INSTITUTO LATINO-AMERICANO
DE CIÊNCIAS DA VIDA E NATUREZA
(ILACVN)

ENGENHARIA FÍSICA

PLATAFORMA INTELIGENTE DE DESIGN ELETRÔNICO AUTOMATIZADO (EDA) BASEADA EM IA PARA PROTOTIPAÇÃO RÁPIDA

Uma abordagem baseada em YOLO e Graph Neural Networks para a geração de netlists SPICE e esquemáticos digitais.

CAIO RAMOS BALLARIN

Foz do Iguaçu, Paraná
2025

INSTITUTO LATINO-AMERICANO
DE CIÊNCIAS DA VIDA E NATUREZA
(ILACVN)

ENGENHARIA FÍSICA

PLATAFORMA INTELIGENTE DE DESIGN ELETRÔNICO AUTOMATIZADO (EDA) BASEADA EM IA PARA PROTOTIPAÇÃO RÁPIDA

Uma abordagem baseada em YOLO e Graph Neural Networks para a geração de netlists SPICE e esquemáticos digitais.

CAIO RAMOS BALLARIN

Trabalho de Conclusão de Curso apresentado ao Instituto Latino-Americano de Ciências da Vida e Natureza da Universidade Federal da Integração Latino-Americana, como requisito parcial à obtenção do título de Bacharel em Engenharia Física.

Orientador: Prof. Dr. João Manoel Lenz Vianna da Silva
Co-orientador: Prof. Dr. Tiago Oliveira Weber

Foz do Iguaçu, Paraná
2025

PLATAFORMA INTELIGENTE DE DESIGN ELETRÔNICO AUTOMATIZADO (EDA) BASEADA EM IA PARA PROTOTIPAÇÃO RÁPIDA

Trabalho de Conclusão de Curso apresentado ao Instituto Latino-Americano de Ciências da Vida e Natureza da Universidade Federal da Integração Latino-Americana, como requisito parcial à obtenção do título de Bacharel em Engenharia Física.

BANCA EXAMINADORA

Orientador: Prof. Dr. João Manoel Lenz Vianna da Silva, UNILA

Coorientador(a): Prof. Dr. Tiago Oliveira Weber, UFRGS

Prof. Dr. Raphael Fortes Infante Gomes, UNILA

Prof. Dr. Adriano Batista de Almeida, UNIOESTE

Foz do Iguaçu, _____ de _____ de _____.

AGRADECIMENTOS

Primeiramente, agradeço a Deus por ter-me protegido e concedido a resiliência necessária para chegar até aqui. Aos meus pais, sou profundamente grato por sempre terem acreditado em mim e nos meus sonhos, por terem-me dado a oportunidade de cursar Engenharia Física e por serem o meu maior apoio — sem eles, eu não teria chegado tão longe; amo-os imensamente. À minha irmã, agradeço por ter estado ao meu lado nos momentos difíceis e por todo o apoio ao longo desta jornada; tenho muito orgulho de ser seu irmão. A uma amiga que caminhou comigo durante metade da graduação, registro minha gratidão pelo suporte e pela esperança que me ajudou a conservar até a conclusão deste trabalho.

Agradeço também aos colegas que colaboraram na construção do dataset e no treinamento dos modelos de inteligência artificial — a contribuição prática de vocês foi fundamental para o desenvolvimento deste projeto. Aos meus professores, expresso minha gratidão pelo conhecimento transmitido e pelo estímulo acadêmico; em especial ao orientador Prof. Dr. João Manoel Lenz Vianna da Silva, por confiar em mim, abraçar a ideia deste projeto e orientar com empenho, e ao coorientador Prof. Dr. Tiago Oliveira Weber, por suas sugestões valiosas na formulação da metodologia e no encaminhamento das próximas etapas da pesquisa.

Por fim, agradeço à Universidade Federal da Integração Latino-Americana (UNILA) pela oportunidade de pesquisar o tema escolhido e pelo auxílio financeiro que tornou possível a realização deste trabalho de conclusão de curso. A todos que, de diferentes formas, contribuíram para que este momento fosse possível, deixo minha sincera e profunda gratidão.

RESUMO

O projeto propõe um software de Automação de Projeto Eletrônico (EDA) capaz de transformar imagens de esquemáticos em arquivos de projeto de placa, reduzindo intervenção humana nas fases iniciais. A arquitetura inclui quatro módulos: (1) reconhecimento de topologia e geração de netlist, (2) seleção de componentes e montagem da PCB, (3) otimização eletrotérmica e simulação, e (4) exportação dos artefatos do projeto. Este estudo foca no módulo (1), usando o conversor Buck como caso de validação.

O reconhecimento combina detecção visual e aprendizado por grafos: um detector YOLO11 aprimorado com o módulo de atenção GAM (YOLO-GAM) extrai símbolos e caixas delimitadoras; um pós-processamento constrói um grafo de componentes e conexões; finalmente, uma Graph Neural Network (GNN) valida e completa a netlist. O conversor Buck foi escolhido por reunir desafios típicos (componentes pequenos, junções e textos sobrepostos, variedade gráfica), nos quais o GAM melhora seletividade e a GNN incorpora restrições topológicas para corrigir ambiguidades e reduzir falsos positivos.

As contribuições principais são: (i) avaliação empírica do impacto do módulo GAM na detecção de símbolos de pequenas dimensões; e (ii) integração prática entre o output do detector e a construção/classificação de grafos via GNN, com métricas que quantificam a fidelidade da netlist. Resultados indicam que a combinação YOLO-GAM + GNN oferece uma solução robusta e eficiente para automatizar a etapa inicial do fluxo EDA, acelerando prototipagem e diminuindo retrabalho.

Palavras-chave: Automação de Projeto Eletrônico; Reconhecimento de esquemáticos; Geração de netlist; YOLO-GAM; Graph Neural Networks; Conversor Buck.

RESUMEN

Este proyecto propone un software de Automatización de Diseño Electrónico (EDA) capaz de transformar imágenes de esquemáticos en archivos de diseño de placas, reduciendo la intervención humana en las etapas iniciales. La arquitectura incluye cuatro módulos principales: (1) reconocimiento de topología y generación de netlist, (2) selección de componentes y ensamblaje de la PCB, (3) optimización y simulación electro-térmica, y (4) exportación de los artefactos del proyecto. Este estudio se centra en el módulo (1), utilizando el convertidor Buck como caso de validación.

El reconocimiento combina detección visual y aprendizaje basado en grafos: un detector YOLO11 mejorado con el módulo de atención GAM (YOLO-GAM) extrae símbolos y cajas delimitadoras; una rutina de posprocesamiento construye un grafo de componentes y conexiones; finalmente, una Red Neuronal de Grafos (GNN) valida y completa la netlist resultante. El convertidor Buck fue elegido porque recoge desafíos típicos de esquemáticos (componentes pequeños, uniones y textos superpuestos, variabilidad gráfica), donde GAM mejora la selectividad del detector y la GNN incorpora restricciones topológicas para corregir ambigüedades y reducir falsos positivos.

Las principales contribuciones del trabajo son: (i) la evaluación empírica del impacto del módulo GAM en la detección de símbolos pequeños; y (ii) la integración práctica entre la salida del detector y la construcción/clasificación del grafo mediante GNN, utilizando métricas que cuantifican la fidelidad de la netlist. Los resultados indican que la combinación YOLO-GAM y GNN representa una solución eficiente y robusta para automatizar la fase inicial del flujo EDA, acelerando la prototipación y reduciendo retrabajos.

Palabras clave: Automatización del Diseño Electrónico; Reconocimiento de esquemáticos; Generación de netlist; YOLO-GAM; Redes Neuronales de Grafos; Convertidor Buck.

ABSTRACT

This project proposes an Electronic Design Automation (EDA) software capable of transforming schematic images into printed circuit design files, reducing human intervention during early design stages. The architecture includes four main modules: (1) topology recognition and netlist generation, (2) component selection and PCB assembly, (3) electrothermal optimization and simulation, and (4) export of project artifacts. This study concentrates on module (1), using the Buck converter as the validation case.

The recognition process combines visual detection and graph-based learning: a YOLO11 detector enhanced with the GAM attention module (YOLO-GAM) extracts symbols and bounding boxes; a post-processing routine constructs a graph representing components and connections; finally, a Graph Neural Network (GNN) validates and completes the resulting netlist. The Buck converter was chosen because it encompasses typical schematic challenges (small components, junctions and overlapping text, graphic variability), where GAM improves detector selectivity and GNN incorporates topological constraints to correct ambiguities and reduce false positives.

The main contributions are: (i) empirical evaluation of the GAM module's impact on detecting small symbols; and (ii) practical integration between detector outputs and graph construction/classification via GNNs, using metrics that quantify netlist fidelity. Results indicate that the combination of YOLO-GAM and GNN offers an efficient and robust solution for automating the initial EDA flow, accelerating prototyping and reducing rework.

Key words: Electronic Design Automation; Schematic Recognition; Netlist Generation; YOLO-GAM; Graph Neural Networks; Buck Converter.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CNN	Rede Neural Convolucional — arquitetura de redes neurais projetada para processar dados com estrutura de grade (por exemplo, imagens), usando filtros convolucionais para extrair características locais.
GNN	Rede Neural sobre Grafos (*Graph Neural Network*) — classe de modelos que realiza aprendizado em estruturas de grafo, propagando e agregando informações entre nós via message passing.
mAP	*mean Average Precision* — métrica de avaliação para detecção de objetos que agrega a precisão média em diferentes níveis de recall (frequentemente usada como mAP@50–95).
YOLO	*You Only Look Once* — família de modelos de detecção de objetos em tempo real que realiza classificação e localização em um único passo de inferência.
GAM	Módulo de Atenção Global (*Global Attention Module*) — bloco de atenção que pode ser acoplado a redes convolucionais para melhorar a modelagem de dependências globais nas features.
GED	*Graph Edit Distance* — medida de similaridade entre grafos definida como o custo mínimo de operações (inserção, remoção, substituição de nós/arestas) para transformar um grafo em outro.
MCS	*Maximum Common Subgraph* (Subgrafo Comum Máximo) — problema/medida que identifica o maior subgrafo comum entre dois grafos; (alternativamente, em outro contexto, MCS pode significar Monte Carlo Simulation).
CC	Corrente Contínua (DC) — forma de corrente elétrica cujo fluxo mantém polaridade constante.
CA	Corrente Alternada (AC) — forma de corrente elétrica cuja polaridade inverte periodicamente.
HVDC	*High Voltage Direct Current* — transmissão em corrente contínua em alta tensão (transmissão CC em alta tensão).
CCM	*Continuous Conduction Mod*e — modo de operação de conversores onde a corrente por um indutor nunca atinge zero durante o ciclo.
DCM	*Discontinuous Conduction Mode* — modo de operação de conversores onde a corrente por um indutor chega a zero em parte do ciclo.
DNN	Rede Neural Profunda (*Deep Neural Network*) — redes neurais com múltiplas camadas contendo grande número de parâmetros, capazes de aprender representações hierárquicas.
CL	(Interpretar conforme contexto) — possíveis expansões: *Classificação (Classification)*; *Command Line (linha de comando)*; *Current Loop (laço de corrente)*. Escolha a que se aplica ao seu caso.
PL	(Interpretar conforme contexto) — possíveis expansões: *Programming Language (linguagem de programação)*; *Power Loss (perda de potência)*; *Pipeline (fluxo de processamento)*. Escolha a que se aplica ao seu caso.
FL	(Interpretar conforme contexto) — possíveis expansões: *Federated Learning (aprendizado federado)*; *Feature Learning (aprendizado de características)*; *Fault Level (nível de falha)*. Escolha a que se aplica ao seu caso.

NMS *Non-Maximum Suppression* — técnica utilizada em detecção de objetos para suprimir caixas sobrepostas e manter apenas as detecções mais confiáveis.

I/O *Input/Output* — entradas e saídas de um sistema (dados, sinais ou dispositivos).

TP *True Positive* — detecção correta de uma instância positiva (verdadeiro positivo).

FP *False Positive* — detecção incorreta onde algo negativo foi classificado como positivo (falso positivo).

FN *False Negative* — falha em detectar uma instância positiva (falso negativo).

F1 F1-score — métrica harmônica entre precisão (precision) e recall: $F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$.

IoU *Intersection over Union* — métrica que mede a sobreposição entre a caixa prevista e a caixa real: razão entre interseção e união das duas caixas.

OCR *Optical Character Recognition* — Reconhecimento Ótico de Caracteres; técnicas para extrair texto de imagens.

CRB *Circuit Reliability Bench* — plataforma usada neste trabalho para validar e simular netlists e verificar consistência/funcionamento dos circuitos gerados.

API *Application Programming Interface* — conjunto de rotinas e padrões que permitem que aplicações se comuniquem entre si.

GAT *Graph Attention Network* — variante de GNN que usa mecanismos de atenção para ponderar mensagens de vizinhos de forma adaptativa.

CLAHE *Contrast Limited Adaptive Histogram Equalization* — técnica de equalização adaptativa de histograma com limitação de contraste, usada para melhorar contraste local em imagens.

SMOTE *Synthetic Minority Over-sampling Technique* — técnica para balanceamento de classes que gera novas amostras sintéticas da classe minoritária.

EDA *Electronic Design Automation* — conjunto de ferramentas computacionais utilizadas para projetar, modelar, verificar e preparar circuitos eletrônicos e sistemas integrados para fabricação, incluindo etapas como esquemático, síntese, simulação e análise.

IA Inteligência Artificial.

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento de sistemas eletrônicos consolidou-se como um dos pilares fundamentais da indústria moderna, sustentando inovações em setores críticos como dispositivos móveis, indústria automotiva, sistemas de energia e telecomunicações. Para suportar essa demanda, ferramentas e fluxos de *Electronic Design Automation* (EDA) tornaram-se indispensáveis nas etapas de planejamento, síntese, verificação e preparação de circuitos para a fabricação. Tais recursos permitem o gerenciamento da crescente complexidade dos projetos e a mitigação de riscos, garantindo a viabilidade econômica e técnica da produção em larga escala [1, 2].

Apesar da sofisticação dessas ferramentas, a etapa de prototipagem que envolve a conversão de conceitos abstratos ou diagramas visuais em arquivos digitais processáveis (listas de conexões ou *netlists*) permanece um gargalo significativo. Frequentemente, as fases iniciais de design, como a definição da topologia e a organização do layout de placas de circuito impresso (PCB), ainda dependem de intervenções manuais intensivas. A literatura aponta que essa dependência manual resulta em processos lentos, propensos a erros humanos e sujeitos a retrabalhos onerosos, o que impacta diretamente o tempo de colocação do produto no mercado e os custos globais do ciclo de desenvolvimento [3, 4].

Na busca por otimizar esse fluxo, pesquisas recentes têm explorado a aplicação de Inteligência Artificial (IA) e Visão Computacional para automatizar a interpretação de esquemáticos. Iniciativas acadêmicas e industriais têm utilizado algoritmos de detecção de objetos, como variantes da arquitetura YOLO, Mask R-CNN e outros modelos profundos, para identificar símbolos e componentes em diagramas, tanto digitais quanto manuscritos [5]. O desafio central abordado por diversos autores reside na complexidade visual desses documentos: esquemáticos eletrônicos frequentemente apresentam alta densidade de elementos, símbolos de pequenas dimensões, junções complexas e sobreposição de textos, condições que degradam a precisão de detectores convencionais e elevam a taxa de falsos positivos.

Para mitigar essas limitações de detecção em cenários complexos, o estado da arte em visão computacional tem evoluído para a incorporação de mecanismos de atenção. Técnicas como o *Global Attention Module* (GAM) têm sido estudadas como formas de aprimorar a extração de características, permitindo que as redes neurais enfatizem canais e regiões espaciais relevantes, suprimindo ruídos de fundo. Diferentemente de mecanismos restritos apenas a canais, abordagens que preservam informações posicionais e contextuais mostram-se promissoras para melhorar a relação sinal/ruído na detecção de pequenos objetos, uma característica crítica para a identificação precisa de componentes eletrônicos [6].

Entretanto, a simples detecção dos componentes não é suficiente para a reconstrução funcional de um circuito. A literatura especializada destaca que circuitos devem ser interpretados como grafos, cujo os componentes são nós e conexões são arestas. Consequentemente, métodos que integram a detecção visual com a inferência estrutural, como Redes Neurais de Grafos (GNN), têm

ganhado destaque para a geração automática de netlists compatíveis com softwares de simulação [7, 8]. Apesar desses avanços teóricos, ainda existe uma carência documentada de sistemas robustos que integrem eficientemente a detecção aprimorada por atenção com a modelagem de grafos, além da escassez de bancos de dados padronizados para o treinamento desses modelos híbridos em topologias de potência.

Diante desse cenário, este trabalho propõe o desenvolvimento de uma abordagem híbrida para a digitalização automática de circuitos. O sistema proposto integra um detector baseado na arquitetura YOLO, aprimorado com módulos de atenção global (GAM-YOLO) para robustez na identificação de componentes, com uma rede neural baseada em grafos para a inferência das conexões elétricas. O foco da aplicação será a detecção e a geração autônoma de netlists de conversores Buck, visando validar a capacidade do sistema em converter imagens estáticas de esquemáticos diretamente em arquivos aptos para simulação elétrica.

1.1 OBJETIVOS

Objetivo geral: Desenvolver e validar um pipeline híbrido (GAM-YOLO + GNN) capaz de transformar imagens de esquemáticos em netlists SPICE verificáveis, reduzindo a intervenção manual na etapa de prototipagem e aumentando a reprodutibilidade do fluxo EDA.

Objetivos específicos

- Implementar e treinar um detector GAM-YOLO para detecção robusta de símbolos em esquemáticos digitalizados e manuscritos.
- Projetar um módulo de pós-processamento que construa representações gráficas (grafos de circuito) a partir das detecções visuais.
- Desenvolver e treinar um modelo GNN para inferência de conectividade e geração de netlists a partir da representação em grafo.
- Avaliar quantitativamente a solução em métricas de detecção (mAP), fidelidade topológica (GED/MCS) e validação funcional (simulação transiente).

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, apresenta-se a base conceitual que sustenta o desenvolvimento deste trabalho. São discutidos os princípios fundamentais de detecção de componentes em esquemáticos eletrônicos, a extração de informações textuais via OCR, a transformação dessas informações em grafos representativos da topologia do circuito e o uso de redes neurais baseadas em grafos (GNN) para inferir sua classificação. Além disso, aborda-se como a geração de netlists e sua validação por simulação fecham o ciclo entre visão computacional e análise elétrica, estabelecendo o fundamento necessário para compreender as etapas, decisões de projeto e resultados obtidos ao longo deste estudo.

2.1 ELETRÔNICA DE POTÊNCIA

Os circuitos de eletrônica de potência convertem energia elétrica entre diferentes níveis de tensão, corrente ou frequência por meio de dispositivos semicondutores operando como chaves, geralmente em altas frequências e controladas. Essa técnica permite adaptar o processamento de energia elétrica às necessidades da carga com alta eficiência, abrangendo aplicações que vão de carregadores portáteis a sistemas de grande porte, como conversores HVDC (do inglês, *High Voltage Direct Current*). A área combina fundamentos de circuitos, semicondutores, controle e fenômenos eletromagnéticos, sendo impulsionada pela demanda por soluções mais eficientes e compactas.

Um conversor funciona como interface entre fonte e carga, adequando a forma de onda, amplitude ou frequência da tensão ou corrente, como mostra na Figura 1. Conversores CC–CC, CC–CA, CA–CC e CA–CA são usados para fornecer à carga condições elétricas específicas, mesmo quando a fonte não atende diretamente esses requisitos. Por exemplo, conversores CA–CA podem regular simultaneamente a tensão e frequência na saída, permitindo alimentar cargas industriais projetadas para padrões distintos da rede convencional.

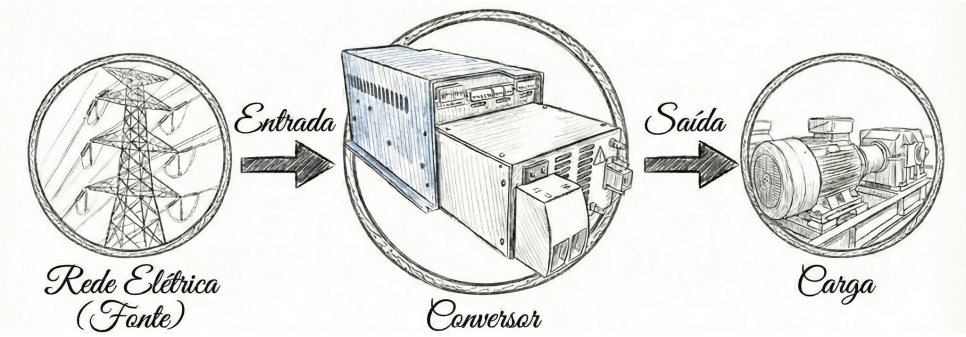


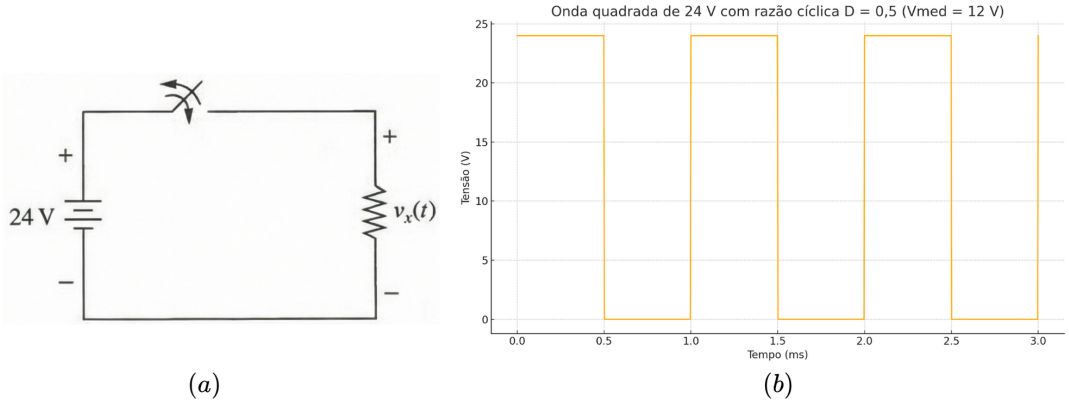
Figura 1: Uma fonte e uma carga com um conversor eletrônico de potência como interface

Fonte: Criado pelo Gemini 3 Pro, baseado no livro Eletrônica de Potência de Hart W. Daniel. [9]

A conversão eficiente é obtida a partir do chaveamento rápido dos dispositivos semicondutores, e não da dissipação contínua de energia. Diferentemente de resistores ou transistores operando na região linear, no qual o controle da tensão implica perdas significativas, dispositivos

semicondutores atuando como chaves operam em dois estados extremos: condução quase sem queda de tensão e corte com corrente nula. A troca entre estes dois estados, de modo controlado, permite regular o processamento de energia entre entrada e saída com pouca dissipação por perdas elétricas.

Para ilustrar esse princípio, a Figura 2 mostra um conversor chaveado básico, cujo o acionamento da chave S é feito de modo a regular a tensão sobre o resistor de saída em 12 V, sendo a alimentação do circuito em 24 V. Soluções lineares, como um resistor série ou um regulador linear, reduzem a tensão somente à custa de dissipação significativa de energia (na forma de calor), enquanto que o conversor chaveado ideal não possui perdas.



(a) (b)

Figura 2: (a) Um circuito chaveado; (b) uma forma de onda da tensão chaveada.

Fonte: O Autor (2025), baseado no livro Eletrônica de Potência de Hart W. Daniel. [9]

O valor médio da tensão da saída é função da tensão de entrada, sendo controlado pela razão cíclica da chave, que é a fração do tempo em que a chave permanece conduzindo dentro do período de chaveamento. Quando a chave está fechada a tensão da saída é de 24 V e 0 V quando está aberta. Se a chave conduzir metade do período de chaveamento (razão cíclica 50%), resulta com que a média da tensão na saída ao longo do tempo seja 12 V.

Para uma onda quadrada que vale V_{in} durante um tempo T_{on} em cada período T e 0V no restante, a tensão média é:

$$V_{\text{med}} = \frac{1}{T}\int_0^T v(t)dt \tag{1}$$

como v(t)= V_{in} apenas no intervalo em que a chave está ligada, temos:

$$V_{\text{med}} = \frac{1}{T} \cdot V_{\text{in}} \cdot T_{\text{on}} \tag{2}$$

definindo a razão cíclica (*duty cycle*) como:

$$D = \frac{T_{\text{on}}}{T} \tag{3}$$

a expressão fica:

$$V_{\text{med}} = D \cdot V_{\text{in}}. \tag{4}$$

2.2 SIMULAÇÃO SPICE

O SPICE (*Simulation Program with Integrated Circuit Emphasis*) é um dos simuladores de circuitos eletrônicos mais utilizados na engenharia elétrica desde sua criação na Universidade da Califórnia, Berkeley. Ele representa circuitos por meio de modelos matemáticos dos componentes com alta complexidade e resolve, numericamente, o conjunto de equações algébricas e diferenciais resultantes. Graças a essa abordagem, o SPICE permite prever o comportamento de tensões, correntes e potências antes mesmo da montagem física do circuito, reduzindo custos, tempo de prototipação e riscos de falha.

Assim, o SPICE tornou-se um dos padrões para simulação em eletrônica por sua precisão, flexibilidade e compatibilidade com modelos de fabricantes. A linguagem SPICE, baseada em *netlists*, descreve o circuito em termos de nós, elementos e parâmetros, permitindo simulações detalhadas de dispositivos semicondutores, conversores de potência, filtros, amplificadores e sistemas analógicos complexos.

O LTspice é uma implementação moderna e otimizada do SPICE, desenvolvida pela Linear *Technology* (hoje *Analog Devices*). Ele reúne a robustez do SPICE com uma interface gráfica que facilita a construção de esquemas, inserção de fontes, definição de análises e visualização de resultados. Embora o usuário interaja principalmente com o ambiente gráfico, o LTspice funciona internamente como um SPICE tradicional: cada circuito desenhado é convertido automaticamente em uma *netlist* SPICE, que é a descrição base do simulador.

Para a construção do esquema no LTspice, cria-se um arquivo de desenho (***.asc*** que descreve a topologia do circuito a ser analisado (ver Figura 3 e 4). Cada componente do esquema pode referenciar um modelo real provido pelo fabricante ou definido pelo usuário, o que torna a simulação mais fiel ao comportamento físico: modelos detalhados permitem estimar perdas por condução e comutação e, consequentemente, obter uma tensão de saída mais representativa da condição real de operação.

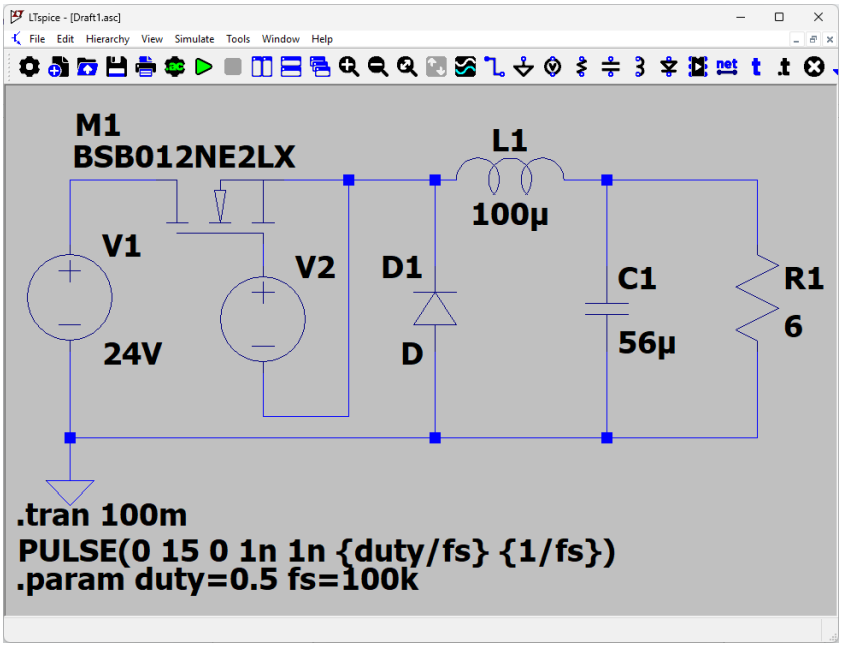


Figura 3: Exemplo de esquemático no software LTspice

Fonte: Autor (2025).

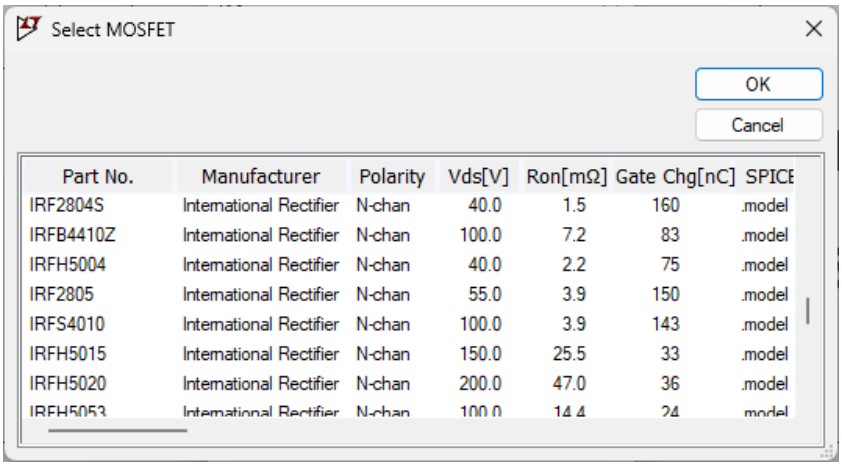


Figura 4: Exemplo da biblioteca de mosfets no software LTspice

Fonte: Autor (2025).

Além do editor esquemático, o LTspice produz a *netlist* do circuito (exportável em *.net*), que contém a listagem textual de instâncias de elementos, nós, parâmetros, nomes de modelos e diretivas de simulação (Figura 5). A netlist é a representação canônica usada pelo motor SPICE para montar o sistema de equações e executar as análises solicitadas.

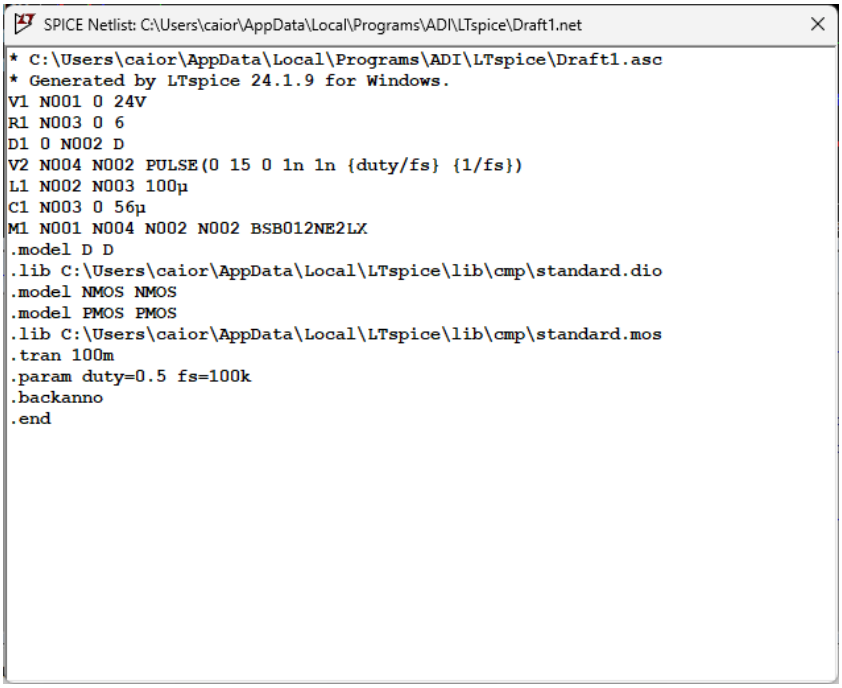


Figura 5: Netlist do circuito de exemplo

Fonte: Autor (2025).

O LTspice disponibiliza diversos tipos de análise: transitória, AC (pequenos sinais em frequência), varredura DC, análise de ruído, entre outras. Para conversores chaveados, a análise transitória é a mais apropriada, pois permite observar a evolução dinâmica do sistema desde a comutação inicial até a convergência ao regime permanente, dentro do software é definida na janela mostrada na Figura 6.

Neste trabalho, adotou-se um tempo de simulação de 200 ms, escolhido não de forma arbitrária, mas com base na finalidade da simulação dentro do fluxo proposto. Como o sistema de automação permite que o usuário defina parâmetros elétricos (por exemplo, L, C, carga e frequência de chaveamento), não é possível prever, a priori, quanto tempo o conversor levará para atingir regime permanente em todos os casos. Assim, utilizou-se um tempo de parada suficientemente elevado para garantir que tanto os transientes iniciais quanto a estabilização da resposta fossem capturados, assegurando a confiabilidade da etapa de verificação automática do circuito gerado.

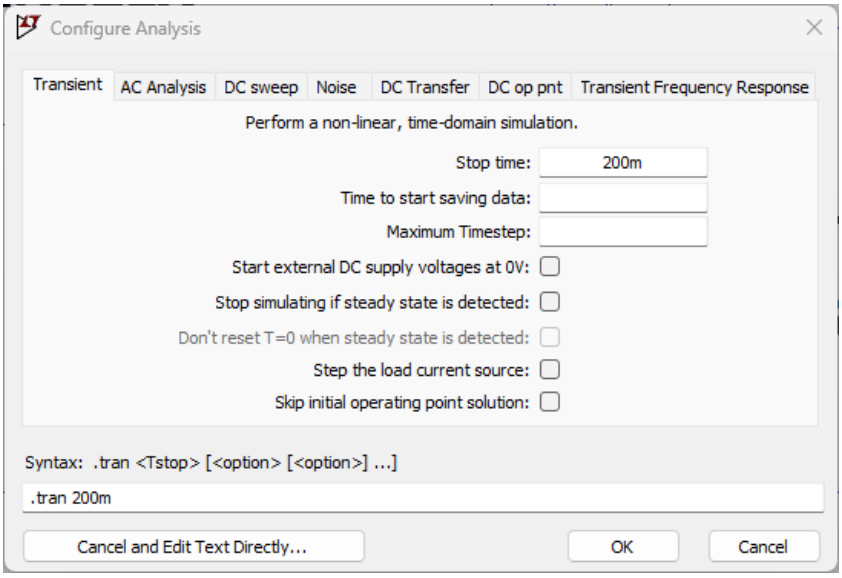


Figura 6: Janela de configuração da simulação SPICE

Fonte: Autor (2025).

2.3 NETLIST

Para que o motor SPICE interprete a topologia de um circuito, sua descrição deve ser fornecida em formato textual por meio de uma *netlist*. A *netlist* é um arquivo de texto no qual cada linha corresponde a um elemento do circuito ou a uma diretiva de simulação; é, portanto, a representação canônica do circuito utilizada pelo computador para montar e resolver as equações elétricas que regem o sistema (ver Fig. 7).

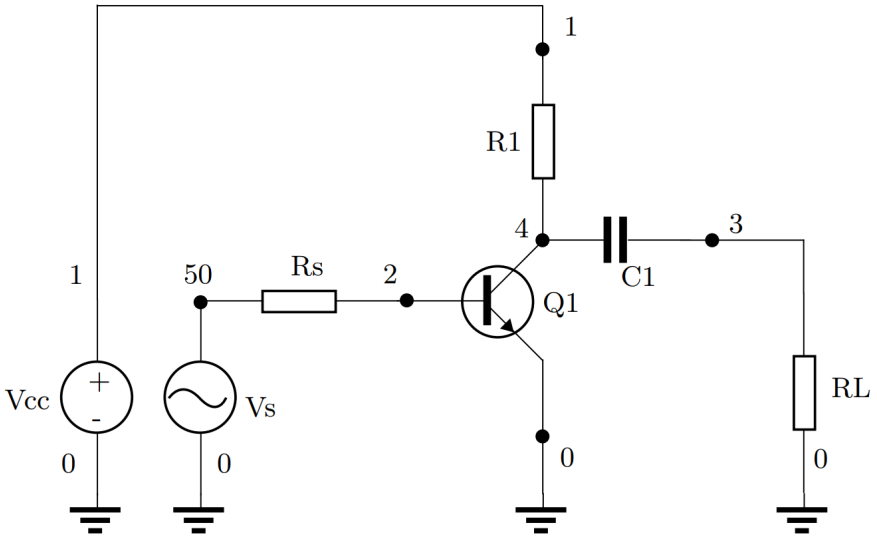


Figura 7: Circuito de exemplo para a netlist

Fonte: Muttoni, Leonardo e outros [10].

```
CIRCUITO DE TESTE
Vcc 1 0 dc 5
Vs 50 0 0.0 ac 1 sin(0 2 10k)
Rs 50 2 100
RL 3 0 2k
Q1 4 2 0 2N3904
R1 1 4 2.0e3
C1 4 3 2.0e-9
.include 2N3904.sp3
.tran 100ns 200us
.save v(3)
```

A linha inicial de uma *netlist* costuma conter apenas o título do projeto, que serve como identificação nas saídas da simulação; alternativamente, o título pode ser definido explicitamente pela diretiva *.title título*. Comentários são permitidos e recomendados para documentar trechos do arquivo: uma linha inteiramente comentada começa com *****, enquanto comentários ao final de uma linha podem ser introduzidos pelos caracteres ***;*** ou ***$***.

Para reutilização e modularidade, a *netlist* admite a inclusão de arquivos externos. A diretiva .include arquivo injeta o conteúdo de outro arquivo tipicamente modelos de componentes ou sub-circuitos ampliando a legibilidade e evitando repetição de parâmetros. Do mesmo modo, a extração de resultados é controlada por comandos de saída; por exemplo, *.save* v_1 v_2 ... v_n indica quais vetores (tensões, correntes, etc.) devem ser gravados no arquivo de saída ao término da simulação.

Cada elemento do circuito é declarado em uma linha distinta da *netlist*. Nessa linha constam, de forma ordenada, o identificador da instância (cuja primeira letra indica o tipo do componente), os nós de conexão e os parâmetros que definem seu comportamento elétrico (conforme a sintaxe resumida na Tabela 1).

Tabela 1: Sintaxe de Elementos do SPICE

Elemento	**Sintaxe**
Fonte de Tensão DC	`VXXX N+ N- <DC> DC/TRAN VALUE`
Fonte de Tensão AC	`VXXX N+ N- <AC <ACMAG <ACPHASE>>>`
Transistor BJT	`QXXX nc nb ne model_name`
Resistor	`RXXX n+ n- valor`
Capacitor	`CXXX n+ n- valor`
Indutor	`LXXX n+ n- valor`

Fonte: Autor (2025), com informações obtidas de VOGT, Holger et al. [11].

O nó de referência é tradicionalmente o nó *0* (terra), enquanto os demais nós são numerados por inteiros positivos. Quando um componente depende de um modelo de maior

complexidade (por exemplo, transistores e diodos de potência), a instância referencia o nome do modelo, cujo conjunto de parâmetros pode estar declarado no próprio arquivo por *.model* ou em um arquivo externo carregado via *.include*.

Componentes semicondutores reais costumam demandar uma descrição paramétrica muito extensa, não é incomum que um modelo possua dezenas ou até cerca de cem parâmetros[1], o que tornaria a *netlist* pouco legível caso cada instância exigisse repetição integral desses valores. Para contornar essa complexidade, o SPICE permite declarar modelos reutilizáveis: um modelo é definido uma única vez com a diretiva *.model*, recebendo um nome e o tipo (por exemplo, NPN, PNP, MOSFET), seguido da lista de parâmetros no formato parâmetro=valor, separados por espaços. Um exemplo típico é:

```
.model BC548 NPN BF=200 IS=1e-10 VBF=50
```

Em projetos práticos, os modelos completos fornecidos pelos fabricantes são muito mais extensos e, por isso, costumam ser mantidos em arquivos distintos. A inclusão desses arquivos na *netlist* é feita com a diretiva .include arquivo, que insere o conteúdo do arquivo externo no ponto da *netlist* em que for declarado, simplificando tanto a manutenção quanto a portabilidade dos projetos.

Após a definição da topologia e o carregamento dos modelos, o próximo passo é especificar qual tipo de análise o simulador deverá executar. As diretivas de análise instruem o motor SPICE sobre o método numérico a empregar e os parâmetros da simulação; a Tabela 2 resume a sintaxe das três análises mais comuns e fornece exemplos de uso. Em linhas gerais:

- A análise transitória (`.tran`) resolve o comportamento no domínio do tempo e é parametrizada por `tstep` (passo de amostragem) e `tstop` (tempo final da simulação), sendo indicada para estudar respostas à comutação e estabelecimento de regime permanente.
- A análise AC (`.ac`) calcula a resposta em frequência (pequeno sinal) sobre um intervalo definido por `fstart` e `fstop`, com `nd` determinando o número de pontos por década.
- A análise DC (`.dc`) realiza varreduras paramétricas: o identificador `srcnam` indica a fonte ou parâmetro a ser varrido (por exemplo, uma fonte de tensão ou temperatura) e os argumentos `vstart`, `vstop` e `vincr` definem, respectivamente, valor inicial, final e passo de varredura.

Tabela 2: Sintaxe dos Comandos de Análise SPICE

Análise	Sintaxe	Exemplo
Análise Transitória	`.tran tstep tstop`	`.tran 150ns 200us`
Análise C.A.	`.ac dec nd fstart fstop`	`.ac dec 10 10 182k`
Análise C.C.	`.dc srcnam vstart vstop vincr`	`.dc vs v 0 1.5 0.01`

Fonte: Autor (2025), com informações obtidas de VOGT, Holger et al. [11].

Dessa forma, o uso combinado de arquivos de modelo (*.include* / *.model*) e diretivas

de análise (*.tran, .ac, .dc*) torna a *netlist* ao mesmo tempo modular, legível e suficientemente expressiva para controlar simulações realistas e reprodutíveis no LTspice.

2.3.1 Topologias

Nos esquemáticos de circuitos eletrônicos, a identificação da topologia de um conversor é possível a partir do arranjo específico de seus componentes e das conexões estabelecidas entre eles. Essa organização estrutural permite antecipar o comportamento elétrico do circuito, incluindo as variações de corrente e tensão em cada elemento. O reconhecimento correto da topologia é fundamental tanto para a classificação do conversor quanto para o desenvolvimento de projetos, pois, uma vez determinado o tipo de conversor, podem-se aplicar diretamente os modelos analíticos e as equações correspondentes para obter os valores desejados, como a tensão de saída e geração da *netlist* correta.

2.3.1.1 Buck

O conversor Buck é empregado quando se deseja obter uma tensão CC de saída inferior à tensão CC de entrada. Sua operação baseia-se no chaveamento controlado de um interruptor eletrônico combinado a um filtro LC, responsável por suavizar a forma de onda pulsada produzida pelo chaveamento, conforme ilustrado na Figura 8(a). O diodo do circuito desempenha o papel de caminho alternativo para a corrente do indutor durante o intervalo em que a chave está desligada, garantindo a continuidade da corrente, como mostrado na Figura 8(b). Quando a chave é ligada, o diodo se polariza reversamente e deixa de conduzir, situação representada na Figura 8(c).

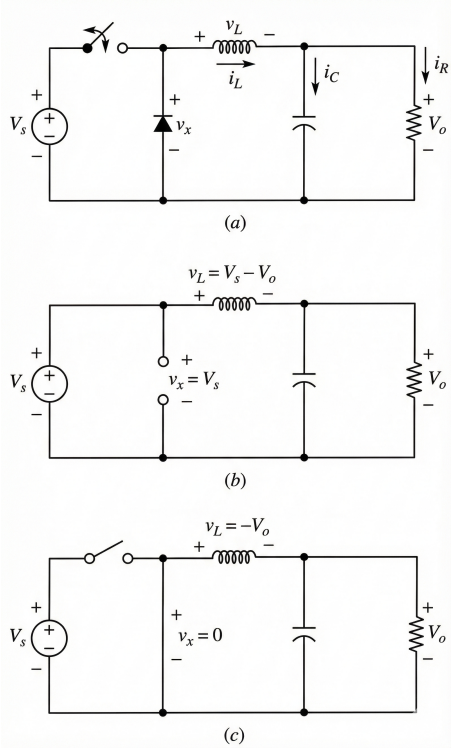


Figura 8: (a) conversor buck CC-CC; (b) circuito equivalente chave fechada; (c) circuito equivalente chave aberta

Fonte: Hart. W Daniel [9].

No regime permanente e em condução contínua (CCM), o conversor Buck apresenta a relação direta entre a tensão de saída V_{out} e de entrada V_{in} dada pelo ciclo de trabalho D:

$$V_{\text{out}} = D \cdot V_{\text{in}} \tag{5}$$

onde

$$D = \frac{t_{\text{on}}}{T} \tag{6}$$

representa a fração do período T em que a chave permanece ligada. A ondulação de corrente no indutor apresenta variação triangular dada por:

$$\Delta I_L = \frac{V_{\text{in}} - V_{\text{out}}}{L} \cdot DT \implies \frac{(V_{\text{in}} - V_{\text{out}})D}{L f_s} \tag{7}$$

e a ondulação de tensão na saída pode ser aproximada por:

$$\Delta V_o \approx \frac{\Delta I_L}{8 C f_s} \tag{8}$$

com f_s = 1/T sendo a frequência de chaveamento.

Para garantir operação em CCM, o indutor deve ser suficientemente grande para que

a corrente não atinja zero durante o período de comutação, assim o indutor mínimo deve satisfazer:

$$L_{\text{min,Buck}} = \frac{(1-D)R}{2f_s} \tag{9}$$

sendo R a resistência da carga. Essa expressão resulta da condição de limite entre CCM e modo de condução descontínua (DCM) ao igualar a variação pico-a-pico da corrente ao dobro da corrente média do indutor.

Essas relações permitem determinar a tensão de saída, a ondulação de corrente e os valores necessários de L e C para garantir a operação adequada, tornando o conversor Buck uma solução eficiente e amplamente utilizada em aplicações que requerem redução estável da tensão contínua.

2.3.1.2 Boost

O conversor Boost, ilustrado na Figura 9, é um conversor CC–CC cuja função é elevar a tensão CC de entrada, fornecendo uma tensão de saída superior. Seu funcionamento baseia-se na comutação periódica de um interruptor eletrônico , que controla o modo como o indutor armazena e transfere energia para a carga.

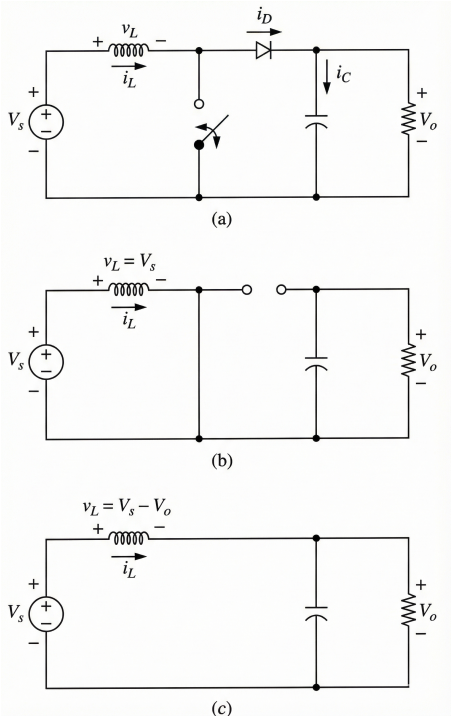


Figura 9: (a) Conversor Boost CC-CC; (b) circuito equivalente chave fechada; (c) circuito equivalente chave aberta

Fonte: Hart. W Daniel [9].

Durante o intervalo em que a chave está ligada, o indutor é conectado diretamente à

fonte de entrada e armazena energia, enquanto o diodo permanece polarizado reversamente, isolando a carga. Quando a chave é desligada, o indutor libera sua energia somando-se à fonte de entrada, polarizando o diodo diretamente e elevando a tensão aplicada ao capacitor e à carga.

No regime permanente e em condução contínua (CCM) a relação entre a tensão de saída V_{out} e a tensão de entrada V_{in} é dada por:

$$V_{\text{out}} = \frac{V_{\text{in}}}{1 - D} \tag{10}$$

onde o ciclo de trabalho D é:

$$D = \frac{t_{\text{on}}}{T} \tag{11}$$

sendo t_{on} o tempo em que a chave permanece ligada e T o período de comutação. Observe que, à medida que D aumenta, o denominador (1-D) diminui, elevando a tensão de saída, sendo a característica típica do conversor Boost.

A variação de corrente no indutor é descrita por:

$$\Delta I_L = \frac{V_{\text{in}}}{L} \cdot DT \tag{12}$$

e a ondulação de tensão na saída pode ser aproximada por

$$\Delta V_{\text{out}} \approx \frac{I_o D}{C f_s} \tag{13}$$

onde f_S = 1/T é a frequência de chaveamento e I_o é a corrente média de carga. Para operação em CCM, o indutor mínimo é dado por

$$L_{\text{min,Boost}} = \frac{DR}{2f_s} \cdot (1 - D)^2 \tag{14}$$

Assim, pode-se definir o comportamento dinâmico do conversor Boost e orientar a escolha dos parâmetros L e C para garantir uma operação estável, baixa ondulação e a elevação de tensão desejada.

2.3.1.3 Buck-Boost

O conversor Buck-Boost, ilustrado na Figura 10, combina os princípios de operação dos conversores Buck e Boost, permitindo regular a tensão de saída para valores maiores ou menores que a tensão de entrada. Essa flexibilidade é alcançada através do chaveamento controlado de um interruptor eletrônico e do uso de um indutor como elemento de armazenamento de energia e transferência indireta de energia entre entrada e saída.

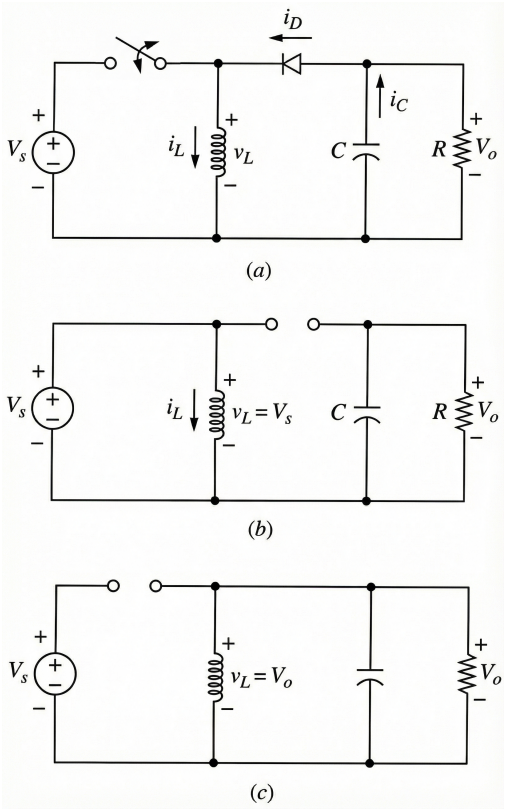


Figura 10: (a) Conversor Buck-Boost; (b) circuito equivalente chave fechada; (c) circuito equivalente chave aberta

Fonte: Hart. W Daniel [9].

Durante o intervalo em que a chave está conduzindo, o indutor fica diretamente conectado à fonte e armazena energia magnética, enquanto o diodo permanece reversamente polarizado, impedindo a transferência de energia para a carga. Quando a chave é desligada, o indutor libera a energia acumulada, polarizando o diodo em condução e transferindo energia para o capacitor e para a carga. Conforme a topologia adotada, essa comutação pode alterar a polaridade da tensão de saída: em topologias inversoras (por exemplo, buck–boost inversor) a tensão de saída é invertida em relação à entrada, enquanto em topologias não inversoras ou em variantes modificadas a polaridade de saída pode coincidir com a da fonte.

No regime permanente e em CCM, a relação entre a tensão de saída V_{out} e a tensão de entrada V_{in} é expressa por:

$$V_{\text{out}} = -V_{\text{in}} \left(\frac{D}{1 - D} \right) \tag{15}$$

com o ciclo de trabalho D definido igual aos conversores anteriores.

O sinal negativo indica a inversão de polaridade característica do conversor Buck-Boost clássico. A expressão mostra que:

- para $D < 0,5$, $|V_{\text{out}}| < |V_{\text{in}}| \rightarrow$ modo buck;
- para $D > 0,5$, $|V_{\text{out}}| > |V_{\text{in}}| \rightarrow$ modo boost;

Para garantir operação em CCM, o indutor deve ser suficientemente grande para que a corrente não atinja zero durante o período de comutação. O valor mínimo é dado por:

$$L_{\min} = \frac{(1-D)^2 R}{2 f_s} \tag{16}$$

Valores de indutância menores que L_{min} colocam o conversor em modo de condução descontínua (DCM), modificando as equações de ganho estático e diminuindo a eficiência.

Portanto, o conversor Buck-Boost destaca-se pela capacidade de fornecer uma tensão de saída maior ou menor que a entrada, com controle direto via ciclo de trabalho. Seu desempenho adequado depende do correto dimensionamento do indutor mínimo, que garante operação em condução contínua, e do capacitor de saída, responsável por limitar a ondulação de tensão.

Além das diferenças nos modos de armazenamento de energia, as topologias Buck, Boost e Buck-Boost apresentam arranjos distintos de ligação entre seus componentes, definindo o comportamento elétrico de cada conversor. Compreender essas configurações estruturais é fundamental para o treinamento do YOLO e da GNN, pois permite reconhecer os padrões visuais e as relações entre componentes que caracterizam cada topologia. Esse entendimento também é essencial para validar a *netlist* gerada, garantindo que as conexões identificadas pelo sistema correspondam ao funcionamento real do conversor.

3 INTELIGÊNCIA ARTIFICIAL

A inteligência artificial (IA) refere-se ao desenvolvimento de sistemas capazes de executar tarefas associadas ao raciocínio humano, como interpretar informações, aprender com dados, tomar decisões e resolver problemas. Na definição de McCarthy (2007), IA é a "ciência e engenharia de produzir sistemas inteligentes", enquanto Bellman (1978) a descreve como a automação de atividades típicas do pensamento humano [12, 13]. Esses sistemas dependem de grandes volumes de dados para ajustar seu comportamento e, assim, operar segundo uma lógica de raciocínio artificial alinhada às capacidades humanas de análise e adaptação.

Dentro desse campo, a visão computacional representa uma área dedicada a permitir que máquinas interpretem e compreendam informações visuais. Fundamentada principalmente em redes neurais e aprendizado profundo, ela possibilita que sistemas identifiquem objetos, extraiam características e tomem decisões a partir de imagens e vídeos já processados na fase de treinamento [14]. Por esse motivo, é considerada um pilar do avanço em detecção e reconhecimento de padrões visuais, desempenhando papel central nas aplicações modernas de IA [15].

3.1 REDES NEURAIS

Um neurônio artificial é a unidade fundamental das redes neurais e foi inspirado, de forma simplificada, no funcionamento dos neurônios biológicos [16]. Ele recebe um conjunto de entradas numéricas, pondera cada uma delas por um peso, soma esses valores juntamente com um viés e aplica uma função de ativação, que determina a saída final. Em termos matemáticos, o neurônio implementa a operação:

$$y = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{17}$$

onde x_i são as entradas, w_i os pesos, b é o viés e ϕ a função de ativação. Na figura 11 observa-se a estrutura de um neurônio artificial:

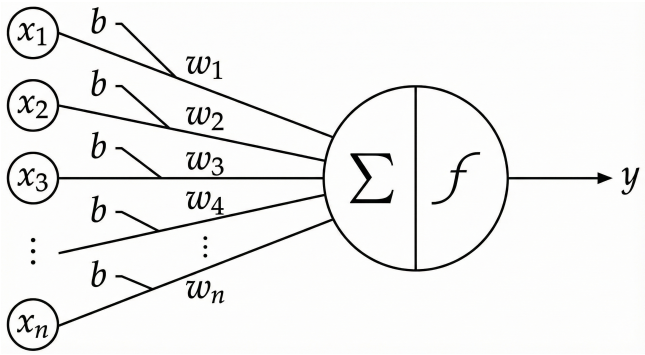


Figura 11: Representação de um neurônio artificial

Fonte: Autor(2025).

Segundo Haykin (2009), essa estrutura permite que o neurônio aprenda padrões a partir de dados, ajustando seus pesos por algoritmos de treinamento para aproximar relações complexas entre entradas e saídas [17]. Durante o treinamento, os pesos das conexões são ajustados para controlar quanto cada entrada influencia a ativação dos neurônios seguintes. Após a soma ponderada e o acréscimo do viés, a função de ativação como sigmoid, ReLU, ELU e etc. determina a intensidade da resposta do neurônio e introduz a não-linearidade necessária para que a rede aprenda padrões complexos. A escolha dessa função afeta diretamente a propagação do gradiente, a velocidade de convergência e a capacidade de generalização do modelo [18, 19]. A figura 12 representa duas redes neurais de exemplo:

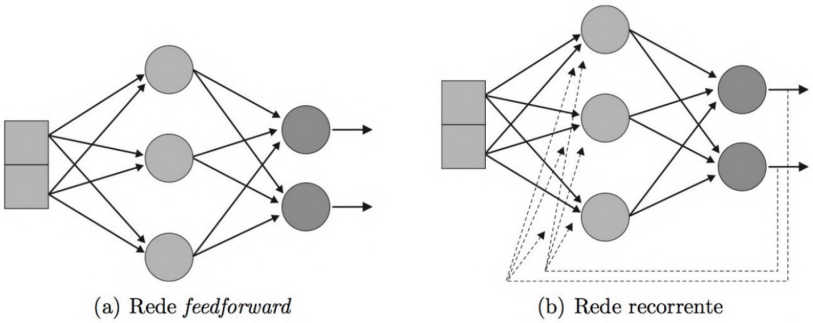


Figura 12: Representação de Redes Neurais feedforward e recorrente

Fonte: FACELI et al., 2021 [20].

3.2 REDES NEURAIS CONVOLUCIONAIS

As redes neurais convolucionais (CNNs) têm se consolidado como uma das soluções mais eficientes para diversas tarefas de visão computacional, incluindo detecção e reconhecimento de objetos, classificação de imagens e restauração de conteúdo visual. Sua eficácia deriva da capacidade de extrair características relevantes por meio de operações convolucionais aplicadas de forma hierárquica, refletindo princípios inspirados na organização do córtex visual humano.

No contexto de aprendizado de máquinas, as CNNs constituem um tipo de rede neural profunda (DNN) especializada em dados bidimensionais. A Figura 13 apresenta a arquitetura *CaffeNet*, uma variação da AlexNet, composta por cinco camadas convolucionais (CLs), duas camadas de max-pooling (PLs) e três camadas totalmente conectadas (FLs) [21]. Essa organização ilustra de forma compacta como CNNs combinam extração de características e mapeamento final para realizar tarefas visuais complexas.

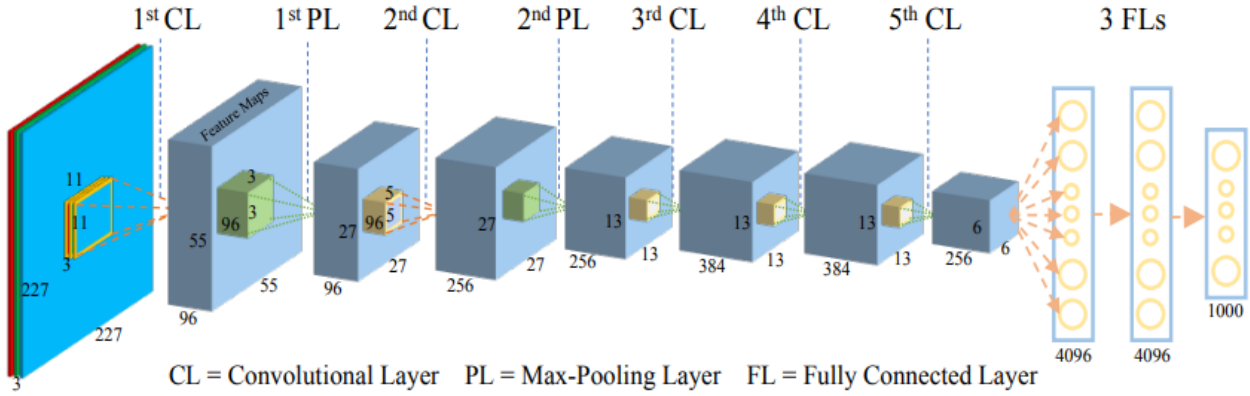


Figura 13: Arquitetura CaffeNet

Fonte: Qin et al. (2018) [22].

3.2.1 Camada Convolucional

Na Figura 13, os blocos amarelos representam os filtros convolucionais, que funcionam como neurônios responsáveis por extrair padrões específicos da imagem. Durante a operação de convolução, esses filtros percorrem a entrada, seja a própria imagem ou mapas de características de camadas anteriores gerando novos *feature maps*, representados pelos blocos azuis. Esse processo permite que a rede aprenda bordas, texturas e estruturas cada vez mais complexas ao longo das camadas.

Na Figura 14(a) é detalhado o funcionamento da primeira camada convolucional. Nessa etapa, cada filtro possui três canais, correspondentes aos componentes RGB da imagem de entrada. Cada canal realiza uma operação de produto ponto sobre uma pequena região da entrada, e os resultados são somados para compor o respectivo *feature channel.*

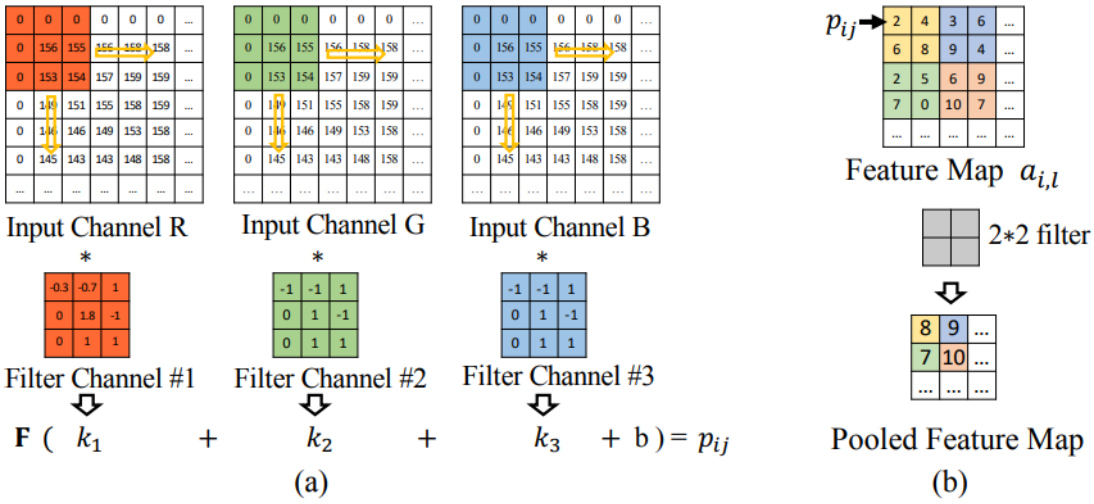


Figura 14: Os processos convolucionais e max-pooling.

Fonte: Qin et al. (2018) [22].

Esse processo permite que a rede extraia características iniciais associadas às combinações de cores e padrões básicos da imagem. Esse processo vem normalmente acompanhado de uma função de ativação, usualmente a ReLU, no qual retorna uma somatória do produto ponto quando positiva e zero caso contrário. Matematicamente, a saída de um filtro i na camada l pode ser expressa por:

$$a_{i,l+1} = F\left(\sum w_{i,l} * a_{i,l} + b_{i,l}\right) \tag{18}$$

onde $w_{i,j}$, $b_{i,j}$ representa os pesos e o viés associados ao filtro, e * representa a operação de convolução. Nesse processo, cada filtro aprende a responder a padrões específicos da entrada, atuando como um extrator de características essenciais para as etapas posteriores de classificação.

3.2.2 Camada Max-Pooling

Na prática, a camada de polimento (*pooling*) é inserida após blocos convolucionais para reduzir a dimensão espacial dos *feature maps* e tornar as representações mais invariantes a pequenas translações. Representadas em verde na Figura 13, as janelas de polimento aplicam uma operação de amostragem local sobre $a_{i,j}$. No *max-pooling* (Figura 14(b)), uma janela 2x2 desloca-se com passo (*stride*) 2 e escolhe o maior valor de cada região, preservando assim as ativações mais salientes. Como consequência, a rede opera com mapas menores e menos redundâncias, o que reduz a complexidade computacional e ajuda a prevenir o sobreajuste (*overfitting*) nas camadas seguintes.

3.2.3 Camada totalmente Conectada

Observando a Figura 13, os círculos amarelos representam os neurônios das camadas totalmente conectadas (FCs), as quais recebem como entrada todas as ativações produzidas pelas camadas convolucionais e de polimento. Essas camadas realizam a integração final das características extraídas, produzindo um vetor de probabilidades de dimensão N, onde N corresponde ao número de classes do problema. Ao final da última camada FC, aplica-se a função Softmax, responsável por transformar as ativações a_i em probabilidades normalizadas:

$$P_i = \frac{e^{a_i}}{\sum_{n=1}^{N} e^{a_n}} \tag{19}$$

Esse vetor P representa a distribuição de probabilidade sobre todas as classes, sendo que o maior valor P_i determina a classe prevista pela rede. Dessa forma, as FCs desempenham o papel de consolidar as informações extraídas pelas camadas anteriores e produzir a decisão final do modelo.

3.3 REDES NEURAIS GRÁFICAS

Para compreender redes neurais gráficas (GNNs), é essencial primeiro definir o conceito de grafo. Um grafo $G = (V, E)$ é formado por um conjunto de nós V e pelas conexões entre eles E. Esse modelo é adequado para qualquer tipo de dado em que as relações entre elementos sejam tão importantes quanto os próprios elementos [23].

As GNNs são redes projetadas para operar diretamente sobre essa estrutura, utilizando a matriz de adjacências A para descrever as conexões e a matriz de atributos X para representar as características de cada nó, como mostrado na figura 15. A aprendizagem ocorre por meio de mecanismos de agregação de vizinhança, que propagam informações entre nós conectados.

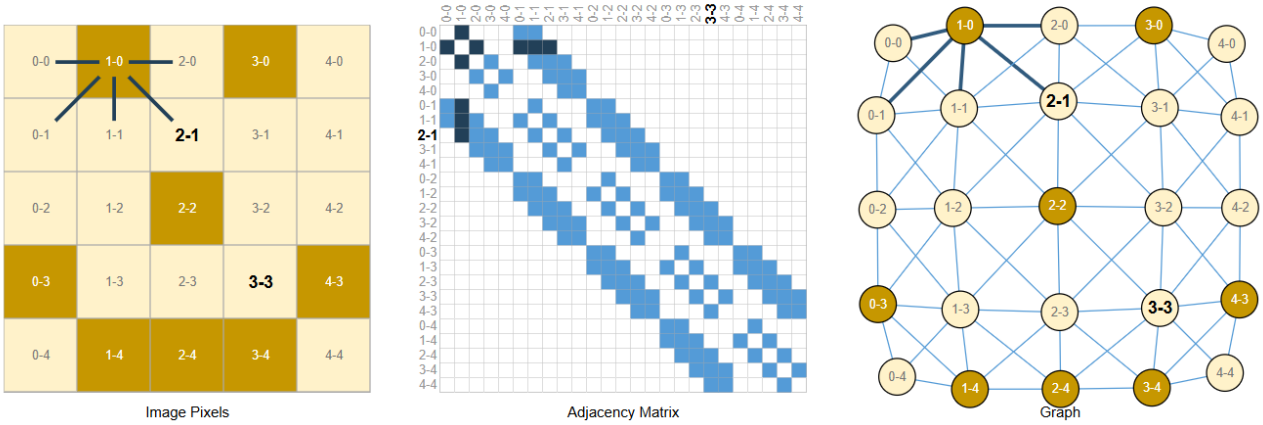


Figura 15: Representação gráfica da imagem em pixels

Fonte: SANCHEZ-LENGELING, Benjamin et al [23].

Embora grafos sejam comumente associados a redes sociais, muitos outros dados podem ser interpretados dessa forma. Uma imagem, por exemplo, pode ser vista não apenas como uma grade $H \times W \times C$, mas também como um grafo regular em que cada pixel é um nó conectado e seus vizinhos (4 ou 8 conexões).

Essa interpretação permite estender o uso das GNNs para domínios estruturados e não estruturados, facilitando a captura das relações espaciais e topológicas necessárias para tarefas como classificação, detecção e análise de estruturas mais complexas, incluindo circuitos elétricos.

3.4 YOU ONLY LOOK ONCE

You Only Look Once conhecida como YOLO é um detector baseado em redes neurais convolucionais (CNNs) que reformula a detecção como um problema de regressão: uma única CNN processa a imagem inteira e, em uma única avaliação prediz coordenadas de caixas delimitadoras (*bounding boxes*) e probabilidades de classe. Essa unificação (*pipeline end-to-end*) contrasta com métodos em duas etapas e permite otimização direta para desempenho de detecção.

Na Figura 16 demonstra os processos da arquitetura da YOLO, a rede divide a imagem em uma grade $S \times S$ no qual cada célula prediz B caixas $(x, y, w, h, confidence)$ e C probabilidades condicionais de classe, codificando as saídas como um tensor $S \times S \times (B \cdot 5 + C)$. As camadas convolucionais extraem características globais enquanto as camadas finais totalmente conectadas realizam a regressão das coordenadas e das probabilidades. Essa escolha favorece o raciocínio contextual sobre a cena inteira e especialização dos preditores por tamanho/forma de objeto.

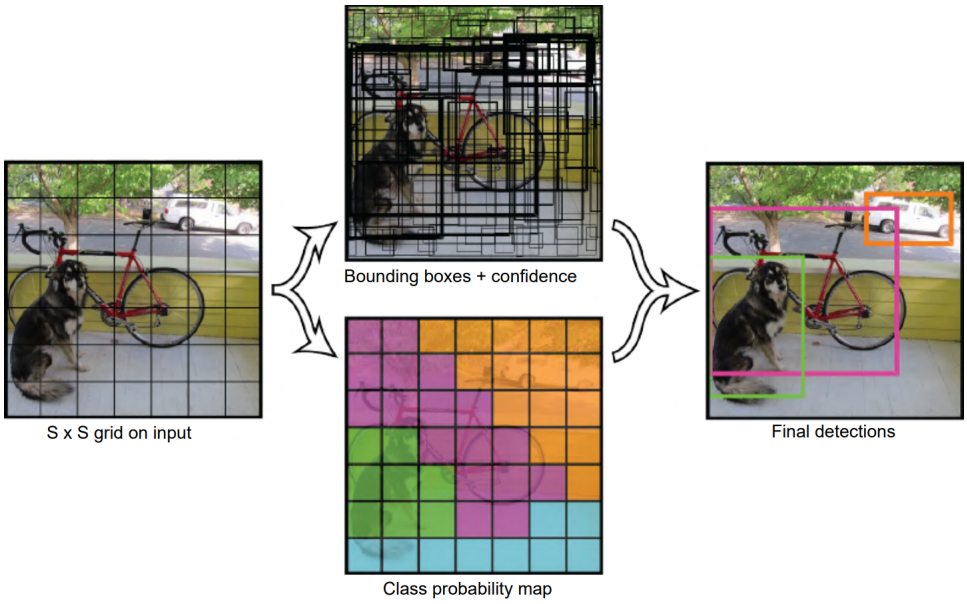


Figura 16: Exemplo do funcionamento YOLO

Fonte: REDMON et al., 2016 [24].

Do ponto de vista de treinamento e inferência, YOLO pré-treina as camadas convolucionais (24 camadas convolucionais seguidas por 2 camadas totalmente conectadas, podendo ser visto na arquitetura da figura 17), adapta a resolução para detecção e usa perdas ponderadas para coordenadas, confiança e classe (com termos λ_{coord} e λ_{noobj}). Em teste, a inferência demanda apenas uma passagem pela rede, resultando em alta taxa de frames por segundo e latência baixa, sendo uma vantagem crítica para aplicações em tempo real.

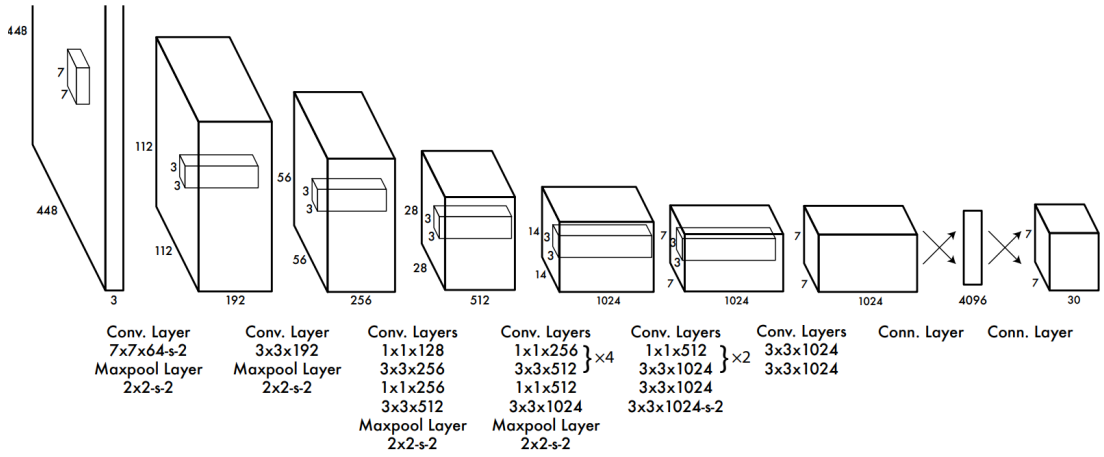


Figura 17: A arquitetura da YOLO

Fonte: REDMON et al., 2016 [24].

Das limitações e *trade-offs*, a arquitetura impõe restrições espaciais (cada célula prevê poucas caixas e uma classe), o que reduz desempenho em objetos muito pequenos ou em grupos densos, o maior erro observado são localizações imprecisas. Essas características orientam escolhas práticas (resolução de entrada maior, *heads* multi-escala, atenção ou módulos como GAM e pós-processamento com NMS) ao adaptar YOLO para problemas específicos, como detecção de componentes em esquemáticos.

3.5 GAM-YOLO

A detecção automática de componentes em esquemáticos elétricos exige modelos capazes de distinguir padrões extremamente sutis, frequentemente composto por traços finos, símbolos pequenos e regiões com forte sobreposição gráfica. O YOLO11 por ser uma versão mais recente da família YOLO, oferece um backbone mais eficiente com módulos de atenção internos (como o C2PSA) e um neck aprimorado, o que resulta em extração de características mais rica e com menor custo computacional. Nesse contexto, a inclusão do *Global Attention Módulo* (GAM) entre o *neck* e o *head* torna-se particularmente relevante, pois reforça ainda mais seletivamente regiões espaciais e canais de informação antes da predição das caixas delimitadoras.

A Figura 18 apresenta a estrutura geral do YOLO11 modificado com o GAM (*backbone* → *neck* → *head*) após a fusão multiescala do *neck* (mapas P1, P2, P3), cujo cada mapa é processado pelo GAM antes de entrar no *decoupled head* do YOLO11. Essa posição é estratégica, pois o *neck* fornece representações já ricas, e o GAM refina essas representações ajustando quais canais (tipos de características) e quais posições espaciais devem ter ganho aumentado ou suprimido.

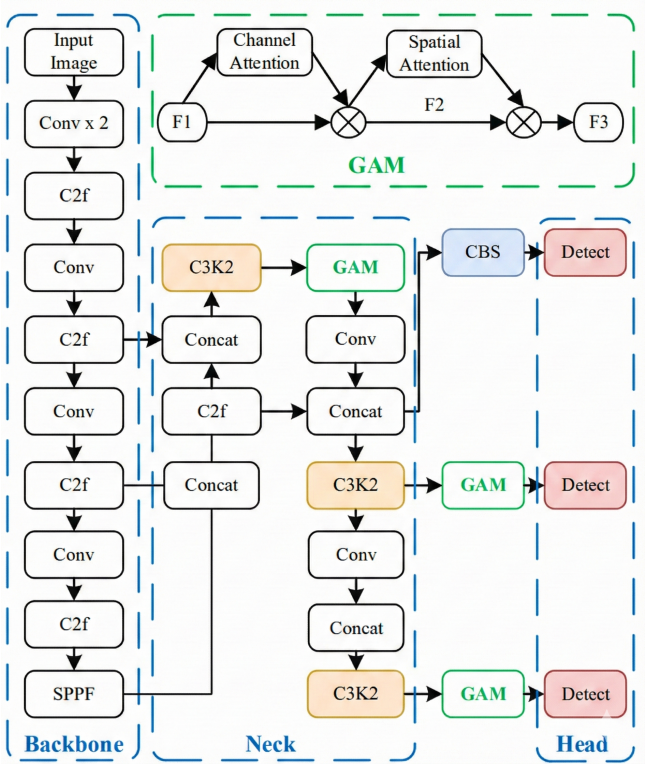


Figura 18: Ilustração da arquitetura GAM-YOLO

Fonte: Autor (2025), adaptada de Adaptado de Gao et al., 2024 [6].

Para transformar as detecções em uma estrutura interpretável eletricamente, as *bounding boxes* e classes são convertidas em uma *netlist* e, em seguida, em um grafo. A Figura 19 mostra esse processo: cada componente detectado (nó) e cada fio/união (aresta) são extraídos a partir das posições e da proximidade geométrica das caixas delimitadoras. Essa representação gráfica é o elo entre a visão (YOLO11+GAM) e a análise topológica (GNN); assim, quanto mais precisas e consistentes forem as caixas delimitadoras e classes, mais fiel será a *netlist* e mais robusta a representação de grafo para classificação de topologia.

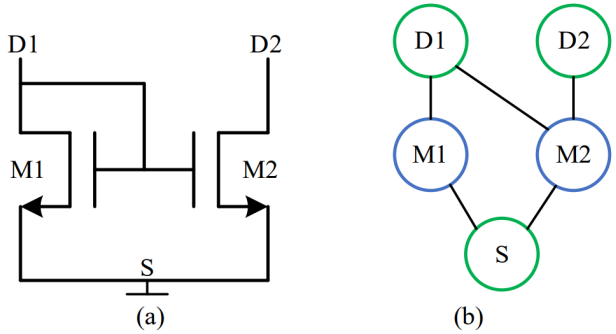


Figura 19: Transformação da representação do circuito em gráfico. (a) Esquemático de um circuito. (b) Representação do grafo do esquemático.

Fonte: Gao et al., 2024 [6].

Mas nem todos os grafos têm o mesmo custo ou fidelidade. A Tabela 3 organiza as estratégias de representação em classes (por exemplo: nós = dispositivos; nós = dispositivos + nets; grafos bipartidos dispositivo ⇔net; pinos com nós; representação alternativa com nets como nós), essas diferenças podem ser observadas na Figura 20.

Tabela 3: Classificação das Representações Gráficas

Classe	Construção de Nó	Construção de vértice
Classe 1	Componente	Net
Classe 2	Componente, net (conectado com pino I/O)	Net (sem pino I/O)
Classe 3	Componente, net	Conexões
Classe 4	Componente, Componente pino	Conexões
Classe 5	Net	Componente

Fonte: Adaptada de Gao et al., 2024 [6].

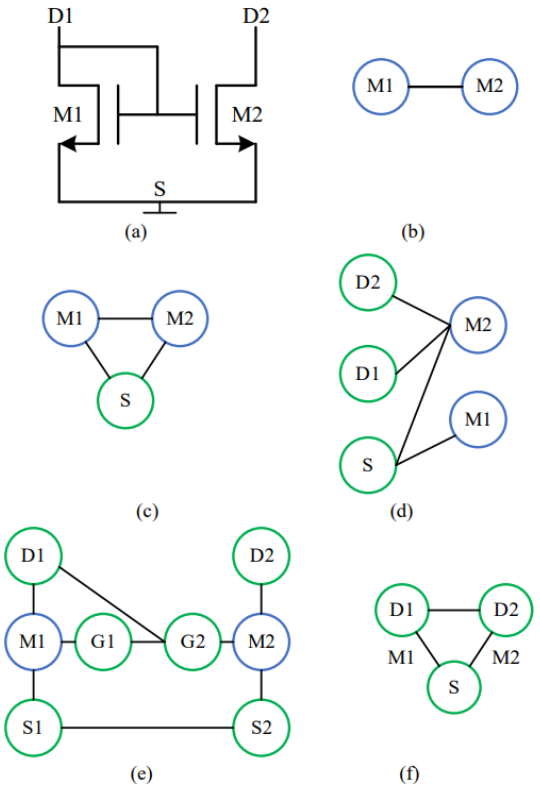


Figura 20: Diferentes classes de representação do circuito. (a) Esquemático do Circuito. (b) Representação gráfica de classe 1. (c) Representação gráfica de classe 2. (d)Representação gráfica de classe de Class 3. (e) Representação gráfica de classe 4. (f) Representação gráfica de classe 5.

Fonte: Gao et al., 2024 [6].

Modelos mais simples (classe 1) reduzem custos computacionais, mas perdem

detalhes elétricos. Para representações mais ricas (classe 3-4) preservam pinos e conexões finais, úteis para análise elétrica detalhada ao custo de maior complexidade e processamento. Essas escolhas orientam como a saída do detector será pós-processado para a GNN.

O ganho de precisão obtido pelo GAM provém de duas operações aprendidas que atuam sobre o mapa de características $F_{\text{in}} \in \mathbb{R}^{H\times W\times C}$ gerado pelo *neck*. Essas operações são:

1. Atenção de canal: Identifica quais canais contêm sinais úteis:

$$g(F)_c = \frac{1}{HW}\sum_{i,j} F(i,j,c), \quad M_c = \sigma\left(W_2\delta\left(W_1 g(F)\right)\right). \tag{20}$$

2. Atenção espacial: Localiza onde no mapa os sinais são relevantes após ajuste de canal:

$$F' = F_{\text{in}} \otimes M_c, \quad M_s = \sigma\left(\text{conv}_{k\times k}\left(\left[\text{AvgPool}_c(F'); \text{MaxPool}_c(F')\right]\right)\right). \tag{21}$$

onde M_s é uma máscara espacial que enfatiza posições com maior probabilidade de conter símbolos ou junções. Fazendo com que a reponderação final seja dada por

$$F_{\text{out}} = F_{\text{in}} \otimes M_c \otimes M_s. \tag{22}$$

A multiplicação por M_C amplifica canais que aumentam a relação sinal/ruído (SNR) para as características relevantes, enquanto canais com baixa utilidade são atenuados, reduzindo as respostas falsas. A variável M_s concentra a energia representacional em regiões de interesse, tornando as ativações mais localizadas e facilitando que o *head* aprenda regressões de caixas mais precisas.

Durante a retropropagação, os gradientes que vêm das perdas de caixas e classe (L_{box}, L_{cls}) atualizam simultaneamente W_1, W_2 e os filtros de convolução do GAM, alinhando as máscaras com os critérios de desempenho final. Isso reduz efetivamente os termos de perda associados a má localização (por exemplo, CIoU) e classificação errônea (VFL ponderado), pois a entrada ao *head* já contém menos informação ambígua.

3.6 MÉTRICAS

Para avaliar quantitativamente a qualidade de um detector é necessário extrair a partir dos dados de saída (caixas delimitadoras, classes e *scores*), medidas que reflitam tanto a capacidade de localização quanto a capacidade de classificação. Em detecção de objetos as decisões não são apenas "certo/errado", elas envolvem emparelhamento entre predições e rótulos, limiares de sobreposição e compromisso entre precisão e sensibilidade. A seguir são apresentadas as métricas principais e como elas se aplicam ao problema tratado neste trabalho.

3.6.1 Matriz Confusão, Precisão, Recall e F1

A avaliação de um modelo de detecção começa definindo como a saída da rede é comparada com os rótulos de verdade. Para isso, utiliza-se a contagem de verdadeiros positivos (TP), falsos positivos (FP) e falsos negativos (FN). Assim, uma predição é considerada.

- **TP (True Positive)**: Quando há uma correspondência correta entre uma caixa predita e um objeto real, isto é, a classe está correta e a sobreposição (IoU) entre as caixas é maior que um limiar pré-definido;
- **FP (False Positive)**: Quando a rede prediz uma caixa que não corresponde a nenhum objeto real (classe errada, IoU insuficiente ou objeto inexistente);
- **FN (False Negative)**: Quando existe um objeto real anotado na imagem, mas o modelo não produz nenhuma predição válida para ele.

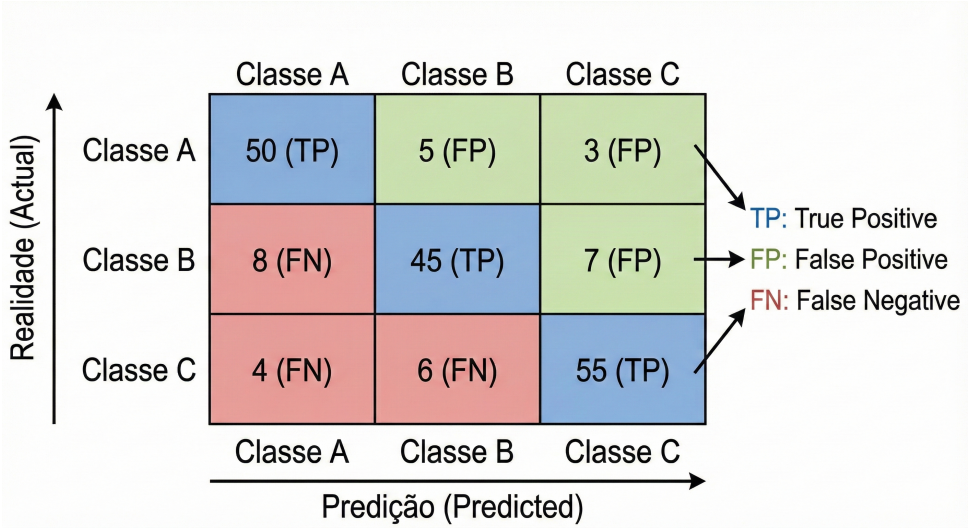


Figura 21: Matriz confusão.

Fonte: Autor (2025).

A partir desses três contadores pode-se montar uma matriz de confusão do modelo, onde a parte superior da diagonal da matriz são os FP, na parte de baixo os FN e sua diagonal é definida pelos TP como mostrado na Figura 21. Também, derivam-se as primeiras métricas fundamentais: Precisão, Recall e F1.

- A **Precisão** mede a proporção de predições corretas entre todas as predições feitas:

$$\text{Precisão} = \frac{TP}{TP + FP}. \tag{23}$$

- O **Recall** mede a proporção de objetos reais que foram de fato detectados:

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{24}$$

- A métrica **F1** combina ambas em uma única medida, como média harmônica:

$$F1 = 2 \cdot \frac{\text{Precisão} \cdot \text{Recall}}{\text{Precisão} + \text{Recall}}. \tag{25}$$

Em termos práticos, alta precisão com baixo recall indica um modelo conservador (erra pouco quando detecta, mas deixa escapar muitos objetos) ou alto recall com baixa precisão indica muitas detecções falsas.

3.6.2 Precisão Media, mAP e Interseção pela União

Em modelos modernos de detecção, como o YOLO, é comum analisar o comportamento do detector ao variar o limiar de confiança das predições. A partir dessa variação obtém-se a curva *Precision x Recall* e, sobre ela, calcula-se a Precisão Média (AP - *Average Precision*) definida como a área sob essa curva:

A métrica AP (*Average Precision*) é o valor médio da Precisão em diferentes níveis de Recall:

$$AP = \int_0^1 \text{Precisão}(\text{Recall}) d(\text{Recall}) \tag{26}$$

estimada numericamente a partir de pontos discretos. Quando há múltiplas classes, calcula-se uma AP por classe e, em seguida, a mAP (*mean Average Precision*):

$$mAP = \frac{1}{C} \sum_{i=1}^{C} AP_i \tag{27}$$

onde C é o número de classes. É prática comum reportar mAP para um limiar de sobreposição fixo (por exemplo, $mAP@0.5$) e também $mAP@0.5:0.95$ (média em vários limiares de IoU), que é mais rígida em relação à qualidade da localização. Finalmente, a noção de Interseção sobre União (IoU - *Intersection over Union*) é a central na definição de TP, FP e FN como mostrado na Figura 22. A IoU entre uma caixa predita B_p e uma caixa real B_{gt} é dada por:

$$IoU(B_p, B_{gt}) = \frac{\text{área}(B_p \cap B_{gt})}{\text{área}(B_p \cup B_{gt})}. \tag{28}$$

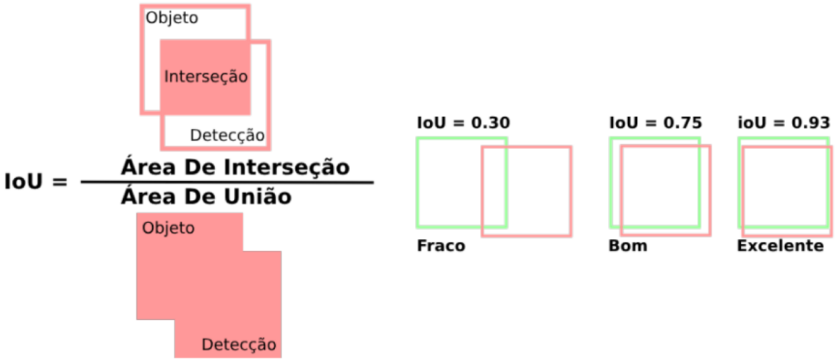


Figura 22: Representação visual da IoU

Fonte: Oliveira, Damasceno e Gondim (2024) [25].

Um par predição-rótulo é considerado TP se $IoU \geq t_{\text{IoU}}$, onde t_{IoU} é um limiar definido (tipicamente 0,5). Esse critério de correspondência é o que determina quantos TP, FP e FN serão contados e, portanto, influencia diretamente todas as métricas anteriores.

4 METODOLOGIA

Esse processo envolve cinco componentes principais: *i)* o estudo da metodologia GAM-YOLO, *ii)* a seleção da versão YOLO mais adequada à tarefa, *iii)* a construção da base de dados utilizada no treinamento dos modelos YOLO e GNN, *iv)* a implementação do script GAM-YOLO e, por fim, *v)* a validação experimental. Além desses elementos, a metodologia também abrange o procedimento de leitura automática dos valores dos componentes do circuito, realizado por meio de OCR (do inglês, *Optic Character Recognition*) utilizando a API (do inglês, *Application Programming Interface*) da empresa OpenAI, responsável por extrair e retornar o valor nominal de cada elemento detectado.

Essa sistematização visa mensurar como as decisões de projeto, especificamente no treinamento da GNN, influenciam a acurácia na detecção do circuito. Dessa forma, torna-se viável validar o modelo e refinar as etapas críticas do processo. A Figura 23 apresenta o fluxograma que detalha visualmente todas as fases desta metodologia.

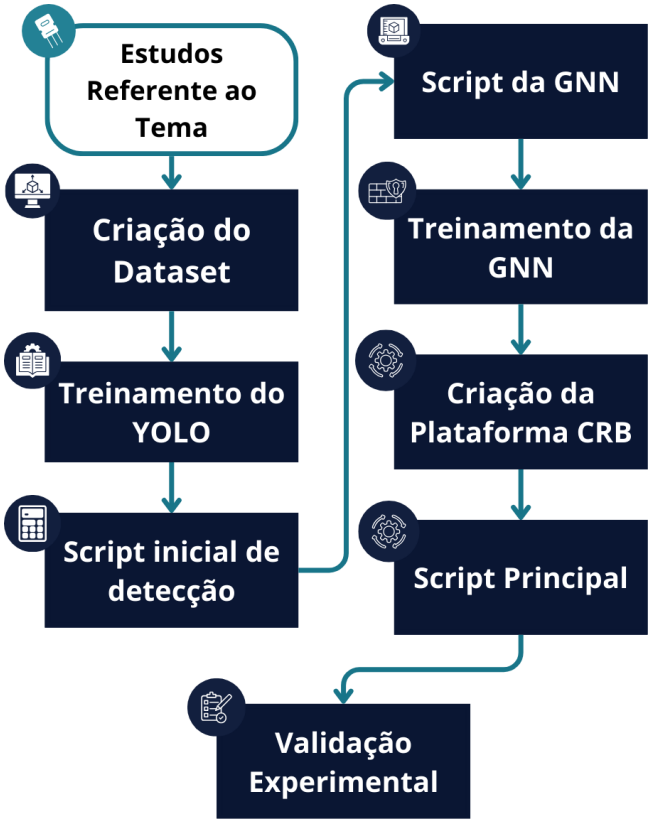


Figura 23: Fluxograma representando a metodologia de trabalho.

Fonte: Autor (2025).

4.1 ESTUDO DO GAM-YOLO

Do estudo teórico do GAM-YOLO, sua aplicação prática é apresentada na Figura 24, onde o diagrama ilustra o pipeline completo: imagens de consulta e da base passam pelo bloco de de-

tecção de componentes (implementado com YOLO11+GAM) e pelo pré-processo de reconhecimento de topologia, que transforma caixas, classes e proximidades geométricas em uma *netlist* inicial e numa representação de grafo. Em sequência, esse grafo recebe atributos (tipo de componente, coordenadas, score) e é normalizado para permitir comparações topológicas reproduzíveis entre a consulta e os esquemáticos da base.

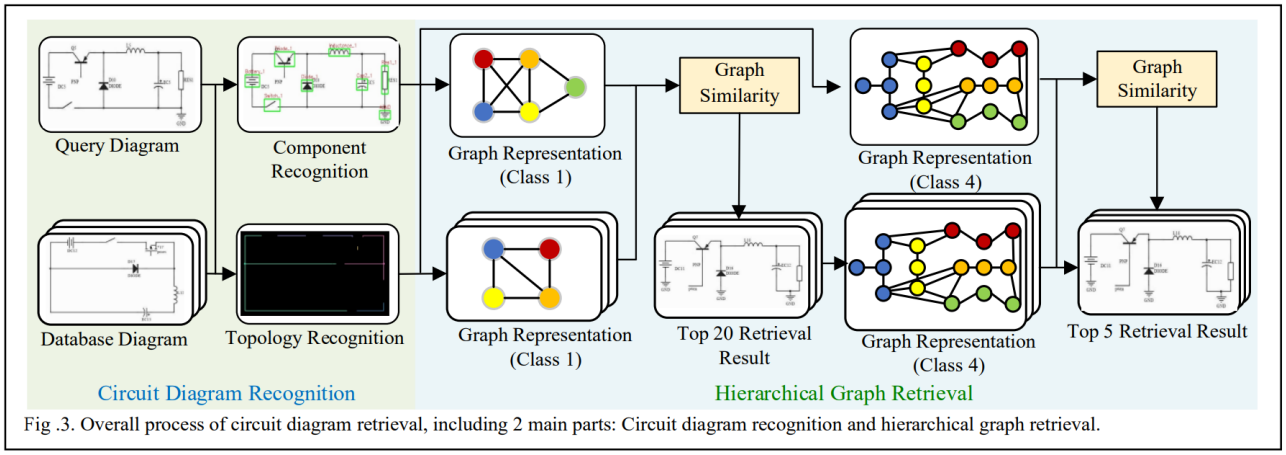


Fig .3. Overall process of circuit diagram retrieval, including 2 main parts: Circuit diagram recognition and hierarchical graph retrieval.

Figura 24: Ilustração da arquitetura GAM-YOLO.

Fonte: Gao et al., 2024 [6].

A Figura também evidencia o uso de representações hierárquicas de grafo (por exemplo, na Classe 1 os nós são os dispositivos, já na Classe 4 os nós são os pinos de conexão), adotadas para uma recuperação *coarse-to-fine*. *Coarse-to-fine* é uma estratégia em múltiplas etapas: inicialmente aplica-se uma comparação rápida e de baixo custo sobre uma representação compacta (a etapa *coarse*) para gerar um conjunto candidato (por exemplo, Top-20); em seguida, sobre esse subconjunto realiza-se uma comparação mais detalhada e dispendiosa (a etapa *fine*), por exemplo, usando *embeddings* de GNN ou *Graph Edit Distance* (GED) para selecionar o Top-5 final. Essa abordagem minimiza custos computacionais ao evitar operações de similaridade custosas em toda a base, ao mesmo tempo em que preserva a fidelidade topológica nas etapas finais.

Na prática, o acoplamento entre um detector mais seletivo (GAM) e um método de comparação baseado em grafos reduz ambiguidades na montagem da *netlist*, menos caixas mal posicionadas geram menos conexões errôneas e melhora a utilidade do sistema para etapas subsequentes de EDA. Para avaliar esse fluxo, recomenda-se reportar métricas como fração de conexões corretas na *netlist* (*netlist fidelity*) e tempo médio de recuperação, pois elas refletem tanto a eficiência da busca hierárquica quanto a qualidade da reconstrução topológica.

4.2 VERSÃO DA YOLO

A versão adotada neste trabalho foi o YOLO11, por representar a evolução mais recente da família YOLO desenvolvida pela Ultralytics. Assim como o YOLOv8, o YOLO11 oferece suporte a múltiplas tarefas como detecção de objetos, segmentação de instâncias, classificação de

imagens, estimativa de pose e *oriented bounding boxes* (OBB), porém incorpora melhorias estruturais significativas, sua estrutura pode ser vista na Figura 25. Essa versão disponibiliza cinco escalas de modelo, variando de nano a extra-large, permitindo ajustar o custo computacional ao cenário de aplicação.

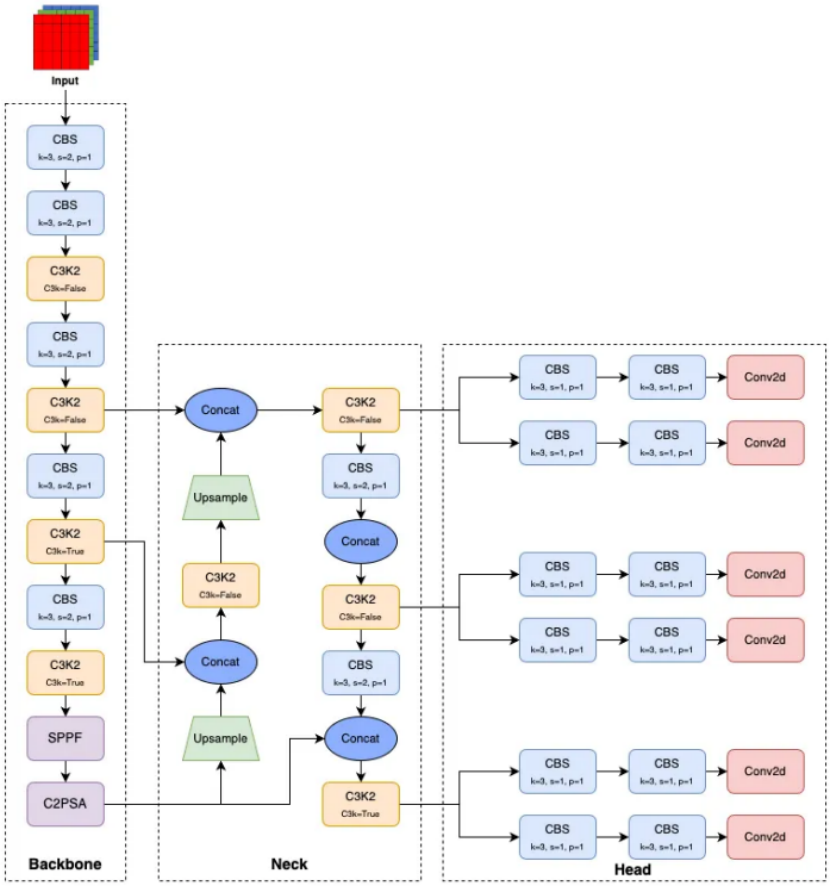


Figura 25: Arquitetura YOLO11 mostrando os novos blocos C3k2 e o módulo C2PSA.

Fonte: Jegham, et al [26].

Entre as principais inovações do YOLO11 destaca-se o módulo C2PSA (*Cross-Stage Partial with self-attention*), que combina as vantagens das conexões parciais em estágios (CSP) com mecanismo de autoatenção. Essa integração melhora a captura de informações contextuais profundas e favorece a detecção de objetos pequenos, densamente distribuídos ou parcialmente sobrepostos, características comuns em esquemáticos eletrônicos. Além disso, o tradicional bloco C2f do YOLOv8 foi substituído pelo bloco C3k2, uma implementação otimizada do CSP *bottleneck* que utiliza duas convoluções menores em vez de uma convolução ampla. Essa mudança mantém a precisão, reduz a complexidade computacional e melhora a velocidade de inferência, resultando em um modelo mais eficiente para aplicações em que precisão e desempenho precisam coexistir. De modo a ilustrar a comparação entre as versões do YOLO, a Figura 26 representa a relação da precisão mAP@50-95 em função do tempo de processamento para cada modelo.

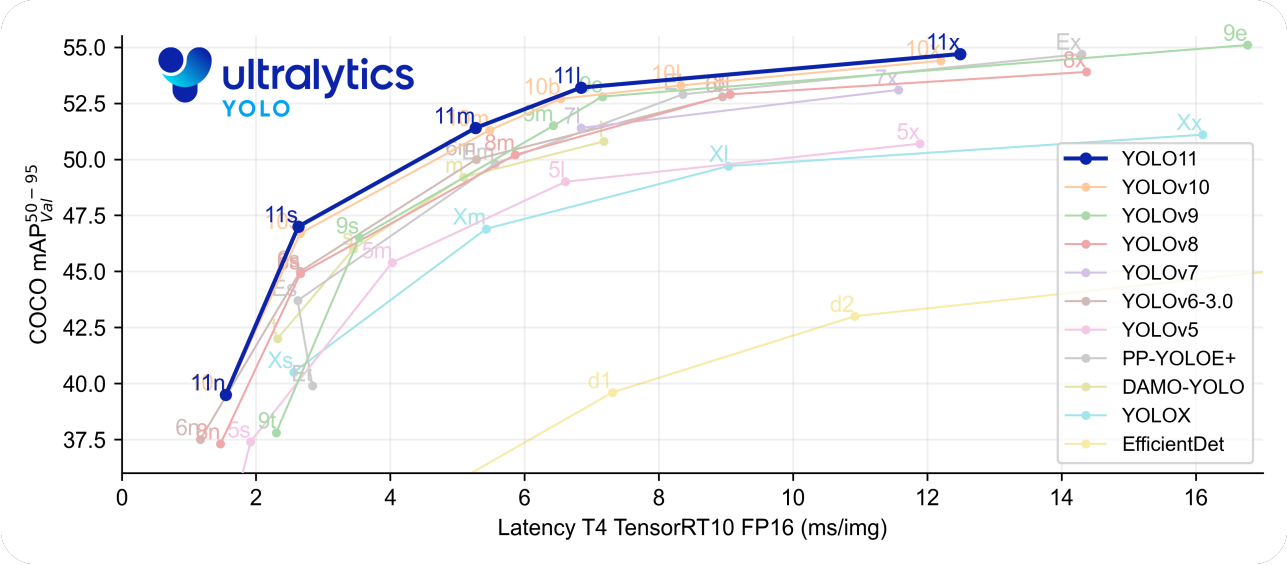


Figura 26: Métricas de desempenhos dos modelos YOLO

Fonte: Ultralytics, YOLO11.

4.3 CONSTRUÇÃO DO BANCO DE DADOS

Para a construção do banco de dados destinado ao modelo de detecção, foram coletadas imagens contendo esquemáticos dos conversores Buck, Boost e Buck–Boost. O conjunto final contém aproximadamente 133 imagens de Buck, 80 de Boost e 64 de Buck–Boost, com variações em resolução, iluminação e cor das anotações à caneta; uma amostra dessas imagens é apresentada na Figura 27.

Antes de iniciar as anotações das classes, todas as imagens passaram por um pré-processamento cujo objetivo foi reduzir ruído e remover amostras deficientes em geral causadas por compressão, desfoque ou arquivos corrompidos que poderiam degradar o treinamento. Imagens com falhas detectadas foram excluídas do conjunto; as restantes foram padronizadas em termos de formato e organizadas com o nome do *dataset* e as classes definidas para a etapa de anotação.

A rotulagem dos dados foi realizada manualmente por meio da plataforma *Roboflow*, selecionada por oferecer recursos integrados para criação, hospedagem e anotação de *datasets*. Embora a plataforma disponha de ferramentas de anotação automatizada, optou-se pelo processo manual com o objetivo de reduzir erros de identificação dos componentes eletroeletrônicos, uma vez que a elevada complexidade visual e a semelhança entre os símbolos exigem uma análise criteriosa e intervenção humana especializada.

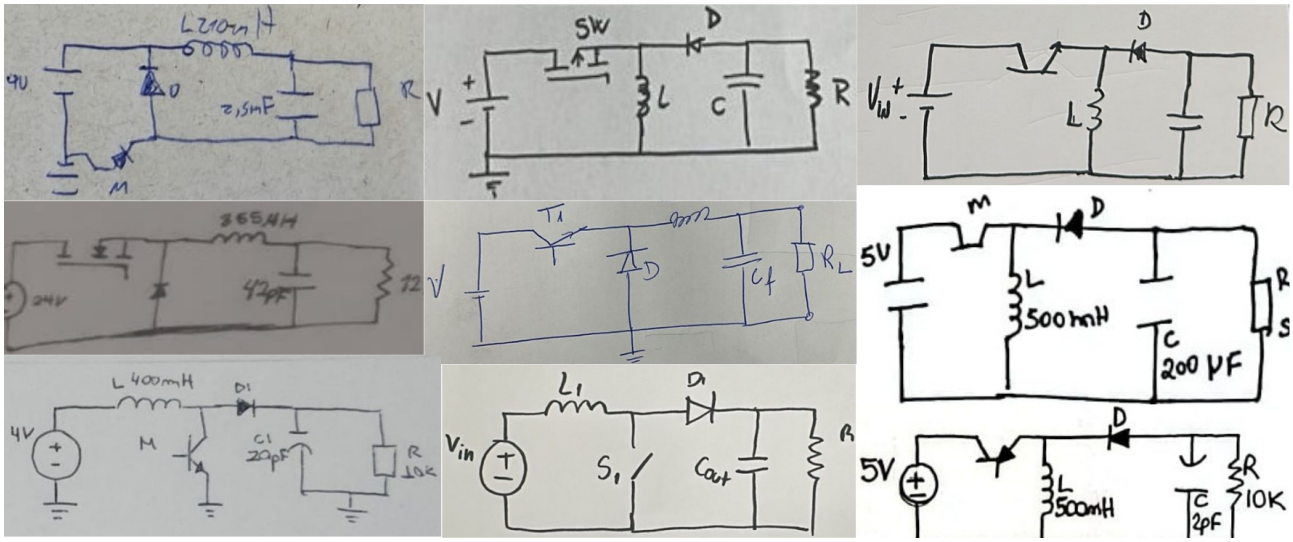


Figura 27: Exemplos de desenhos para o banco de imagens.

Fonte: Autor (2025).

Por fim, o *dataset* foi verificado quanto à consistência das anotações e exportado no formato compatível com o pipeline de treino (arquivos de imagem + labels no padrão YOLO), garantindo prontidão para as etapas de treino, validação e testes experimentais.

4.4 ESCOLHA DA YOLO E GPU

Após a preparação do conjunto de imagens, o próximo passo consistiu na seleção de um modelo YOLO para treinamento. O YOLO11 possui cinco variantes: nano, pequeno (*small*), médio (*medium*), grande (*large*) e extra grande (*extra large*). A versão *large* foi selecionada principalmente porque apresenta um mAP satisfatório em relação ao tempo de processamento na CPU. Além disso, possui uma arquitetura melhorada com módulos de atenção integrados. Embora o tempo de processamento e arquitetura sejam importantes, a precisão do modelo foi o fator decisivo. Isso inclinou a escolha pelo YOLO11-large. A Tabela 4 apresenta os parâmetros de cada variante, com destaque para a métrica de precisão, a qual demonstra a superioridade da versão grande.

Tabela 4: Tabela de Desempenho dos Modelos YOLO11 (Detalhado)

Model	Size (px)	$mAP^{val}_{50\text{-}95}$	Speed CPU (ms)	Speed T4 (ms)	Params (M)	FLOPs (B)
YOLO11n	640	39,5	$56,1 \pm 0,8$	$1,5 \pm 0,0$	2,6	6,5
YOLO11s	640	47,0	$90,0 \pm 1,2$	$2,5 \pm 0,0$	9,4	21,5
YOLO11m	640	51,5	$183,2 \pm 2,0$	$4,7 \pm 0,1$	20,1	68,0
YOLO11l	640	53,4	$238,6 \pm 1,4$	$6,2 \pm 0,1$	25,3	86,9
YOLO11x	640	54,7	$462,8 \pm 6,7$	$11,3 \pm 0,2$	56,9	194,9

Para o treinamento da YOLO11l foi criado um notebook no Google Colab, fazendo a importação do *dataset* via api da *Roboflow*. Devido a demora e exigência de memória RAM foi

necessário fazer a assinatura do Google Colab Pro para utilizar melhor hardware para o treinamento, dentro das máquinas disponíveis como mostrado na Tabela 8 foi escolhida a GPU NVIDIA A100 da Google por causa da sua capacidade computacional significativamente superior em operações de precisão mista (FP16/TF32) nos Tensor Cores, entregando centenas de TFLOPS dedicados a cargas de trabalho de *deep learning*, valor bem acima das GPUs T4 e L4 disponibilizadas na mesma infraestrutura indicados na Tabela 5, 6 e 7, com os valores de cada GPU em cada tarefa. Além disso, a A100 oferece capacidade de memória (40-80 GB de HBM2e) e largura de banda na ordem de terabytes por segundo substancialmente superiores às GPUs T4 (16 GB) e L4 (24 GB). Esse recurso é especialmente relevante no treinamento de redes Yolo, que demandam grande quantidade de memória para processar imagens em alta resolução e *batches* maiores, reduzindo o risco de estouro de memória e a necessidade de reduzir o tamanho do lote ou a resolução das imagens ainda mais.

Tabela 5: Opções de GPU disponíveis no ambiente de execução

Modelo da GPU
NVIDIA A100
NVIDIA L4
NVIDIA T4

Fonte: Autor (2025).

Tabela 6: Desempenho de computação

Modelo de GPU	**FP64**	**FP32**	**FP16**	**INT8**
A100 40 GB	9,7 TFLOPS	19,5 TFLOPS		
L4	0,5 TFLOPS	30,3 TFLOPS		
T4	0,25 TFLOPS	8,1 TFLOPS		

Fonte: Google Colab Machines.

Tabela 7: Desempenho de Tensor Core

Modelo de GPU	**FP64**	**FP32**	**FP16/FP32 de precisão mista**	**INT8**	**INT4**	**FP8**
A100 40 GB	19,5 TFLOPS	156 TFLOPS	312 TFLOPS	624 TOPS	1248 TOPS	
L4		120 TFLOPS	242 TFLOPS	485 TOPS	260 TOPS	485 TFLOPS
T4			65 TFLOPS	130 TOPS	260 TOPS	

Fonte: Google Colab Machines.

4.5 TREINAMENTO DA YOLO E GNN

Concluída a fase de preparação do dataset, iniciou-se o treinamento do YOLO11 e da GNN. Ambos os modelos foram treinados e avaliados no ambiente do Google Colab, escolhido por sua acessibilidade e pela facilidade na configuração de ambientes virtuais, permitindo a instalação de todas as dependências necessárias, como PyTorch, TensorFlow, OpenCV e CUDA. Além disso, o Colab disponibiliza aceleração por GPU e memória ampliada, reduzindo de forma significativa o tempo de treinamento e de inferência.

No caso específico da GNN, empregou-se a técnica de *oversampling* para compensar o desbalanceamento entre topologias e garantir melhor capacidade de generalização do modelo. Os parâmetros utilizados em cada treinamento encontram-se sintetizados nas Tabelas 8 e 9, permitindo a reprodução e comparação dos experimentos.

Tabela 8: Tabela de parâmetros - YOLO11

Parâmetro	**Valor**
Epochs	120
Otimizador	AdamW
Batch Size	16
Image Size	640x640px
Learning rate	0.0001
Dropout rate	0.15
Data Split	80-20

Fonte: Autor (2025).

Tabela 9: Tabela de parâmetros - GNN

Parâmetro	Valor
Epochs	500
Otimizador	Adam
Weight decay	0.0005
Scheduler	ReduceLROnPlateu
Batch Size	16
Dropout rate	0.3
Data Split	80-20

Fonte: Autor (2025).

Apesar dessas vantagens, o ambiente apresenta limitações de uso, especialmente em tarefas que demandam longas sessões de processamento. Nesses casos, a assinatura do Colab Pro torna-se necessária para acesso a recursos mais robustos. Para contornar essas restrições e maximizar o desempenho dos treinamentos deste trabalho, optou-se pela utilização da máquina A100 com alta RAM, que oferece maior capacidade de cálculo e estabilidade durante execuções prolongadas. Nas Figuras 28 e 29 são apresentados os resultados dos treinamentos do YOLO11 e da GNN.

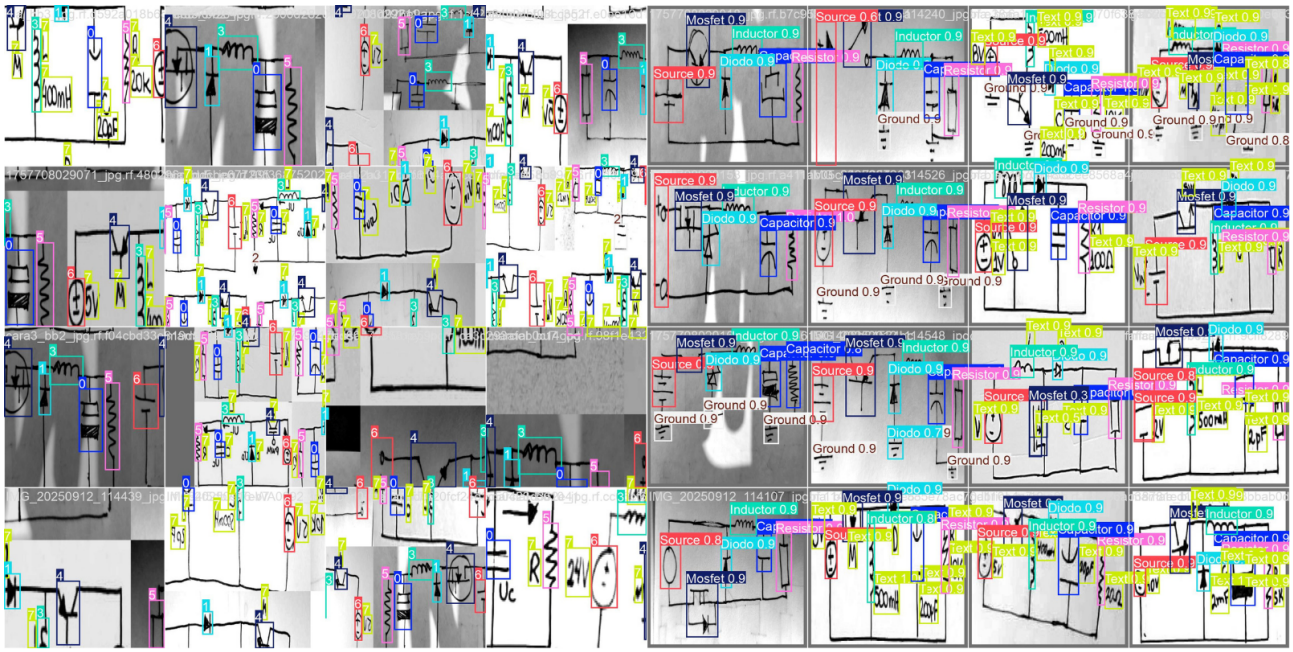
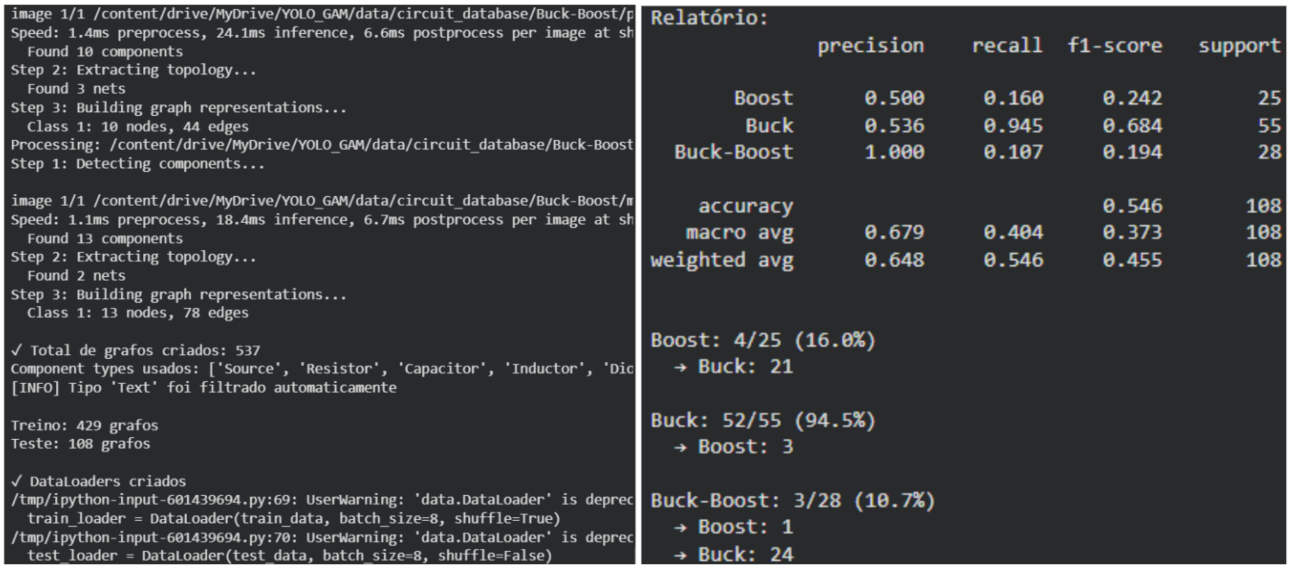


Figura 28: Resultados da detecção do treinamento.

Fonte: Autor (2025).



(a) (b)

Figura 29: (a) Treinamento da GNN no Colab. (b) Relatório do treinamento com suas métricas.

Fonte: Autor (2025).

4.6 DESENVOLVIMENTO DA PLATAFORMA CRB

Durante a ampliação do *dataset*, foi desenvolvido o CRB (*Circuit Reliability Bench*), um *framework* em Python baseado no LTspice. O CRB recebe *netlists* SPICE, válida a sintaxe e topologia, executa simulações transientes via PyLTspice (*Spicelib*), extrai sinais relevantes, detecta automaticamente a janela de regime estacionário, calcula métricas elétricas (tensão média, corrente média, *ripple*, potência) e gera saídas em gráficos PNG, relatórios TXT/CSV e validação JSON. A Figura 30 apresenta a arquitetura.

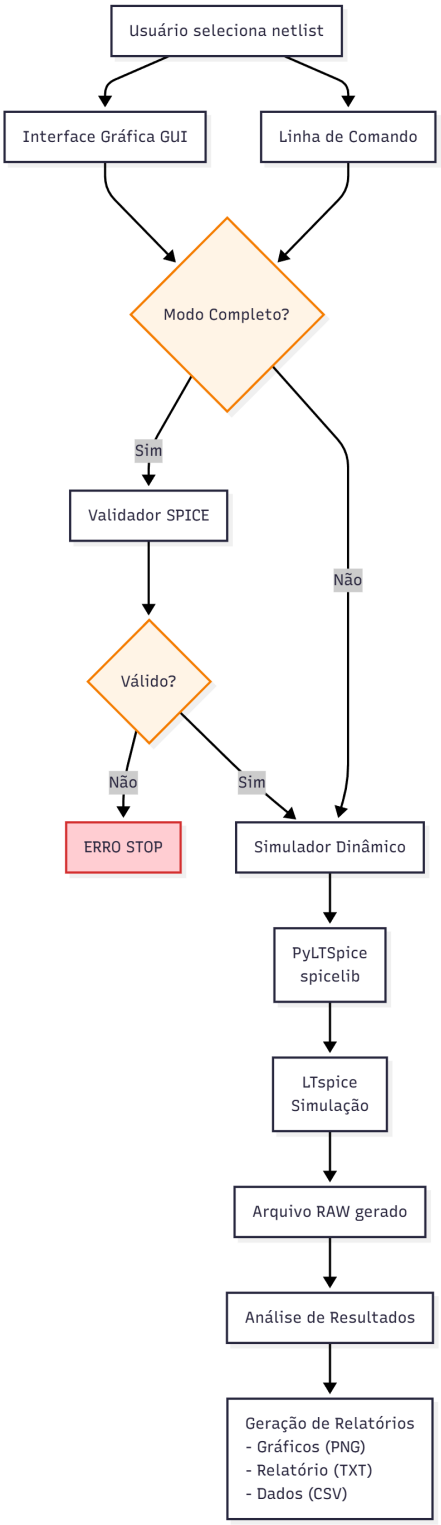


Figura 30: Fluxograma do processo de trabalho completo.

Fonte: Autor (2025).

O sistema possui três blocos principais: (1) Verificador de arquivos SPICE, (2) Simulador dinâmico e (3) Interface gráfica. O usuário seleciona o arquivo de circuito, realiza uma verificação preliminar, quando necessário, executa o PyLTspice (que invoca o LTspice), lê os resultados

gerados, analisa os dados, extrai os valores de interesse e gera gráficos e relatórios.

4.6.1 Função Validador SPICE

O desenvolvimento do Validador SPICE foi motivado pela necessidade de garantir não apenas a correção sintática, mas a integridade funcional do circuito recuperado. Como o processo de reconhecimento visual via YOLO e GNN é probabilístico, a má identificação dos elementos ou conexões pode resultar em diferentes categorias de erro: *i)* componentes incorretos, onde a natureza do dispositivo é confundida; *ii)* topologias divergentes da desenhada; *iii)* circuitos funcionais, porém distintos do desejado (falsos positivos funcionais); e *iv)* circuitos não funcionais ou eletricamente inválidos.

Enquanto o item *(iv)* resultaria em um erro explícito de convergência no simulador, sendo de fácil detecção, os itens *(i)* a *(iii)* representam riscos críticos, pois podem gerar netlists que o simulador aceita e processa, entregando resultados fisicamente coerentes, mas que não correspondem ao projeto original do usuário. Portanto, o Validador SPICE atua como uma camada intermediária de verificação semântica e topológica, projetada para interceptar inconsistências antes da etapa de simulação, assegurando que a netlist proposta seja não apenas executável, mas representativa de um circuito funcional válido.

Internamente, ele é modularizado em componentes: *SPICEParser* (tokenização e extração de elementos), SyntaxValidator (número de pinos, formatos de valor, existência de modelos), *TopologyValidator* (nós flutuantes, ilhas desconectadas, presença do nó de referência 0), *ElectricalValidator* (*loops* e ilhas) e *ModelValidator* (subcircuitos e modelos referenciados). Cada validação produz objetos *ValidationIssue* com severidade (ERROR, WARNING, INFO) e sugestões de correção. Um exemplo pode ser visto na Figura 32, onde o *SyntaxValidator* flagra um erro comum: o número incorreto de pinos em um componente, que poderia facilmente passar despercebido em verificações manuais, mas certamente resultaria em simulações falhas. O fluxograma (Figura 31) da função validador descreve esse encadeamento de parse → validações → relatório. Separar validação em camadas (sintaxe/topologia/elétrica) permite capturar desde erros simples (ex.: número errado de pinos) até problemas conceituais (ex.: falta de caminho DC para terra), o que evita simulações inválidas e orienta correção automática pelo usuário.

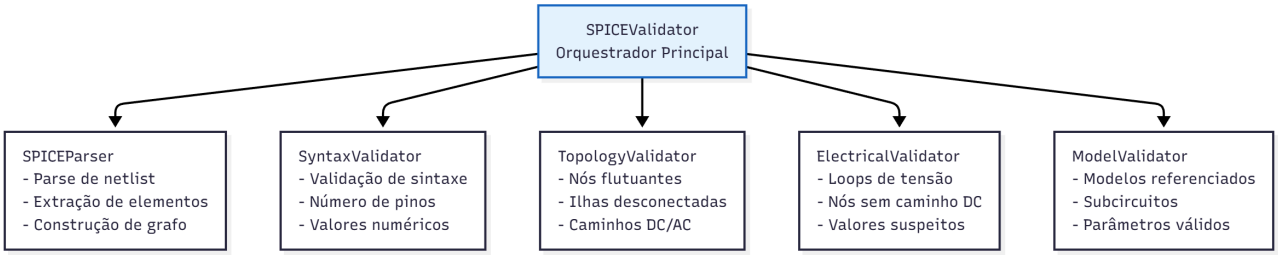


Figura 31: Arquitetura da função SPICEValidato.

Fonte: Autor (2025).

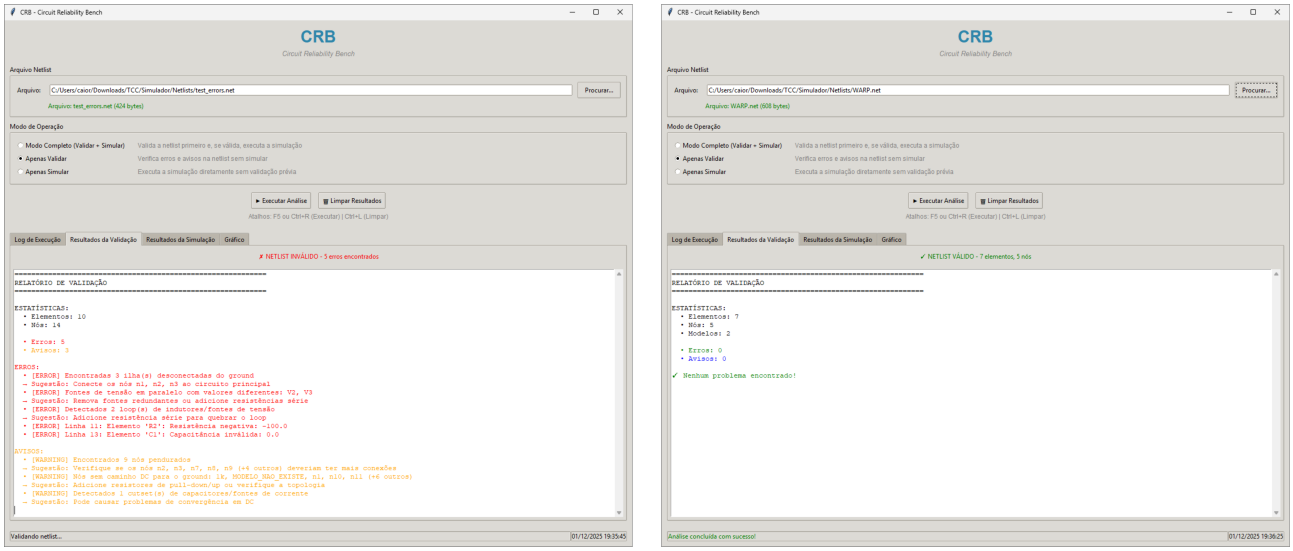


Figura 32: CRB somente validação de netlist(a) Netlist inválida; (b) Netlist correta.

Fonte: Autor (2025).

O núcleo da simulação é a função *run simulation*, tendo como principal comportamento:

1. Executar LTspice via *PyLTspice (spicelib)*: o projeto utiliza *SimRunner* para invocar o simulador, coletar *raw* e *log* e encapsular os resultados em objetos de fácil manipulação. A migração para a API orientada a objetos do *PyLTspice 5.0+* garante *thread-safety* e instanciação controlada.
2. Seleção automática de *probes*: após a leitura dos *traces*, o simulador aplica heurísticas para identificar sinais de interesse automaticamente, por exemplo, a procura por $V(\text{out})$, a detecção de resistores de carga e a seleção de $I(\text{Rload})$ para corrente. Quando múltiplos candidatos existem, uma heurística baseada em nomes, número de conexões e magnitude estimada prioriza o *probe* que melhor representa a saída do conversor.
3. Detecção de estabilização e cálculo de *ripple*: o algoritmo *detectar_estabilizacao* inicia a busca após uma fração do tempo total (*fracao_inicio_busca*, *default*), define uma janela mínima (*tempo_min_estavel*), e percorre janelas sequenciais calculando o *ripple* (pico a pico). Ao encontrar uma janela com *ptp* dentro da ***tolerancia_volts***, marca o índice como início da região estável e calcula métricas estatísticas (média, min, max, Vpp). Isso permite detectar automaticamente o regime permanente em sinais com transitórios e ruídos.

Como saída, a função retorna um dicionário com *tensao_media*, *corrente_media*, *potencia_media*, *tempo_estabilizacao*, além de salvar gráficos (PNG) e um relatório TXT com resumo e parâmetros da simulação demonstrados na Figura 33.

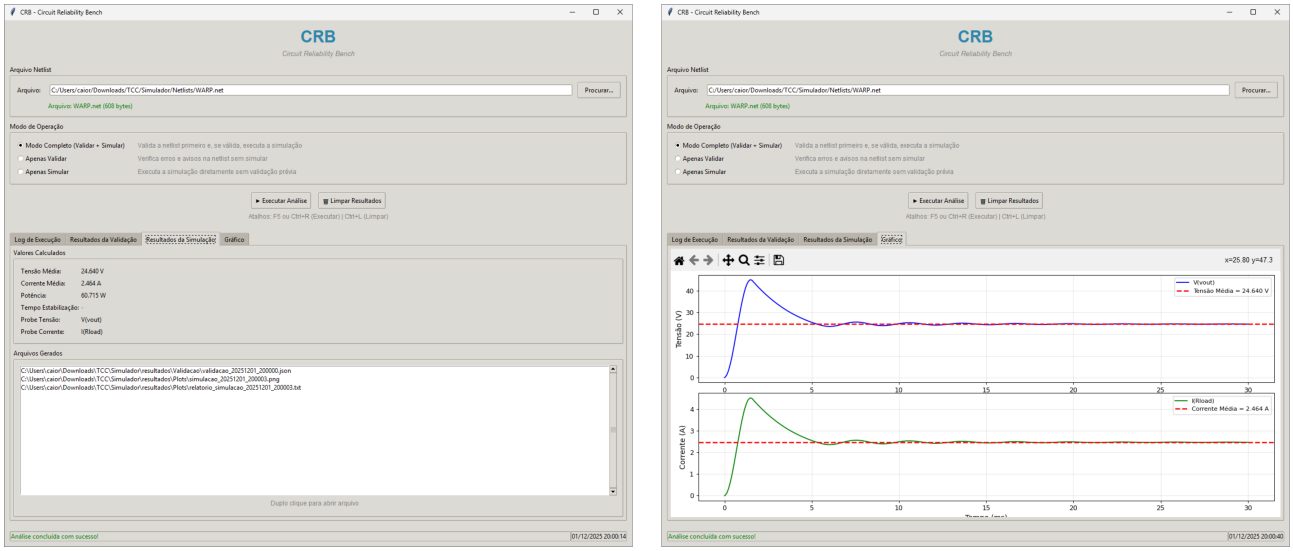


Figura 33: Processo completo de trabalho no CRB. (a) Aba de resultados da simulação com o local dos arquivos salvos; (b) Aba de plotagem dos gráficos de tensão e corrente da netlist.

Fonte: Autor (2025).

4.6.2 Interface Gráfica do CRB

A GUI foi implementada em *Tkinter* e organizada em abas: Log, Validação, Simulação e Gráfico. Ela permite três modos de operação: Modo Completo (Validar + Simular), Apenas Validar e Apenas Simular. O fluxo na GUI segue o seguinte padrão: usuário seleciona arquivo → escolhe modo → precisa executar → barra de progresso → apresentação dos resultados nas abas. A interface também fornece atalhos (F5 / Ctrl+R) e uma apresentação amigável das mensagens de validação (com cores por severidade), representada na Figura 34.

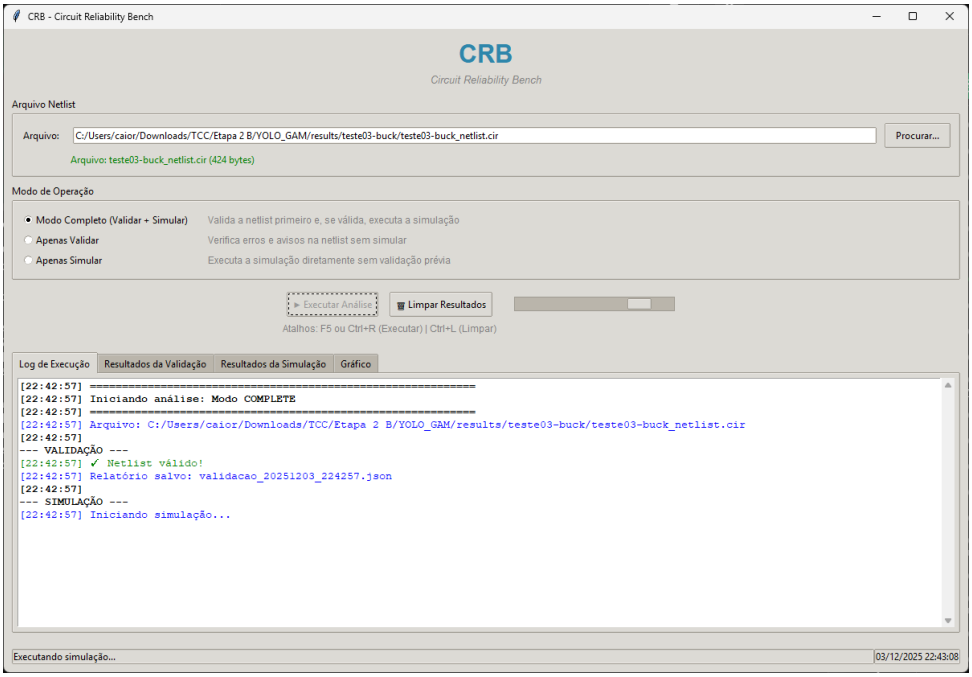


Figura 34: Interface principal do CRB.

Fonte: Autor (2025).

4.6.3 Problemas e limitações

Durante a implementação, foi necessária a adaptação para PyLTspice 5.0+, que mudou para uma API orientada a objetos. A refatoração envolveu a substituição de chamadas globais por instâncias de SimRunner e LTspice, além da normalização da leitura de traces para compatibilidade retroativa. Para garantir que a refatoração não quebrasse o fluxo, foram criados testes automatizados (*test_refactoring.py*) que verificam *imports* e a capacidade de instanciar as APIs; esses testes compõem o primeiro nível de integridade do projeto. Das limitações conhecidas, uma delas é em relação à dependência do LTspice; o simulador depende do executável do LTspice e, em ambientes Unix, o uso de *Wine* pode introduzir fragilidade.

Por fim, a última limitação seria em relação à detecção de regime estacionário: o algoritmo atual usa janela fixa; para sinais muito ruidosos ou com baixa variação devem ser implementados critérios estatísticos como desvio padrão e análise espectral. Assim, será explorado o uso de simulações de sinais com características variadas para identificar os melhores parâmetros estatísticos, o que pode envolver a implementação de testes com sinais ruidosos e melhorias no algoritmo de análise espectral.

4.7 MONTAGEM DO SCRIPT PRINCIPAL E VALIDAÇÃO EXPERIMENTAL

A integração de todas as etapas em um único script resultou na arquitetura final do projeto, ilustrada na Figura 35, que detalha a ordem de execução dos processos. Na primeira etapa do processo, o usuário envia uma imagem de entrada contendo um circuito eletrônico. Essa imagem

é submetida a um pré-processamento que inclui redimensionamento para 640×640 pixels, remoção de ruídos e normalização de contraste. Em seguida, a imagem processada é encaminhada à etapa de detecção de objetos, realizada pelo modelo GAM+YOLO, o qual, após a inferência, gera arquivos de configuração e de depuração, armazenados no diretório de resultados. A imagem com os componentes detectados é então utilizada na etapa subsequente, na qual é realizada a construção do grafo por meio de um algoritmo geométrico. Em paralelo, a imagem de entrada previamente processada é enviada à plataforma da OpenAI para a execução do OCR e extração dos valores dos componentes. Por fim, a última etapa consiste em fornecer à GNN tanto a representação do grafo quanto os valores extraídos dos componentes, permitindo a detecção da topologia do circuito e resultando na geração dos arquivos de saída, incluindo a netlist e as imagens de depuração.

O script principal integra diferentes tecnologias, cujas versões estão listadas na Tabela 10: *Ultralytics*/YOLO, NetworkX/PyG (*PyTorch Geometric*) para representação e conversão de grafos, PyTorch para carregamento e avaliação do *checkpoint* da GNN e um módulo OCR. A principal função desse script é coordenar essas tecnologias, instanciar os modelos, gerenciar entrada e saída de dados e selecionar o fluxo de OCR conforme a configuração definida.

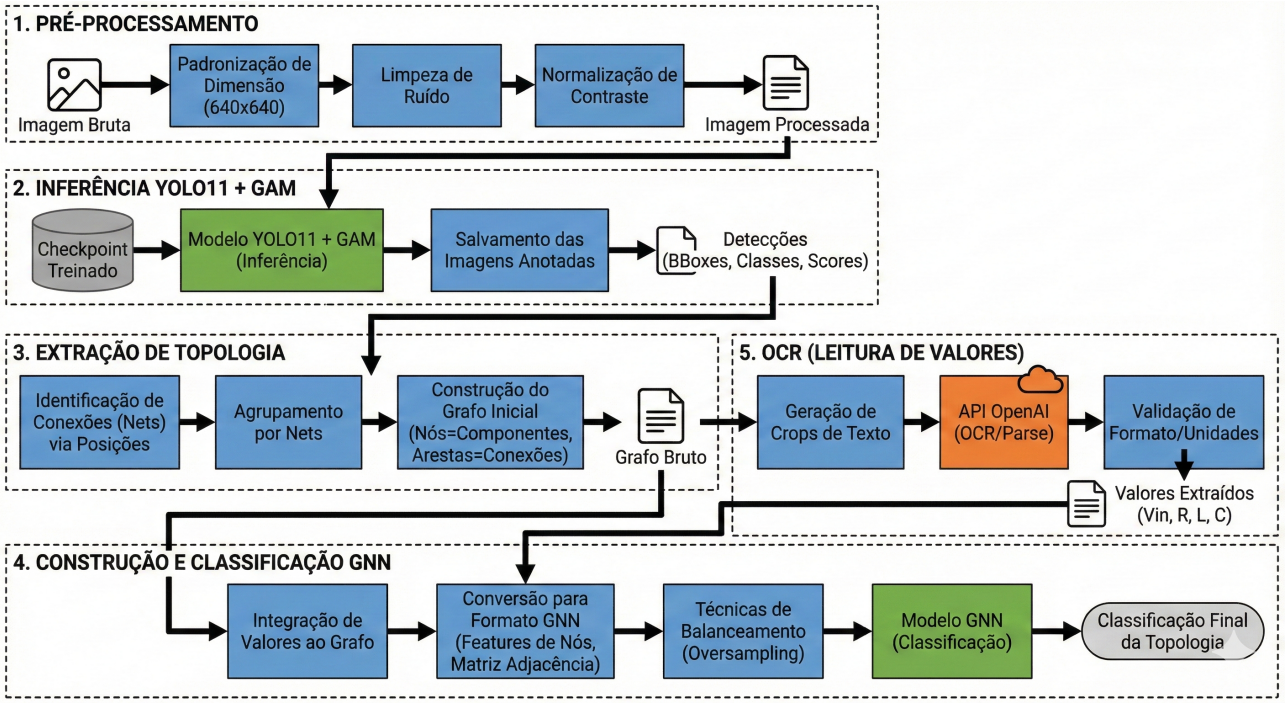


Figura 35: Fluxograma da pipeline do Script Principal

Fonte: Gerada pelo Gemini Pro 3.

Tabela 10: Versionamento dos componentes utilizados

Componente	Tecnologia	Versão	Propósito
Detecção	YOLO11 (Ultralytics)	11.x	Object detection em tempo real
Grafos	NetworkX	3.x	Representação e manipulação de grafos
GNN	PyTorch Geometric	2.x	Redes neurais em grafos
Deep Learning	PyTorch	2.x	Framework de ML
OCR	Google Gemini / OpenAI GPT-4V	API	Reconhecimento de texto em imagens
Visualização	Matplotlib, SVGWrite	-	Geração de gráficos e SVGs

Fonte: Autor (2025).

A função *main* atua como ponto de entrada do script principal, validando argumentos, preparando pastas de resultados, carregando *checkpoints* dos modelos YOLO e GNN e definindo parâmetros como *thresholds* e modo de OCR. Em seguida, processa as entradas, sejam imagens individuais ou pastas, executando em sequência: detecção, análise de *crops/threshold* (opcional), OCR, construção do grafo e inferência de topologia pela GNN. A função também centraliza logs, captura exceções de alto nível (IO/API) e registra um resumo de cada execução em JSON para fins de rastreabilidade.

Passando para a função de classificação de topologia, ela é o elo entre a representação de grafo e a GNN, carregando o *checkpoint* do modelo e detectando automaticamente qual arquitetura usar (simples / *oversampling* / GAT / *GraphSage*), instanciando o modelo correspondente, carregando os pesos e colocando o modelo ativo. Depois, ela solicita ao detector a construção do grafo e converte esse grafo para o formato PyG. Finalmente, ela realiza a inferência (*forward*) e retorna: rótulo predito, vetor de probabilidades, o detector instanciado e o objeto data (PyG) usado, permitindo salvar o grafo na inferência. Deve-se tratar os dados gerados do PyG porque a GNN espera tensores compactos, para isso é necessário uma função que utiliza o *networkx* e converte para PyG, normalizando a ordem dos nós e construindo uma matriz de *features* x (*one-hot* tipo + grau normalizado), montando um *index* de vértices e duplicando arestas para tornar o grafo não-direcionado e retorna o objeto dada do PyG.

Após essa etapa, a função de geração da *netlist* recebe a topologia inferida e os valores extraídos pelo OCR para preencher um *template* SPICE correspondente (Buck, Boost, Buck-Boost). Calcula parâmetros temporais (período, ton) a partir da frequência, organiza componentes com os nós corretos e escreve o arquivo na extensão .cir. Essa função é chamada somente após GNN + OCR estarem concluídos e produz o arquivo que será simulado.

5 RESULTADOS E DISCUSSÃO

Inicialmente, teve-se como objetivos validar o fluxo de anotação e pré-processamento de imagens, além de demonstrar a viabilidade do detector em localizar componentes em esquemas desenhados à mão (ver Figura 37). Para isso, foi criado um projeto público no *Roboflow*. No projeto, foram anotadas 26 imagens (15 Buck, 6 Boost, 5 Buck-Boost), padronizadas para 640×640 pixels, convertidas para escala de cinza e submetidas a data *augmentation* com rotações de ±5°. Essas rotações foram escolhidas para preservar a geometria das linhas e expor o modelo a variações típicas de esboços manuais. A padronização e o uso de escala de cinza garantem a uniformidade das entradas e facilitam a detecção sob diferentes iluminações. Isso aumenta a variabilidade do conjunto e a robustez do modelo. A Figura 36 ilustra a interface de anotação e exemplos do *dataset*.

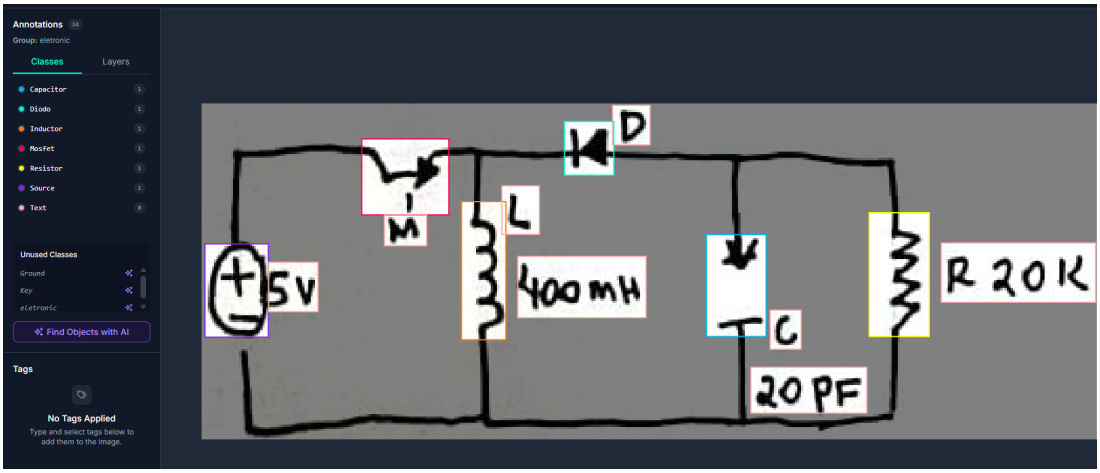


Figura 36: Exemplo de amostra após anotações das classes.

Fonte: Autor (2025).

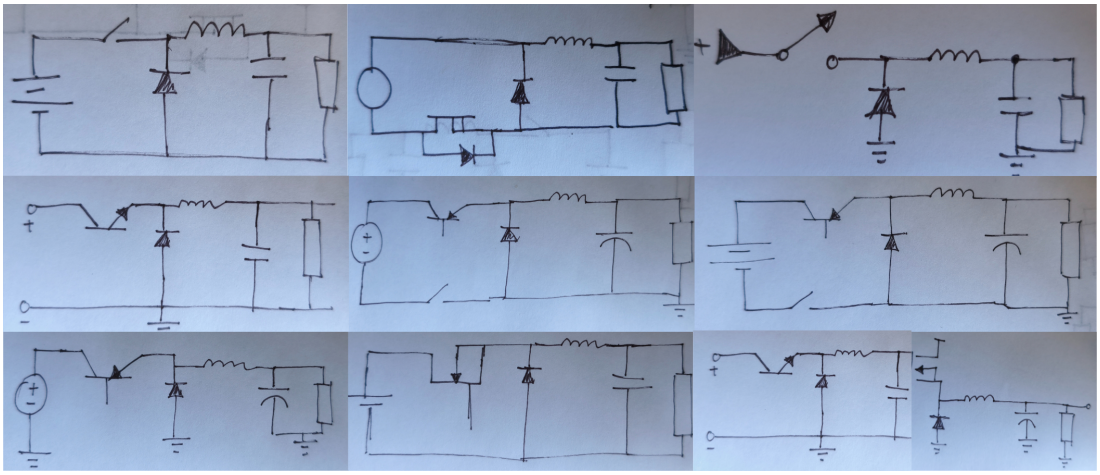


Figura 37: Amostras do dataset inicial.

Fonte: Autor (2025).

Observou-se que as caixas anotadas apresentaram variações de proporção e presença de ruído de fundo, como manchas e linhas adicionais. Essas características motivaram a implementação de um pipeline de limpeza posterior. Esse pipeline incluiu *inpainting* em regiões saturadas e a aplicação de filtro bilateral quando necessário para garantir recortes mais adequados para OCR e extração de feições visuais. Para ilustrar esse processo, considere uma imagem problemática de exemplo: inicialmente, ela apresenta proporções desiguais e várias manchas que obscurecem componentes importantes. Na primeira etapa de limpeza, aplica-se o *inpainting* para suavizar áreas saturadas, removendo manchas sem alterar traços críticos. Em seguida, utiliza-se o filtro bilateral para reduzir o ruído de fundo e acentuar detalhes essenciais para OCR. Após a limpeza, a imagem torna-se mais nítida, facilitando a detecção e extração dos elementos visuais. Cada projeto hospedado na plataforma *Roboflow* é acessado por meio de um link, como o exemplo https://universe.roboflow.com/tcc-giiu8/conversor-buck-gid5p/, que dá acesso ao *dataset* mais recente. O proprietário recebe uma chave de API para acesso remoto, permitindo downloads e uploads no *workspace*. Na página do projeto, é possível visualizar os pré-processamentos realizados antes do download, conforme exemplificado na Tabela 11.

Tabela 11: Configurações de Pré-processamento e Aumentação de Dados

Etapa	**Configurações**
Preprocessing	Auto-Orient: Applied
	Resize: Stretch to 640x640
	Auto-Adjust Contrast: Using Contrast Stretching
	Grayscale: Applied
Augmentations	Outputs per training example: 3
	Grayscale: Apply to 100% of images
	Blur: Up to 2px

Fonte: Autor (2025).

5.1 ANÁLISE DAS MÉTRICAS DOS TESTES

Devido ao tamanho reduzido do *dataset*, o treinamento foi concluído em aproximadamente 14 minutos, abrangendo 192 épocas com o relatório do treino apresentado na Tabela 12. O modelo alcançou um mAP50 de 0,85 para limiares (*threshold*) até 0,60, demonstrando estabilidade satisfatória em precisão e recall para diferentes limiares de confiança conforme a Figura 38.

Tabela 12: Resultados por classe no conjunto de validação

Class	Img	Inst	Precision (P)	Recall (R)	mAP@50	mAP@50-95
All	7	53	0,904	0,881	0,920	0,591
Capacitor	7	8	0,860	0,875	0,889	0,539
Diodo	7	7	0,993	1,000	0,995	0,584
Ground	4	11	0,988	0,818	0,980	0,548
Inductor	7	7	0,971	1,000	0,995	0,579
MOSFET	7	7	0,954	0,857	0,964	0,646
Resistor	7	7	0,842	0,767	0,773	0,606
Source	6	6	0,717	0,848	0,847	0,636

Fonte: Autor (2025).

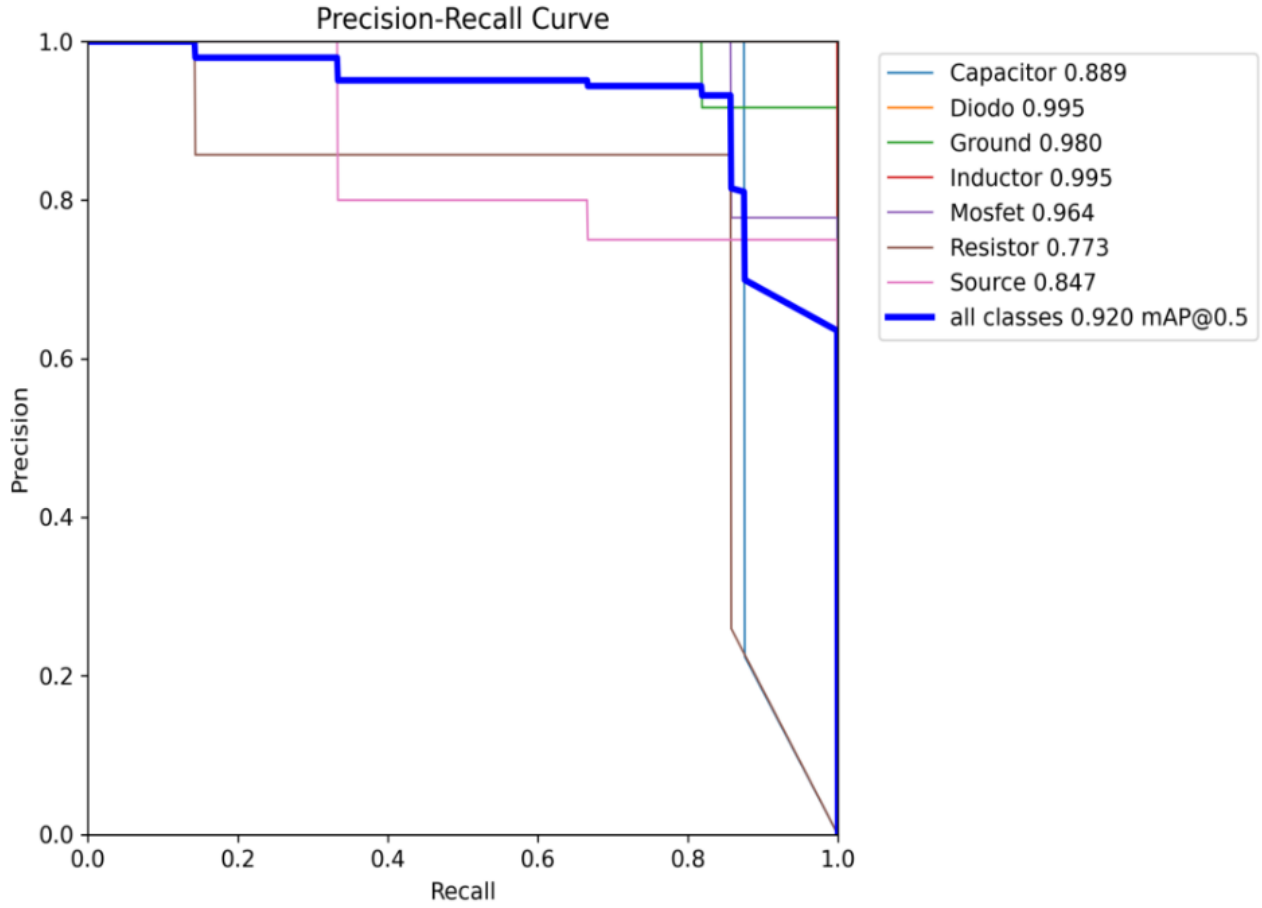


Figura 38: Curva Precision x Recall.

Fonte: Autor (2025).

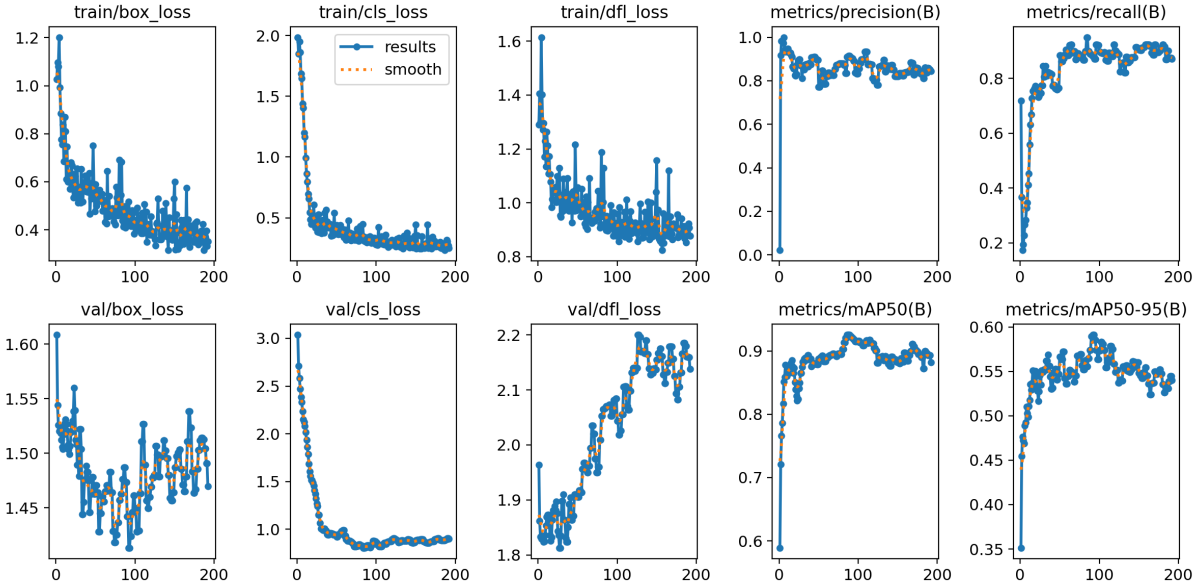


Figura 39: Curvas de treino e validação

Fonte: Autor (2025).

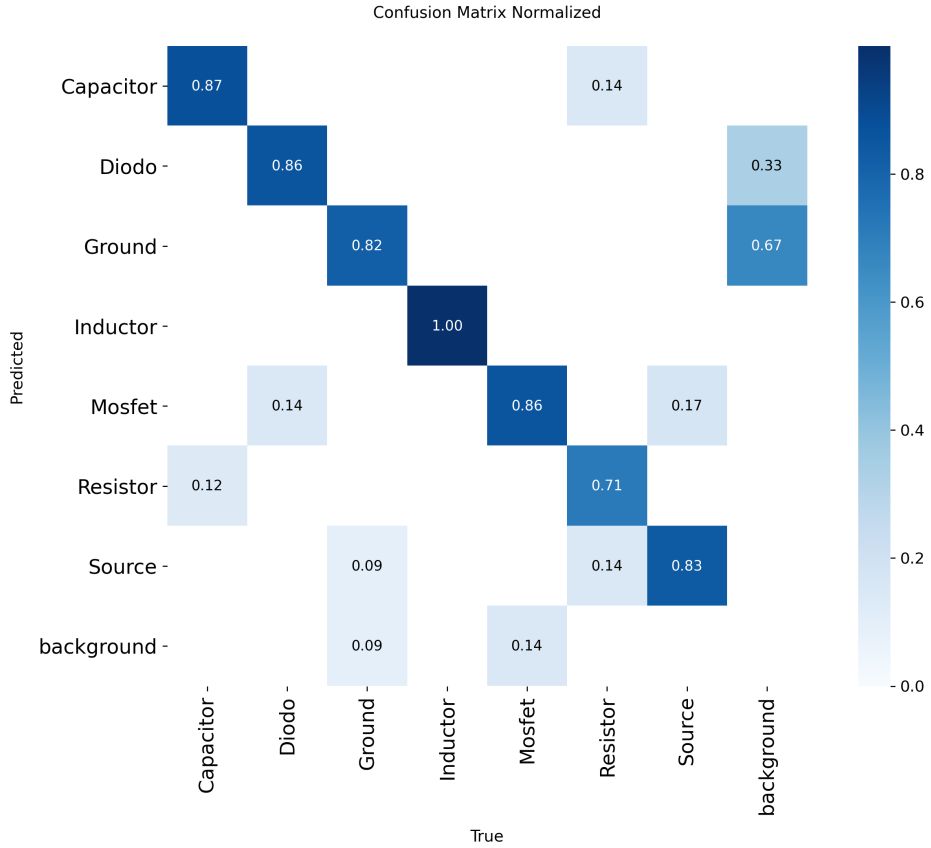


Figura 40: Matriz de confusão normalizada.

Fonte: Autor (2025).

Após o treino inicial, o modelo apresentou rápida diminuição de erros e alta pontuação mAP50 (Figura 39). Porém, houve aumento do erro de validação e queda no mAP50-95, indicando sobreajuste. A matriz de confusão (Figura 40) indica bom desempenho para indutor e capacitor, mas confusões entre Fonte e *Background* e entre Resistor e Fonte. O tamanho reduzido do *dataset* gerou baixa variabilidade e dificultou a generalização. O desbalanceamento de classes e a ausência de exemplos variados afetam negativamente a robustez do modelo, especialmente para categorias visualmente semelhantes. Ampliar o *dataset*, equilibrar as classes e diversificar as anotações são passos cruciais para avaliar melhor a generalização do modelo.

Para contextualizar esses resultados, é importante compará-los com *benchmarks* de trabalhos semelhantes na literatura. Por exemplo, o trabalho de Padilla et al. (2021) apresentou um mAP50 de 0,82 com um *dataset* similar em um estágio inicial, também enfrentando desafios de sobreajuste [27]. Da mesma forma, estudos como o de Oksuz et al. (2021) relataram a importância de um *dataset* equilibrado para evitar confusões entre classes como Fonte e *Background.* Considerando esses *benchmarks*, observa-se que nosso modelo demonstrou desempenho competitivo na detecção de componentes, mas ainda apresenta potencial de melhorias referentes ao equilíbrio de classes e à diversidade de dados [28].

Para ampliar a base de imagens de treinamento, criou-se uma campanha de coleta de desenhos esquemáticos, conforme demonstrado no apêndice A, com regras para máxima padronização. Ao fim da campanha foram obtidas 136 amostras com cerca de 20 participantes.

Após o recebimento das amostras, foi utilizada a plataforma do *Roboflow* para fazer as novas anotações nas imagens, agora com a adição de uma nova classe de texto para fazer o uso do reconhecimento dos valores dos componentes pelo OCR. Ao concluir a rotulação, foi necessário exportar o *dataset*, mas antes passaram-se por etapas de preparação até obter a versão final. Inicialmente, aplicou-se um *hold-out* split 80/20, reservando 80% das amostras para treinamento e 20% para teste/validação. Em seguida, todas as imagens foram padronizadas para 640×640 pixels, dimensão compatível com a entrada do modelo YOLO, o que é suficiente para preservar detalhes de componentes pequenos durante a extração de características.

Posteriormente, executou-se *data augmentation* para aumentar a diversidade do conjunto e reduzir o *overfitting*: aplicaram-se flips horizontais e rotações controladas, além de variações de brilho/contraste quando necessário para simular diferentes condições de captura. Essa etapa dobrou o número total de imagens, resultando no conjunto final composto por 266 imagens Buck, 160 imagens Boost e 128 imagens Buck–Boost, representando um incremento de 100% no volume de amostras disponíveis para treinamento. Ao retreinar o modelo YOLO11l, foi possível fazer a comparação com os resultados anteriores com *dataset* menor, as Figuras 41, 42 e 43 mostram a comparação do modelo treinado antes e depois do aumento do *dataset*:

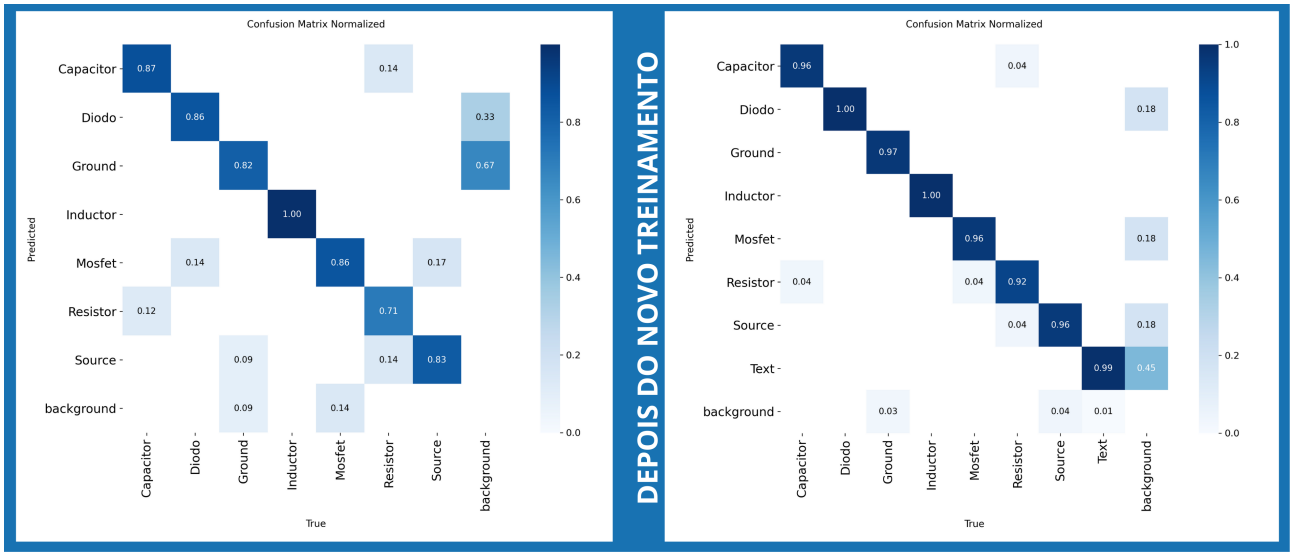


Figura 41: Comparação de matrizes de confusão dos treinos.

Fonte: Autor (2025).

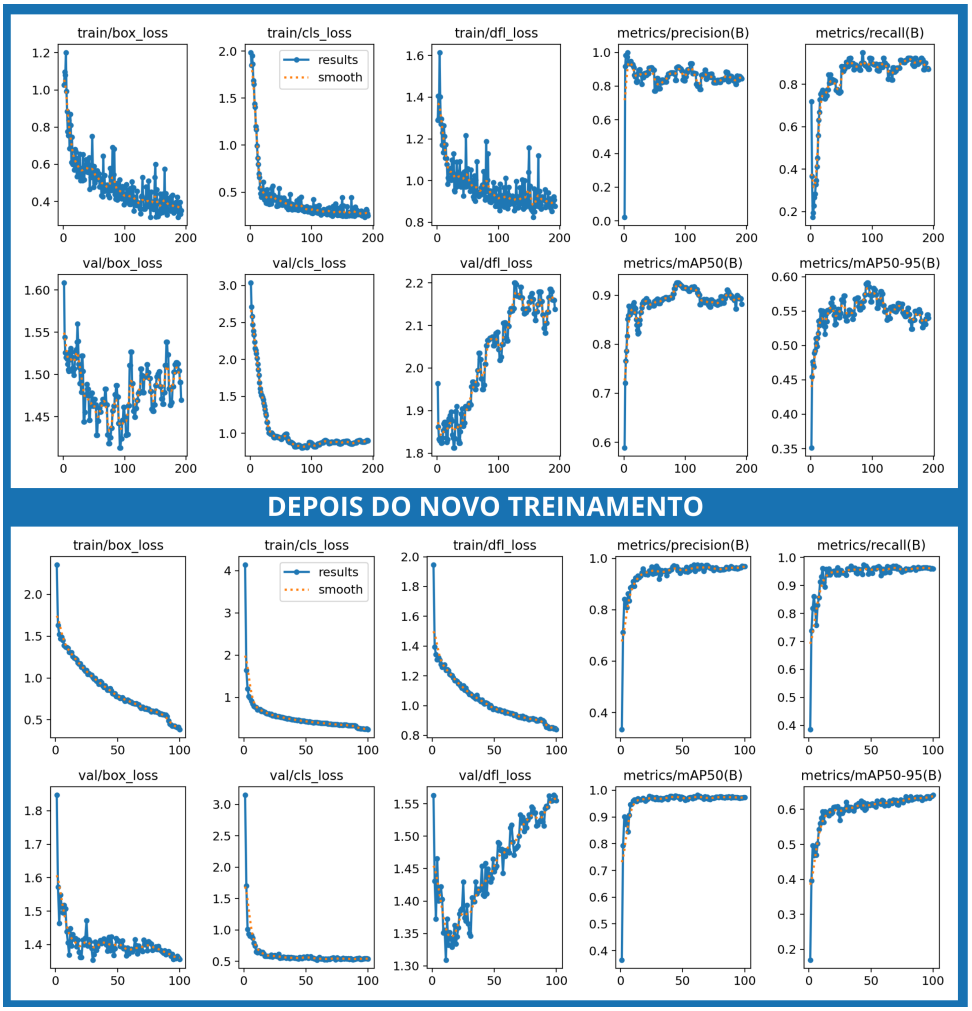


Figura 42: Comparação das matérias antes e depois do novo treino.

Fonte: Autor (2025).

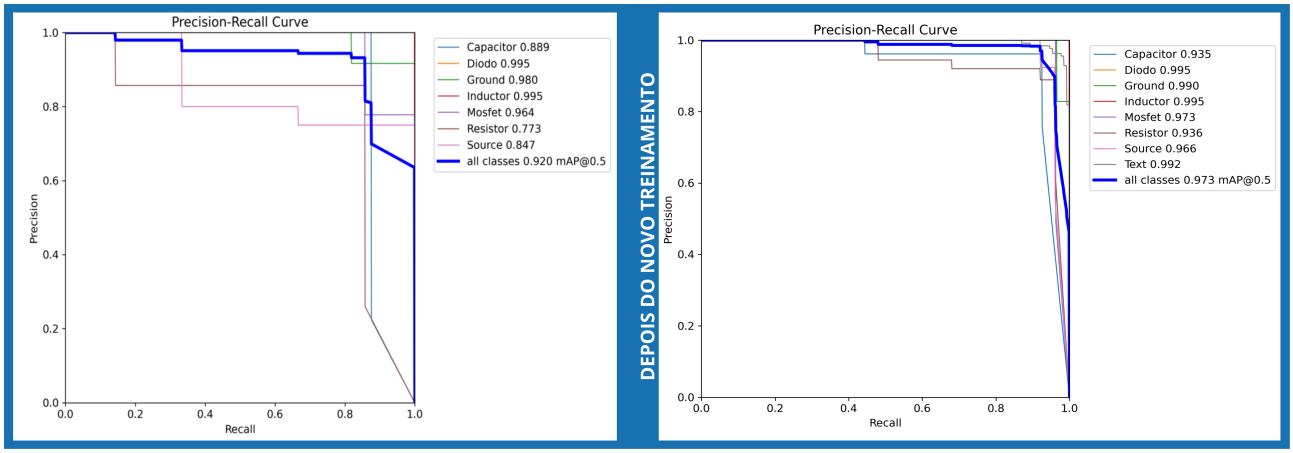


Figura 43: Comparação da métrica Precision vs. Recall antes e depois do novo treino.

Fonte: Autor (2025).

A comparação entre os treinamentos com conjuntos de imagens reduzidos e ampliados revela ganhos significativos em todas as métricas avaliadas. O mAP50 global aumentou de 0,920 para 0,973, com melhorias semelhantes na robustez de *recall* e precisão por classe (por exemplo, Resistor de 0,773 para 0,936 e *Source* de 0,847 para 0,966). A matriz de confusão evidencia redução das confusões entre Source e Background, bem como entre Resistor e classes próximas. As curvas de Precisão versus Recall também indicam maior precisão em níveis elevados de *recall*. Essas melhorias decorrem principalmente do aumento da quantidade e diversidade de imagens anotadas, do balanceamento das classes e de ajustes experimentais, como *augmentations* e hiperparâmetros. Na prática, esse incremento no mAP indica uma redução significativa de erros ao classificar componentes em esquemáticos reais. Com menos componentes etiquetados incorretamente, a eficiência operacional e a confiabilidade dos projetos aumentam, além de minimizar os custos e o tempo associados a correções manuais.

Contudo, uma análise detalhada dos casos de falha mostra que algumas limitações persistem mesmo após o aumento do *dataset*. Por exemplo, o modelo ainda apresenta confusões quando há sobreposição de componentes, rotulação manual inconsistente ou presença de artefatos provenientes de digitalizações de baixa qualidade. Casos em que os componentes são parcialmente ocultados por linhas de conexão ou anotados de maneira ambígua também levam a erros de classificação, indicando limites do modelo em cenários menos padronizados. Além disso, a precisão diminui quando as imagens apresentam variações extremas de iluminação ou quando há presença de ruído visual intenso, como rasuras e manchas, dificultando a identificação correta dos elementos.

Essas observações apontam para a necessidade de estratégias adicionais, como normalização avançada de imagens, técnicas de *augmentation* voltadas para simular cenários adversos e curadoria mais criteriosa das anotações. Por fim, o modelo continua encontrando dificuldades em distinguir entre classes visualmente semelhantes, o que pode afetar a precisão ao lidar com novos tipos de componentes não representados no conjunto de dados atual.

Essa evolução é corroborada qualitativamente pela figura de comparação das detecções (Figura 44): no painel da esquerda observam-se *bounding boxes* menos confiáveis e com pontuações (scores) mais baixos, como por exemplo MOSFET e Resistor , caixas que englobam excesso de fundo e algumas omissões; no painel da direita, as mesmas regiões aparecem com confiança substancialmente maior (ex.: MOSFET e Resistor), caixas mais apertadas ao contorno dos traços do componente e menos sobreposição com o fundo. Essa mudança visível indica que o modelo aprendeu filtros de características mais discriminativas (melhor contraste sinal-fundo e formas de traço) e melhorou a localização espacial das caixas, resultando em maior consistência no aumento observado em mAP50-95.

Apesar dos avanços qualitativos e quantitativos, ainda há oportunidades de aprimoramento na precisão da regressão de caixas (mAP50-95). Em casos extremos de texto ou sobreposição, recomenda-se, como próximos passos, o ajuste de *anchors* e da resolução de entrada, o refinamento do peso da função de perda de regressão (DFL/CIoU) e a inspeção manual dos falsos positivos e negativos remanescentes para orientar a coleta adicional de dados.

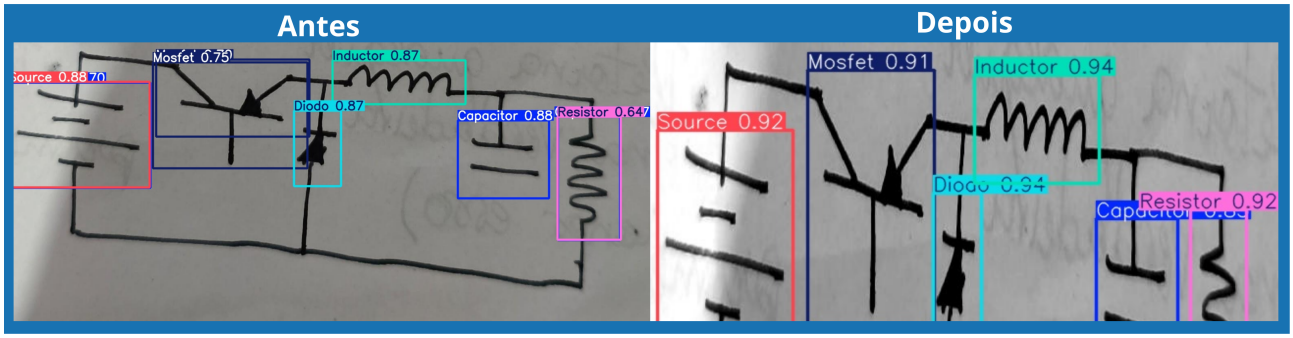


Figura 44: Resultado das detecções antes e depois do novo treinamento.

Fonte: Autor (2025).

5.2 DESENVOLVIMENTO DO SCRIPT DE DETECÇÃO E PRÉ-PROCESSAMENTO PARA OCR

Para realizar os testes de detecção foi necessária a criação de um script que utiliza o modelo treinado. O arquivo ***yolo_detect.py*** é o módulo responsável por executar a inferência do detector YOLO, desenhar resultados e salvar os arquivos para análise posterior. Ele começa com uma transcrição de argumentos que torna o uso flexível (modelo, fonte de imagem ou vídeo, limiar de confiança, resolução de saída, opção de salvar os recortes da imagem, pasta de saída etc.). Em seguida, valida o caminho do modelo e instância a classe do *Ultralytics*: YOLO (***model_path***, ***task='detect'***). O código detecta automaticamente o tipo de entrada (arquivo de imagem, pasta, arquivo de vídeo, webcam) e configura a captura/resolução conforme o necessário.

Durante o loop principal, cada frame é pré-processado passando ao modelo e as detecções retornadas são iteradas. Para cada detecção com confiança acima do limiar, o script desenha a caixa delimitadora, monta um rótulo de exibição e conta objetos. Quando uma imagem/quadrados

devem ser salvos, o script grava: (1) a imagem com as caixas desenhadas, (2) um arquivo JSON com metadados (lista de detecções com bbox, conf, cls, name).

O módulo de recortes e normalização recebe como entrada as imagens e os JSONs de detecção produzidos pelo passo anterior e tem por objetivo padronizar e preparar os crops destinados à extração de texto (OCR). O primeiro problema tratado é a diversidade de formatos de caixas delimitadoras: o modelo detecta automaticamente se as caixas estão em formato xyxy, xywh ou normalizado e converte para coordenadas absolutas seguras, com *clipping* para os limites da imagem. Em seguida aplicar validações (bbox degenerada, área inválida) e salvar cada recorte com nome seguro e borda configurável demonstrado na Figura 45. Antes de salvar, cada crop passa por uma rotina de limpeza: remoção de marcas coloridas (*inpainting* em regiões saturadas), *upscaling* se o recorte for muito pequeno, filtragem para reduzir ruído (bilateral filter) e outras operações leves que preservam os traços dos desenhos. O módulo também inclui utilitários que constroem um mapa de categorias a partir de formatos COCO-like ou JSONs simples e identificam quais detecções pertencem à categoria texto (por nome, **label** ou ***category_id***). O resultado é uma pasta estruturada com recortes prontos para OCR e um log informando quantas detecções de texto foram encontradas e processadas. Esse módulo é essencial para transformar detecções brutas em imagens apropriadas para reconhecimento de valores.

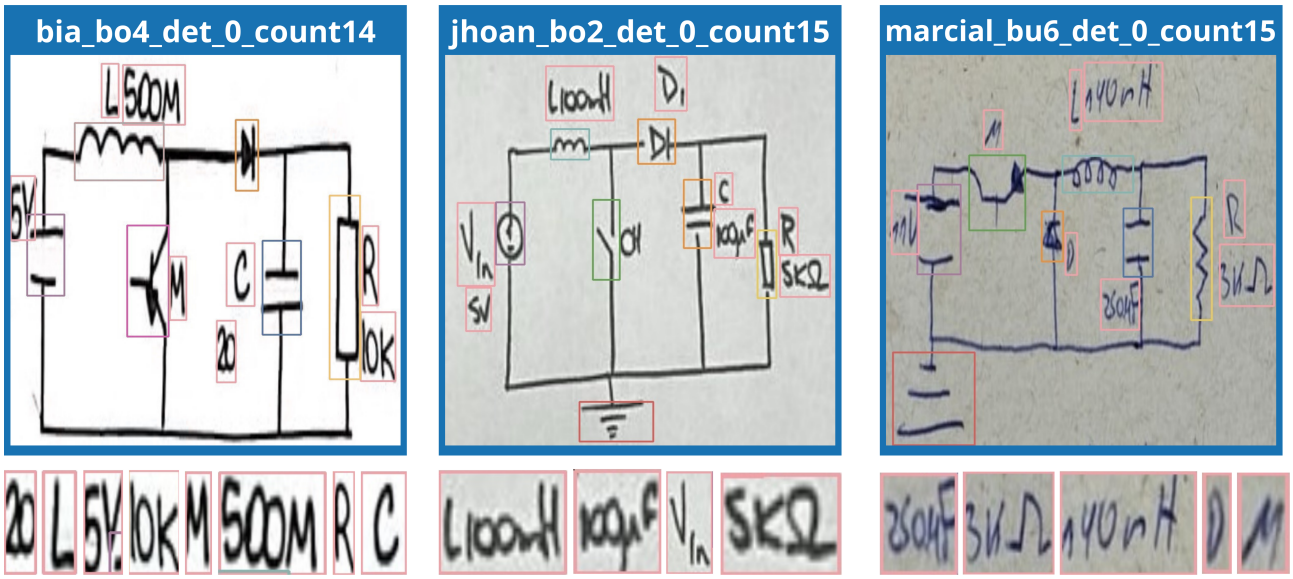


Figura 45: Crops após detecção

Fonte: Autor (2025).

Após a geração dos recortes, a ferramenta de análise de *thresholds* foi utilizada para explorar a melhor escolha dos parâmetros de binarização e pré-processamento ideais para OCR sobre os *crops* gerados. O script converte a imagem para *grayscale*, aplica *blur*, CLAHE (equalização adaptativa) e operações morfológicas, e então varre *thresholds* de 0 a 255 calculando a porcentagem de pixels considerados *foreground* para cada *threshold*. Os resultados são salvos em CSV e uma figura-resumo é gerada contendo a imagem original, o *grayscale*, a imagem processada e exemplos

de binarização, gerando também um gráfico “percentual vs *threshold*”que marca os *thresholds* de interesse apresentados na figura 46-47 apresenta o resultado das análises.

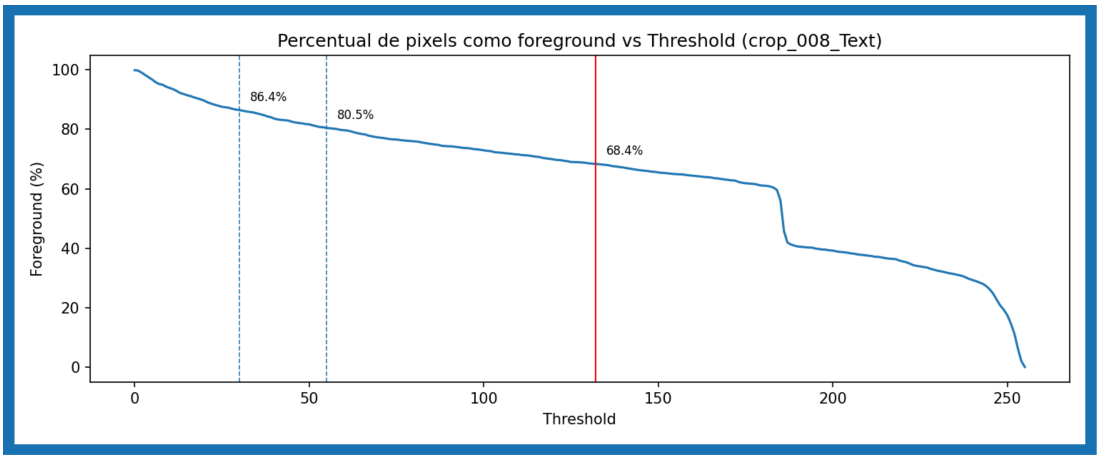


Figura 46: Gráfico do percentual de pixels detectados como foreground em função do threshold

Fonte: Autor (2025).

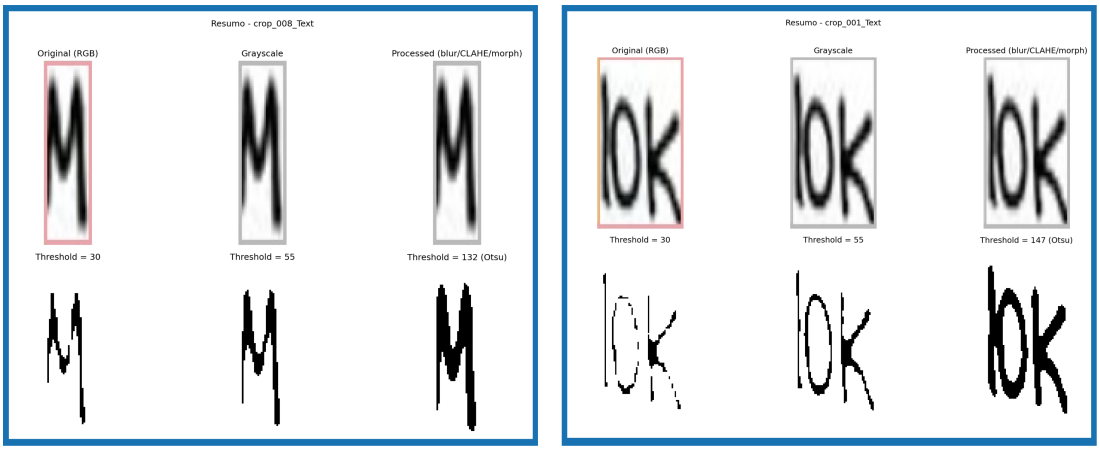


Figura 47: Resultados da análise de threshold para um recorte.

Fonte: Autor (2025).

O gráfico “percentual de pixels como *foreground* vs *threshold*” mostra como a proporção de pixels considerados primeiro plano muda conforme o limiar de binarização varia. A curva decrescente é esperada, pois *thresholds* menores classificam mais pixels como *foreground* (regiões escuras), enquanto *thresholds* maiores deixam apenas os traços mais escuros. No exemplo do ***crop_008,*** o *threshold* automático de Otsu marcado em vermelho resulta em cerca de 68% de pixels como *foreground*. Já *threshold* mais baixos marcados em tracejado, deixam entre e de pixels como *foreground*.

Ao inspecionar os binarizados gerados por esses *thresholds*, observou-se que *thresholds* muito baixos preservam praticamente todos os vestígios dos traços, mas também mantêm muito ruído e manchas do fundo; o limiar de Otsu (produz máscaras muito “espessas” que conectam

traços próximos e apagam detalhes finos, prejudicando a segmentação de caracteres manuscritos). Já um *threshold* intermediário, aplicado após uma etapa de equalização local de contraste (CLAHE) e um desfoque leve, preserva bem os contornos das letras/traços enquanto reduz o ruído de fundo. Esses efeitos ficam evidentes na figura 47, onde o binarizado com *threshold* 55 mantém as formas do "M" e "10K" sem fundir elementos próximos, resultando em uma máscara mais apropriada para OCR.

Com base na análise empírica automatizada realizada pelo script, foram definidos pipelines de pré-processamento que combinam: (1) conversão para escala de cinza, (2) aplicação de CLAHE para aumento do contraste local, (3) filtro bilateral leve para redução de ruído e (4) binarização com *threshold* fixo (valor 55) ou adaptativo, conforme o nível de iluminação do recorte. Esse processo foi aplicado tanto aos recortes individuais quanto à otimização da imagem completa do circuito antes do envio para OCR.

5.3 RESULTADOS DO OCR

Após gerar recortes limpos pelo de recortes e análise de threshold, foram testados dois motores de OCR locais amplamente utilizados em Python:

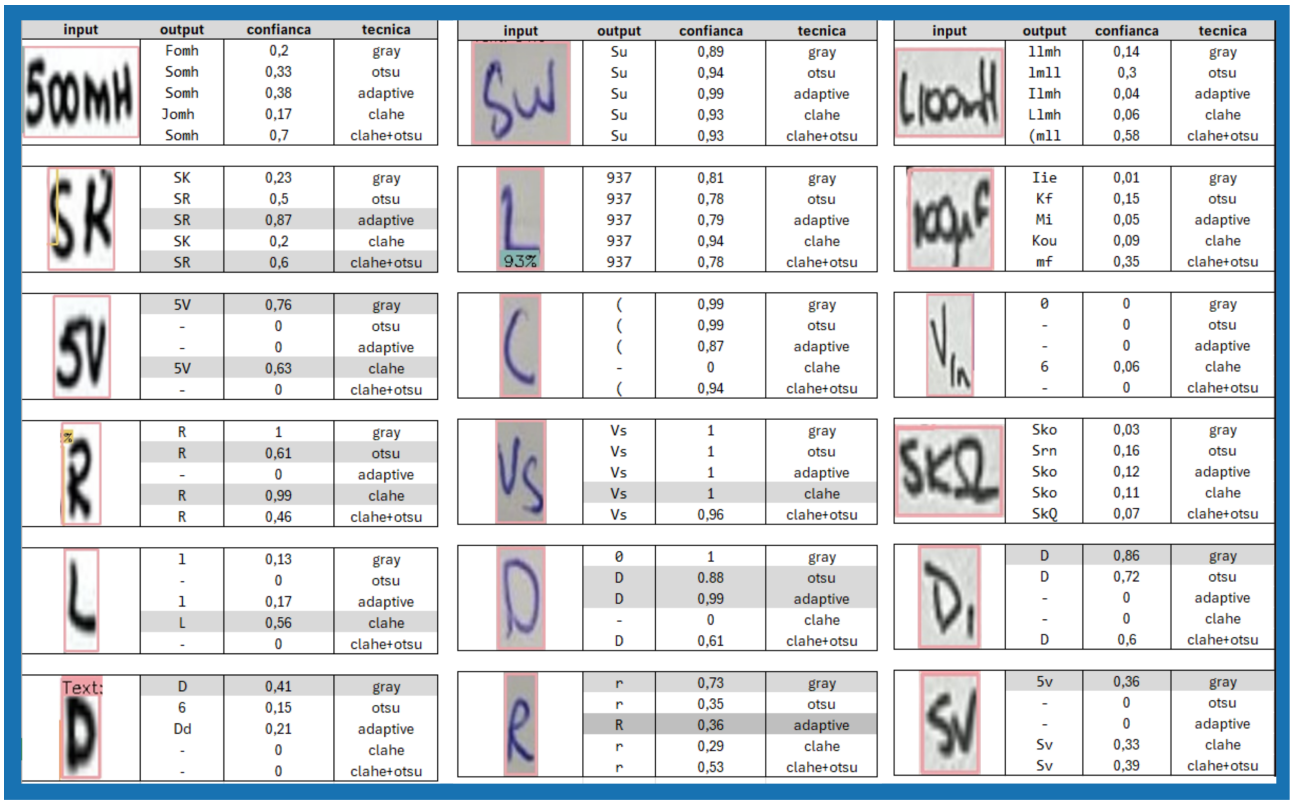
1. ***Tesseract*** (pelo ***pytesseract***) - requer o binário do ***Tesseract*** instalado no sistema e é invocado via *wrapper Python*.
2. ***EasyOCR*** (biblioteca ***easyocr***) - modelo baseado em aprendizado profundo (*deep learning*) pronto para uso, instalável por *pip*.

Ambos os motores foram instalados e executados diretamente no pipeline Python sobre os recortes. Os resultados consolidados (Figura 48) evidenciam o padrão típico de problemas com manuscritos: baixa acurácia para caracteres e números manuscritos, alta incidência de erros de substituição, fragmentação de dígitos, omissões e baixa confiabilidade na leitura de valores de componentes. A alteração dos filtros de pré-processamento não resultou em melhorias significativas nas leituras do OCR.

Em síntese, Tesseract e EasyOCR apresentam bom desempenho com texto impresso e traços nítidos, mas cometem erros sistemáticos em manuscritos com variação de espessura, inclinação e ruído do papel. A análise dos resultados revelou elevado número de falsos positivos e negativos, além de baixa taxa de leitura correta por recorte em valores manuscritos, mesmo após aplicação de CLAHE e binarização otimizada. Portanto, esses motores não atingiram desempenho aceitável para automação sem revisão humana. Para melhorar os resultados de OCR, recomenda-se implementar pipelines de pré-processamento com segmentação adaptativa de palavras ou caracteres antes do reconhecimento, bem como testes sistemáticos com diferentes técnicas de binarização adaptativa específicas para fundos complexos.

Também seria pertinente treinar modelos OCR personalizados utilizando conjuntos de dados anotados localmente, aplicando *transfer learning* a partir de modelos pré-treinados em manuscritos, com ajuste fino orientado aos padrões gráficos presentes nos esquemas analisados. Outra

estratégia seria o uso combinado de múltiplos mecanismos de OCR em ensemble, onde apenas reconhecimentos com consenso elevado são aceitos, reduzindo falsos positivos. Além disso, a aplicação de pós-processamento baseado em dicionários técnicos ou regras de validação para valores de componentes pode filtrar resultados incoerentes, aumentando a precisão geral do sistema. Por fim, técnicas avançadas de *augmentation*, como geração sintética de manuscritos utilizando GANs, podem expandir a diversidade dos dados de treinamento, promovendo maior robustez do modelo OCR em cenários adversos.



input	output	confianca	tecnica
500MH	Fomh	0,2	gray
	Somh	0,33	otsu
	Somh	0,38	adaptive
	Jomh	0,17	clahe
	Somh	0,7	clahe+otsu
SR	SK	0,23	gray
	SR	0,5	otsu
	SR	0,87	adaptive
	SK	0,2	clahe
	SR	0,6	clahe+otsu
5V	5V	0,76	gray
	-	0	otsu
	-	0	adaptive
	5V	0,63	clahe
	-	0	clahe+otsu
R	R	1	gray
	R	0,61	otsu
	-	0	adaptive
	R	0,99	clahe
	R	0,46	clahe+otsu
L	1	0,13	gray
	-	0	otsu
	1	0,17	adaptive
	L	0,56	clahe
	-	0	clahe+otsu
Text: D	D	0,41	gray
	6	0,15	otsu
	Dd	0,21	adaptive
	-	0	clahe
	-	0	clahe+otsu

input	output	confianca	tecnica
Sw	Su	0,89	gray
	Su	0,94	otsu
	Su	0,99	adaptive
	Su	0,93	clahe
	Su	0,93	clahe+otsu
L 93%	937	0,81	gray
	937	0,78	otsu
	937	0,79	adaptive
	937	0,94	clahe
	937	0,78	clahe+otsu
C	(	0,99	gray
	(	0,99	otsu
	(	0,87	adaptive
	-	0	clahe
	(	0,94	clahe+otsu
Vs	Vs	1	gray
	Vs	1	otsu
	Vs	1	adaptive
	Vs	1	clahe
	Vs	0,96	clahe+otsu
D	Ø	1	gray
	D	0.88	otsu
	D	0,99	adaptive
	-	0	clahe
	D	0,61	clahe+otsu
R	r	0,73	gray
	r	0,35	otsu
	R	0,36	adaptive
	r	0,29	clahe
	r	0,53	clahe+otsu

input	output	confianca	tecnica
L100nH	llmh	0,14	gray
	lmll	0,3	otsu
	Ilmh	0,04	adaptive
	Llmh	0,06	clahe
	(mll	0,58	clahe+otsu
100µF	Iie	0,01	gray
	Kf	0,15	otsu
	Mi	0,05	adaptive
	Kou	0,09	clahe
	mf	0,35	clahe+otsu
V_{in}	Ø	0	gray
	-	0	otsu
	-	0	adaptive
	6	0,06	clahe
	-	0	clahe+otsu
SKΩ	Sko	0,03	gray
	Srn	0,16	otsu
	Sko	0,12	adaptive
	Sko	0,11	clahe
	SkQ	0,07	clahe+otsu
D_1	D	0,86	gray
	D	0,72	otsu
	-	0	adaptive
	-	0	clahe
	D	0,6	clahe+otsu
5V	5v	0,36	gray
	-	0	otsu
	-	0	adaptive
	Sv	0,33	clahe
	Sv	0,39	clahe+otsu

Figura 48: Alguns dos resultados da análise das respostas dos OCRs por crops.

Fonte: Autor (2025).

5.4 MIGRAÇÃO PARA OCR VIA API (OPENAI)

Devido à limitação dos OCRs locais em reconhecer manuscrito, foi testado o OCR via API da OpenAI utilizando o modelo GPT 5.1, onde cada imagem é codificada em base64 e enviada junto com a requisição de reconhecimento. Na prática, o envio é feito convertendo a imagem binária para base64 e incluindo esse *payload* na chamada HTTP à API. Os testes comprovaram que a componente de OCR do serviço da OpenAI entrega transcrições visivelmente melhores em manuscritos: maior taxa de caracteres corretos, melhor separação de dígitos e menor necessidade de pós-edição manual. Entretanto, existem restrições operacionais importantes:

- **Custo em tokens**: cada recorte enviado por vez via API consome uma quantidade significativa de *tokens*. Ao enviar um recorte individual, gera aproximadamente 1400 *tokens* de custo de processamento.

- **Custo por circuito**: se um circuito tem em média 8 recortes, enviar um recorte por requisição implicaria em aproximadamente 11200 *tokens* por circuito, o que resulta em custos monetários relativamente elevados quando escalados para centenas de circuitos, em comparação ao utilizar somente uma imagem do circuito completo para realizar o OCR, custando em média 800 *tokens*. Outro problema ocorre ao decodificar o recorte, ocasionando perda de informação da imagem, como visualizado nas figuras do apêndice B. Para utilizar a plataforma da OpenAI, é necessário debitar no mínimo US$5,00, em que o custo típico de *input token* é de US$0,50 por 1 milhão de *tokens*, dando um total de 10 milhões de *tokens* de entrada. Utilizando 11200 *tokens* por circuito, o número máximo de imagens que poderá ser enviado é de cerca de 893 imagens, enquanto para a imagem completa do circuito seria 12500 imagens.
- **Latência e overhead**: múltiplas chamadas sequenciais aumentam a latência total do *pipeline* e complicam o gerenciamento de erros/tentativas.

Diante dessas limitações e com base na análise de thresholds, foi adotada uma estratégia de redução de custos: o pipeline passou a enviar a imagem completa do circuito, já pré-processada, como um único arquivo base64, em vez de enviar cada crop individualmente.

5.5 RESULTADOS DO TREINAMENTO DA GNN

Para treinar a GNN foi criado o notebook no Google Colab, onde foi organizado em blocos claros e reprodutíveis: Carregamento e pré-processamento dos dados, construção dos grafos por circuito (nós = componentes, arestas = conexões) e extração/normalização das *features* nodais e de arestas; separação explícita dos conjuntos (treino/teste/validação) com cálculo de estatísticas de classe; implementação das estratégias de balanceamento (entre elas o *oversampling*) e suas variantes de *augmentation*; definição modular dos modelos testados (baseline diagnóstico, GAT e GraphSAGE) com encapsulamento das rotinas *forward/backward, logging* de métricas por época, *callbacks* e por fim avaliação e geração de relatórios para diagnóstico. Essa organização permite trocar uma única célula do notebook para reexecutar experimentos, como mudar o balanceamento, o modelo ou os hiperparâmetros, sem afetar o restante do fluxo.

A Figura 49, em conjunto com os resultados apresentados nas Tabelas 13, 14 e 15, permite uma análise aprofundada do impacto das diferentes metodologias de treinamento no desempenho da GNN, especialmente diante das limitações do *dataset* disponível. Em linhas gerais, os experimentos demonstram que o desempenho do modelo é fortemente condicionado pelo desbalanceamento das classes e pela baixa representatividade estrutural de algumas topologias, o que restringe a capacidade de generalização da rede independentemente da arquitetura testada.

A Tabela 13, relativa ao treinamento com o conjunto original, sem balanceamento, evidencia esse viés: a classe Buck, por ser a mais representada, apresenta desempenho relativamente superior, enquanto as classes Boost e Buck-Boost exibem *recall* significativamente reduzidos. A matriz de confusão associada confirma a alta taxa de confusão entre essas topologias, indicando

que o modelo tende a aprender padrões dominantes do conjunto de treino sem captar características estruturais suficientes das classes minoritárias. Esse cenário caracteriza um problema clássico de representatividade, em que a quantidade e a diversidade das amostras têm papel central na capacidade de discriminação da rede.

Na Tabela 14, que resume os resultados após a aplicação de *oversampling*, mostra um aumento expressivo no *recall* da classe Buck (valores superiores a 90%), com melhora concomitante em métricas agregadas como acurácia e *F1-score*. Entretanto, os ganhos para Boost e Buck-Boost foram apenas marginais, sinalizando que a replicação de amostras ajuda a reduzir o viés em relação à classe majoritária, mas não compensa a falta de variabilidade estrutural nas classes menos representadas. Em outras palavras, o *oversampling* melhora a sensibilidade para Buck de forma rápida e reprodutível, porém não cria novos padrões estruturais que o modelo possa aprender para distinguir adequadamente as demais topologias.

E por fim, a Tabela 15 apresenta os resultados obtidos com a substituição da arquitetura GAT por GraphSAGE, indica uma redistribuição dos erros entre as classes, mas sem ganho substancial nas métricas globais quando comparada aos experimentos com *oversampling*. Esse comportamento sugere que, nas condições atuais do *dataset*, mudanças arquiteturais têm efeito limitado: a qualidade e a quantidade dos exemplos e a riqueza da representação do grafo exercem influência mais determinante sobre o desempenho do que a escolha entre as arquiteturas testadas.

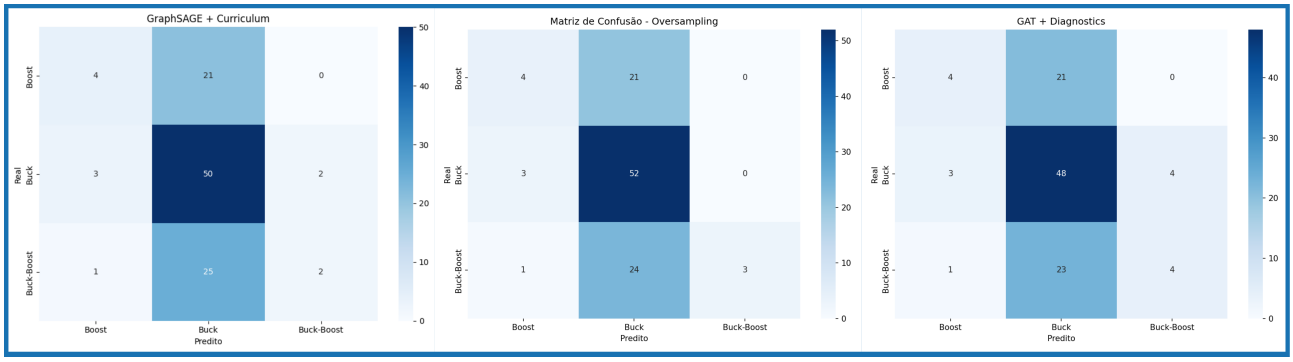


Figura 49: Comparação das matrizes de confusão entre metodologias. (a) Oversampling; (b) GAT + Diagnostics; (c) GraphSAGE + Curriculum.

Fonte: Autor (2025).

Tabela 13: Relatório de classificação e detalhamento: Oversampling

Classe	Precision	Recall	F1-score	Support
Boost	0,500	0,160	0,242	25
Buck	0,536	0,945	0,684	55
Buck-Boost	1,000	0,107	0,194	28
Accuracy			**0.546**	**108**
Macro avg	0,679	0,404	0,373	108
Weighted avg	0,648	0,546	0,455	108
Detalhamento de Classificação (Acertos e Erros)				
Boost: 4/25 (16,0%) → Classificado como Buck: 21				
Buck: 52/55 (94,5%) → Boost: 3				
Buck-Boost: 3/28 (10,7%) → Boost: 1, Buck: 24				

Fonte: O Autor (2025).

Tabela 14: Relatório de classificação e detalhamento: GAT+ Diagnostics

Classe	Precision	Recall	F1-score	Support
Boost	0,500	0,160	0,242	25
Buck	0,522	0,873	0,653	55
Buck-Boost	0,500	0,143	0,222	28
Accuracy			**0.519**	**108**
Macro avg	0,507	0,392	0,373	108
Weighted avg	0,511	0,519	0,446	108
Detalhamento de Classificação (Acertos e Erros)				
Boost: 4/25 (16.0%) → Classificado como Buck: 21				
Buck: 48/55 (87.3%) → Boost: 3, Buck-Boost: 4				
Buck-Boost: 4/28 (14.3%) → Boost: 1, Buck: 23				

Fonte: Autor (2025).

Tabela 15: Relatório de classificação e detalhamento: GraphSAGE + Curriculum

Classe	Precision	Recall	F1-score	Support
Boost	0,500	0,160	0,242	25
Buck	0,521	0,909	0,662	55
Buck-Boost	0,500	0,071	0,125	28
Accuracy			**0.519**	**108**
Macro avg	0,507	0,380	0,343	108
Weighted avg	0,511	0,519	0,426	108
Detalhamento de Classificação (Acertos e Erros)				
Boost: 4/25 (16.0%) → Classificado como Buck: 21				
Buck: 50/55 (90.9%) → Boost: 3, Buck-Boost: 2				
Buck-Boost: 2/28 (7.1%) → Boost: 1, Buck: 25				

Fonte: Autor (2025).

A escolha do *oversampling* como estratégia principal foi tomada com base em critérios práticos e empíricos que se alinham aos resultados acima descritos. A técnica atua diretamente sobre a escassez de exemplos de Buck, é simples de implementar e reproduzível no notebook, e proporcionou melhorias mensuráveis nas métricas sem necessidade de reengenharia da arquitetura o que se refletiu claramente na Tabela 13. Foram, no entanto, considerados e testados métodos alternativos: *undersampling* das classes majoritárias, ajuste de pesos na função de perda e técnicas de geração sintética como SMOTE. O *undersampling* mostrou risco de perda de informação relevante das classes majoritárias; o ajuste de pesos e as abordagens sintéticas, nos testes preliminares realizados, proporcionaram ganhos menos expressivos e mais sensíveis a ajustes finos de hiperparâmetros.

Para mitigar o risco de sobreajuste decorrente da replicação de amostras, o *pipeline* incluiu salvaguardas empíricas cuja eficácia é refletida nos resultados observados: perturbação controlada das *features* das amostras replicadas (*feature jitter*), regularização por *dropout* e *weight decay*, além de *early stopping*. Essas medidas reduziram a tendência do modelo a memorizar instâncias replicadas e contribuíram para que os ganhos verificados na Tabela 14 se mostrassem mais robustos em validação.

Em conclusão, os resultados das 13, 14 e 15 e da Figura 49 suportam a adoção do *oversampling* como solução operacional de melhor relação custo-benefício no contexto experimental aqui descrito: trouxe melhorias rápidas, de baixo custo computacional e alta reprodutibilidade, sem a necessidade de reestruturar a arquitetura. Ainda assim, suas limitações são claras — replicação não

suprime a necessidade de exemplos reais e de maior variabilidade estrutural. Recomenda-se, portanto, como próximos passos a priorização da ampliação e diversificação do *dataset* (coleta de mais exemplos para Boost e Buck-Boost), o enriquecimento da representação dos grafos (atributos adicionais em nós/arestas e modelagem estrutural mais detalhada) e uma busca sistemática de hiperparâmetros; essas medidas, combinadas ao pipeline de regularização já empregado, apresentam maior probabilidade de promover ganhos substantivos e generalizáveis no desempenho da GNN.

5.6 RESULTADOS EXPERIMENTAIS

Após compilar o repositório e gerar todos os arquivos necessários do projeto, foram testados desenhos esquemáticos do conversor Buck para verificar o funcionamento do pipeline completo. A imagem de entrada enviada por linha de comando está representada na figura 50 e 51. Subsequentemente, o resultado da identificação dos componentes pode ser observado na figura 52. O modelo conseguiu identificar com êxito todos os componentes da figura.

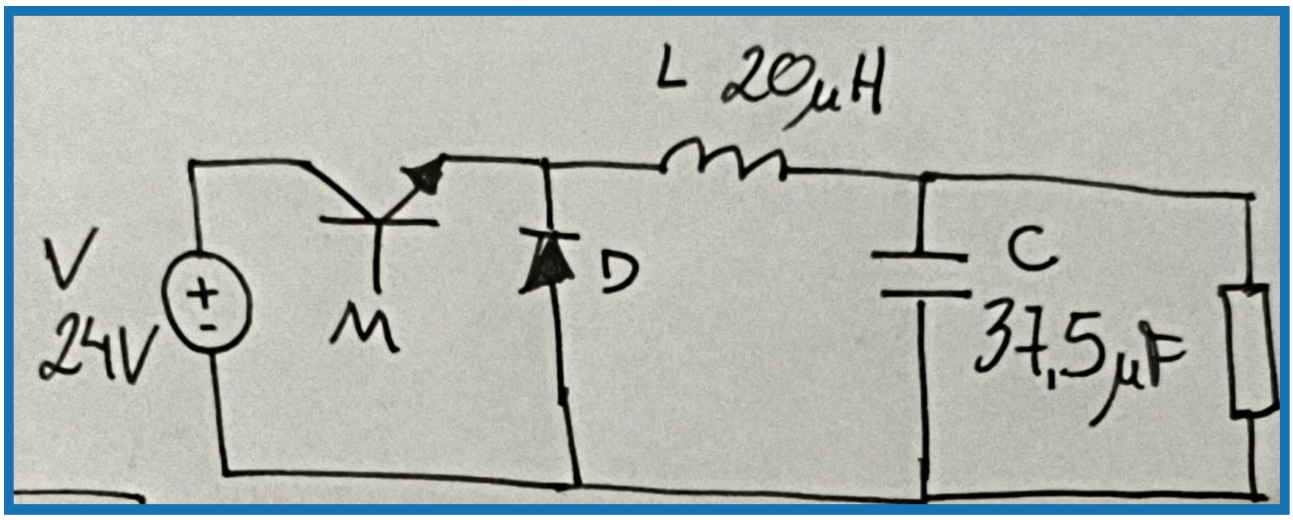


Figura 50: Imagem de entrada de um conversor Buck.

Fonte: Autor (2025).

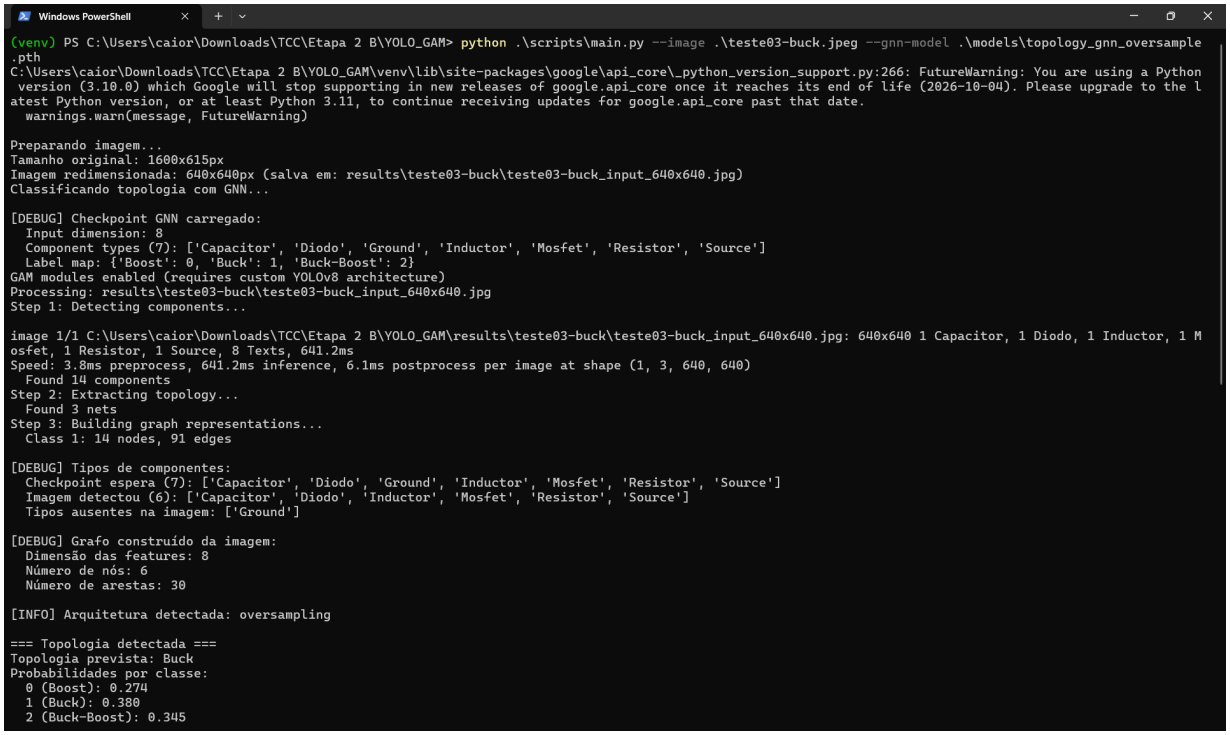


Figura 51: Execução do script principal por linha de comando, passando a imagem de entrada e escolhendo o modelo de GNN a ser utilizado.

Fonte: Autor (2025).

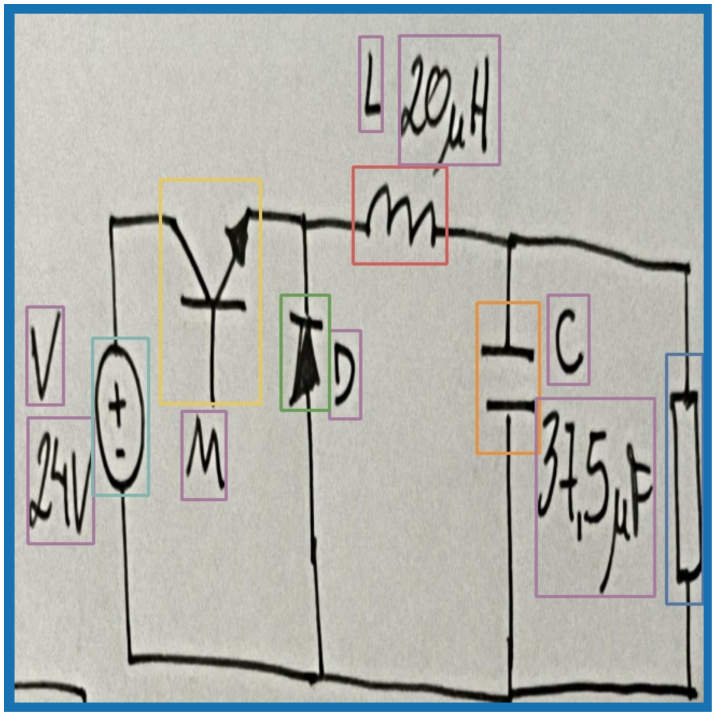


Figura 52: Imagem do resultado das detecções dos componentes.

Fonte: Autor (2025).

A identificação de topologia, como visto no fim da Figura 51, mostra que a GNN conseguiu inferir corretamente o tipo de conversor que está sendo analisado, tendo como probabilidade aproximadamente 38% de certeza de Buck, 27% de ser Boost e 35% de ser Buck-Boost. Durante a inferência, foi salvo o grafo gerado pela GNN para identificar a topologia representada na figura 53.

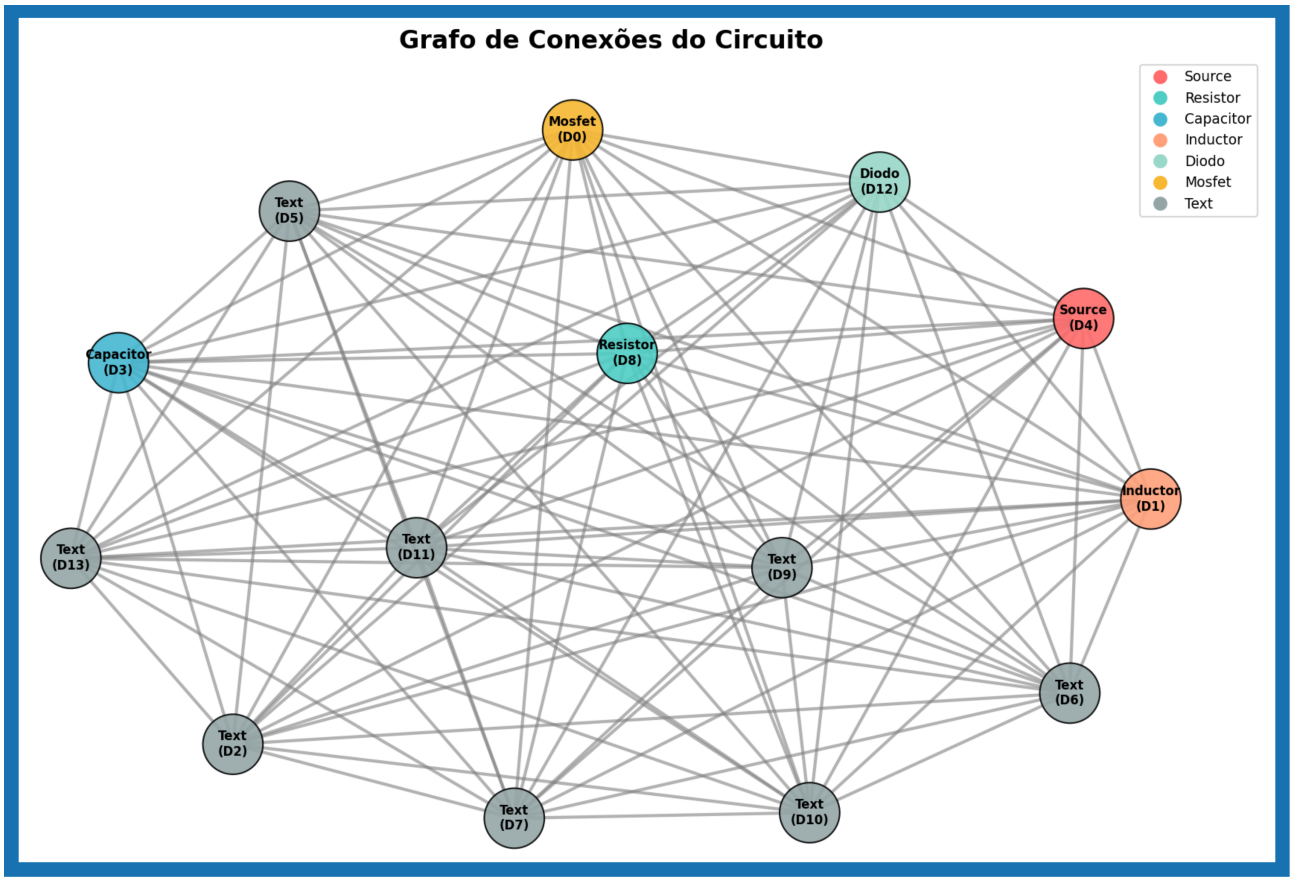


Figura 53: Representação gráfica das conexões da imagem de entrada do conversor buck

Fonte: Autor (2025).

A última parte do pipeline é a confirmação dos valores obtidos pelo OCR. Se não for possível a identificação dos valores ou se apresentar algum dado errado, o usuário pode decidir se continua o pipeline ou se arruma manualmente, e isso foi seguido com sucesso, como apresentado na figura 51. E por fim, o último processo é o salvamento dos arquivos gerados no processo na pasta de resultados, demonstrado na Figura 54, como a *netlist* correspondente à imagem de entrada (Fig. 55), o grafo e a detecção, acompanhados dos arquivos JSON e da probabilidade do tipo de topologia para análises mais profundas.

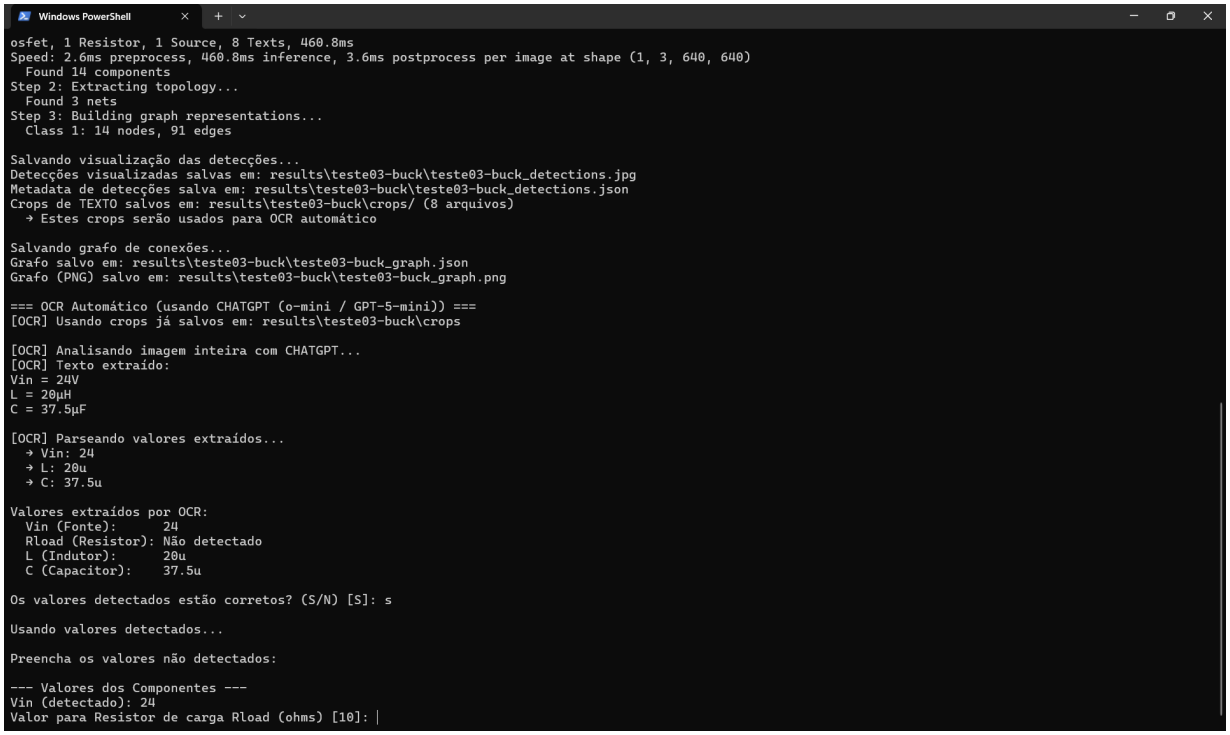


Figura 54: Validação dos valores obtidos do OCR pelo usuário.

Fonte: Autor (2025).

```
* Topologia detectada: Buck
* Netlist gerada automaticamente para teste03-buck (corrigida)
* Frequência de chaveamento: 100k
* Tempo de simulação: 200ms

Vin vin 0 DC 24
Ssw vin n_sw gate 0 SWmod
D1 0 n_sw Dmodel
L1 n_sw vout 20u
C1 vout 0 37.5u
Rload vout 0 8
Vgate gate 0 PULSE(0 10 0 50n 50n 5u 10u)

.model Dmodel D(Is=1e-9 N=1)
.model SWmod SW(Ron=1m Roff=1e6 Vt=5 Vh=0)

.tran 0 200m 0 1u
.end
```

Figura 55: Netlist correspondente ao circuito da imagem de entrada.

Fonte: Autor (2025).

A plataforma CRB foi utilizada para verificar a *netlist* e realizar a simulação elétrica do circuito correspondente, conforme ilustrado na Figura 56. No cenário teórico, com uma tensão de entrada de 24 V, estimou-se uma tensão de saída de 12 V, considerando os valores do indutor e do capacitor extraídos da imagem e uma carga de 8 ohms. Para avaliar a robustez e precisão do pipeline, adotaram-se critérios rigorosos de validação, incluindo a comparação quantitativa entre os valores simulados e os valores teóricos, calculados a partir das equações analíticas do conversor Buck. Além disso, resultados experimentais obtidos em testes anteriores serviram como referência adicional para avaliar a exatidão da simulação. As simulações abrangeram diferentes condições de carga, permitindo analisar não apenas a tensão e corrente de saída, mas também a eficiência energética do circuito em situações diversas. Observou-se que os resultados simulados apresentaram desvio relativo inferior a 3% em relação aos valores teóricos e experimentais de saída, confirmando a precisão do pipeline. Esta coincidência quantitativa valida não apenas o correto funcionamento do projeto como também a sua capacidade de generalização frente a distintos conjuntos de parâmetros elétricos.

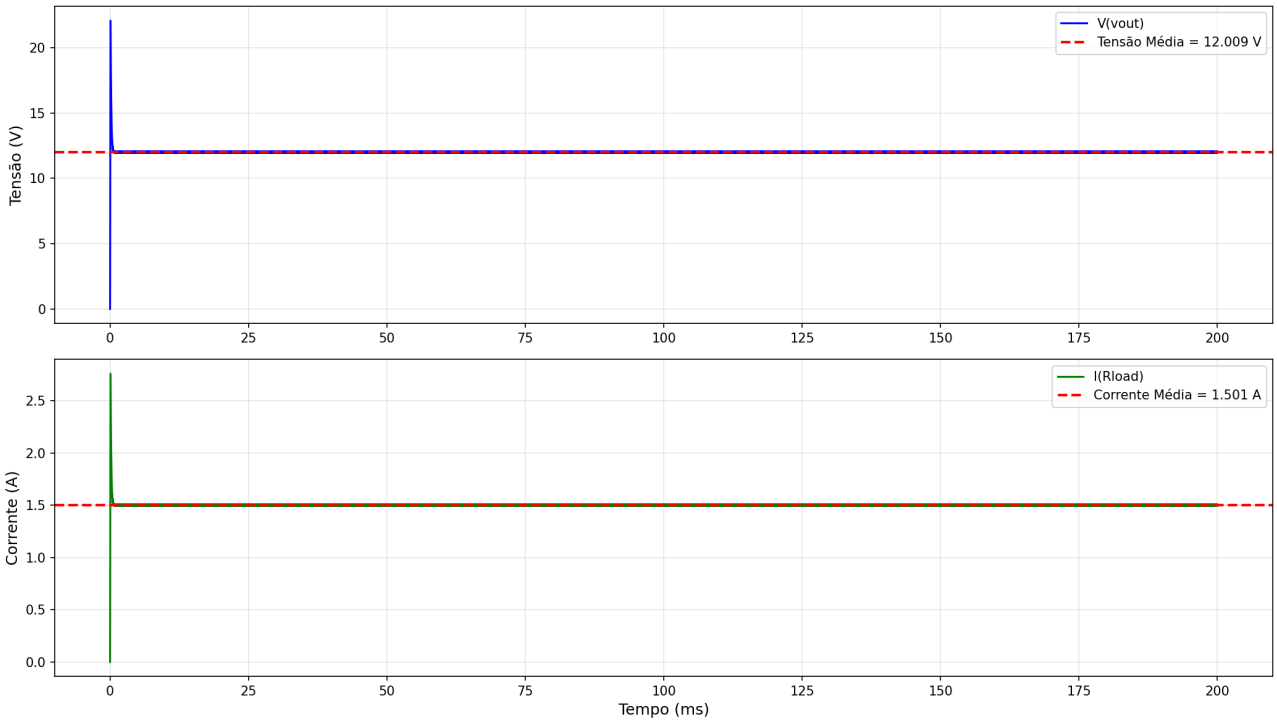


Figura 56: Resultado da simulação elétrica da netlist.

Fonte: Autor (2025).

6 CONCLUSÕES

O desenvolvimento da plataforma CRB integrou visão computacional e simulação SPICE para automatizar a etapa de prototipagem de conversores de potência. O treinamento do detector YOLO11, realizado com um conjunto final de 554 imagens manuscritas (266 Buck, 160 Boost e 128 Buck-Boost), resultou em um mAP@0.5 de 97,3%, com o tempo médio de inferência de 641 ms por imagem. Na etapa de classificação topológica, o modelo GNN com oversampling atingiu uma acurácia global de 54,6%. Enquanto a identificação da topologia Buck apresentou alto desempenho, com recall de 94,5%, as topologias Boost e Buck-Boost obtiveram resultados inferiores (recall abaixo de 16%), evidenciando a dificuldade do modelo em distinguir estruturas com grafos de representação simples (Classe 1). Por fim, a validação elétrica das netlists geradas corretamente demonstrou consistência, apresentando um erro relativo inferior a 3% na tensão de saída simulada em comparação aos valores teóricos calculados.

Para superar a baixa acurácia na classificação das topologias Boost e Buck-Boost, é necessária a migração para representações de grafos mais detalhadas (Classe 3 ou 4), que incluam nós de conexão explícitos em vez de apenas componentes. Adicionalmente, a implementação de um motor OCR local treinado especificamente para dígitos manuscritos deve substituir a dependência atual de APIs externas, reduzindo custos e latência. A expansão do dataset, focada no balanceamento das classes minoritárias, é o requisito principal para elevar a métrica mAP@50-95 e garantir a generalização do detector para novos estilos de desenho.

6.1 TRABALHOS FUTUROS

Os próximos passos devem priorizar a melhoria da precisão e da confiabilidade no reconhecimento de topologias, por meio do aumento do detalhamento das representações gráficas e do fortalecimento da confiança da GNN. Planeja-se a integração completa da plataforma CRB ao pipeline principal, visando à automação de processos e à geração de relatórios, bem como o desenvolvimento de um OCR local para ampliar a autonomia do sistema e expandir o suporte a um conjunto mais abrangente de topologias eletrônicas. Está igualmente prevista a criação de um banco de dados de componentes eletrônicos comerciais, contendo informações elétricas, encapsulamentos e *footprints*, com o objetivo de subsidiar a etapa de criação e automação do layout de PCBs. A implementação de uma interface gráfica também é considerada essencial para facilitar o uso por usuários não técnicos e ampliar o alcance da solução.

Adicionalmente, foram identificadas limitações relacionadas à execução e às heurísticas adotadas pelo sistema. Para mitigar a dependência do Wine e reduzir a fragilidade operacional, propõe-se o desenvolvimento de um script nativo para Linux, com potencial de melhorar o desempenho geral. No que se refere à seleção automática de *probes*, observou-se que, em circuitos não padronizados, as heurísticas atuais podem falhar; como estratégias de mitigação, sugere-se a disponibilização de uma opção de seleção manual na interface gráfica, bem como o aprimoramento das heurísticas por

meio de regras baseadas em topologias, experimentação com diferentes algoritmos e ampliação das bases de dados utilizadas. O impacto dessas melhorias será avaliado por métricas objetivas, como acurácia, precisão e mAP do detector YOLO, além de indicadores de usabilidade e desempenho, de modo a assegurar o atendimento aos objetivos de eficiência, confiabilidade e acessibilidade do sistema.

REFERÊNCIAS

[1] ZHONG, Ruizhe et al. A Comprehensive Survey on Graph Neural Networks for Electronic Design Automation. **arXiv preprint arXiv:2311.05043**, 2023.

[2] HUANG, Guohao et al. Machine Learning for Electronic Design Automation: A Survey. **ACM Transactions on Design Automation of Electronic Systems**, 2021. Comprehensive ML-based EDA survey.

[3] LIN, Rui et al. Beyond Schematic Capture: A Human-Centered Approach to Circuit Design Tools. *In:* PROCEEDINGS of the ACM Design Automation Conference (DAC). 2019. Discusses limitations of manual schematic capture workflows.

[4] HEMKER, Daniel et al. Electrical Circuit Analysis Using Deep-Learning Methods: Opportunities and Challenges. **Advances in Radio Science**, 2024. Highlights challenges in automating circuit interpretation and dataset scarcity.

[5] PEREIRA, Oliveira P. V.; F. G., de. Abordagem YOLOv5 para Detecção e Classificação de Esferas de Solda no Encapsulamento de Semicondutores. **Anais do Congresso da Sociedade Brasileira de Computação (SBC)**, 2024. Disponível em: https://sol.sbc.org.br/index.php/connect/article/view/27965.

[6] GAO, Ming et al. Circuit Diagram Retrieval Based on Hierarchical Circuit Graph Representation. **IEEE**, 2024. Uses GAM-YOLO for circuit component recognition and hierarchical graph retrieval strategy.

[7] PEREIRA, S. R. W. et al. Explorando Graph Neural Networks (GNNs): um mapeamento sistemático da literatura para análise de domínios de aplicação e métricas de avaliação. **Journal Brasileiro de Sistemas de Informação**, 2025. DOI: 10.5540/03.2025.011.01.0396.

[8] DONG, Z. et al. **CktGNN: Circuit Graph Neural Network for Electronic Design Automation**. 2023. arXiv preprint arXiv:2308.16406. Disponível em: https://arxiv.org/abs/2308.16406.

[9] HART, Daniel W. **Eletrônica de potência: análise e projetos de circuitos**. McGraw Hill Brasil, 2016.

[10] MUTTONI, Leonardo et al. Desenvolvimento e avaliação de um framework modular para síntese automática de circuitos analógicos: aplicação do algoritmo recozimento simulado com evolução geométrica de circuitos. Universidade Federal de Uberlândia, 2024.

[11] VOGT, Holger et al. **Ngspice User's Manual Version 34 (ngspice release version)**. 2021.

[12] BELLMAN, Richard Ernest. Artificial intelligence: Can computers think?, 1978.

[13] MCCARTHY, John. **What is Artificial Intelligence?** 2007. Stanford University. Disponível em: http://jmc.stanford.edu/articles/whatisai.html.

[14] COELHO, A.; SOBRENOME, B.; OUTROS, C. Introdução à Visão Computacional e Redes Neurais Profundas. **Revista Brasileira de Computação (Exemplo)**, v. 10, p. 100–110, 2017.

[15] SANTOS, J.; SILVA, M. Reconhecimento de Padrões Visuais em Aplicações Modernas. *In:* ANAIS do Encontro Nacional de Inteligência Artificial (Exemplo). SBC, 2017.

[16] MCCULLOCH, Warren S; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943.

[17] HAYKIN, Simon. **Neural networks and learning machines, 3/E**. Pearson Education India, 2009.

[18] GOODFELLOW, Ian et al. **Deep learning**. MIT press Cambridge, 2016. v. 1.

[19] CLEVERT, Djork-Arné; UNTERTHINER, Thomas; HOCHREITER, Sepp. Fast and accurate deep network learning by exponential linear units (elus). **arXiv preprint arXiv:1511.07289**, v. 4, n. 5, p. 11, 2015.

[20] FACELI, Katti et al. Inteligência artificial: uma abordagem de aprendizado de máquina, 2021.

[21] KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. **Communications of the ACM**, ACM, New York, NY, USA, v. 60, n. 6, p. 84–90, 2017. DOI: 10.1145/3065386. Disponível em: https://doi.org/10.1145/3065386.

[22] QIN, Zhuwei et al. How convolutional neural network see the world-A survey of convolutional neural network visualization methods. **arXiv preprint arXiv:1804.11191**, 2018.

[23] SANCHEZ-LENGELING, Benjamin et al. A gentle introduction to graph neural networks. **Distill**, v. 6, n. 9, e33, 2021.

[24] REDMON, Joseph et al. You Only Look Once: Unified, Real-Time Object Detection. *In:* PROCEEDINGS of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, USA: IEEE, jun. 2016. p. 779–788. DOI: 10.1109/CVPR.2016.91.

[25] OLIVEIRA, Arthur Germano de; DAMASCENO, Alexandro Lima; GONDIM, Ruan Dos Santos. **Um Estudo Analítico entre Versões do Algoritmo de Visão Computacional YOLO**. 2024. Monografia (Trabalho de Conclusão de Curso) – Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE).

[26] JEGHAM, Nidhal et al. Evaluating the evolution of yolo (you only look once) models: A comprehensive benchmark study of yolo11 and its predecessors. **arXiv e-prints**, arxiv–2411, 2024.

[27] PADILLA, Rafael et al. A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit. **Electronics**, v. 10, n. 3, 2021. ISSN 2079-9292. DOI: 10.3390/electronics10030279. Disponível em: https://www.mdpi.com/2079-9292/10/3/279.

[28] CRASTO, Nieves. Class Imbalance in Object Detection: An Experimental Diagnosis and Study of Mitigation Strategies. **arXiv preprint arXiv:2403.07113**, 2024. Disponível em: https://arxiv.org/abs/2403.07113.

APÊNDICE A – Campanha de Aumento do Dataset

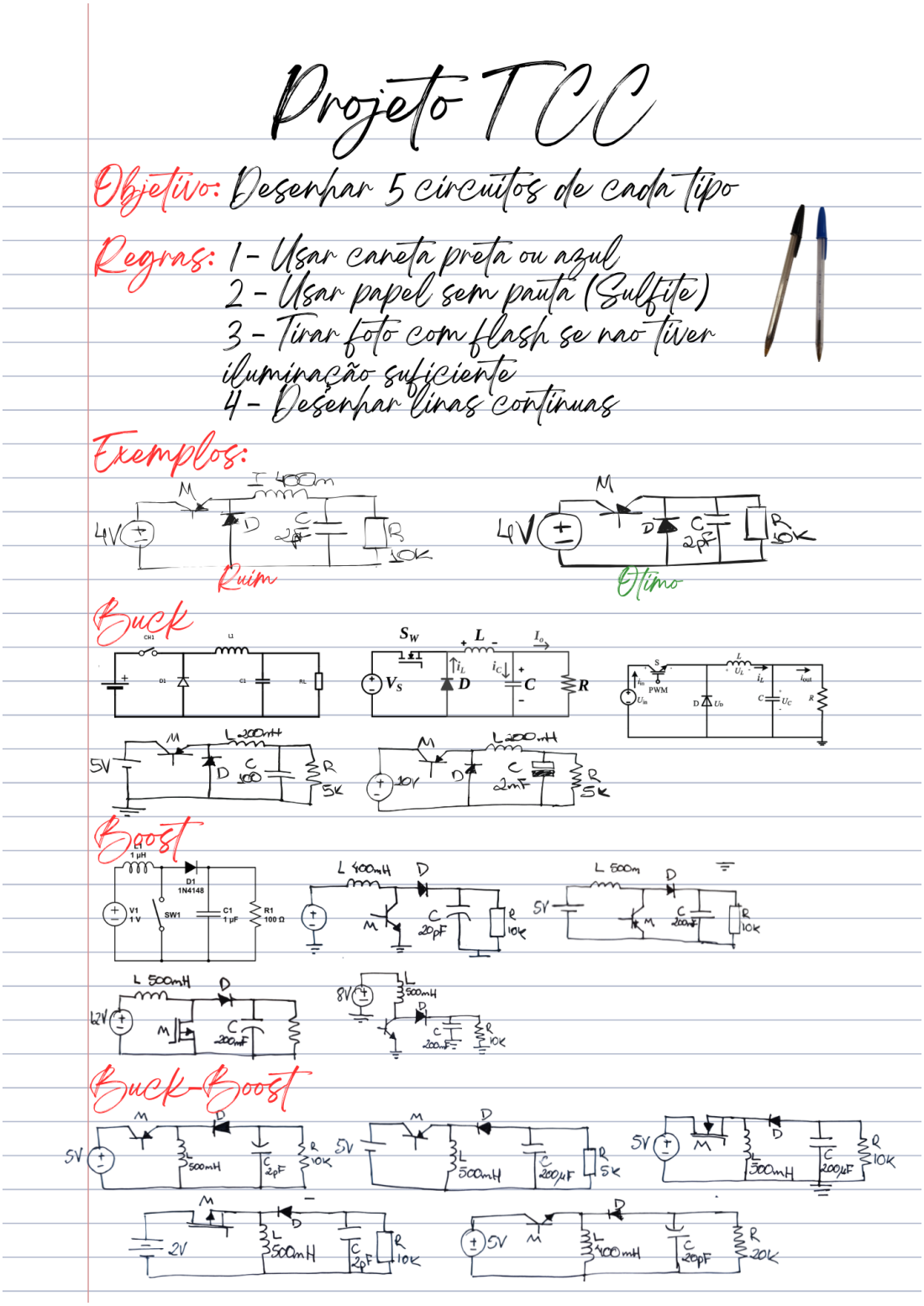


Figura 57: Campanha para aumentar a quantidade de imagens.

Fonte: Autor (2025).

APÊNDICE B – Logs de envio de imagens para API da OpenAI

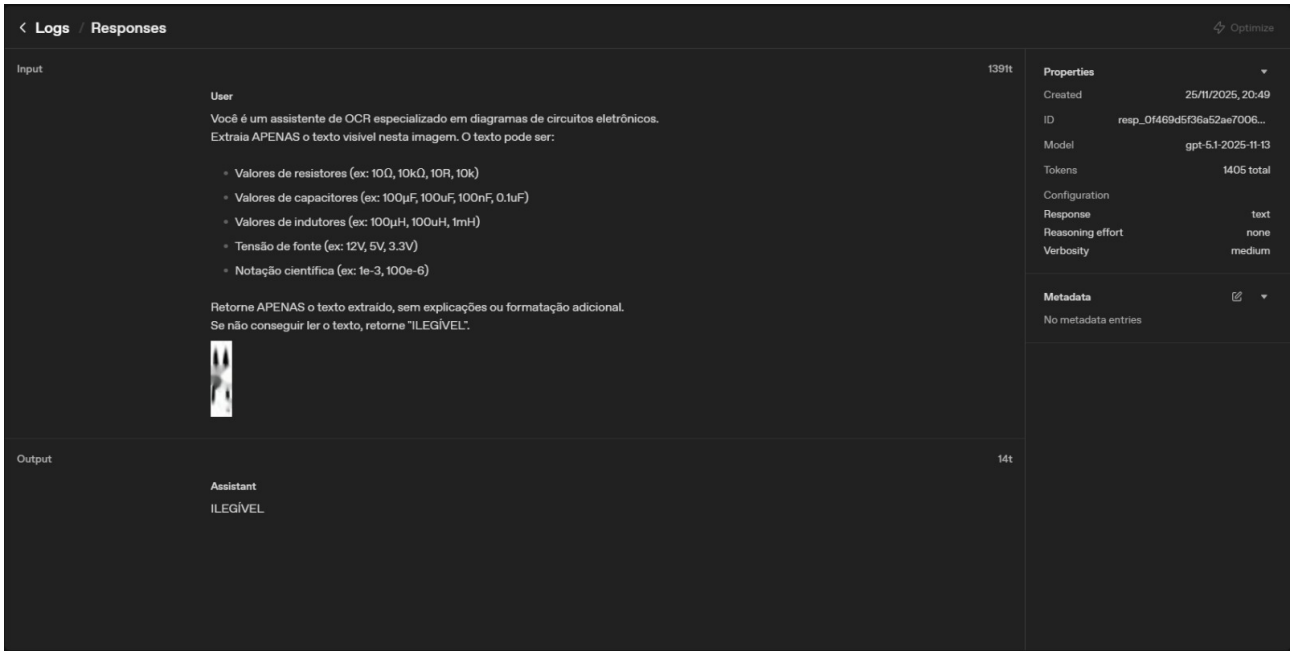


Figura 58: Envio de um crop individual para API da OpenAI.

Fonte: Autor (2025).

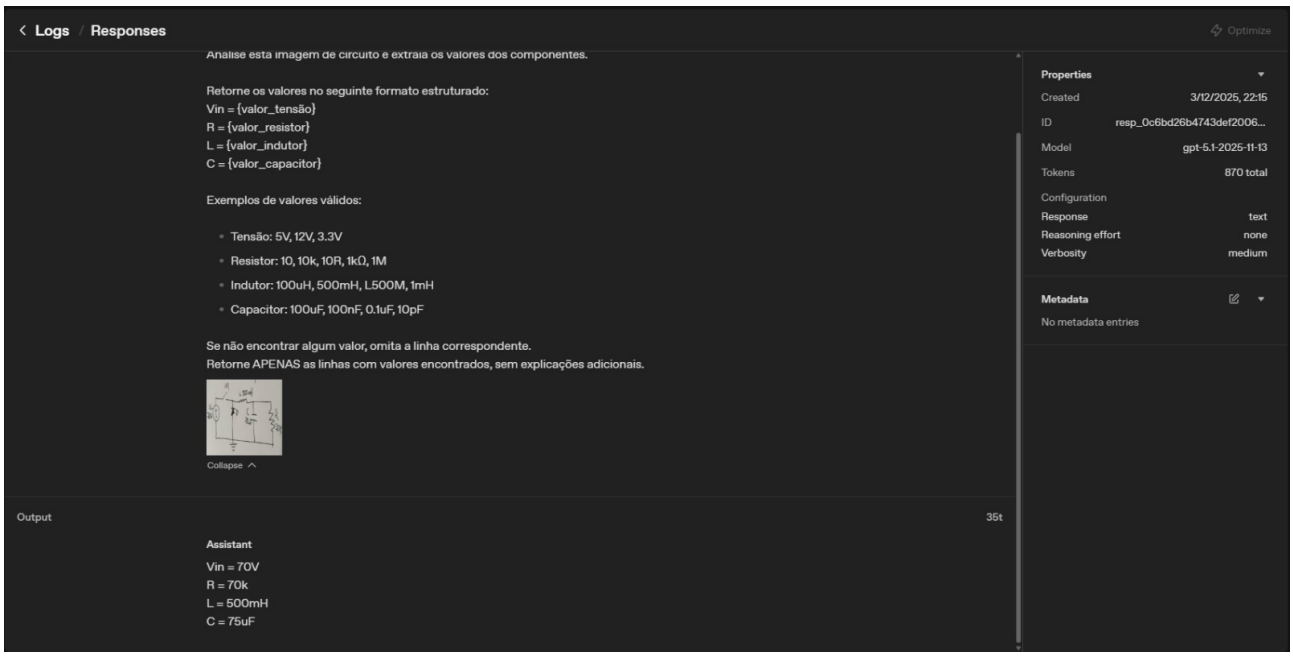


Figura 59: Envio do circuito inteiro para a API da OpenAI.

Fonte: Autor (2025).