Rekonstrukcja semantyczna grafik naukowych: od obrazu rastrowego do edytowalnego SVG i PowerPoint

dr inż. Mateusz Pomianek, wersja 2.0.1

Streszczenie

Współczesne modele generatywne tworzą coraz bardziej przekonujące ilustracje naukowe, lecz ich rezultat często stanowi jedynie obraz rastrowy. Dla czytelnika może on wyglądać jak gotowy diagram, natomiast dla autora publikacji pozostaje zbiorem pikseli: nie można niezależnie edytować napisów, przesuwać węzłów, zmieniać kierunku połączeń ani konsekwentnie dostosowywać typografii. Pozornie prosta operacja konwersji PNG do SVG okazuje się zatem problemem rekonstrukcji informacji, a nie tylko zmiany formatu pliku. Niniejszy przegląd wyjaśnia różnicę między trasowaniem konturów (*bitmap tracing*), rekonstrukcją prymitywów geometrycznych i rekonstrukcją semantyczną (*semantic reconstruction*). Omawia klasyczne wektoryzatory oraz nowsze modele wizyjno-językowe VFIG, StarVector i RefineSVG, a następnie przedstawia architekturę łączącą rozpoznawanie tekstu (OCR), widzenie komputerowe (CV), rozpoznawanie relacji przez VLM i typowany graf sceny (*scene graph*). Szczególną uwagę poświęcono zachowaniu dokładnej treści napisów, poprawnej orientacji strzałek, odtwarzaniu równań i różnicy między edytowalnym SVG a natywnym dokumentem PPTX. Zaproponowano zestaw miar jakości, procedurę porównania metod na wspólnym zbiorze testowym i zasady kontroli błędów mogących zmieniać znaczenie informacji naukowej. Wniosek zasadniczy brzmi: najpewniejszym rozwiązaniem nie jest automatyczne trasowanie wszystkich pikseli, lecz rozpoznanie obiektów i ich relacji, a następnie deterministyczne wygenerowanie odpowiednich formatów wynikowych.

Słowa kluczowe: grafika wektorowa, SVG, wektoryzacja, multimodalne modele językowe, rekonstrukcja semantyczna, graf sceny, OCR, VFIG, StarVector, RefineSVG, PowerPoint, walidacja diagramów naukowych.

1. Wprowadzenie: dlaczego zmiana formatu nie wystarcza

Schemat neuronowy, architektura systemu informatycznego, wykres pomiarowy czy rysunek techniczny mają dwie warstwy. Pierwsza jest widzialna: barwy, linie, prostokąty, strzałki i napisy. Druga jest logiczna: określa, który obiekt reprezentuje neuron, które połączenie jest pobudzające, jaki tekst stanowi etykietę osi i które figury należą do jednej grupy. W obrazie PNG obie warstwy są sprowadzone do wartości pikseli. W natywnie utworzonym diagramie mogą pozostać oddzielnymi, edytowalnymi obiektami.

Wyobraźmy sobie ilustrację przedstawiającą połączenie dwóch obszarów mózgu. Napis „kora przedczołowa” może być czytelny, a strzałka idealnie narysowana. Klasyczny wektoryzator potrafi zastąpić kontur liter krzywymi Béziera i dzięki temu zapewnić płynne powiększanie. Jednak autor nadal nie może zmienić treści podpisu w zwykłym polu tekstowym. Co istotniejsze, program nie wie, że grot strzałki powinien pozostać połączony z konkretnym węzłem podczas przestawiania elementów. Otrzymujemy **wektorową kopię wyglądu**, a nie **odzyskany diagram**.

Rozróżnienie to nabiera znaczenia w publikacjach naukowych. Zamiana „10 µM” na „10 mM”, pominięcie znaku minus, przestawienie indeksu dolnego albo odwrócenie strzałki może całkowicie zmienić interpretację doświadczenia. Co więcej, błąd bywa niezauważalny w ogólnych wskaźnikach

podobieństwa obrazu. Dla grafiki naukowej jakość nie może więc oznaczać wyłącznie podobieństwa wizualnego. Musi obejmować dokładność treści, topologię relacji i możliwość późniejszej edycji.

Zainteresowanie takim podejściem wzrosło wraz z rozwojem modeli wizyjno-językowych (*vision-language models*, VLM). VFIG rekonstruuje złożone figury jako SVG, wykorzystując uczenie od prostych obiektów do struktur złożonych, StarVector generuje kod SVG na podstawie obrazu i tekstu, natomiast RefineSVG stosuje informację zwrotną po renderowaniu. Są to ważne osiągnięcia, ale samo generowanie SVG nie gwarantuje poprawności merytorycznej ani kompatybilności z edycją w PowerPoint [1]–[3].

Celem artykułu jest zatem wyjaśnienie: **co dokładnie powinno zostać odtworzone, dlaczego obecne podejścia zawodzą, które metody sprawdzają się dla określonych elementów i jak udowodnić, że wynik zachował pierwotną informację naukową**. Rozważania koncentrują się na schematach blokowych, diagramach neurobiologicznych, elektronice, infografikach, rysunkach konstrukcyjnych i slajdach dydaktycznych.

2. Trzy znaczenia „konwersji PNG do SVG”

2.1. Trasowanie pikseli: wektorowa reprezentacja wyglądu

Najstarszy i najłatwiej automatyzowany wariant to trasowanie obrazu rastrowego (*raster-to-vector tracing*). Program wyszukuje granice barw lub obszarów, a następnie przybliża je odcinkami, wielokątami i krzywymi. Dobrze działa przy logo, ikonach, sylwetkach, prostych rysunkach liniowych i jednorodnych obszarach koloru. Wynik skaluje się niezależnie od rozdzielczości, a jego krzywe można modyfikować w edytorze graficznym.

Nie oznacza to jednak odzyskania struktury. Litera „A” może stać się zbiorem kilku zamkniętych ścieżek `<path>`, a napis „Neuron A” zespołem kilkunastu niezależnych konturów. Pojedynczy prostokąt, którego wierzchołki w rastrze są lekko rozmyte, może zostać opisany ścieżką o dziesiątkach punktów zamiast elementem `<rect>`. Taka reprezentacja zachowuje wygląd, lecz utrudnia zmianę tekstu, formatowanie i dalsze użycie w dokumentach naukowych.

Klasycznymi narzędziami są Potrace i VTracer. Pierwszy jest szczególnie przydatny do monochromatycznych konturów; drugi obsługuje kolor i bardziej złożone obszary. Współczesne narzędzia komercyjne, w tym Adobe Illustrator Image Trace i Vectorizer.AI, oferują lepsze dopasowanie prymitywów i optymalizację geometrii, ale również nie należy utożsamiać ich automatycznie z rozpoznawaniem tekstu lub logicznych połączeń [4]–[7].

2.2. Rekonstrukcja prymitywów: odzyskiwanie intencji geometrycznej

Kolejny poziom polega na rozpoznaniu, że pewna grupa pikseli przedstawia prostokąt, elipsę, okrąg, odcinek czy łuk. Różnica jest zasadnicza. Edytowalny element `<rect>` można powiększać bez deformowania przeciwległych krawędzi; elipsa ma środek i promienie, a odcinek dwa końce. Złożony kontur po trasowaniu nie daje tak prostego modelu edycji.

Rekonstrukcja geometryczna może wykorzystywać wykrywanie krawędzi, transformację Hougha, aproksymację konturów, segmentację instancji i dopasowanie modeli parametrycznych. Ograniczeniem pozostaje niejednoznaczność. Ta sama linia może być osią wykresu, krawędzią figury, strzałką albo dekoracją. Jej znaczenie wymaga uwzględnienia sąsiednich obiektów i kontekstu całej ilustracji.

2.3. Rekonstrukcja semantyczna: odzyskanie obiektów i relacji

Rekonstrukcja semantyczna (*semantic reconstruction*) próbuje odpowiedzieć nie tylko na pytanie, jak obiekt wygląda, lecz również **czym jest i do czego jest połączony**. Wynikiem może być lista węzłów, napisów, łączników, grup i relacji źródło–cel. Jeżeli podczas edycji przesuniemy węzeł B, prawidłowo zrekonstruowany łącznik A→B powinien podążyć za nim. Jeżeli zmienimy styl typograficzny, napis pozostanie tym samym ciągiem znaków.

Taką ideę realizuje m.in. system Draw with Thought, który odtwarza diagramy jako semantyczny XML środowiska mxGraph, zamiast traktować PNG jedynie jako zbiór konturów [8]. To ważna wskazówka metodologiczna: **SVG jest bardzo dobrym formatem prezentacyjnym, lecz sam w sobie nie jest pełną, przenośną reprezentacją znaczenia diagramu**. Związki między obiektami powinny być utrzymywane w niezależnej strukturze danych.



Rys. 1. Przejście od pikseli przez trasowanie konturów do edytowalnego modelu obiektowego. Przykład koncepcyjny: przedstawione kontury i obiekty mają charakter wyłącznie schematyczny.

3. Czym w istocie jest edytowalny SVG

3.1. Edycja pliku, obiektu i znaczenia to różne wymagania

Sformułowanie „edytowalny SVG" bywa używane w co najmniej czterech znaczeniach. Plik może być: (1) skalowalny bez utraty jakości; (2) złożony z kształtów, których węzły można przesuwać; (3) wyposażony w prawdziwe pola tekstowe i prymitywy geometryczne; (4) oparty na modelu semantycznym, który zachowuje powiązania po zmianie układu. Są to coraz silniejsze własności. Spełnienie pierwszej nie pociąga za sobą spełnienia pozostałych.

W standardzie SVG 2 tekst jest reprezentowany elementami `<text>` i `<tspan>`, natomiast obrazy rastrowe mogą być osadzane w `<image>`. Samo rozszerzenie `.svg` nie świadczy więc o odzyskaniu struktury: plik zawierający wyłącznie `<image href="...">` jest w istocie bitmapą umieszczoną w opakowaniu XML. Podobnie tysiące ścieżek mogą przedstawiać napisy, choć żadnego semantycznego napisu już nie ma [9].

Minimalny przykład prawdziwych elementów edytowalnych jest następujący:

```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 640 240">
  <g id="group-neuron-a">
    <rect id="neuron-a" x="40" y="70" width="200" height="90"/>
    <text id="label-neuron-a" x="60" y="120">Neuron α</text>
  </g>
  <line id="edge-a-b" x1="240" y1="115" x2="410" y2="115"
        marker-end="url(#arrowhead)"/>
</svg>
```

Można tu niezależnie zmienić geometrię, napis i styl. Nie można jednak z samej obecności `<line>` wywnioskować, że linia łączy dwa logiczne węzły. Dopiero metadane `source_id`, `target_id` i `direction` umożliwiają bezpieczne przeliczanie topologii diagramu po edycji.

3.2. Identyfikatory, grupy, warstwy i ślad pochodzenia

Profesjonalna grafika powinna przechowywać stabilne identyfikatory obiektów, logiczne grupy i uporządkowane warstwy. Osobno warto zapisywać pochodzenie informacji (*provenance*): czy napis rozpoznał OCR, czy figurę dopasował algorytm geometryczny, a relację zasugerował VLM. Ta ostatnia informacja zwykle wykracza poza standardowe możliwości samodzielnego SVG i uzasadnia pośredni graf sceny.

Przy projektowaniu należy rozdzielić wygląd od treści. Styl tekstu może być niepewny, jeśli oryginalny font nie jest dostępny, ale jego znaki muszą pozostać dokładne. Odwrotnie, wierny kolor i kontur nie są wystarczające, jeżeli zmieniono etykietę, znak nierówności lub kierunek połączenia. W dokumentacji naukowej poprawność treści ma zwykle wyższy priorytet niż dokładność dekoracji.

3.3. Unicode, typografia i matematyka

Rozpoznawanie etykiet na rysunkach technicznych wymaga precyzyjnej obsługi Unicode. Greckie ν i łacińskie v mogą być podobne wizualnie, lecz nie są tym samym znakiem. Podobnie µM i mM oznaczają zupełnie inne stężenia, a znak − nie powinien być automatycznie zastępowany dywizem. Szczególną uwagę należy poświęcić indeksom, wykładnikom, stopniom, strzałkom chemicznym i symbolom jednostek SI.

Zalecaną praktyką jest zachowanie dosłownie rozpoznanej treści, a wszelką normalizację przeprowadzanie jako osobną, jawnie kontrolowaną operację. Dla równań stosuje się rozpoznawanie matematyczne i reprezentację LaTeX/MathML, ale wynik wymaga weryfikacji matematycznej. Zamiana pojedynczego symbolu może bowiem przejść niezauważona przez ocenę podobieństwa obrazów, a jednocześnie odwrócić sens całego wzoru.

4. Dlaczego automatyczna rekonstrukcja jest trudna

4.1. Niejednoznaczność mapowania obraz–struktura

Odtwarzanie struktury z bitmapy jest zadaniem odwrotnym (*inverse graphics problem*). Wiele różnych plików SVG może po renderowaniu prowadzić do niemal identycznego rastra. Prostokąt z zaokrąglonymi rogami da się przedstawić jako `<rect rx="...">`, złożoną ścieżkę Béziera albo niewielką bitmapę. Podobnie napis może być tekstem, ścieżką lub osadzonym obrazkiem. Ze zbioru pikseli nie da się jednoznacznie odtworzyć decyzji autora, jeśli nie zastosuje się dodatkowych założeń o strukturze.

Klasyczne trasowanie optymalizuje przede wszystkim podobieństwo barw i konturów. Modele VLM wnoszą informację o typowych układach obiektów, ale wraz z nią pojawia się nowy rodzaj ryzyka: mogą dopowiadać elementy, które są prawdopodobne semantycznie, lecz nieobecne w źródle. W schemacie neuronowym brakująca strzałka nie powinna zostać „naprawiona” tylko dlatego, że model zna typowe połączenia anatomiczne. Zadaniem jest wierna rekonstrukcja, a nie twórcza korekta.

4.2. Utrata informacji na etapie rasteryzacji

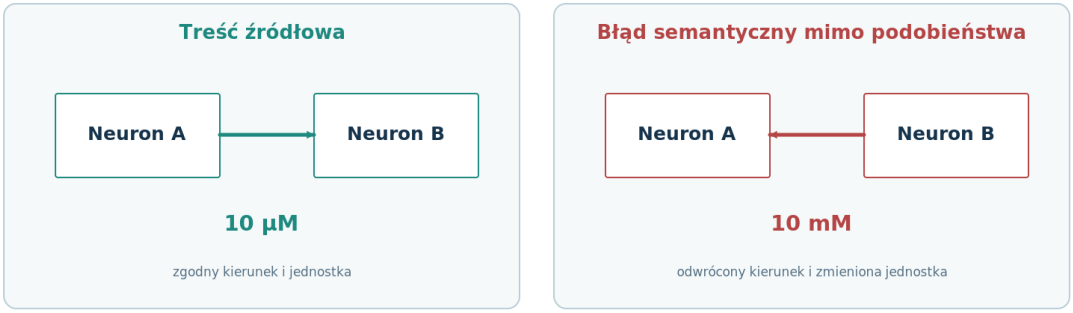
Rasteryzacja redukuje rozdzielczość przestrzenną, scala warstwy, zaciera kontury i usuwa identyfikatory. Najbardziej podatne na degradację są małe litery, indeksy, cienkie linie, groty strzałek oraz linie o bardzo podobnych barwach. Wraz ze spadkiem rozdzielczości rośnie liczba przypadków, w których kilka

różnych interpretacji jest zgodnych z tym samym obrazem. Nie istnieje ogólna gwarancja odzyskania informacji, której faktycznie nie ma już w sygnale.

Dlatego kontrola jakości powinna umożliwiać uczciwą odpowiedź „element niepewny". Takie oznaczenie jest lepsze niż przedstawienie zgadywanego napisu jako prawdziwego. Przy małych obrazach i mikrotekście dopuszczalne jest zatrzymanie procesu i skierowanie wybranych fragmentów do weryfikacji przez człowieka.

4.3. Konflikt między wyglądem, a znaczeniem

Jeżeli celem optymalizacji jest wyłącznie zgodność pikselowa, algorytm może poprawić wynik przez dodanie wielu drobnych ścieżek, nie przywracając sensu grafiki. Może także odtworzyć ogólną sylwetkę strzałki, ale nie zachować jej właściwej orientacji. Badania VFIG wskazują na potrzebę oceny typów obiektów, położenia, układu i połączeń zamiast samej rekonstrukcji wyglądu [1].



Rys. 2. Syntetyczny przykład dwóch krytycznych błędów: odwrócenia połączenia oraz zamiany jednostki µM na mM. Oryginalny sens naukowy został zmieniony mimo zachowania podobnego układu graficznego.

5. Od klasycznych algorytmów do modeli generujących SVG

5.1. Potrace, VTracer i zaawansowane trasowanie

Potrace jest sprawdzonym rozwiązaniem do trasowania bitmap, szczególnie przy grafice monochromatycznej. VTracer obsługuje kolorowe obrazy i integrację przez narzędzia wiersza poleceń lub biblioteki programistyczne. ImageTracerJS dostarcza podobne możliwości w środowisku JavaScript. Zaletami tej rodziny metod są szybkość, powtarzalność i możliwość użycia bez dużego modelu GPU; wadami — ograniczone rozumienie tekstu, geometrii i relacji [4], [5], [10].

Adobe Illustrator Image Trace i Vectorizer.AI rozszerzają tę rodzinę o bardziej zaawansowane dopasowanie kształtów i upraszczanie krzywych. Takie narzędzia mogą być bardzo użyteczne przy konturach anatomicznych, ikonach, nieregularnych obszarach i bogatej grafice dekoracyjnej. Nie należy jednak oczekiwać, że automatycznie utworzą poprawne, semantycznie połączone węzły diagramu i wszystkie napisy jako edytowalny tekst [6], [7].

5.2. StarVector: SVG jako język generowany przez model

StarVector traktuje kod SVG jako język wyjściowy modelu wizyjno-językowego. Zamiast kolejno wyszukiwać kontury i aproksymować je krzywymi, model może bezpośrednio wybierać prymitywy: linię, elipsę, wielokąt, tekst czy gradient. Autorzy przedstawili zbiór SVG-Stack oraz benchmark SVG-Bench; badania opublikowano na CVPR 2025 [2]. Takie podejście może prowadzić do bardziej zwartego i znaczeniowo bogatszego SVG niż klasyczne trasowanie.

Nie usuwa to problemu poprawności. Model autoregresyjny może pomylić współrzędne, domknąć niewłaściwą ścieżkę, zmienić etykietę albo przerwać generowanie. W dłuższym kodzie drobne błędy

mogą kumulować się i ujawniać dopiero podczas renderowania. Dlatego plik wynikowy musi przejść walidację składni, renderowanie oraz porównanie z wejściem.

5.3. VFIG: modele wyspecjalizowane w figurach naukowych

VFIG z 2026 r. jest szczególnie istotny dla dokumentacji naukowej, ponieważ został zaprojektowany do odtwarzania złożonych figur, a nie wyłącznie ikon i prostych obrazów. Zbiór VFIG-Data obejmuje 66 579 par figura–SVG, w tym 60 034 złożone diagramy i 6545 przykładów kształtów i strzałek. Publikacja opisuje uczenie od prostych prymitywów do pełnych struktur (*coarse-to-fine curriculum*) oraz późniejszą optymalizację z uwzględnieniem zgodności wizualnej i topologicznej [1], [11].

Autorzy raportują wynik VLM-Judge równy 0,829 dla wariantu SFT+RL na VFIG-Bench. Jest to wartość z ich własnego protokołu ewaluacji, a nie uniwersalny wskaźnik jakości każdej figury. W oryginalnym raporcie źródłowym przywołano również wartość 0,548 dla StarVector-8B w tej konfiguracji porównawczej. Liczby te wskazują potencjał specjalizacji domenowej, ale nie zastępują niezależnego testu na grafice elektronicznej, neuronaukowej czy rysunkach z bardzo małą czcionką.

Dostępny wariant VFIG-4B może stanowić punkt wyjścia do lokalnego prototypu. Model ma około czterech miliardów parametrów, lecz sam rozmiar wag nie określa pełnego zapotrzebowania na pamięć GPU. Praca w BF16 wymaga około 8 GB na same wagi, a dodatkowy budżet jest potrzebny dla toru wizyjnego, aktywacji i buforów. W źródle zaproponowano 12–16 GB VRAM jako orientacyjne założenie inżynieryjne; nie należy tego mylić z opublikowaną gwarancją sprzętową [12].

5.4. RefineSVG: model, który ogląda rezultat własnej pracy

RefineSVG rozwija architekturę generacji ze sprzężeniem zwrotnym (*render–observe–correct*). Model najpierw tworzy kod SVG, następnie zewnętrzny renderer zamienia go na obraz, a powstałą różnicę względem wejścia przedstawia jako sygnał do korekty. Autorzy opisują mapę różnic (*Diff-Map*) oraz trening łączący nadzorowane dostrajanie i uczenie ze wzmocnieniem. Wstępny wynik jest więc weryfikowany, zamiast być traktowany jako ostateczny [3].

Rozwiązanie to jest szczególnie ważne przy długim kodzie SVG: pozycje obiektów i połączeń mogą stopniowo odchylać się od źródła. Samo porównanie renderów nie wystarczy jednak do rozstrzygnięcia, czy napisy oraz kierunki połączeń są poprawne. Pętla korekcyjna powinna korzystać również z niezależnego OCR i walidatora grafu relacji, a liczba iteracji musi być ograniczona.

5.5. Uzupełniające kierunki badań

W literaturze pojawiają się rozwiązania takie jak LLM4SVG, OmniSVG czy AutoFigure-Edit. Ich znaczenie polega przede wszystkim na szukaniu bardziej odpowiedniej reprezentacji kodu SVG, treningu na strukturach wektorowych i zwiększaniu zakresu edytowalnych obiektów [13]–[15]. Narzędzia natywnie generujące wektory, np. Recraft, rozwiązują nieco inny problem: są atrakcyjne, gdy figurę można stworzyć od nowa na podstawie opisu, ale nie należy utożsamiać ich z gwarancją wiernej rekonstrukcji konkretnego obrazu źródłowego [16].

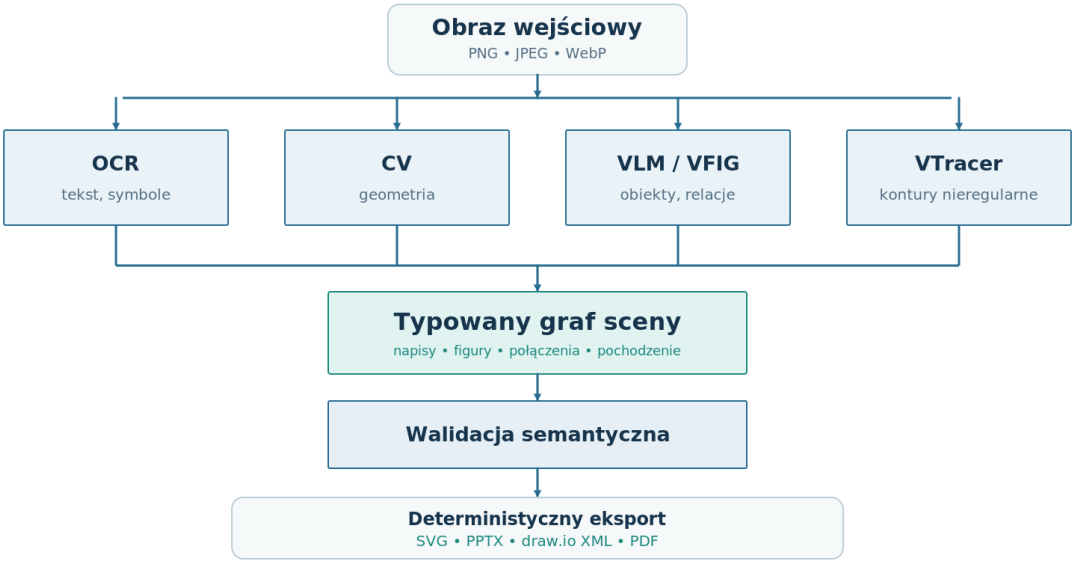
Wniosek: modele multimodalne są cennym mechanizmem interpretacji, lecz w zastosowaniach naukowych nie powinny samodzielnie rozstrzygać o ostatecznej treści tekstu ani wykonywać jednocześnie wszystkich operacji geometrycznych, semantycznych i formatowania. Bardziej kontrolowalna jest architektura, w której każdy z tych problemów ma osobny moduł.

6. Architektura hybrydowa: graf sceny jako centralna reprezentacja

6.1. Podział zadań między wyspecjalizowane moduły

Najbardziej obiecującym rozwiązaniem jest hybrydowy kompilator grafiki naukowej (*semantic graphics compiler*). Nie zaczyna się on od wygenerowania pliku SVG, lecz od odczytania obrazu i przekształcenia go w model sceny. Moduł OCR identyfikuje napisy i symbole, algorytmy CV mierzą położenie figur, VLM interpretuje obiekty i połączenia, a klasyczny wektoryzator jest stosowany jedynie do obszarów trudnych do opisania prostymi prymitywami.

Podział ten ogranicza zarówno błędy generatywne, jak i nadmierne komplikowanie grafiki. Rozpoznany prostokąt powstaje jako <rect>, podpis jako <text>, a relacja jako łącznik z jednoznacznie określonym kierunkiem. Dopiero poza warstwą interpretacji deterministyczny generator tworzy SVG, PPTX, XML draw.io czy wariant do LaTeX.



Rys. 3. Autorska architektura hybrydowa: niezależne kanały rozpoznawania są scalane w grafie sceny, poddawane walidacji i eksportowane do formatów docelowych. Schemat jest propozycją projektową, a nie rezultatem pomiaru.

6.2. Jawny graf sceny (*scene graph*)

Graf sceny powinien zawierać stabilne identyfikatory, geometrię, tekst, kolejność warstw, relacje, źródło rozpoznania, miarę niepewności oraz status weryfikacji. Przykładowy zapis może wyglądać tak:

```
{
 "nodes": [
  {"id": "n1", "type": "rect", "bbox": [0.10, 0.20, 0.24, 0.12]},
  {"id": "n2", "type": "rect", "bbox": [0.60, 0.20, 0.24, 0.12]}
 ],
 "texts": [
  {"id": "t1", "parent": "n1", "value": "Neuron α",
   "source": "OCR", "verified": true}
 ],
 "edges": [
  {"id": "e1", "source_id": "n1", "target_id": "n2",
   "direction": "forward", "verified": false}
 ]
}
```

Współrzędne znormalizowane do zakresu 0–1 zwiększają niezależność od rozdzielczości wejściowej. Model relacji powinien odróżniać brak grotu strzałki od informacji, że grot jest nieczytelny. Niepewne połączenia nie mogą być automatycznie uznawane za prawdziwe tylko dlatego, że diagram wygląda dzięki nim spójniej.

6.3. Hierarchia dowodów i rozstrzyganie konfliktów

W raporcie źródłowym zaproponowano praktyczną hierarchię: jawnie widoczna treść rozpoznana przez OCR, następnie pomiary geometrii, dalej relacje potwierdzone zarówno przez VLM, jak i detektor strzałek, a na końcu inferencja samego VLM. Jest to sensowna polityka projektowa, ale nie sztywne prawo: OCR również popełnia błędy, szczególnie przy małej rozdzielczości, grece i indeksach. Dlatego rozbieżność między modułami powinna skutkować oznaczeniem konfliktu oraz kontrolą fragmentu obrazu.

Przykładowo jeśli OCR odczytuje „0,1 mV", a VLM „0,1 µV", najbezpieczniejszy system nie wybiera automatycznie jednej wersji. Przedstawia obie hipotezy, wycinek źródła i oznacza etykietę jako nierozstrzygniętą. Analogicznie brak pewności co do orientacji strzałki wymaga weryfikacji grotu, a nie odwołania do typowej wiedzy o analizowanej dziedzinie.

6.4. Deterministyczne generowanie formatów wynikowych

Po zaakceptowaniu grafu sceny etap eksportu powinien być deterministyczny. Te same obiekty i parametry mają prowadzić do tych samych instrukcji rysowania. Można dzięki temu osobno testować parser obrazu i osobno generator SVG lub PPTX. Dla SVG należy kontrolować poprawność XML, zgodność z namespace, unikatowość identyfikatorów, obecność rzeczywistych napisów oraz brak niezamierzonych osadzonych bitmap.

W PowerPoint te same dane powinny utworzyć natywne pola tekstowe, figury i łączniki. W LaTeX można pozostawić tekst i wzory do złożenia przez silnik dokumentu, zachowując geometrię w postaci wektorowej. Taki podział umożliwia późniejszą zmianę formatu bez ponownego uruchamiania OCR i modelu multimodalnego.

7. PowerPoint, draw.io i LaTeX: różne znaczenia edytowalności

7.1. Dlaczego poprawny SVG może pozostać słabo edytowalny w PowerPoint

Microsoft 365 obsługuje wstawianie SVG, ale operacja „Konwertuj na kształt" (*Convert to Shape*) nie gwarantuje zachowania tekstu jako zwykłych pól tekstowych dla każdego wariantu SVG i każdej platformy. Dokumentacja Microsoft wskazuje ograniczenia zakresu dostępności funkcji, a zgłaszane problemy obejmują m.in. przemieszczenie znaków po rozgrupowaniu na macOS [17], [18].

Należy odróżniać import SVG jako skalowalnego obrazu od otrzymania rzeczywistego slajdu zawierającego osobne obiekty. Jeżeli użytkownik chce przestawiać elementy, edytować napisy i aktualizować łączniki, bezpośrednie utworzenie natywnego PPTX jest zwykle pewniejszym rozwiązaniem niż import SVG, nawet gdy ten drugi zawiera poprawne <text>.

7.2. Bezpośrednie generowanie natywnego PPTX

Biblioteka python-pptx udostępnia pola tekstowe, kształty, własności fontów i obiekty łączników; alternatywą jest PptxGenJS. W praktyce tworzenie slajdów z grafu sceny pozwala zachować prawdziwe elementy Office, a następnie testować ich liczbę i zawartość po rozpakowaniu dokumentu PPTX jako archiwum ZIP [19], [20].

Pewną trudnością pozostaje zakotwiczanie łączników w punktach połączeń figur i wierne odwzorowanie niestandardowych efektów SVG. Dla specjalistycznych krzywych konieczne mogą być obiekty swobodne (*freeform*) albo świadome pozostawienie wybranych regionów jako ilustracji wektorowej. Proces powinien więc maksymalizować edytowalność tam, gdzie ma ona rzeczywistą wartość, a nie wymuszać semantyzację każdego ozdobnego fragmentu.

7.3. Diagramy draw.io: semantyczny format źródłowy

Import gotowego SVG do draw.io nie przywraca automatycznie wewnętrznego modelu połączeń. Dokumentacja narzędzia wyraźnie rozróżnia graficzny SVG od semantycznej struktury diagramu, którą można zachować m.in. w formacie mxGraph XML [21]. To dodatkowy argument za przechowywaniem grafu sceny niezależnie od postaci renderowanej.

7.4. Skład publikacji w LaTeX

W dokumentach LaTeX i Typst ważne jest nie tylko zachowanie wektorowej geometrii, lecz także zgodność fontów, matematyki i stylu wydawniczego. Inkscape i pakiet `svg` dla LaTeX wspierają przepływ pracy, w którym geometria może być eksportowana do PDF, a napisy składane osobno [22]. Podobną zasadę warto stosować przy własnym generatorze: elementy tekstowe powinny mieć źródłową treść, natomiast silnik składu odpowiadać za końcowe łamanie, indeksy i fonty.

Wymaganie	Zwykły SVG	Natywny PPTX	draw.io XML	PDF + tekst LaTeX
Skalowanie bez strat	tak	tak dla figur	tak	tak
Edycja napisów	jeśli <text>	tak	tak	w źródle LaTeX
Logiczne łączniki	tylko z metadanymi	możliwe	tak	nie
Zachowanie układu	zwykle dobre	zależne od renderera	dobre	bardzo dobre
Edycja w środowisku docelowym	edytor SVG	PowerPoint	draw.io	edytor źródła

Tab. 1. Zestawienie właściwości formatów. „możliwe” oznacza funkcję zależną od sposobu wygenerowania dokumentu, a nie gwarancję każdego pliku.

8. Kontrola jakości: od pikseli do poprawności znaczenia

8.1. Dlaczego nie ma jednej wystarczającej metryki

Ocena odtworzonej grafiki musi odzwierciedlać cel edycji. SSIM i miary odległości percepcyjnej, takie jak LPIPS, opisują podobieństwo obrazu. Nie informują jednak, czy pojedynczy wyraz został zapisany poprawnie ani czy strzałka prowadzi do odpowiedniego obiektu. W praktyce należy zatem osobno oceniać wierność wizualną, tekstową, geometryczno-topologiczną oraz edytowalność. Takie podejście jest spójne z kierunkiem testów strukturalnych VFIG-Bench [1], [23].



Rys. 4. Cztery niezależne wymiary oceny konwersji. Skróty są nazwami metryk, a przedstawiony układ nie zakłada równych wag.

8.2. Tekst: dokładność znaków, jednostek i symboli

Standardową miarą błędów rozpoznawania napisów jest współczynnik błędów znakowych (*character error rate*, CER):

$$\mathrm{CER} = \frac{S + D + I}{N},$$

gdzie S oznacza liczbę substytucji, D usunięć, I wstawień, a N liczbę znaków tekstu wzorcowego. Jako uzupełnienie stosuje się WER, czyli współczynnik błędów słownych (*word error rate*), oraz osobne testy znaków Unicode, indeksów i symboli jednostek. W grafice naukowej należy dodatkowo raportować błędy krytyczne, np. zmianę jednostki wielkości fizycznej, odwrócenie znaku lub pomylenie greckiej litery z łacińską. Przydatna jest także miara udziału tekstu zachowanego jako prawdziwe elementy edytowalne. Jej prosta wersja może mieć postać:

$$E_{\mathrm{text}} = \frac{N_{\mathrm{editable}}}{N_{\mathrm{reference}}},$$

gdzie licznik zlicza znaki przechowywane jako tekst, a mianownik wszystkie znaki, które zgodnie z wzorcem powinny zostać odtworzone. Aby uniknąć sztucznie wysokiego wyniku, nie wolno liczyć do licznika napisów błędnie rozpoznanych: wskaźnik edytowalności i poprawności tekstu należy prezentować łącznie.

8.3. Geometria: dokładność obiektów i złożoność ścieżek

Geometrię ocenia się za pomocą precyzji i czułości rozpoznawania typów figur, współczynnika nakładania obszarów (*intersection over union*, IoU), błędu położenia środka, błędu wymiarów i odchylenia kątowego odcinków. Warto także mierzyć liczbę punktów kontrolnych oraz stosunek liczby ścieżek do liczby prostych prymitywów. Dwa pliki mogą wyglądać identycznie, lecz jeden zawierać dziesięć logicznych obiektów, a drugi kilka tysięcy fragmentów krzywych.

Nie należy jednak karać za każdą ścieżkę. Kontur organiczny może wymagać wielu punktów, natomiast zastąpienie go prostokątem byłoby deformacją obrazu. Interpretacja złożoności musi więc zależeć od rodzaju przedstawionego elementu, a nie wyłącznie od rozmiaru pliku.

8.4. Struktura: weryfikacja połączeń i orientacji

Dla diagramu reprezentowanego grafem $G = (V, E)$ kluczowe są zgodność węzłów, zgodność krawędzi, identyfikacja źródła i celu oraz kierunek strzałek. Można stosować F1 dla węzłów i krawędzi, dokładność źródło–cel (*source–target accuracy*) i kompletność połączeń. Przed obliczeniem metryk potrzebne jest jednoznaczne dopasowanie węzłów rekonstrukcji do elementów wzorcowych. Nie można porównywać krawędzi bez ustalenia, które obiekty są ich końcami.

Najważniejszym testem merytorycznym jest to, czy rekonstruowany graf zachowuje tę samą treść. Odwrócenie strzałki może wywołać niewielką zmianę pikseli, ale w diagramie przepływu informacji oznacza zmianę relacji przyczynowej. Powinno zatem stanowić błąd krytyczny niezależnie od wysokiego SSIM.

8.5. Edytowalność i miary złożone

Proponowany w raporcie źródłowym wskaźnik łączny ma postać:

$$E = 0{,}30T + 0{,}25P + 0{,}20C + 0{,}15G + 0{,}10L,$$

gdzie T oznacza jakość odzyskania tekstu, P udział prostych prymitywów, C poprawność łączników, G grupowanie logiczne, a L jakość warstw i identyfikatorów. Podane wagi są **propozycją metodologiczną**, nie wynikiem walidacji psychometrycznej ani standardem branżowym. Powinny zostać uzasadnione wymaganiami zastosowania i ustalone przed analizą danych. Dla neuroanatomii i elektroniki można racjonalnie przyznać większą wagę poprawności etykiet i relacji niż kosztowi edycji kolorów.

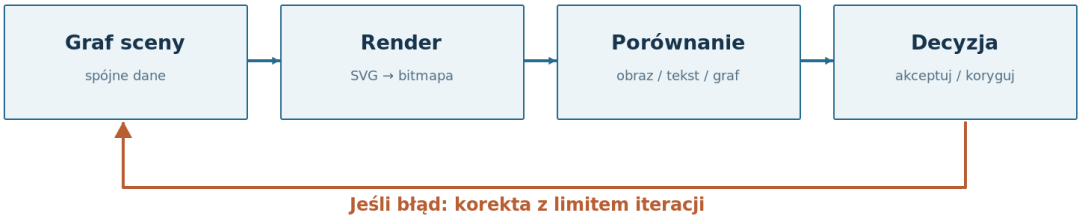
Nawet dobrze zaprojektowany wskaźnik syntetyczny nie powinien zastępować progów bezpieczeństwa. Jeżeli pojedynczy znak w nazwie jednostki jest błędny lub niepewna pozostaje orientacja połączenia, system nie powinien uznać figury za poprawną wyłącznie dlatego, że średni wynik jest wysoki.

9. Pętla walidacji i kryteria zatrzymania

9.1. Od jednorazowej generacji do kontrolowanej korekty

Prawidłowy proces wytwarzania nie kończy się na zapisie SVG. Po wygenerowaniu pliku należy go wyrenderować w niezależnym silniku, porównać z wejściem, przeprowadzić OCR na wyniku i odczytać strukturę XML. Następnie należy sprawdzić spójność napisów, geometrii i grafu relacji. Dopiero taki zestaw dowodów uzasadnia akceptację figury.

Pętla naprawcza (*bounded repair loop*) powinna otrzymywać konkretny raport różnic: tekst A rozpoznany jako Λ, błędny identyfikator celu strzałki, przesunięcie węzła o 14 px, pominięta etykieta. Naprawa musi być lokalna, ograniczona do dotkniętych obiektów, aby nie pogarszać poprawnych wcześniej fragmentów. Liczba powtórzeń oraz maksymalna dopuszczalna zmiana struktury powinny być z góry ustalone.



Rys. 5. Walidacja w pętli zamkniętej: renderowanie, porównanie z wejściem, kontrola semantyczna i ograniczona korekta. Zawiera zasadę pierwszeństwa wierności informacji nad podobieństwem pikselowym.

9.2. Co powinno automatycznie blokować akceptację

Do błędów krytycznych zalicza się zmianę jednostki, wartości, symbolu matematycznego, niezgodny cytowany tekst, brak wymaganej etykiety, zmianę kierunku strzałki oraz dodanie niewidocznego w źródle połączenia. Błędy drugorzędne to np. umiarkowana różnica odcienia lub niedokładny promień zaokrąglenia. Warto rozdzielić te klasy już w kontrakcie walidacyjnym, zamiast dopiero na etapie końcowego przeglądu.

Szczególnie niebezpieczna jest automatyczna „korekta merytoryczna” przez VLM. W odtwarzaniu dokumentu naukowego system powinien przede wszystkim wiernie przepisywać (*faithful transcription*), a nie poprawiać domniemane błędy autora. Jeżeli źródło zawiera niepoprawny opis, powinien on zostać zachowany w rekonstrukcji i ewentualnie oznaczony w osobnym raporcie uwag.

10. Pięć praktycznych sposobów wdrożenia

10.1. Wariant całkowicie otwarty i lokalny

Pipeline: przygotowanie obrazu → PaddleOCR → detekcja geometrii → VFIG-4B → VTracer dla niewyjaśnionych obszarów → graf sceny → generator SVG/PPTX → walidacja. Jest to najbardziej obiecująca konfiguracja badawcza bez opłat licencyjnych, ale wymaga zintegrowania kilku komponentów i GPU o wystarczającej pamięci. Nie jest gotowym produktem, którego przewagę eksperymentalną nad wszystkimi alternatywami już wykazano. Wartościowe elementy walidatora można uruchamiać na CPU, a inferencję VLM kierować do osobnego węzła GPU.

10.2. Ogólny model multimodalny i własny generator

Pipeline: obraz → model multimodalny → JSON zgodny ze schematem → deterministyczny kod Python → SVG/PPTX → render i kontrola. Zaletą jest elastyczność w interpretacji nietypowych diagramów. Wadami są niepewność generacji, koszt zewnętrznego API i ryzyko wnioskowania o niewidocznych szczegółach. Prompt powinien wprost zabraniać dopowiadania połączeń i korekty odczytanych napisów; prawdziwą gwarancją pozostają jednak walidatory, a nie deklaracje w instrukcji dla modelu.

10.3. Wariant hybrydowy jako architektura domyślna

Najbardziej kompletna koncepcja łączy OCR, detektory geometryczne, VLM i selektywne trasowanie z niezależnym walidatorem semantycznym. Jest najlepszym punktem wyjścia do prac rozwojowych, ponieważ rozdziela źródła błędów i pozwala niezależnie ulepszać rozpoznawanie tekstu, węzłów i relacji. Cena tej elastyczności to nakład inżynieryjny na scalanie sprzecznych detekcji i utrzymanie spójności grafu.

10.4. Wariant komercyjny wspomagany ręczną kontrolą

W pracy o dużym znaczeniu estetycznym lub przy bardziej organicznych konturach uzasadnione może być połączenie semantycznego rozpoznania z Adobe Illustrator lub Vectorizer.AI. Ręczna kontrola nie stanowi porażki automatyzacji: jest świadomym etapem zapewnienia jakości, szczególnie przy rycinach anatomicznych, gradientach i drobnych opisach. Informacje o licencjach i cenach są zmienne i należy je weryfikować przed zakupem, zamiast traktować historyczne zestawienie jako aktualny cennik.

10.5. Wariant skierowany bezpośrednio do PowerPoint

Jeżeli rzeczywistym produktem końcowym jest slajd, zalecane jest: obraz → graf sceny → natywny PPTX. Pozwala to uniknąć nieprzewidywalności konwersji <text> w importerze SVG. Ten sam graf można dodatkowo wyeksportować do SVG do publikacji, a oryginalny plik PPTX zachować jako edytowalny materiał dydaktyczny. Mechanizmy generowania tekstu i figur są dobrze udokumentowane w python-pptx i PptxGenJS [19], [20].

Konfiguracja	Główna przewaga	Ograniczenie	Najlepszy scenariusz
VTracer / Potrace	Prostota i powtarzalność	Brak semantyki	Kontury i ikony
OCR + klasyczny tracer	Zachowanie treści napisów	Słabe relacje	Proste figury z tekstem
VFIG / StarVector	Interpretacja struktury	Błędy generatywne	Złożone figury
VLM + graf sceny	Kontrolowany model danych	Wymaga integracji	Diagramy techniczne
Hybryda + walidacja	Wielowarstwowa kontrola	Koszt opracowania	Publikacje naukowe
Graf sceny → PPTX	Natywna edytowalność	Różnice w renderowaniu	Slajdy i dydaktyka

Tab. 2. Porównanie sposobów realizacji. Ocena jest jakościową syntezą właściwości architektur, a nie rankingiem opartym na wspólnym benchmarku.

11. Metodyka badawcza: jak uczciwie porównać rozwiązania

11.1. Dwie warstwy materiału testowego

Rzetelny eksperyment wymaga obrazów, dla których znamy prawdziwą strukturę, oraz obrazów rzeczywistych, których struktura jest dostępna tylko częściowo. Pierwsza warstwa powinna powstać przez utworzenie kontrolowanych diagramów bezpośrednio w SVG/PPTX, zapis grafu obiektowego, a następnie ich rasteryzację w kilku rozdzielczościach. Umożliwia to pomiar wszystkich rodzajów błędu wobec precyzyjnego wzorca (*ground truth*).

Druga warstwa obejmuje faktyczne ilustracje generowane przez modele AI i obrazy pochodzące z praktyki naukowej. Warto rozważyć publiczny VFIG-Bench, którego źródłowy raport opisuje 400 obrazów z referencyjną strukturą i osobny zbiór 198 przykładów spoza rozkładu treningowego (*out-of-distribution*, OOD) [23]. W przypadku materiałów bez wiarygodnego grafu wzorcowego część ocen musi być wykonywana przez ekspertów, z oddzielnym raportowaniem zgodności ocen.

11.2. Klasy trudności i czynniki eksperymentalne

Proponuje się osiem klas: schemat blokowy, architektura systemu AI, neuronauka i anatomia, elektronika, wielotekstowa infografika, rysunek techniczny z wymiarami, wzory matematyczne oraz mikrotekst z gęstymi strzałkami. Dla każdej klasy należy uwzględnić różne rozdzielczości, stopień zagęszczenia, liczbę napisów i rodzaj połączeń. Przykładowy plan obejmuje 20–50 figur na klasę i 2–3 rozdzielczości; jest to propozycja eksperymentu, a nie pomiar zrealizowany w niniejszym przeglądzie.

Należy oddzielić zbiór rozwojowy od testowego, zamrozić obrazy i wersje narzędzi, dokumentować parametry inferencji, liczbę wywołań API oraz ustawienia OCR i tracerów. Wszystkie konfiguracje powinny otrzymać ten sam obraz i porównywalny budżet interakcji. Nie należy sumować wartości opublikowanych przez różnych producentów w jeden ranking, ponieważ ich wyniki pochodzą z odmiennych zbiorów danych i procedur.

11.3. Ablacja i analiza kosztu

Szczególnie pouczające będą porównania: (a) sam VTracer; (b) VTracer + OCR; (c) sam VFIG; (d) VFIG + OCR; (e) VFIG + OCR + CV; (f) hybryda z walidatorem; (g) ogólny VLM + graf sceny; (h) eksport natywnego PPTX. Taka ablacja odpowiada na pytanie, który moduł rzeczywiście odpowiada za poprawę tekstu, połączeń i edytowalności.

Oprócz średniej jakości trzeba raportować rozkład wyników, najgorsze przypadki, częstość błędów krytycznych, medianę i P95 czasu przetwarzania, koszt na figurę oraz liczbę interwencji człowieka. Testy statystyczne powinny uwzględniać sparowany charakter obserwacji: te same grafiki są oceniane w różnych konfiguracjach. W analizie należy stosować przedziały ufności, rozmiary efektu i korektę wielokrotnych porównań, a nie wyłącznie pojedynczą wartość p.

11.4. Walidacja zgodności z PowerPoint

Praktyczny test edytowalności PPTX powinien obejmować: (1) otwarcie pliku w docelowej wersji Office; (2) zmianę wybranego napisu bez ingerencji w geometrię; (3) przesunięcie węzła i kontrolę łącznika; (4) ponowne zapisanie pliku; (5) inspekcję struktur XML wewnątrz PPTX; (6) renderowanie slajdu i porównanie. Należy wykonać test na środowisku Windows, a w razie wymogu wieloplatformowości również na macOS. Działające w danym edytorze rozgrupowanie SVG nie powinno być automatycznie uznawane za zachowanie semantycznych łączników.

12. Zalecana implementacja i plan rozwoju

Implementacja produkcyjna powinna mieć czytelną separację odpowiedzialności. Proponowany pakiet image2sciencevector składa się z modułów przygotowania obrazu, OCR, geometrii, analizy semantycznej, grafu sceny, scalania, eksporterów SVG/PPTX, renderowania i walidacji. Każdy moduł powinien mieć własne testy jednostkowe oraz kontrakt danych. Dodatkowy moduł benchmarkowy porównuje całą architekturę z konfiguracjami kontrolnymi.

- **Etap pierwszy** polega na opracowaniu parsera obrazów, modelu danych oraz generatora SVG/PPTX. Warto zacząć od diagramów zawierających prostokąty, strzałki i napisy, ponieważ dla nich łatwo utworzyć bezdyskusyjny wzorzec strukturalny.
- **Etap drugi** rozszerza OCR o grekę, symbole i matematykę oraz wprowadza detekcję grotów i relacji.
- **Etap trzeci** dodaje model VLM/VFIG, ale tylko jako komponent interpretacyjny, oraz mechanizm kontroli konfliktów.
- **Etap czwarty** obejmuje renderowanie i walidację regresyjną na stałym zestawie grafik, wersjonowanym razem z implementacją.

Zaletą tej architektury jest niezależność od konkretnego dostawcy modelu. VFIG może zostać zastąpiony innym modelem multimodalnym, a silnik SVG innym generatorem, bez przepisywania struktur danych i testów. To również ułatwia użycie modeli lokalnych: ciężkie wnioskowanie wizualne może działać na GPU, a deterministyczna walidacja i eksport na maszynie o znacznie skromniejszych zasobach.

Kryterium gotowości nie powinien być pojedynczy atrakcyjny przykład. System jest gotowy do zastosowania w dokumentacji naukowej, gdy na zamrożonym zestawie testowym osiąga ustalone progi zgodności tekstu i relacji, nie dopisuje niewidocznych obiektów oraz pozwala użytkownikowi wykryć i skorygować przypadki niepewne. Osobnego progu wymaga eksport do PowerPoint, ponieważ wizualnie poprawny slajd może nadal nie zawierać edytowalnych obiektów.

13. Dyskusja: ograniczenia i perspektywy

Najważniejsze ograniczenie wszystkich opisanych metod ma charakter informacyjny. Jeśli piksele są rozmyte, napis w oryginale był nieczytelny, a grot strzałki zniknął przy kompresji, nie istnieje algorytm zdolny bez dodatkowych założeń zawsze odzyskać prawdziwą wersję. Model generatywny może stworzyć przekonującą hipotezę, lecz w pracy naukowej należy ją odróżnić od odczytu potwierdzonego źródłem.

Drugim ograniczeniem jest niejednoznaczność pojęcia „edytowalny”. Grafika składająca się z kilku tysięcy ścieżek jest technicznie wektorowa, ale jej aktualizacja bywa bardziej pracochłonna niż ręczne przerysowanie. Z kolei natywny PPTX może być bardzo wygodny do edycji, ale nie zachowuje wszystkich specjalistycznych efektów SVG. Nie istnieje jeden uniwersalny format najlepszy dla wszystkich zastosowań. Racjonalne jest utrzymywanie wspólnego, semantycznego modelu źródłowego oraz wielu formatów wyjściowych.

Trzecie ograniczenie dotyczy wiarygodności publikowanych porównań. Benchmarki VFIG i StarVector są wartościowe, ale ich wyniki nie mogą zostać bezpośrednio przeniesione na dowolne obrazy naukowe. Narzędzia komercyjne i otwarte często prezentują inne metryki, a dostępne demonstracje selekcjonują przykłady. Dlatego przed wyborem technologii do wdrożenia należy przeprowadzić własny test na typach figur rzeczywiście używanych przez odbiorcę.

Najbardziej interesującym kierunkiem rozwoju jest **grafika naukowa jako kompilowalna reprezentacja wiedzy**. Model wizyjny odzyskuje obiekty, OCR i walidatory kontrolują treść, a system typów i graf relacji zachowują znaczenie. SVG, PPTX i rysunki publikacyjne stają się jedynie różnymi sposobami wyświetlenia tej samej struktury. Umożliwia to formalną walidację figur, porównywanie zmian semantycznych pomiędzy wersjami i częściowo automatyczne przystosowywanie ilustracji do wymagań czasopisma.

14. Wnioski

Konwersja rastra do SVG jest łatwa tylko wtedy, gdy celem jest skalowanie wyglądu. Jeśli potrzebne są prawdziwie edytowalne napisy, figury, logiczne połączenia i pełna wierność znaczenia, zadanie staje się rekonstrukcją semantyczną. W tej klasie problemów klasyczne trasowanie pozostaje narzędziem pomocniczym, wykorzystywanym głównie do nieregularnych konturów. Tekst powinien być odzyskiwany przez OCR, geometria przez algorytmy dopasowania prymitywów, a relacje przez modele VLM i niezależną weryfikację.

Wyniki VFIG i RefineSVG uzasadniają inwestowanie w modele wyspecjalizowane w figurach naukowych oraz zamkniętą pętlę renderowania i korekty. Nie uzasadniają natomiast przekonania, że wysoki wskaźnik podobieństwa obrazu automatycznie oznacza poprawność naukową. **Najważniejszy jest brak zmiany treści:** nazw, jednostek, symboli, kierunków strzałek i powiązań między obiektami.

Dla zastosowań wymagających późniejszej edycji szczególnie rekomenduje się jeden, wersjonowany graf sceny z niezależnymi generatorami SVG i natywnego PPTX. Walidacja powinna osobno raportować wierność wizualną, tekstową, topologiczną i użytkową, a brak pewności traktować jako przesłankę do sprawdzenia, nie do zgadywania.

Aneks A. Przykładowy model danych i kontrola SVG

Poniższe fragmenty mają charakter referencyjny. Przed zastosowaniem produkcyjnym wymagają walidacji typów, bezpieczeństwa importu plików XML, testów Unicode oraz kontroli konkretnych typów SVG.

```python
from dataclasses import dataclass, field

@dataclass
class TextItem:
    id: str
    value: str
    bbox: tuple[float, float, float, float]
    source: str
    verified: bool = False

@dataclass
class Node:
    id: str
    kind: str
    bbox: tuple[float, float, float, float]

@dataclass
class Edge:
    id: str
    source_id: str
    target_id: str
    direction: str
    verified: bool = False

@dataclass
class Scene:
    nodes: list[Node] = field(default_factory=list)
    texts: list[TextItem] = field(default_factory=list)
    edges: list[Edge] = field(default_factory=list)
```

Podstawowy audyt struktury pliku powinien wykrywać obecność tekstu i osadzonych bitmap oraz wskazywać przypadki wymagające dalszej analizy. Sama liczba ścieżek nie dowodzi istnienia lub nieistnienia tekstu zamienionego na kontury.

```
from collections import Counter
from pathlib import Path
from defusedxml import ElementTree as ET  # bezpieczny parser XML

SVG = '{http://www.w3.org/2000/svg}'

def audit_svg(path: str | Path) -> dict:
    root = ET.parse(str(path)).getroot()
    tags = ('text', 'tspan', 'rect', 'circle', 'ellipse',
            'line', 'polyline', 'polygon', 'path', 'image', 'g')
    counts = {name: len(root.findall(f'.//{SVG}{name}'))
              for name in tags}
    embedded = []
    for node in root.findall(f'.//{SVG}image'):
        href = node.get('href') or node.get(
            '{http://www.w3.org/1999/xlink}href', '')
        if href.startswith('data:image/'):
            embedded.append(node.get('id', '<bez id>'))
    ids = [el.get('id') for el in root.iter() if el.get('id')]
    return {
        'counts': counts,
        'embedded_rasters': embedded,
        'duplicate_ids': [x for x, n in Counter(ids).items() if n > 1],
        'has_semantic_text': counts['text'] > 0,
        'manual_review_required': counts['text'] == 0 and counts['path'] > 20,
    }
```

Wynik audytu jest jedynie pierwszą kontrolą. Powinien zostać połączony z odczytem tekstu z pliku źródłowego oraz z renderowanego SVG. Produkcyjny walidator powinien także ograniczać rozmiary i referencje zewnętrzne, wykrywać niedozwolone konstrukcje XML i wymagać unikatowych identyfikatorów.

Aneks B. Protokół oceny i minimalna lista kontrolna

Obszar	Pytanie kontrolne	Warunek akceptacji
Tekst	Czy wszystkie napisy zostały odtworzone dosłownie?	Brak błędów krytycznych OCR
Jednostki	Czy greka, µ, m, indeksy i znaki są zgodne?	Zgodność z obrazem wzorcowym
Figury	Czy proste obiekty są prymitywami?	Brak nieuzasadnionej zamiany na ścieżki
Połączenia	Czy zachowano źródło, cel i kierunek?	Brak błędów w krawędziach krytycznych
Struktura	Czy każdy węzeł ma stabilny identyfikator?	Kompletna i spójna referencja
SVG	Czy jest poprawnym XML bez fałszywej „wektoryzacji” bitmapą?	Parser + audyt DOM
PowerPoint	Czy napisy i figury są natywnie edytowalne?	Test w docelowej wersji Office
Wizualizacja	Czy render zachowuje czytelność?	Ocena niezależna od semantyki
Reprodukowalność	Czy znane są wejścia, ustawienia i wersje?	Zamrożone dane i manifest
Niepewność	Czy niejednoznaczne elementy są oznaczone?	Brak nieujawnionych domysłów

Zalecany zapis jednego testu powinien uwzględniać identyfikator figury, kategorię, rozdzielczość, wersje modeli, hash źródła, zgodność tekstu, zgodność relacji, liczbę obiektów edytowalnych, wynik renderowania i listę błędów krytycznych. Takie dane umożliwiają późniejszą analizę regresji oraz porównanie nowych wersji modułów bez utraty interpretowalności wyników.

Aneks C. Katalog narzędzi i repozytoriów pomocniczych

Wybór narzędzia powinien wynikać z dominującego rodzaju elementów na rycinie. Dla minimalistycznego schematu blokowego wartość ma przede wszystkim poprawne odtworzenie napisów, prostych figur i połączeń. W ilustracji anatomicznej większy udział mają nieregularne kontury, stąd klasyczny tracer lub ręczna rekonstrukcja mogą być konieczne. Nie ma jednej biblioteki, która z jednakową niezawodnością rozwiązuje oba problemy.

Narzędzie / projekt	Rola w procesie	Krytyczne ograniczenie
VFIG-4B	semantyczna rekonstrukcja złożonych figur	wymagane niezależne sprawdzanie tekstu i relacji
StarVector	bezpośrednia generacja kodu SVG	błędy składni i geometrii wymagają renderowania
RefineSVG	korekta na podstawie różnicy renderów	informacja pikselowa nie rozstrzyga wszystkich konfliktów
VTracer	trasowanie nieregularnych konturów	brak jawnej semantyki tekstu i połączeń
Potrace / Inkscape	szybkie wektoryzowanie grafiki liniowej	napisy zwykle pozostają konturami
PaddleOCR	odczyt etykiet i symboli	mały tekst, matematyka i słabe kontrasty
Illustrator Image Trace	profesjonalna obróbka złożonych konturów	nie zastępuje semantycznego modelu diagramu
Vectorizer.AI	automatyczne trasowanie poprzez API	nie należy zakładać odzyskania prawdziwego tekstu
BioRender	ręczne tworzenie i korekta rycin biologicznych	zależność od środowiska autora i licencji
Figma	ręczna edycja i inspekcja warstw SVG	import nie odzyskuje brakujących znaczeń
draw.io	semantyczny edytor schematów	wymaga natywnego modelu diagramu
python-pptx / PptxGenJS	natywne slajdy PowerPoint	konieczność osobnej implementacji łączników i stylów

Projekty społecznościowe mogą dostarczać użytecznych pomysłów na poszczególne moduły, nawet jeśli nie mają jeszcze dojrzałych benchmarków. px-image2pptx wykorzystuje rozpoznawanie tekstu, usuwanie napisów z tła (*inpainting*) oraz tworzenie natywnych pól tekstowych [24]. image2editable rozwija rekonstrukcję do edytowalnych formatów z własną strukturą pośrednią [25]. FigureWeave, Image-to-PPT i ClusterGlyph pokazują inne warianty integracji segmentacji, OCR i generowania plików [26]–[28]. Nie należy traktować samej dostępności repozytorium jako dowodu skuteczności; przed wdrożeniem potrzebny jest audyt zależności, licencji oraz test na własnych obrazach.

W zastosowaniach naukowych specjalistyczny edytor, taki jak BioRender, może być końcowym środowiskiem pracy człowieka. Zaletą jest dostęp do bibliotek obiektów dziedzinowych i wygodna edycja, ale nie jest to równoważne automatycznemu odtworzeniu istniejącego diagramu. Podobnie Figma ułatwia kontrolę warstw i szczegółów graficznych, nie gwarantując jednak odtworzenia połączeń logicznych z przypadkowego SVG [29], [30].

Aneks D. Minimalny przykład natywnego eksportu do PowerPoint

Poniższy przykład pokazuje różnicę między wstawieniem SVG jako obrazu a wygenerowaniem rzeczywistych kształtów Office. Napisy pozostają edytowalne, a łącznik jest elementem slajdu. Nie jest to pełny generator grafu sceny: w wersji produkcyjnej współrzędne i identyfikatory powinny być pobierane z zatwierdzonej reprezentacji pośredniej.

```
from pptx import Presentation
from pptx.util import Inches, Pt
from pptx.enum.shapes import MSO_SHAPE, MSO_CONNECTOR

prs = Presentation()
slide = prs.slides.add_slide(prs.slide_layouts[6])

left = slide.shapes.add_shape(MSO_SHAPE.ROUNDED_RECTANGLE, Inches(0.7), Inches(1.4), Inches(2.3), Inches(1.0))
```

```
left.text = "Neuron A"
left.text_frame.paragraphs[0].runs[0].font.size = Pt(18)

right = slide.shapes.add_shape(MSO_SHAPE.ROUNDED_RECTANGLE, Inches(5.1), Inches(1.4), Inches(2.3), Inches(1.0))
right.text = "Neuron B"
right.text_frame.paragraphs[0].runs[0].font.size = Pt(18)

edge = slide.shapes.add_connector(MSO_CONNECTOR.STRAIGHT, Inches(3.0), Inches(1.9), Inches(5.1), Inches(1.9))
prs.save("diagram_native.pptx")
```

W praktycznym zastosowaniu łączniki muszą otrzymywać również style grotów, a ich współrzędne aktualizować się po zmianach układu. Równoległy eksport tego samego grafu do SVG pozwala użyć jednej reprezentacji źródłowej w prezentacji i artykule. Test końcowy powinien potwierdzić, że etykiety są edytowalnymi obiektami tekstowymi, a nie bitmapami ani zestawami konturów.

Bibliografia

[1] Q. He *et al.*, “VFIG: Vectorizing Complex Figures in SVG with Vision-Language Models,” *arXiv preprint*, arXiv:2603.24575, 2026. Dostęp: https://arxiv.org/abs/2603.24575.

[2] J. A. Rodriguez *et al.*, “StarVector: Generating Scalable Vector Graphics Code from Images and Text,” in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, 2025, pp. 16175–16186.

[3] S. Liu *et al.*, “RefineSVG: Visual Feedback-Driven Reinforcement Learning for Image-to-SVG Generation,” *arXiv preprint*, arXiv:2607.27699, 2026. Dostęp: https://arxiv.org/abs/2607.27699.

[4] P. Selinger, “Potrace: A polygon-based tracing algorithm,” dokumentacja projektu. Dostęp: https://potrace.sourceforge.net/.

[5] Visioncortex, “VTracer: Raster to Vector Graphics Converter,” repozytorium oprogramowania. Dostęp: https://github.com/visioncortex/vtracer.

[6] Adobe, “Trace images to convert raster into vector artwork,” dokumentacja Illustrator, 2026. Dostęp: https://helpx.adobe.com/illustrator/desktop/manage-objects/traces-mockups-symbols/trace-images-to-convert-raster-into-vector-artwork.html.

[7] Vectorizer.AI, “API Documentation,” dokumentacja techniczna. Dostęp: https://vectorizer.ai/api/documentation.

[8] Z. Cui *et al.*, “Draw with Thought: Unleashing Multimodal Reasoning for Scientific Diagram Generation,” *arXiv preprint*, arXiv:2504.09479, 2025. Dostęp: https://arxiv.org/abs/2504.09479.

[9] W3C, “Scalable Vector Graphics (SVG) 2: Text,” *W3C Candidate Recommendation*, 2026. Dostęp: https://www.w3.org/TR/2026/CR-SVG2-20261006/text.html.

[10] A. Jankovics, “ImageTracerJS,” repozytorium oprogramowania. Dostęp: https://github.com/jankovicsandras/imagetracerjs.

[11] Q. He *et al.*, “VFIG-Data,” zbiór danych, Hugging Face, 2026. Dostęp: https://huggingface.co/datasets/QijiaHe/VFIG-Data.

[12] X. Liu *et al.*, “VFIG-4B,” karta modelu, Hugging Face, 2026. Dostęp: https://huggingface.co/XunmeiLiu/VFIG-4B.

[13] X. Xing *et al.*, “Empowering LLMs to Understand and Generate Complex Vector Graphics,” in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, 2025, pp. 19487–19497, doi: 10.1109/CVPR52734.2025.01815. Dostęp:

https://openaccess.thecvf.com/content/CVPR2025/html/Xing_Empowering_LLMs_to_Understand_and_Generate_Complex_Vector_Graphics_CVPR_2025_paper.html.

[14] OmniSVG, “OmniSVG,” model i dokumentacja projektu. Dostęp: https://huggingface.co/OmniSVG/OmniSVG.

[15] Z. Lin *et al.*, “AutoFigure-Edit: Generating Editable Scientific Illustration,” *arXiv preprint*, arXiv:2603.06674, 2026. Dostęp: https://arxiv.org/abs/2603.06674.

[16] Recraft, “Pricing and API vector models,” dokumentacja dostawcy, 2026. Dostęp: https://www.recraft.ai/pricing?tab=api.

[17] Microsoft, “Edit SVG images in Microsoft 365,” dokumentacja pomocy technicznej. Dostęp: https://support.microsoft.com/en-us/office/graphics-visuals/edit-svg-images-in-microsoft-365.

[18] Microsoft Learn Q&A, “Unexpected behaviour of text elements when ungrouping SVG in PowerPoint,” dyskusja techniczna, 2026. Dostęp: https://learn.microsoft.com/en-nz/answers/questions/5697014/unexpected-behaviour-of-text-elements-when-ungroup.

[19] python-pptx, “Working with shapes,” dokumentacja programistyczna. Dostęp: https://python-pptx.readthedocs.io/en/latest/api/shapes.html.

[20] PptxGenJS, “API: Text and Shapes,” dokumentacja programistyczna. Dostęp: https://gitbrent.github.io/PptxGenJS/.

[21] diagrams.net, “Insert SVG images,” dokumentacja draw.io. Dostęp: https://www.drawio.com/docs/manual/insert/svg-insert/.

[22] CTAN, “svg: Include and extract SVG pictures in LaTeX documents,” dokumentacja pakietu. Dostęp: https://ctan.org/pkg/svg.

[23] Q. He *et al.*, “VFIG-Bench,” zbiór testowy, Hugging Face, 2026. Dostęp: https://huggingface.co/datasets/QijiaHe/VFIG-Bench.

[24] JadeLiu-tech, “px-image2pptx,” repozytorium projektu. Dostęp: https://github.com/JadeLiu-tech/px-image2pptx.

[25] guangzhaoli, “image2editable,” repozytorium projektu. Dostęp: https://github.com/guangzhaoli/image2editable.

[26] Krisocer, “FigureWeave,” repozytorium projektu. Dostęp: https://github.com/Krisocer/FigureWeave.

[27] 7zSun, “Image-to-PPT,” repozytorium projektu. Dostęp: https://github.com/7zSun/Image-to-PPT.

[28] vtavakkoli, “ClusterGlyph,” repozytorium projektu. Dostęp: https://github.com/vtavakkoli/ClusterGlyph.

[29] BioRender, “Pricing and product information,” dokumentacja dostawcy. Dostęp: https://www.biorender.com/pricing.

[30] Figma, “Add images and videos to designs,” dokumentacja pomocy. Dostęp: https://help.figma.com/hc/en-us/articles/360040028034-Add-images-and-videos-to-designs.