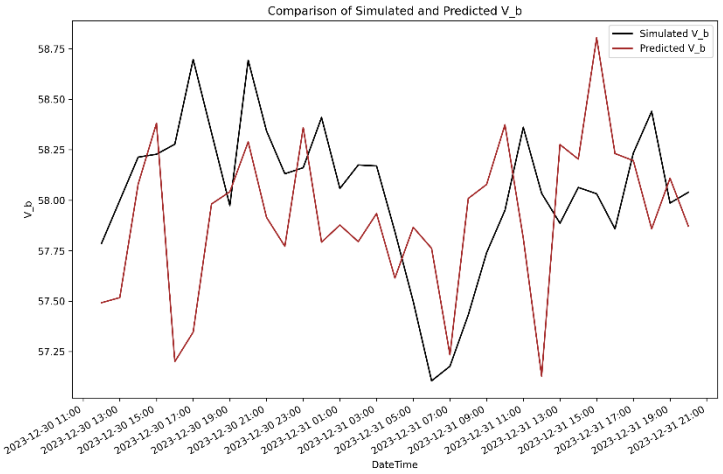

Comparison of Simulated and Predicted I_a
I_a
DateTime
Simulated I_a
Predicted I_a

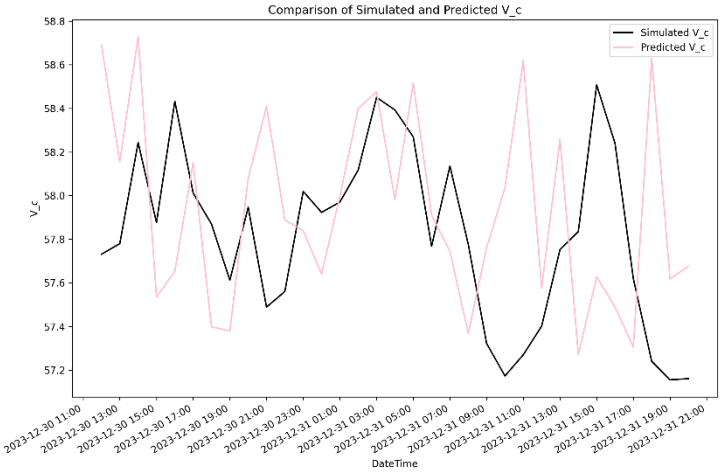

Comparison of Simulated and Predicted I_b
I_b
DateTime
Simulated I_b
Predicted I_b

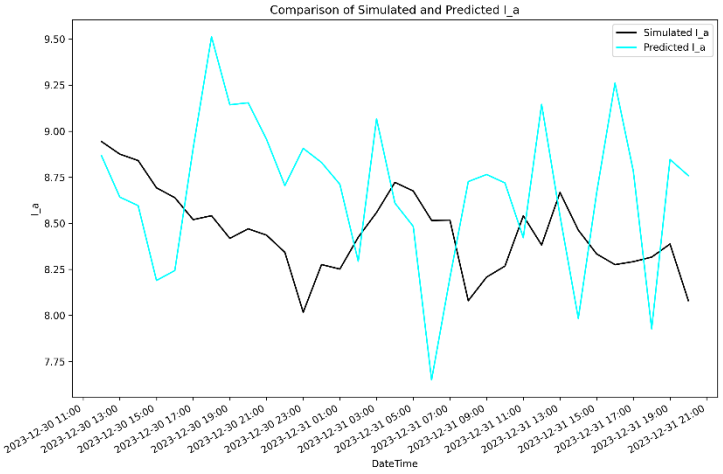

Comparison of Simulated and Predicted I_c
Simulated I_c
Predicted I_c
I_c
DateTime

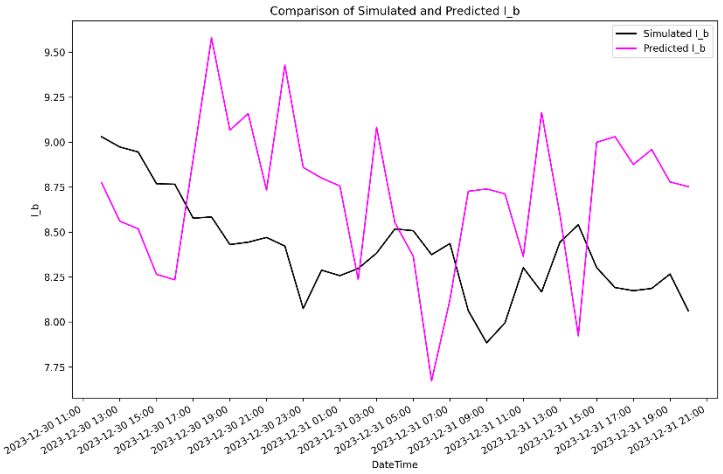
1 hora

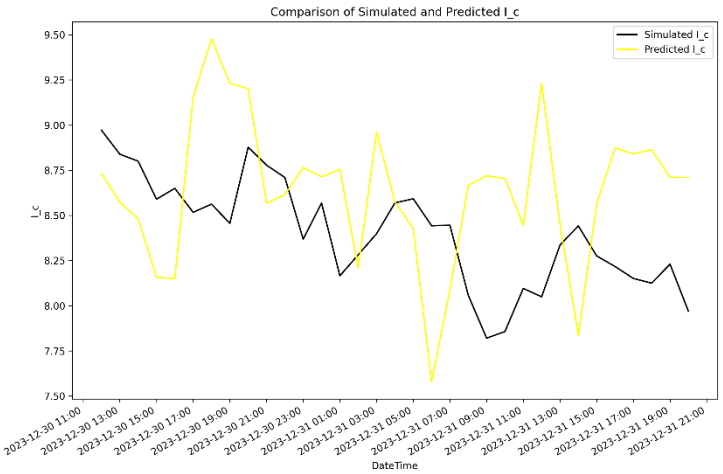

Comparison of Simulated and Predicted V_a
Simulated V_a
Predicted V_a
V_a
DateTime


Comparison of Simulated and Predicted V_b
Simulated V_b
Predicted V_b
V_b
DateTime


Comparison of Simulated and Predicted V_c
Simulated V_c
Predicted V_c
V_c
DateTime

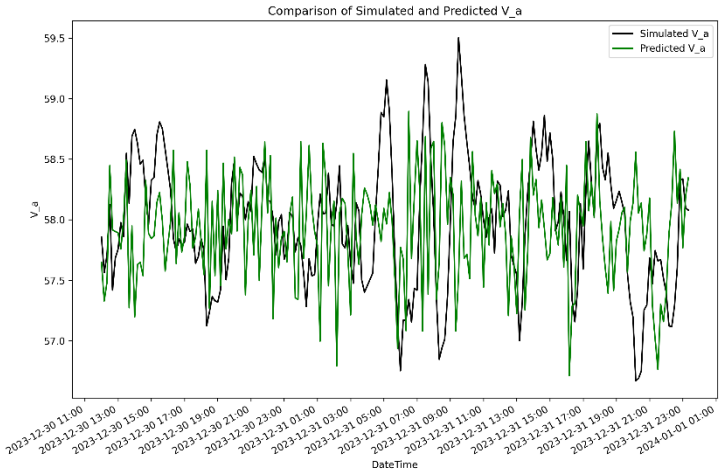

Comparison of Simulated and Predicted I_a
Simulated I_a
Predicted I_a
I_a
DateTime

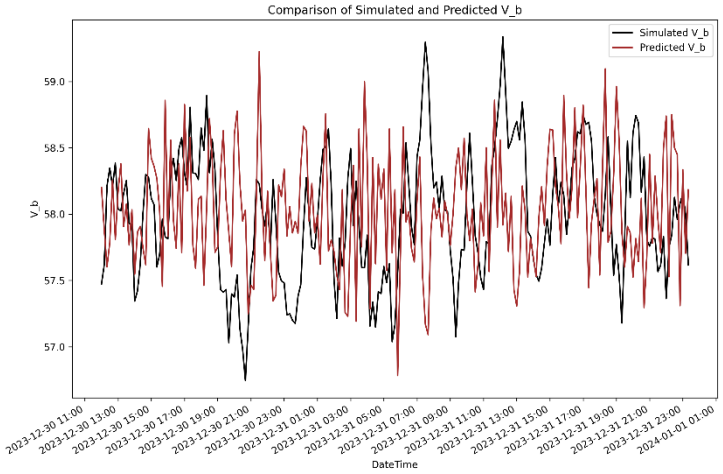


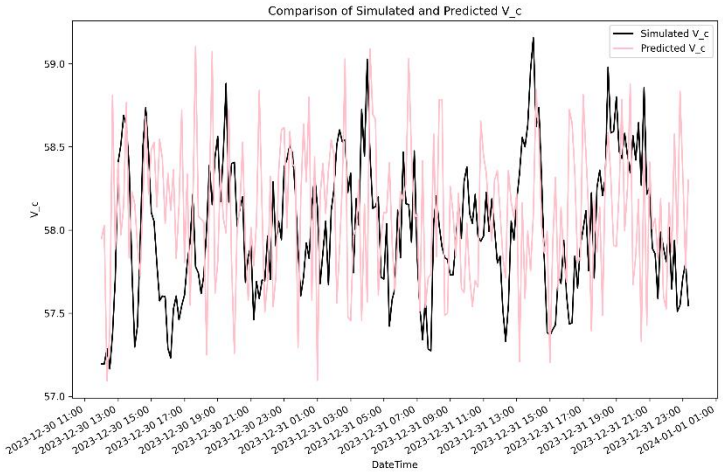


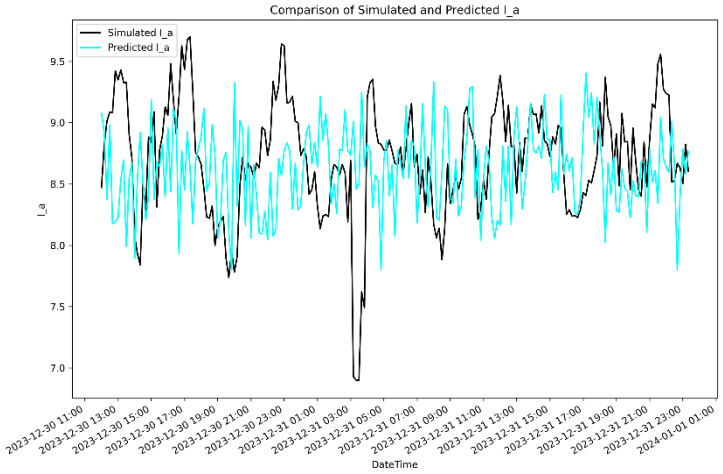
Anexo C - Gráficos gerados cenário 2 (*test-size* 0.05, 0.15 e 0.20)

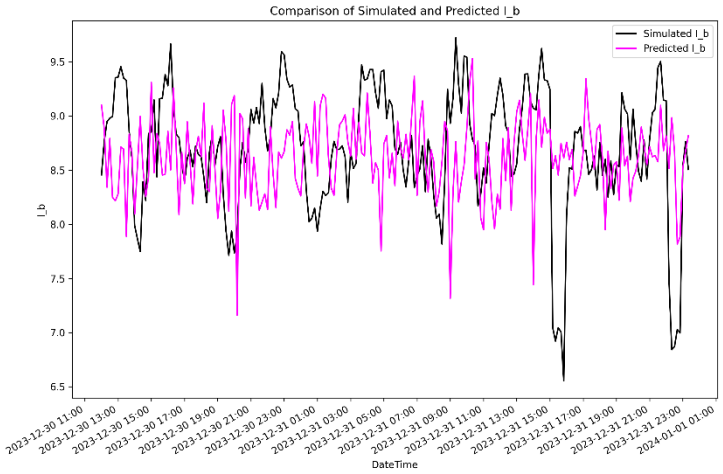
Test-size 0.05

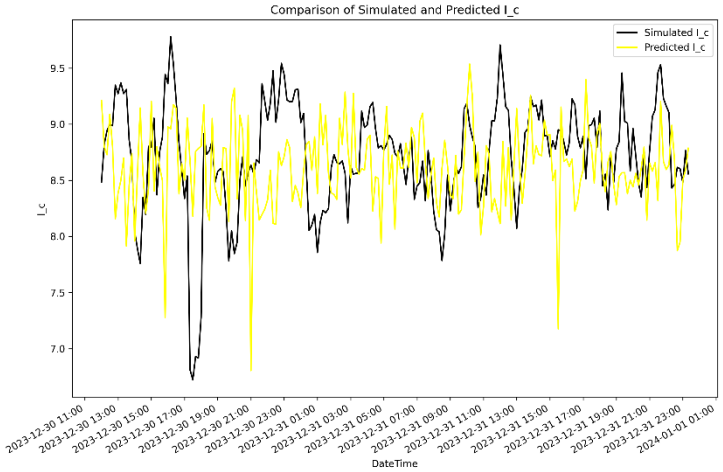



Comparison of Simulated and Predicted V_b
Simulated V_b
Predicted V_b
V_b
DateTime

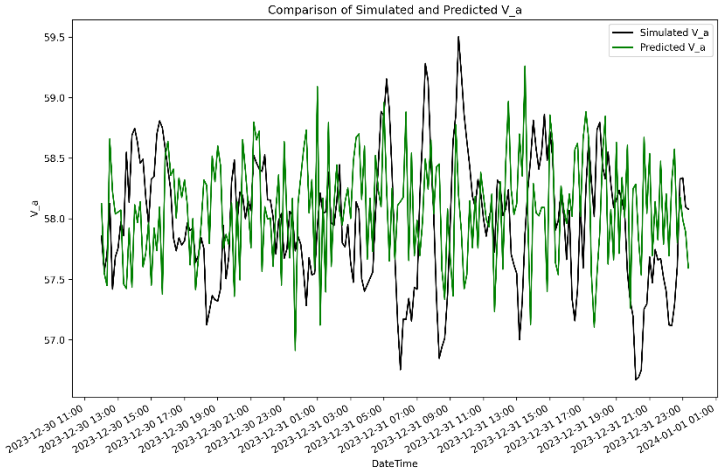

Comparison of Simulated and Predicted V_c
Simulated V_c
Predicted V_c
V_c
DateTime

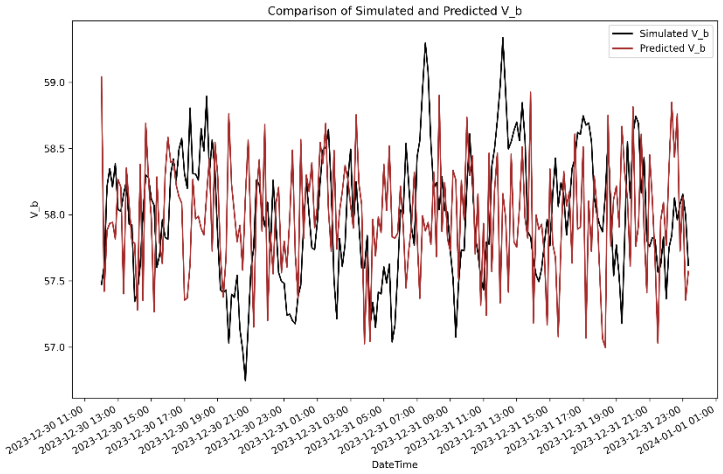

Comparison of Simulated and Predicted I_a
Simulated I_a
Predicted I_a
I_a
DateTime

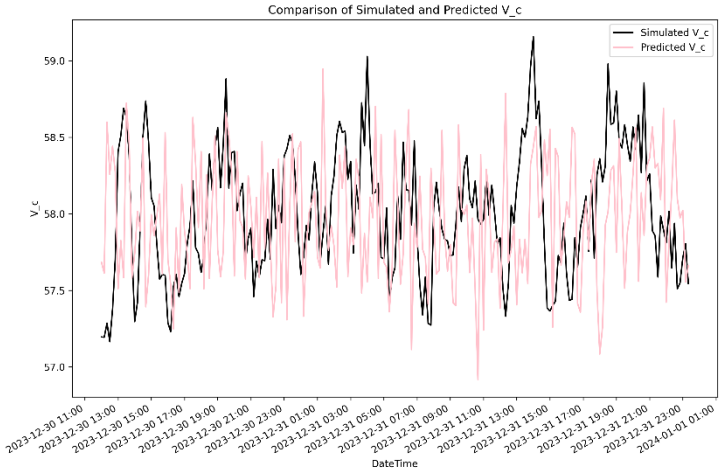


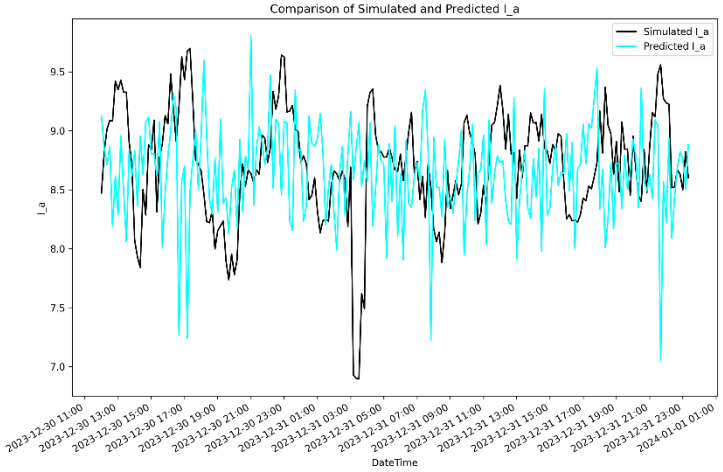


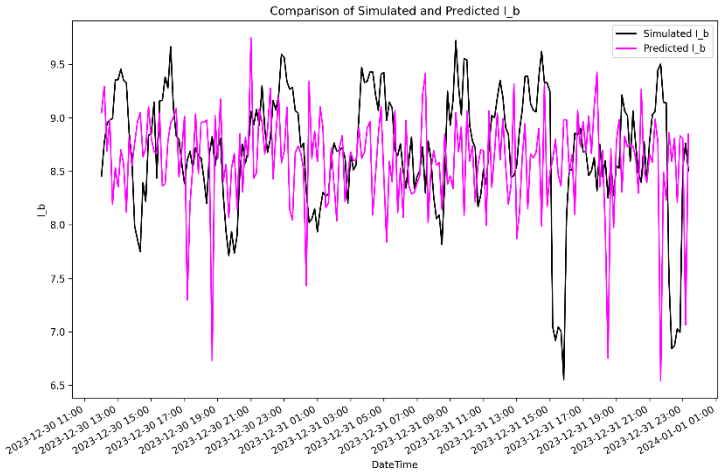
Test-size 0.15

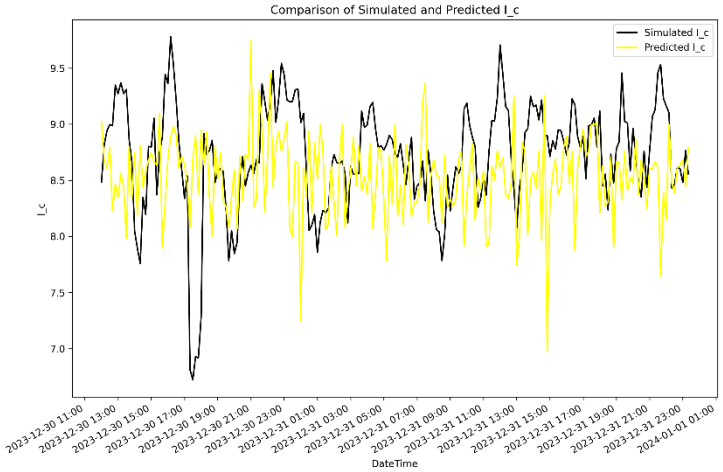



Comparison of Simulated and Predicted V_b
Simulated V_b
Predicted V_b
V_b
DateTime

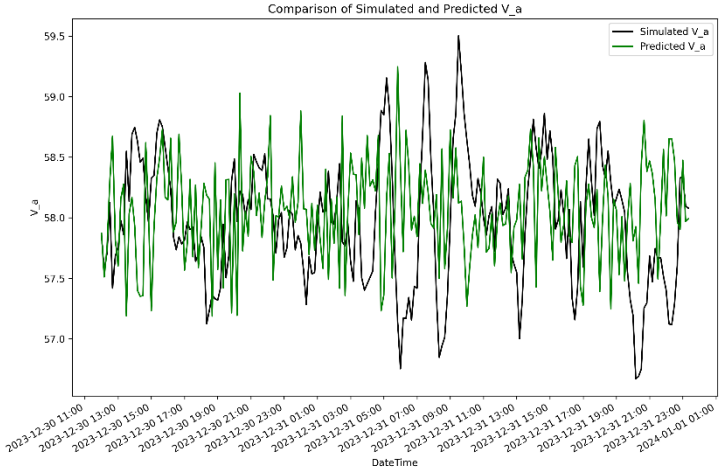

Comparison of Simulated and Predicted V_c
Simulated V_c
Predicted V_c
V_c
DateTime

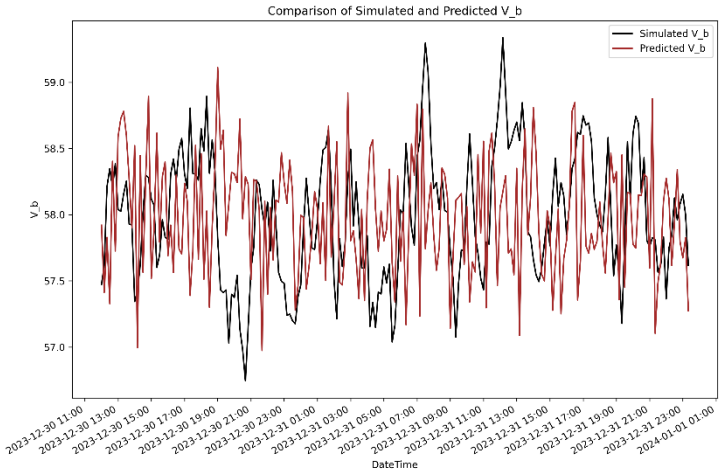

Comparison of Simulated and Predicted I_a
Simulated I_a
Predicted I_a
I_a
DateTime

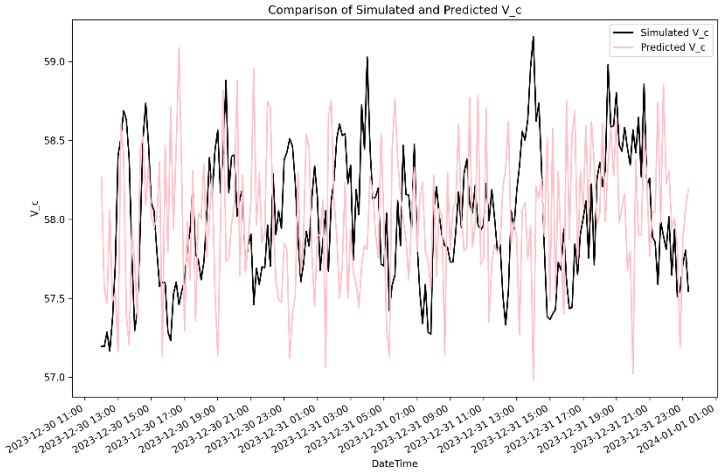


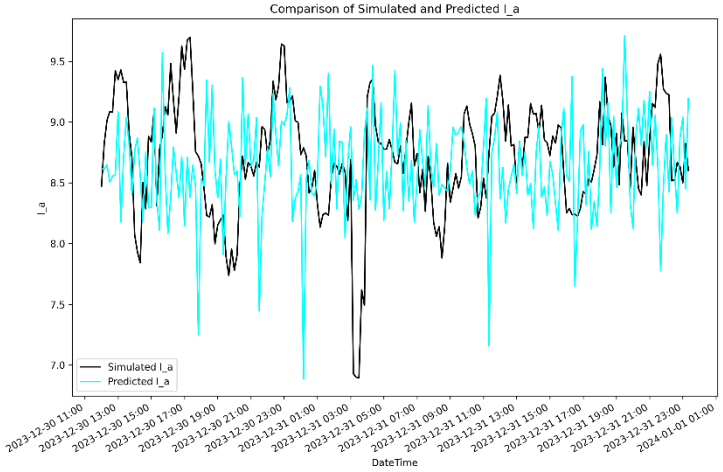


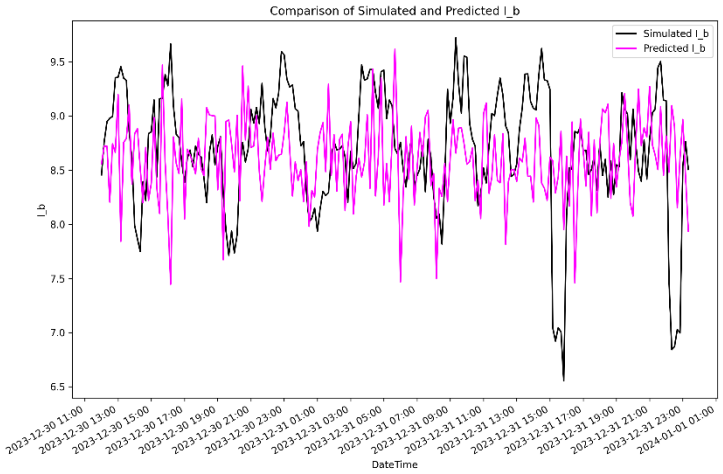
Test-size 0.20

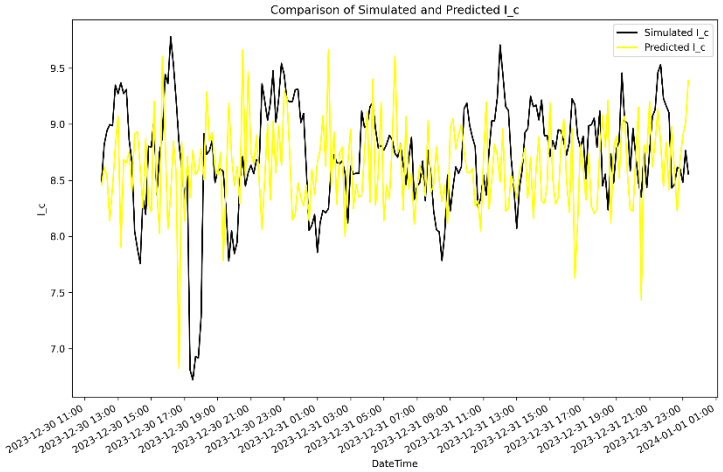



Comparison of Simulated and Predicted V_b
Simulated V_b
Predicted V_b
V_b
DateTime

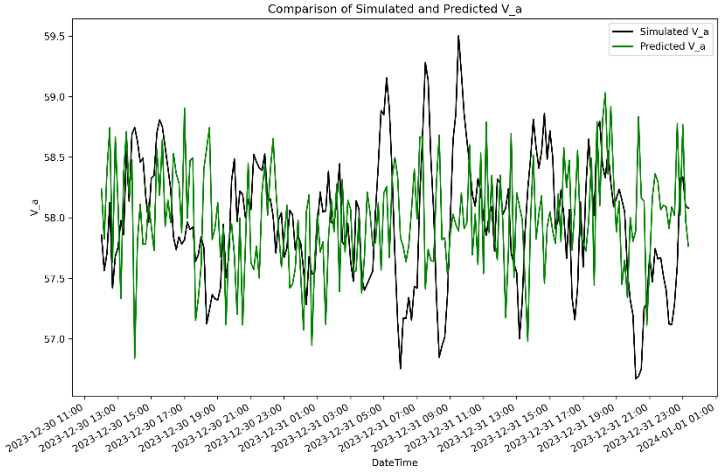

Comparison of Simulated and Predicted V_c
Simulated V_c
Predicted V_c
V_c
DateTime

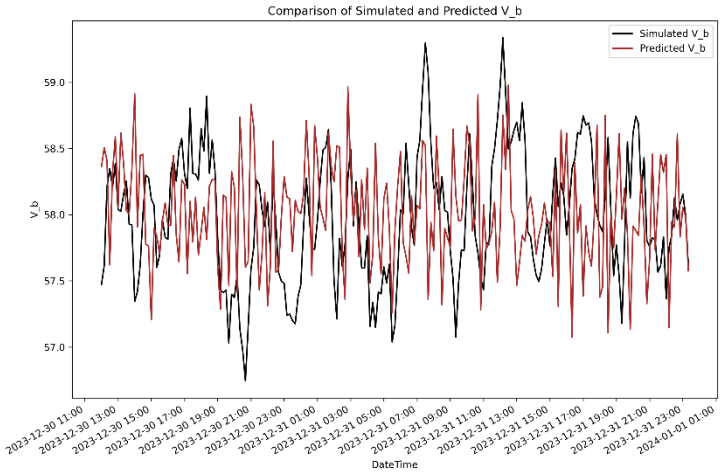

Comparison of Simulated and Predicted I_a
Simulated I_a
Predicted I_a
I_a
DateTime

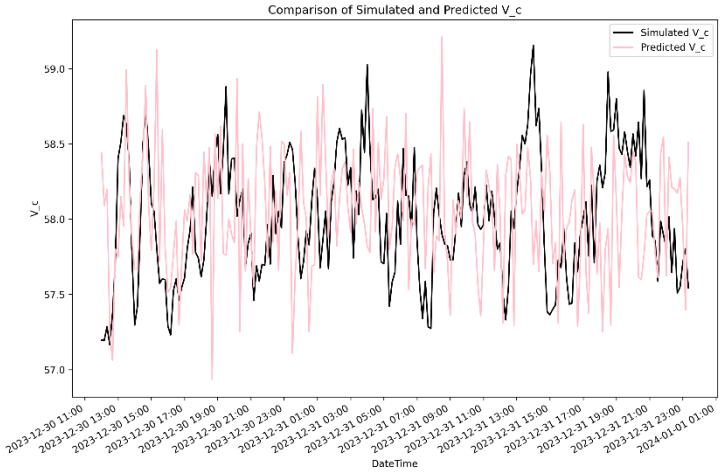


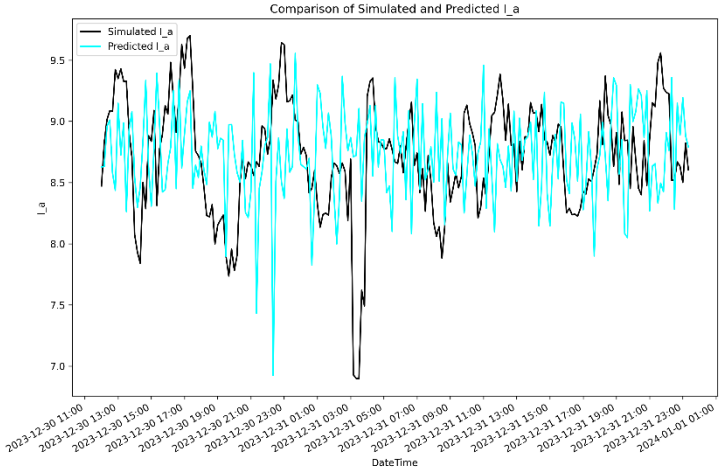


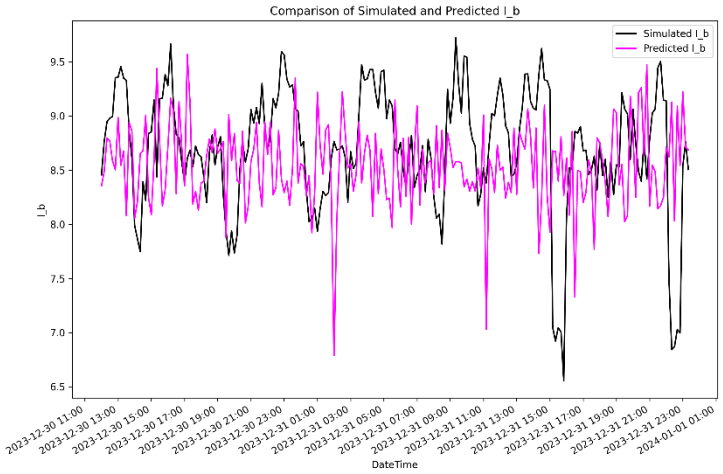
Anexo D : Gráficos gerados cenário 3 (*batch-size* 1, 8, 16 e 64)

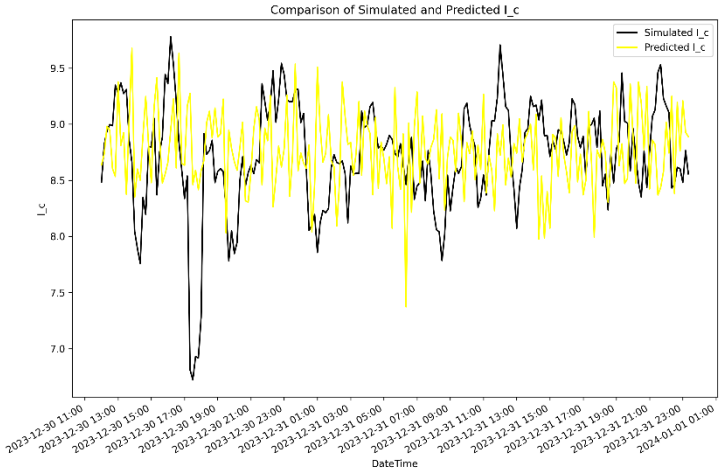



Comparison of Simulated and Predicted V_b
Simulated V_b
Predicted V_b
V_b
DateTime

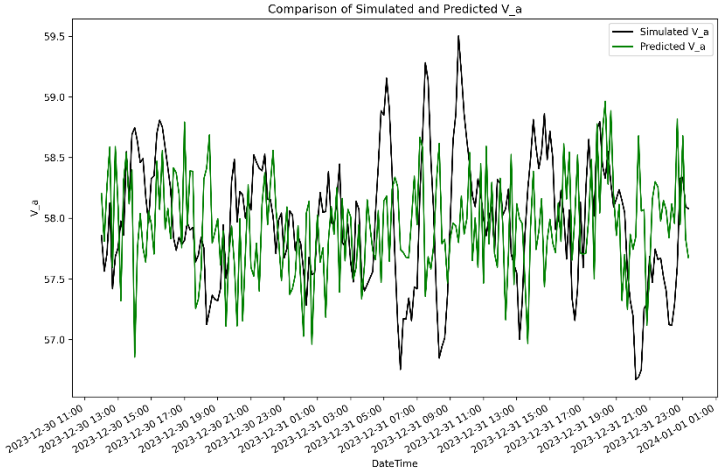

Comparison of Simulated and Predicted V_c
Simulated V_c
Predicted V_c
V_c
DateTime

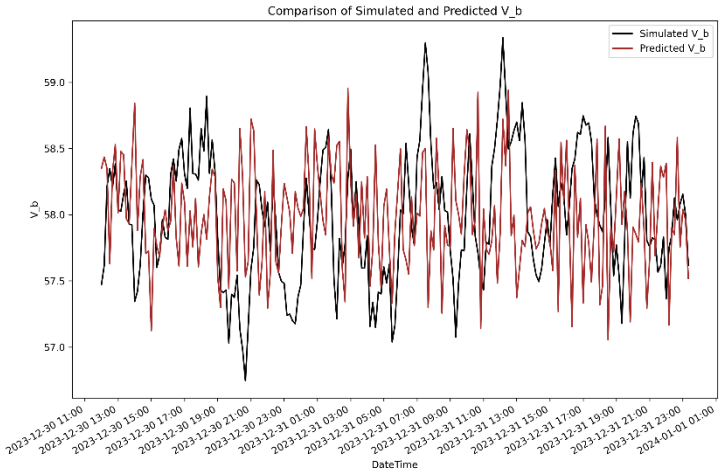

Comparison of Simulated and Predicted I_a
Simulated I_a
Predicted I_a
I_a
DateTime

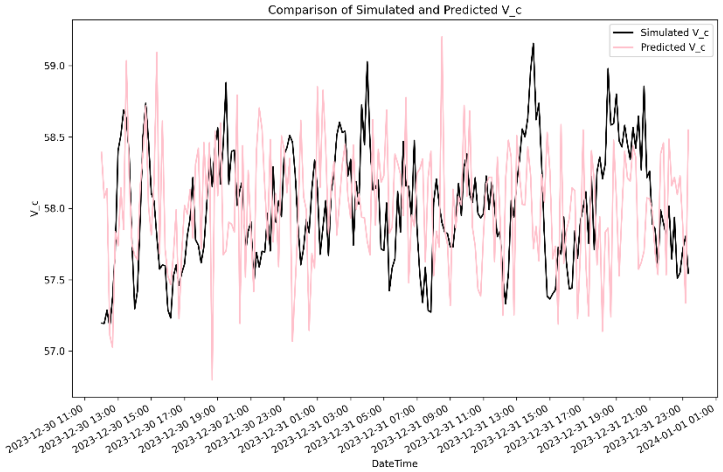

Comparison of Simulated and Predicted I_b
Simulated I_b
Predicted I_b
I_b
DateTime

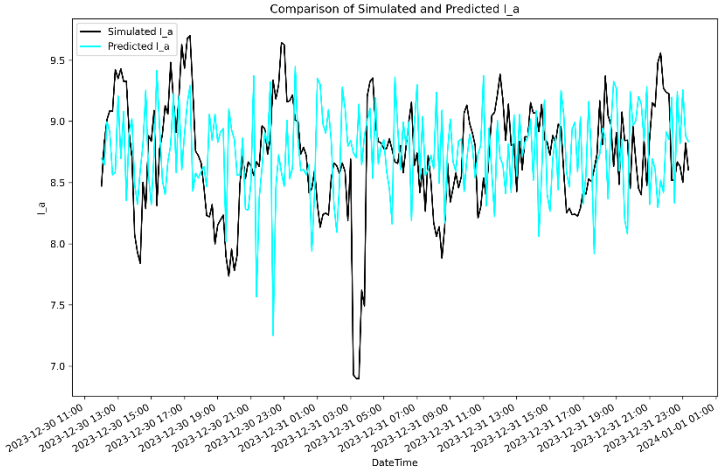

Comparison of Simulated and Predicted I_c
Simulated I_c
Predicted I_c
I_c
DateTime

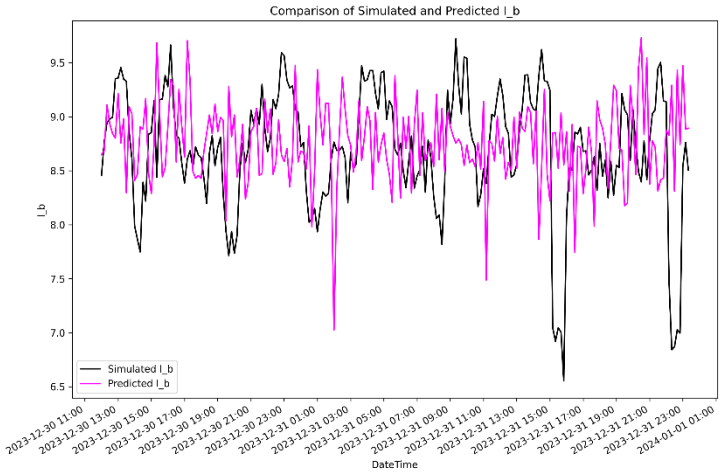
Batch-size 8

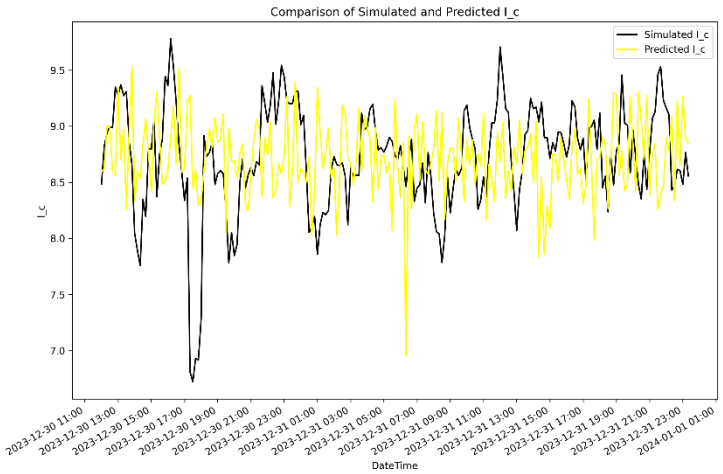

Comparison of Simulated and Predicted V_a
Simulated V_a
Predicted V_a
V_a
DateTime


Comparison of Simulated and Predicted V_b
Simulated V_b
Predicted V_b
V_b
DateTime

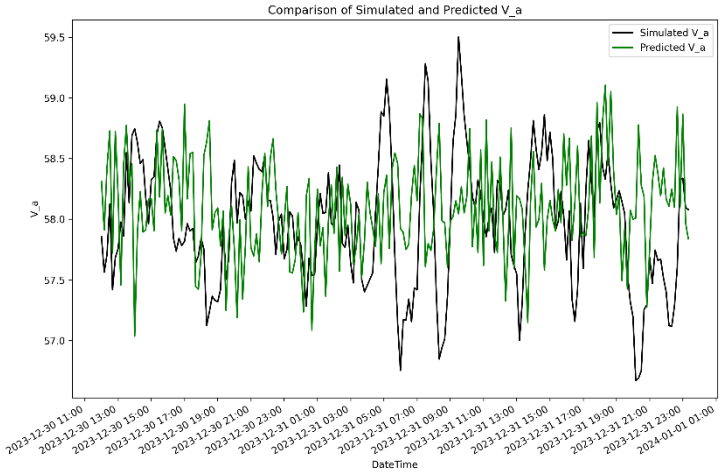

Comparison of Simulated and Predicted V_c
Simulated V_c
Predicted V_c
V_c
DateTime

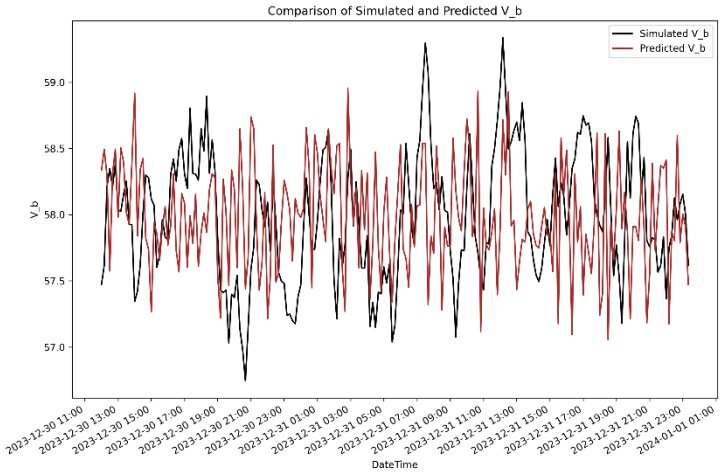

Comparison of Simulated and Predicted I_a
Simulated I_a
Predicted I_a
I_a
DateTime

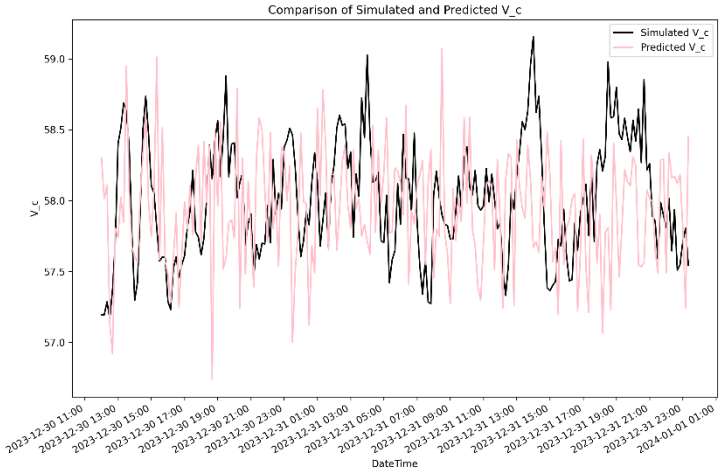


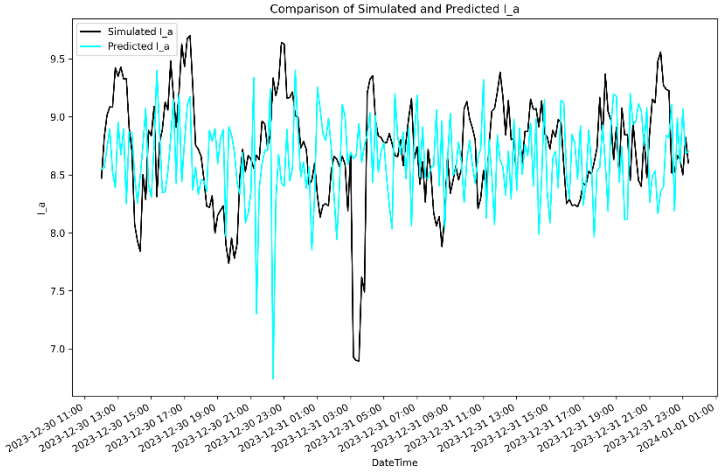


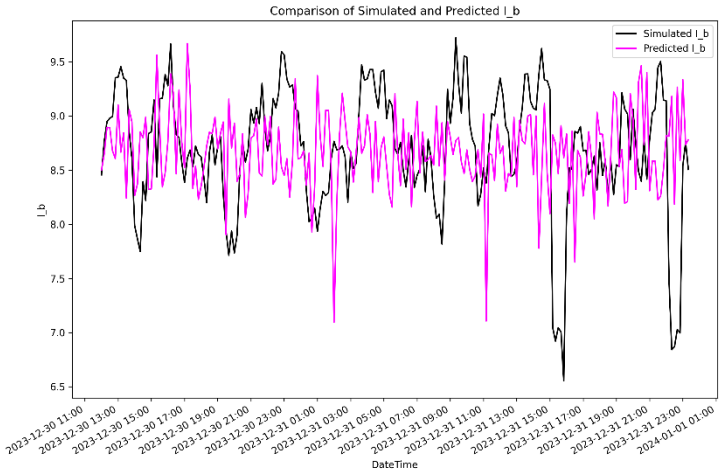
Batch-size 16

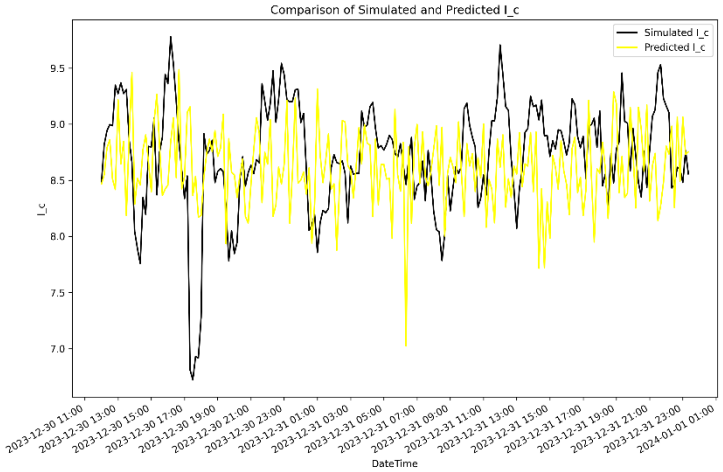



Comparison of Simulated and Predicted V_b
Simulated V_b
Predicted V_b
V_b
DateTime

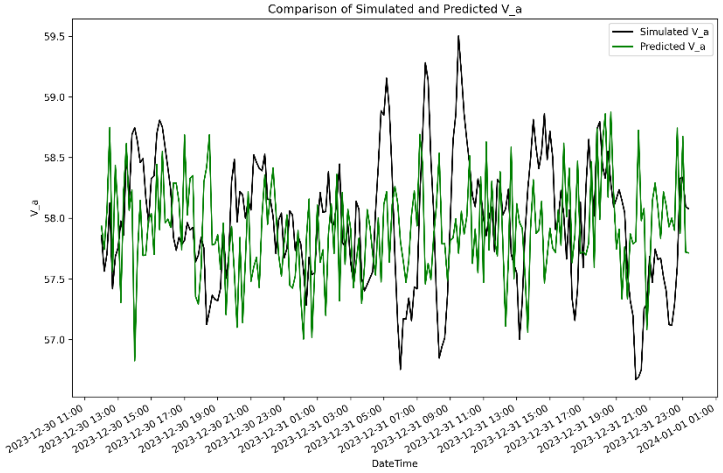

Comparison of Simulated and Predicted V_c
Simulated V_c
Predicted V_c
V_c
DateTime

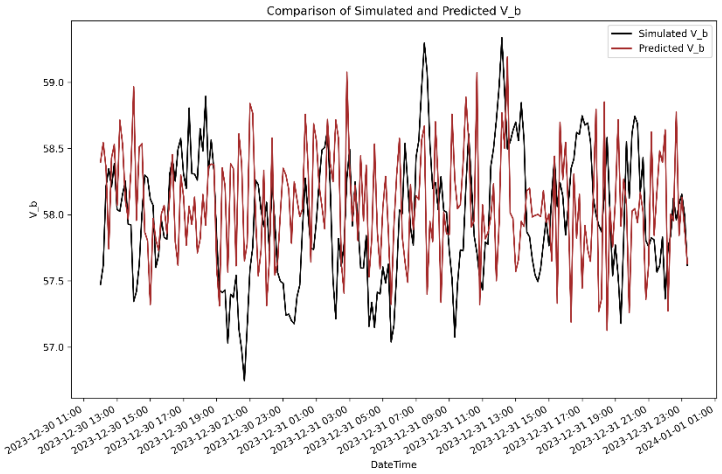

Comparison of Simulated and Predicted I_a
Simulated I_a
Predicted I_a
I_a
DateTime

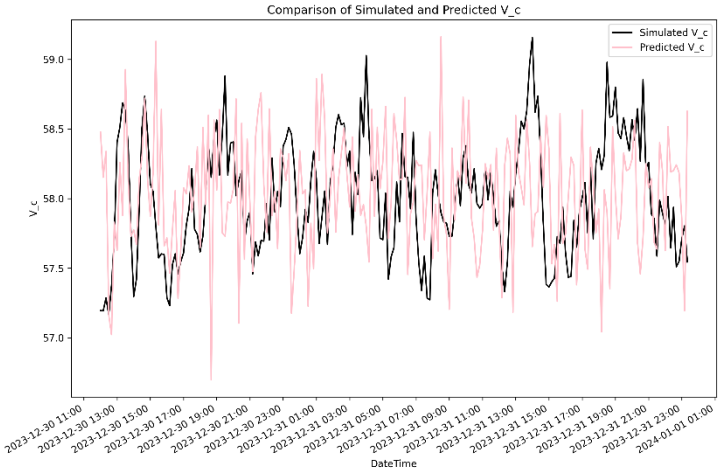


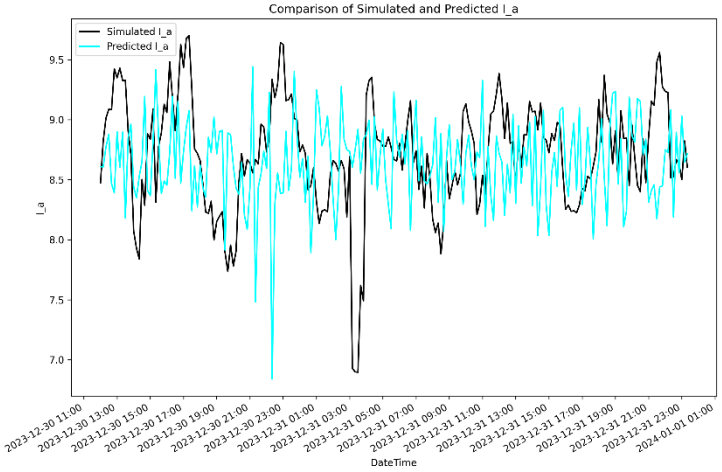


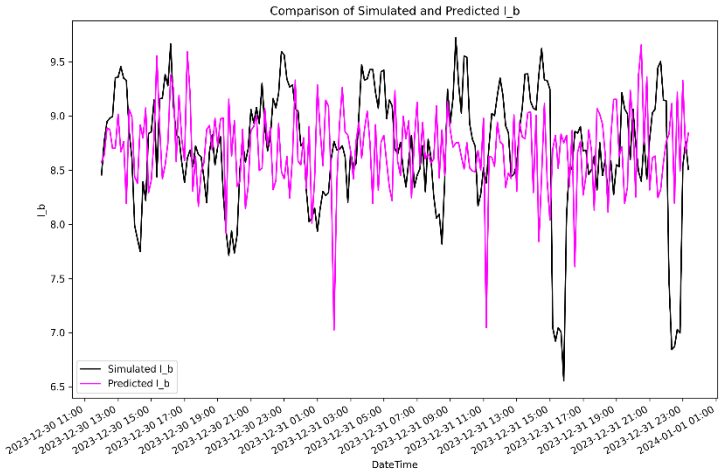
Batch-size 64

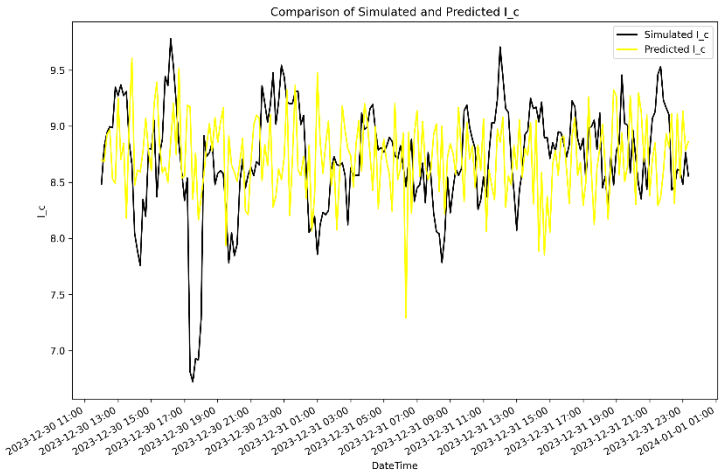



Comparison of Simulated and Predicted V_b
Simulated V_b
Predicted V_b
V_b
DateTime


Comparison of Simulated and Predicted V_c
Simulated V_c
Predicted V_c
V_c
DateTime

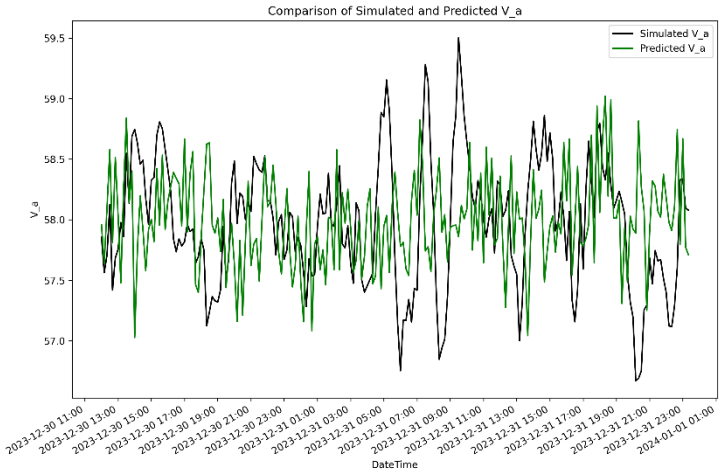

Comparison of Simulated and Predicted I_a
Simulated I_a
Predicted I_a
I_a
DateTime

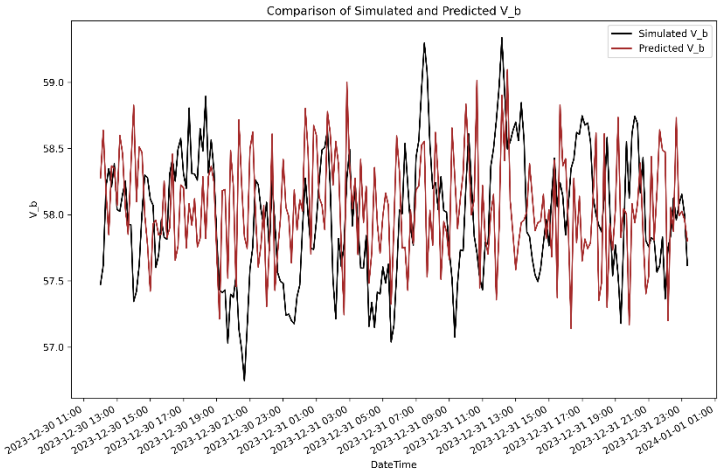


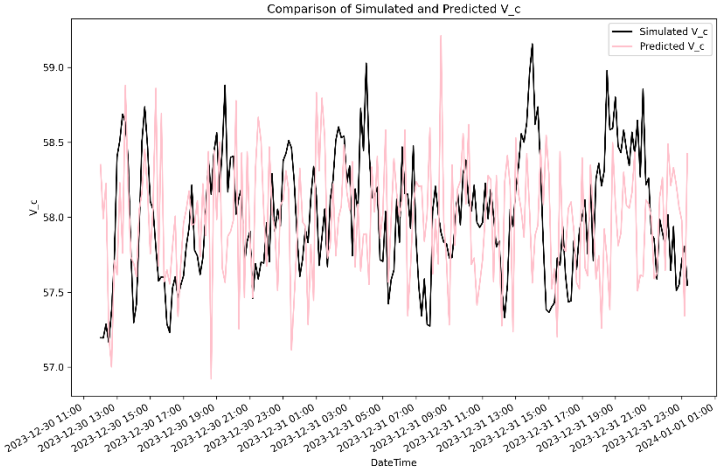


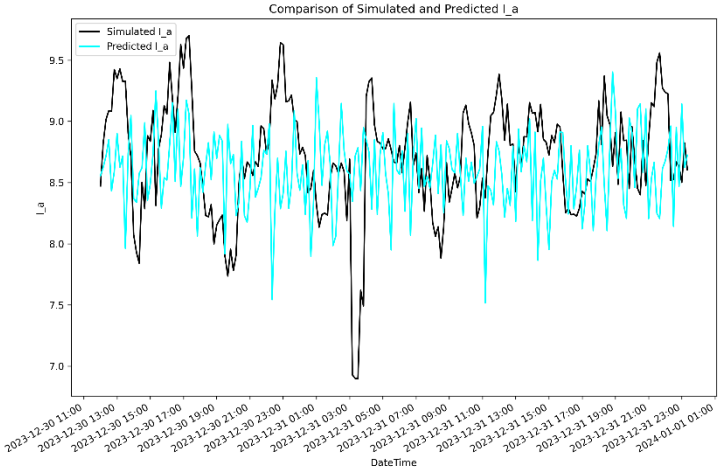
Anexo E : Gráficos gerados cenário 4 (Neuronios 8,32,128)

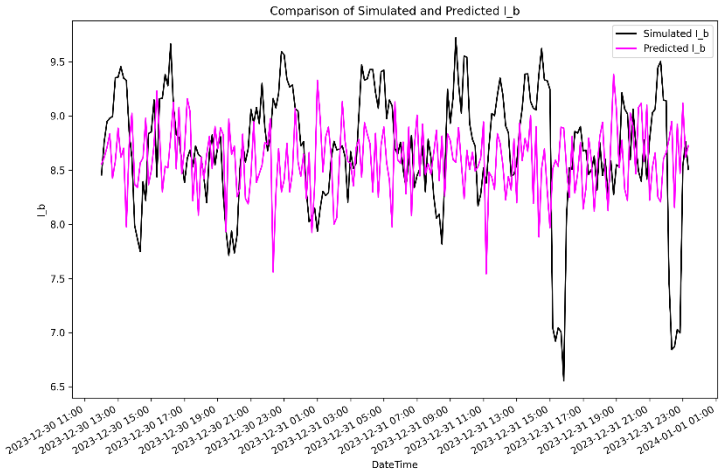
8 Neurónios

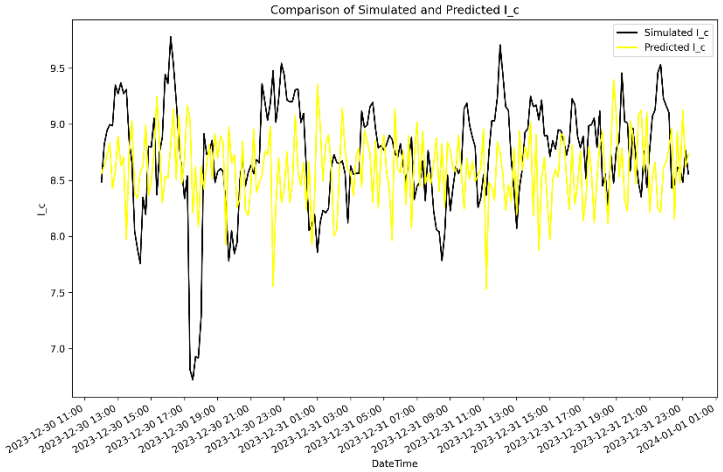



Comparison of Simulated and Predicted V_b
Simulated V_b
Predicted V_b
V_b
DateTime

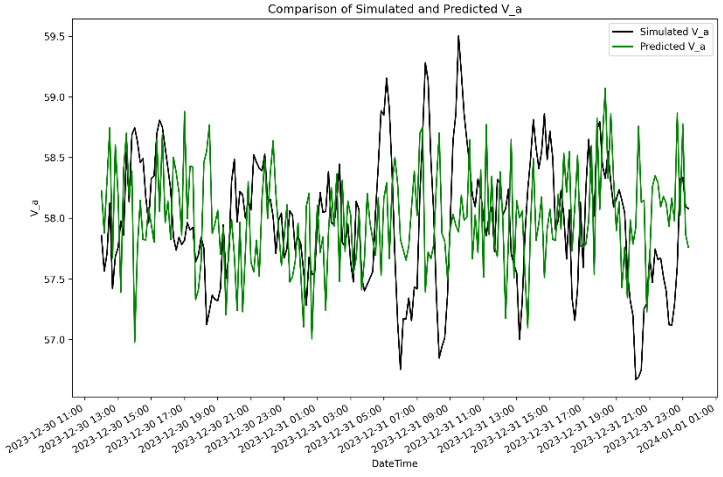

Comparison of Simulated and Predicted V_c
Simulated V_c
Predicted V_c
V_c
DateTime

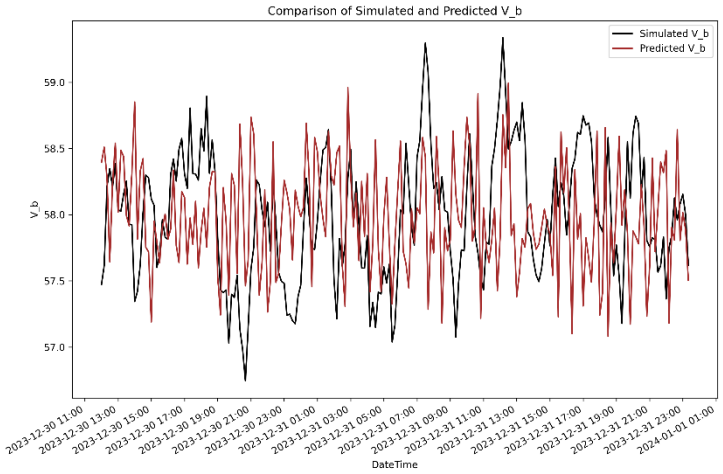

Comparison of Simulated and Predicted I_a
Simulated I_a
Predicted I_a
I_a
DateTime

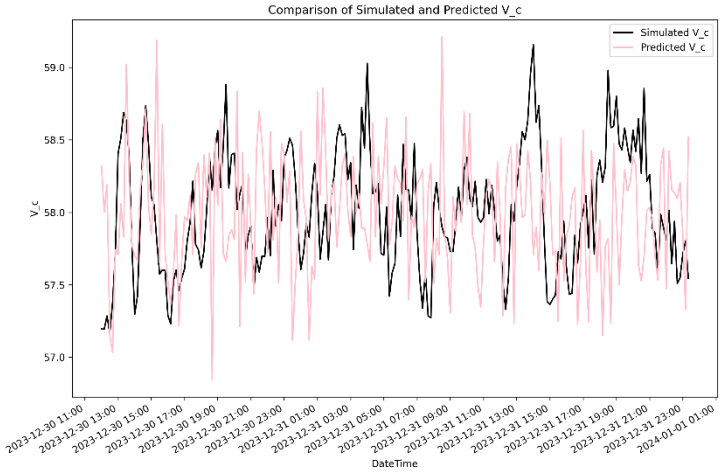


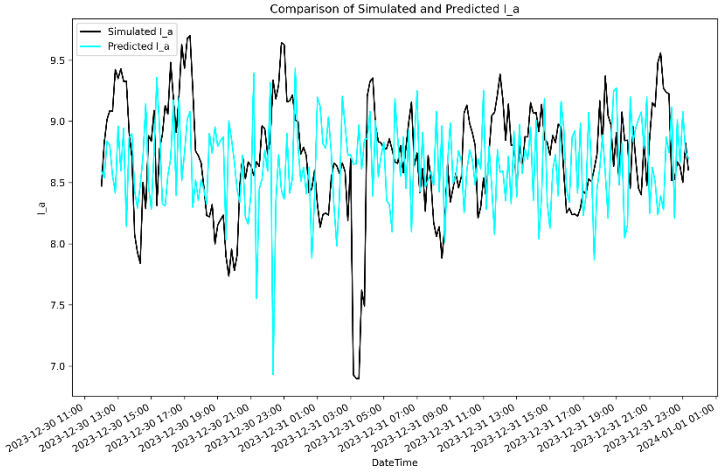


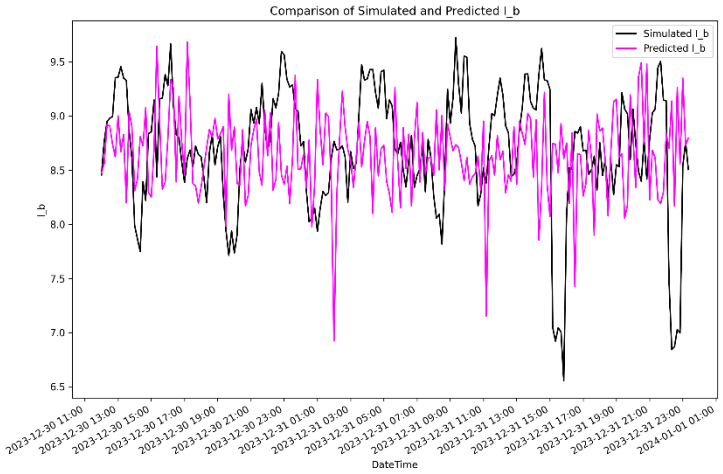
32 neurónios

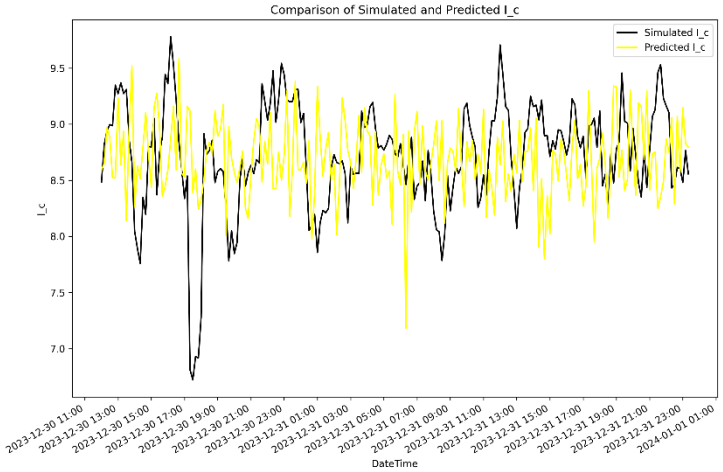



Comparison of Simulated and Predicted V_b
Simulated V_b
Predicted V_b
V_b
DateTime

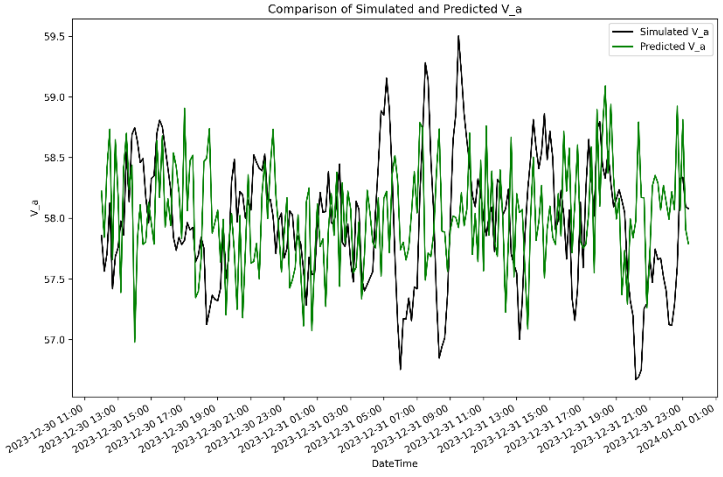

Comparison of Simulated and Predicted V_c
Simulated V_c
Predicted V_c
V_c
DateTime

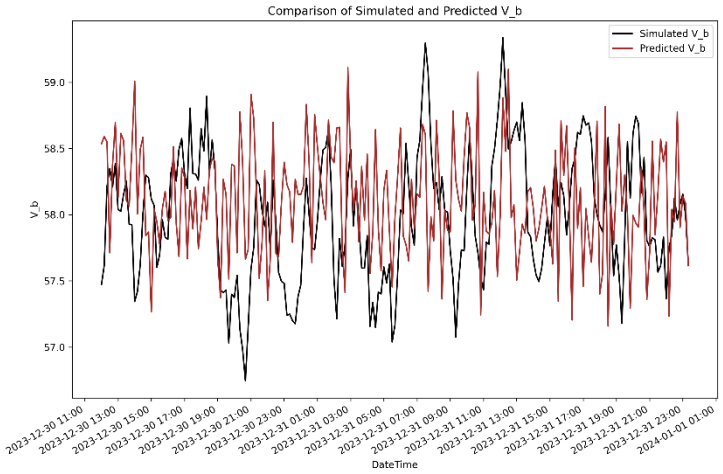

Comparison of Simulated and Predicted I_a
Simulated I_a
Predicted I_a
I_a
DateTime

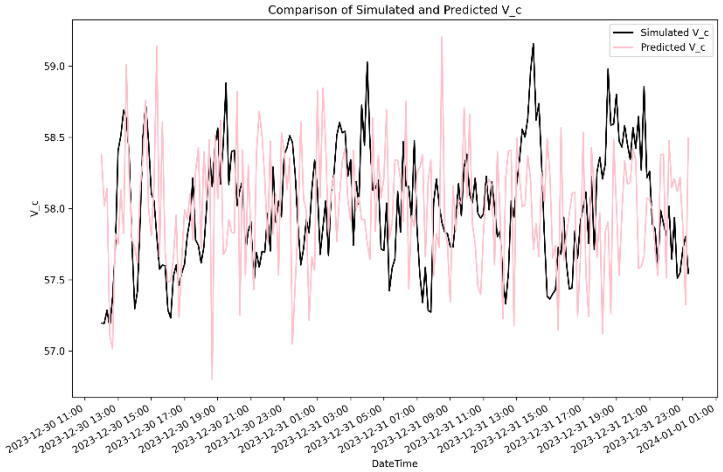


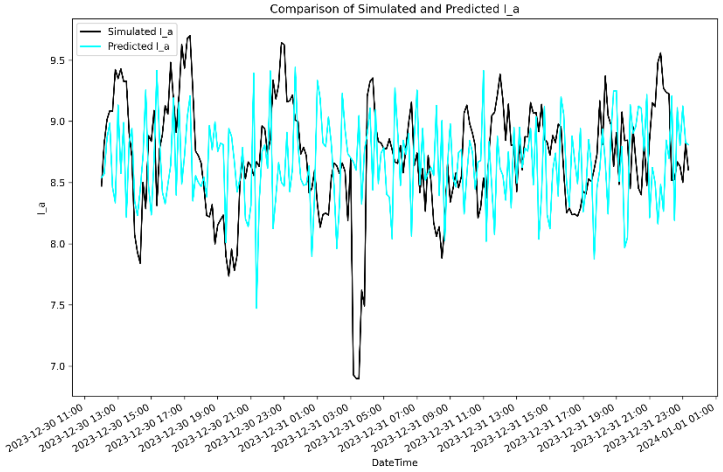


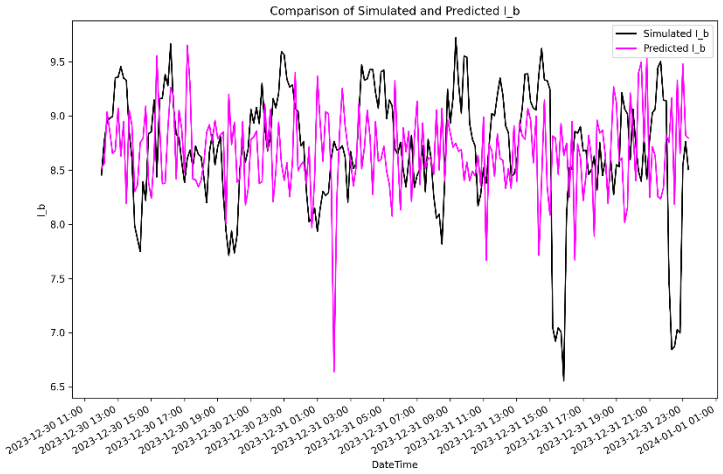
128 neurónios

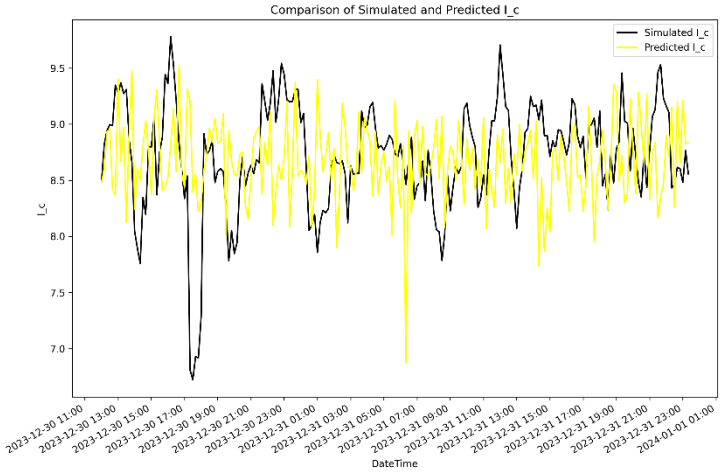



Comparison of Simulated and Predicted V_b
Simulated V_b
Predicted V_b
V_b
DateTime

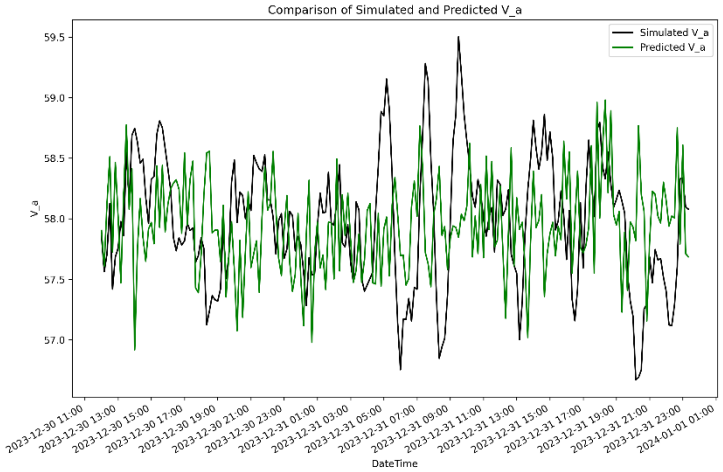

Comparison of Simulated and Predicted V_c
Simulated V_c
Predicted V_c
V_c
DateTime

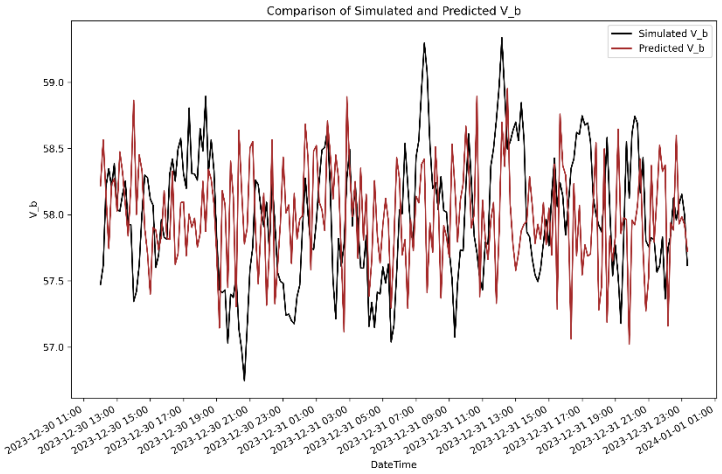

Comparison of Simulated and Predicted I_a
Simulated I_a
Predicted I_a
I_a
DateTime

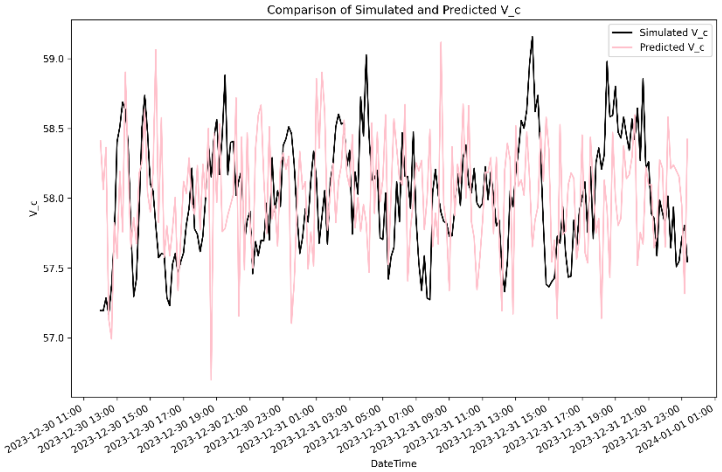


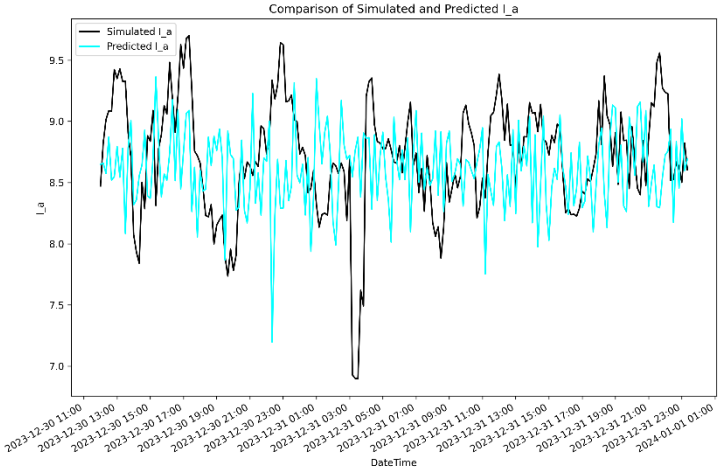


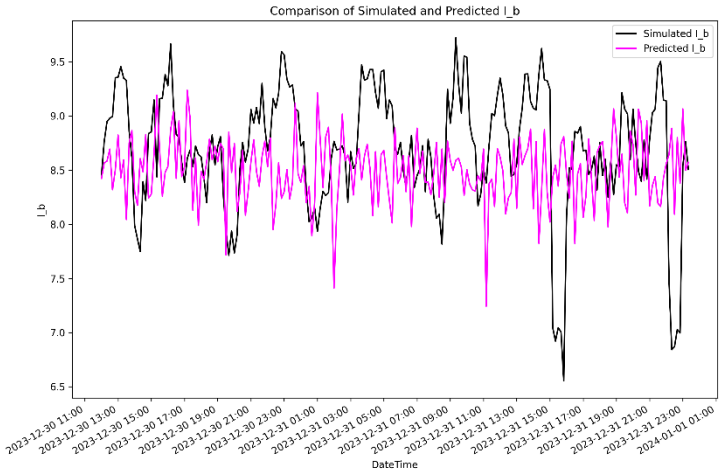
Anexo F : Gráficos gerados cenário 5 (*Epochs* 1, 10 e 20)

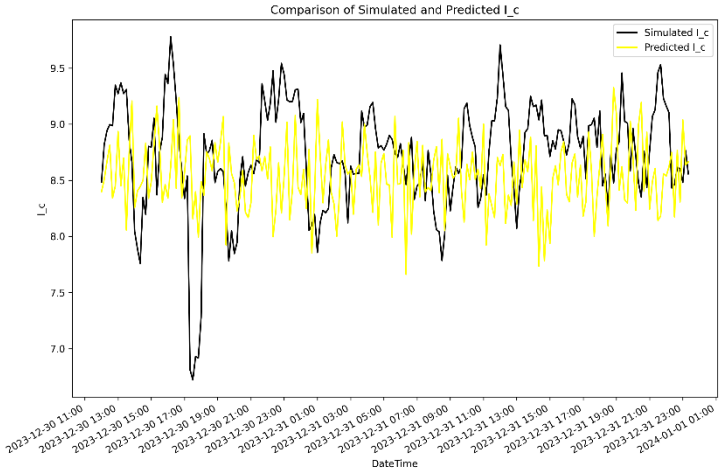



Comparison of Simulated and Predicted V_b
Simulated V_b
Predicted V_b
V_b
DateTime

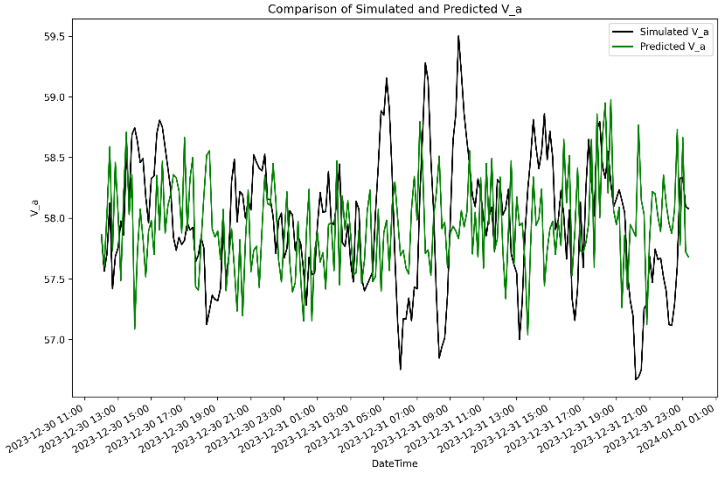

Comparison of Simulated and Predicted V_c
Simulated V_c
Predicted V_c
V_c
DateTime

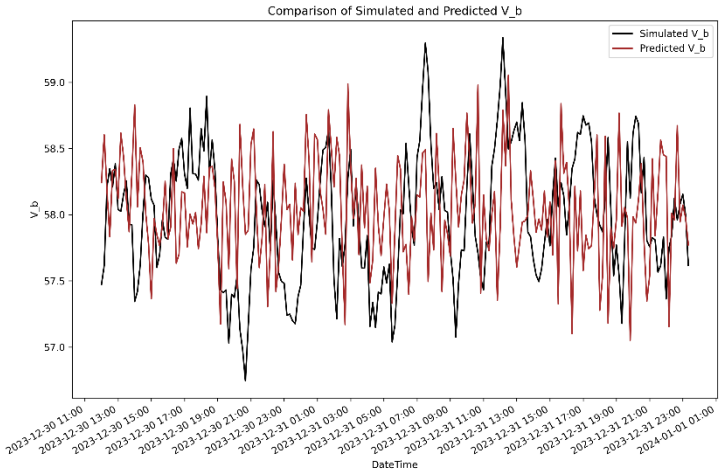

Comparison of Simulated and Predicted I_a
Simulated I_a
Predicted I_a
I_a
DateTime

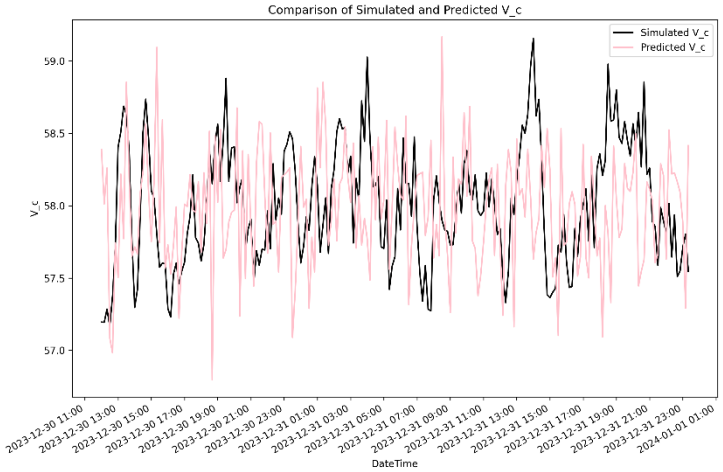

Comparison of Simulated and Predicted I_b
Simulated I_b
Predicted I_b
I_b
DateTime

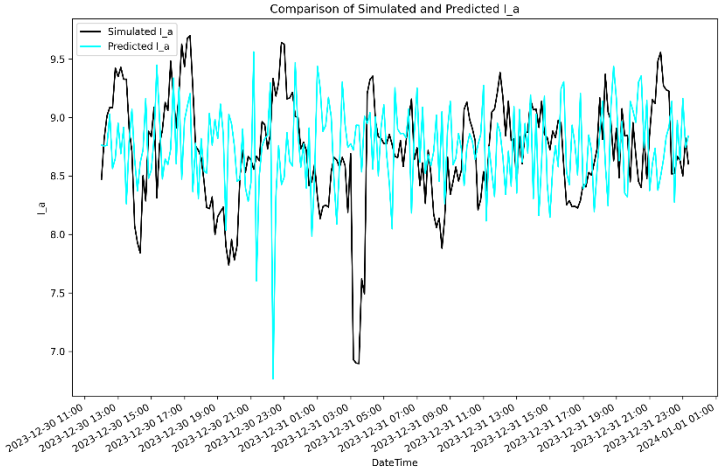

Comparison of Simulated and Predicted I_c
Simulated I_c
Predicted I_c
I_c
DateTime

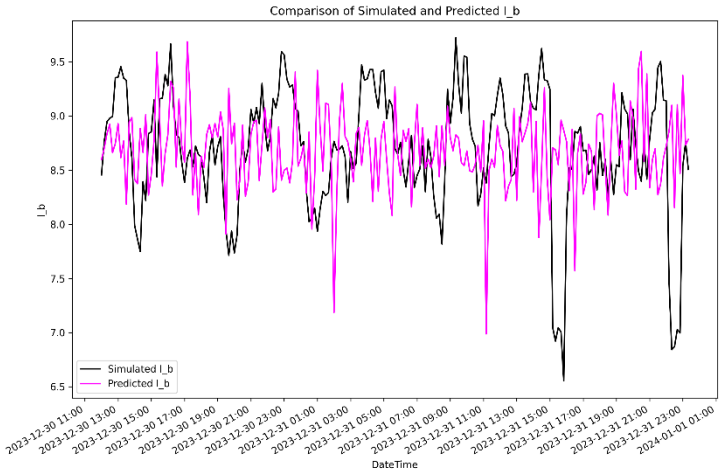
15 *epochs*

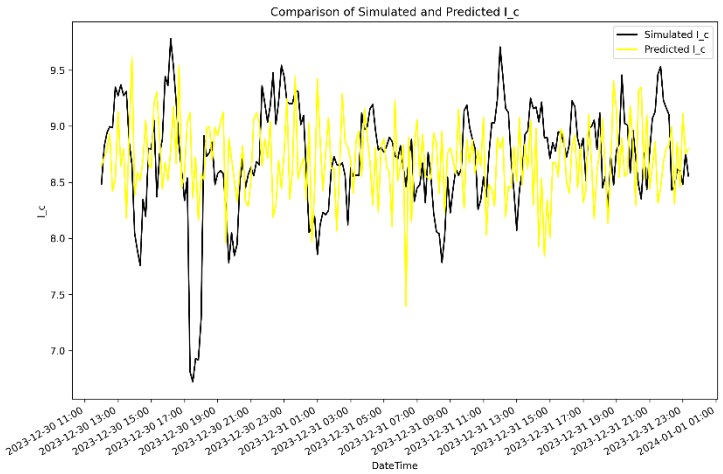

Comparison of Simulated and Predicted V_a
Simulated V_a
Predicted V_a
V_a
DateTime


Comparison of Simulated and Predicted V_b
Simulated V_b
Predicted V_b
V_b
DateTime

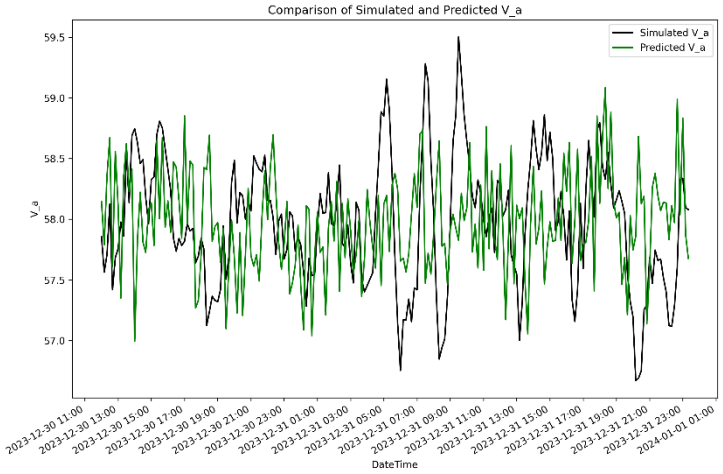

Comparison of Simulated and Predicted V_c
Simulated V_c
Predicted V_c
V_c
DateTime

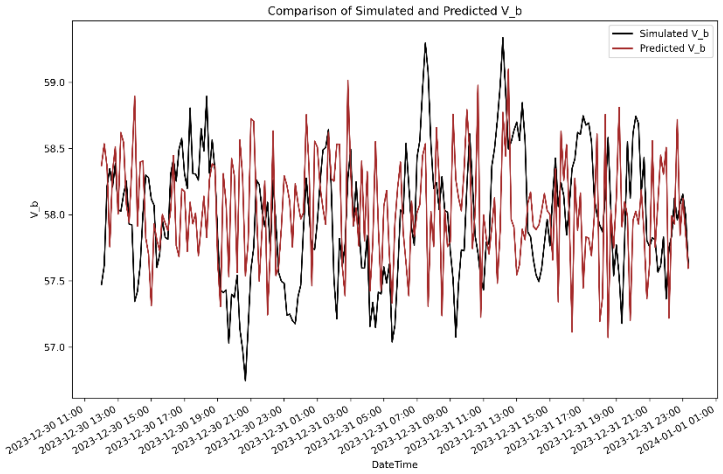

Comparison of Simulated and Predicted I_a
Simulated I_a
Predicted I_a
I_a
DateTime

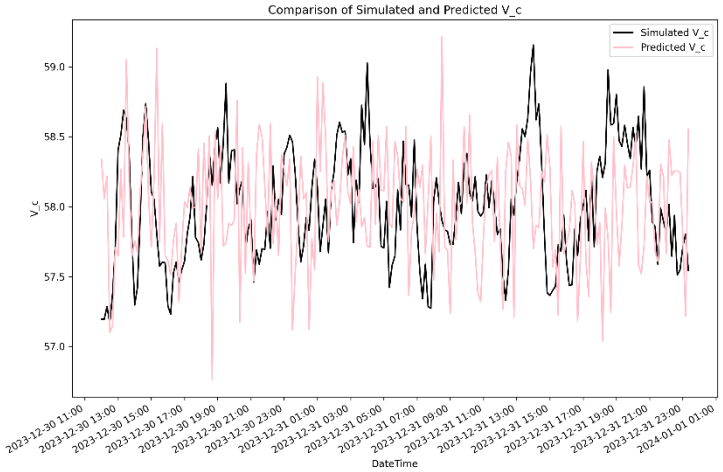

Comparison of Simulated and Predicted I_b
I_b
DateTime
Simulated I_b
Predicted I_b

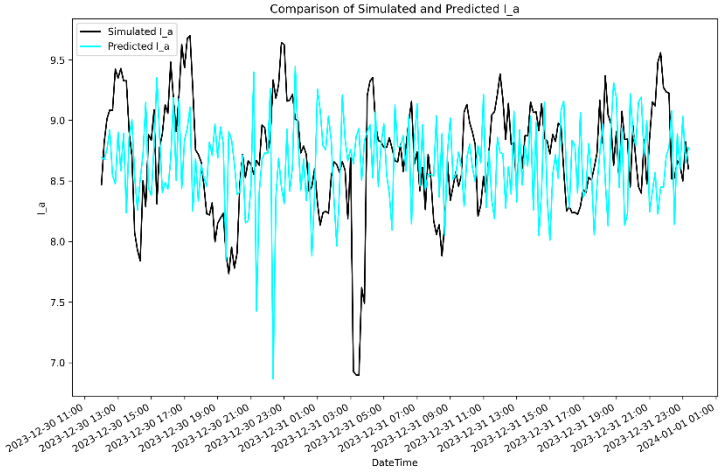

Comparison of Simulated and Predicted I_c
I_c
DateTime
Simulated I_c
Predicted I_c

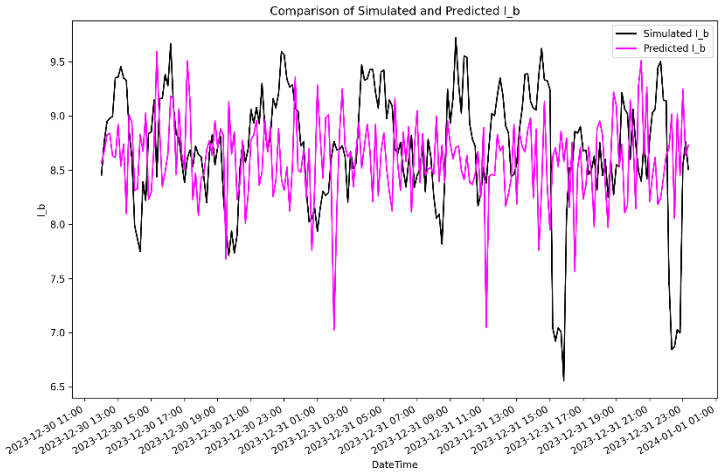
20 epochs

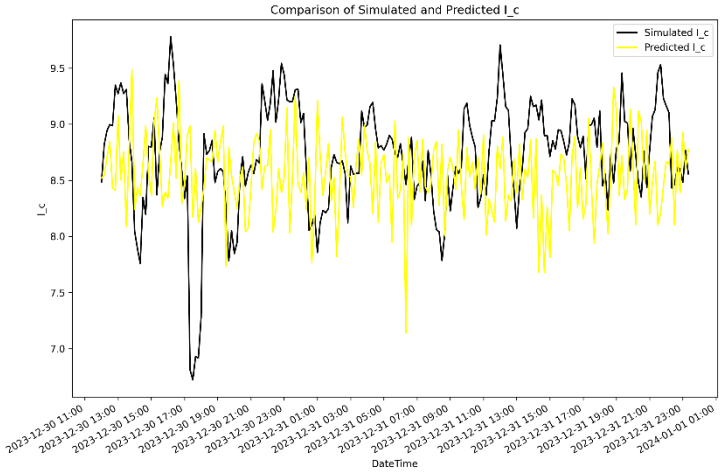

Comparison of Simulated and Predicted V_a
V_a
DateTime
Simulated V_a
Predicted V_a


Comparison of Simulated and Predicted V_b
Simulated V_b
Predicted V_b
V_b
DateTime


Comparison of Simulated and Predicted V_c
Simulated V_c
Predicted V_c
V_c
DateTime


Comparison of Simulated and Predicted I_a
Simulated I_a
Predicted I_a
I_a
DateTime


Comparison of Simulated and Predicted I_b
Simulated I_b
Predicted I_b
I_b
DateTime


Comparison of Simulated and Predicted I_c
Simulated I_c
Predicted I_c
I_c
DateTime

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

P.PORTO

DECLARAÇÃO DE INTEGRIDADE

DECLARAÇÃO DE INTEGRIDADE

Declaro ter conduzido este trabalho académico com integridade. Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração.

Declaro que o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim.

Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P.PORTO.

Pedro João Pereira Alves

ISEP, Porto, 29 de setembro de 2024