Walidacja i aktualizacja modeli BPMN za pomocą LLM

Michał Jurzak

Informatyka i Systemy Inteligentne, Inżynieria Oprogramowania

# Cel projektu

Zarządzanie procesami biznesowymi (BPM) jest fundamentalnym elementem strategii operacyjnej nowoczesnych organizacji. Notacja BPMN (Business Process Model and Notation) 2.0 stała się de facto standardem w wizualnym modelowaniu tych procesów, umożliwiając jednoznaczną komunikację między interesariuszami biznesowymi a deweloperami systemów IT. Mimo precyzji notacji, ręczne tworzenie i walidacja modeli BPMN jest procesem czasochłonnym i podatnym na błędy ludzkie, takie jak błędy logiczne, zakleszczenia, niespójności czy niezgodności z dobrymi praktykami.

Celem projektu jest automatyczna analiza, porównanie oraz poprawa modeli BPMN z wykorzystaniem narzędzi programistycznych oraz wsparcia modeli językowych (LLM). Projekt umożliwia wykrywanie błędów w plikach BPMN, generowanie raportów, wizualizację różnic oraz automatyczne nanoszenie poprawek za pomocą narzędzi.

Repozytorium projektu: *https://github.com/michaljurzak1/LLM-BPMN-Checker*

# Kluczowe funkcjonalności

1. **Interaktywny system wsparcia dla BPMN**

Projekt umożliwia użytkownikowi interaktywną pracę z diagramami BPMN – od wczytania pliku, przez analizę, po automatyczną poprawę i wizualizację zmian.

1. **Integracja dużego modelu językowego (LLM) z narzędziami BPMN**

System łączy możliwości LLM (np. GPT) z dedykowanymi narzędziami do analizy i edycji BPMN, co pozwala na zaawansowaną interpretację, diagnozę i modyfikację modeli procesów biznesowych.

1. **Wizualizacja i śledzenie zmian**

Każda zmiana w modelu jest wizualizowana, a użytkownik może łatwo porównać stan przed i po modyfikacji oraz zaakceptować lub odrzucić poprawki.

1. **Czat z agentem AI**

Użytkownik komunikuje się z agentem AI w języku naturalnym, uzyskując wyjaśnienia, rekomendacje oraz możliwość zadawania pytań dotyczących modelu BPMN.

1. **Automatyczna walidacja i naprawa błędów**

System wykrywa typowe błędy w diagramach BPMN (np. niezgodności ze standardem, błędy strukturalne) i automatycznie je naprawia lub sugeruje poprawki.

1. **Elastyczność i konfigurowalność**

Użytkownik może dostosować zachowanie agenta AI poprzez zmianę promptu systemowego oraz pracować na własnych plikach BPMN.

1. **Wsparcie dla nauki i testowania**

Projekt zawiera narzędzia i przykłady do testowania, debugowania oraz eksploracji działania systemu, co czyni go przydatnym zarówno w praktyce biznesowej, jak i edukacji.

# Opis działania programu

Aplikacja napisana została w pythonie przy użyciu biblioteki LangChain pozwalającej na zintegrowanie modelu LLM z programem. Do wizualizacji użyty został moduł Streamlit, a także bpmn-js użyty do wizualizowania modelu.

Obraz zawierający tekst, zrzut ekranu, diagram, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 1. Interfejs użytkownika

System opiera się na trójwarstwowej architekturze, której schemat interakcji jest przedstawiony na Rysunku 2. Komponenty systemu to:

1. **Warstwa Prezentacji (Użytkownik):** Interfejs użytkownika (UI) zrealizowany jako aplikacja webowa. Umożliwia on wybór pliku BPMN, jego wizualizację (**Rysunek 1**), interakcję z agentem poprzez czat oraz zatwierdzanie lub odrzucanie proponowanych zmian.
2. **Warstwa Logiki Aplikacji (Agent - BPMNAgent):** Centralny element systemu, który pełni rolę orkiestratora. Agent, napędzany przez LLM, otrzymuje zapytania od użytkownika w języku naturalnym. Przetwarza je w kontekście załadowanego diagramu BPMN, korzystając z zestawu dostępnych narzędzi do analizy i modyfikacji. Odpowiada za prowadzenie dialogu, formułowanie rekomendacji i planowanie kroków modyfikacji.
3. **Warstwa Narzędziowa (Narzędzia BPMN - BPMNTools):** Zestaw wyspecjalizowanych funkcji (API), które agent może wywoływać. Funkcje te operują bezpośrednio na strukturze XML pliku BPMN, realizując atomowe operacje, takie jak odczyt informacji o węzłach, dodawanie nowych elementów, usuwanie istniejących czy modyfikacja przepływów sekwencji.

**Przepływ interakcji:** Użytkownik wysyła zapytanie (np. "Znajdź błędy w tym procesie"). Agent analizuje zapytanie i stan diagramu. Decyduje o użyciu odpowiednich narzędzi (np. get\_all\_nodes, get\_node\_info). Na podstawie uzyskanych danych formułuje odpowiedź i propozycję zmian. Po akceptacji użytkownika, agent wywołuje narzędzia modyfikujące (np. add\_tasks, remove\_nodes), które aktualizują model BPMN. Zaktualizowany diagram jest prezentowany użytkownikowi.

Obraz zawierający tekst, diagram, zrzut ekranu, linia

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 2. Uproszczony schemat interakcji komponentów systemu.

## Zastosowane Technologie

* Python**:** Główny język programowania, wykorzystany do implementacji logiki backendowej i integracji komponentów.
* LangChain**:** Kluczowy framework do budowy aplikacji opartych na LLM. Umożliwił stworzenie agenta, zdefiniowanie dla niego narzędzi oraz zarządzanie stanem konwersacji i łańcuchami wywołań (chains).
* Streamlit**:** Biblioteka do szybkiego tworzenia interaktywnych aplikacji webowych w Pythonie. Została użyta do budowy całego interfejsu graficznego.
* bpmn-js: Biblioteka JavaScript do renderowania i interakcji z diagramami BPMN. Została zintegrowana z aplikacją Streamlit w celu wizualizacji modeli.
* Duży Model Językowy (LLM): Wykorzystano model z rodziny GPT – ***o4-mini*** (przez API) jako silnik rozumowania dla agenta AI.

Agent AI miał do dyspozycji zestaw narzędzi pozwalających na kompleksową interakcję z modelem BPMN. Kluczowe narzędzia to:

* get\_all\_nodes: Zwraca listę wszystkich elementów (zadań, bramek, zdarzeń) w modelu.
* get\_node\_info: Dostarcza szczegółowych informacji o konkretnym węźle (ID, typ, nazwa, połączenia przychodzące/wychodzące).
* add\_tasks, remove\_nodes, edit\_nodes: Narzędzia do modyfikacji struktury modelu.
* add\_sequence\_flows: Umożliwia tworzenie nowych połączeń między węzłami.
* get\_bpmn\_as\_image: Generuje obraz diagramu, który model może wykorzystać do analizy wizualnej.

W toku prac stwierdzono, że choć dodanie narzędzia get\_bpmn\_as\_image zwiększa koszt obliczeniowy, jest ono niezwykle użyteczne dla LLM w ocenie ogólnej czytelności i złożoności wizualnej diagramu. Ograniczenie liczby narzędzi analitycznych do tych najbardziej uniwersalnych zapobiegało zapętleniom agenta i poprawiało jego skuteczność.

# Testowanie i ewaluacja

Ewaluację przeprowadzono na zbiorze 30 rzeczywistych modeli BPMN, które zawierały zróżnicowane błędy typowe dla praktyki modelowania, w tym:

* **Błędy semantyczne i logiczne:** nieprawidłowe użycie bramek (np. brak synchronizacji po bramce równoległej), zakleszczenia.
* **Błędy ciągłości procesu:** osierocone ścieżki, brak zdarzeń końcowych.
* **Błędy w użyciu zdarzeń:** niepoprawne umiejscowienie zdarzeń przerywających i nieprzerywających.
* **Błędy strukturalne i notacyjne:** brak etykiet, nieczytelny układ ("spaghetti flow").

Testy podzielono na dwie główne fazy:

1. **Faza 1: Identyfikacja Błędów:** Ocena zdolności modelu do wykrywania problemów.
   * *Scenariusz A:* Wykrywanie wszystkich możliwych błędów w modelu.
   * *Scenariusz B:* Wymuszenie na modelu wskazania jednego, najistotniejszego błędu.
2. **Faza 2: Autonomiczna Korekta:** Ocena zdolności agenta do naprawy modelu z użyciem dostępnych narzędzi.
   * *Scenariusz C:* Implementacja poprawek na podstawie precyzyjnych instrukcji od użytkownika.
   * *Scenariusz D:* W pełni autonomiczna korekta (samodzielne wykrycie i naprawa błędów).

W przypadku wykrywania kilku błędów model językowy uzyskał średnią ocenę w skali 0-10 wynoszącą około 6.6, w niektórych przypadkach osiągając maksymalne noty wykrywając dokładnie błędy o których była mowa. Oceniany był również przypadek propozycji rozwiązania. Tu LLM posiada nieco gorsze rezultaty otrzymując wynik 5.7. Propozycje rozwiązania często mijają się z celem, lub również nieco pogarszając obecny model BPMN w przypadku gdy model językowy ma wolną rękę.

Przypadek zmuszenia modelu językowego do podania wyłącznie jednego, najistotniejszego błędu wykazuje znaczną nieskuteczność. Model językowy w większości przypadków (70%) nie uznaje błędu sugerowanego jako najważniejszy. Natomiast w przypadkach w których model zaklasyfikował najistotniejszy błąd, wykazuje się największą precyzją w zdefiniowaniu błędu oraz propozycji rozwiązania.

W kolejnej fazie testów, sprawdzone zostały najlepsze rezultaty z poprzedniej części – wykryty jeden z kilku najlepiej pasujący błąd do obecnego zaproponowanego. Ta faza testów obejmowała możliwości programu – modelu wraz z dostępnymi narzędziami – do autonomicznej poprawy modelu BPMN. Testy te rozbiły się o dwie kategorie: zdolność do implementowania wskazanych poprawek oraz zdolność do autonomicznego wykrywania i korygowania błędów.

W pierwszej kategorii, dotyczącej implementacji poprawek na podstawie konkretnych instrukcji oraz ogólnej autonomicznej korekty, LLM wykazał się wyjątkową biegłością, gdy otrzymał precyzyjne wytyczne. W wielu przypadkach, gdy program został poinstruowany, jaką zmianę ma wprowadzić (np. zamienić bramkę XOR na AND, dodać brakującą bramkę scalającą, przenieść zdarzenie czasowe, dodać etykiety, czy użyć bramki opartej na zdarzeniach), osiągał perfekcyjne wyniki. . Co więcej, w niektórych scenariuszach (np. rezygnacja z przerywającego zdarzenia czasowego, poprawa struktury przepływu), LLM przekraczał dosłowne instrukcje, dostarczając jeszcze bardziej poprawne i solidne rozwiązania, co świadczy o głębokim zrozumieniu semantyki BPMN.

Jednakże, jego autonomiczne wykrywanie i korygowanie błędów (bez konkretnych instrukcji) było znacznie bardziej zmienne. Chociaż potrafił samodzielnie zidentyfikować i poprawnie naprawić niektóre istotne problemy, często mylnie diagnozował problem (np. mylił przyczynę zakleszczenia), wprowadzał nowe, krytyczne błędy prowadzące do zakleszczeń, lub tworzył chaotyczne wizualnie "spaghetti flow", które choć technicznie poprawiały problem (np. dodanie bramki scalającej), były niemal niemożliwe do odczytania. Pojawiały się również przypadki, gdzie LLM skupiał się na drugorzędnych kwestiach, ignorując główne błędy strukturalne (np. brak podziału puli uczestników), lub nie podejmował żadnych działań.

Duże modele językowe są w stanie wykryć kilka najistotniejszych błędów strukturalnych modelu BPMN, natomiast podczas wykorzystania narzędzi najlepsze rezultaty model otrzymywał gdy otrzymywał jasne instrukcje. Konieczne są dalsze prace badające temat wraz z wykorzystaniem bardziej zaawansowanych technik (np. few shot prompting, poszerzenie o bazy RAG lub prompt tuning) w celu uzyskiwania lepszych rezultatów.

# Rezultaty i przykładowe zastosowania

Głównym rezultatem projektu jest w pełni funkcjonalna aplikacja webowa, która stanowi studium przypadku integracji LLM w procesie inżynierii oprogramowania. System:

* **Skraca czas walidacji modeli BPMN**, oferując natychmiastowe sugestie i wykrywając błędy, które mogłyby zostać przeoczone.
* **Działa jako interaktywne narzędzie edukacyjne**, pozwalając użytkownikom (szczególnie początkującym) na zrozumienie błędów i naukę dobrych praktyk modelowania.
* **Wspiera iteracyjną pracę nad procesami**, umożliwiając szybkie prototypowanie zmian i wizualizację ich wpływu na model.

## Przykładowe zastosowania:

* **Analitycy biznesowi:** Mogą wykorzystać narzędzie do szybkiej weryfikacji swoich modeli przed ich formalnym zatwierdzeniem lub przekazaniem do działu IT.
* **Zespoły deweloperskie:** Mogą zintegrować podobne mechanizmy w swoich potokach CI/CD do automatycznej weryfikacji dokumentacji procesowej.
* **Instytucje edukacyjne:** Aplikacja może służyć jako pomoc dydaktyczna na kursach z zakresu zarządzania procesami biznesowymi i modelowania.

# Wnioski i możliwości rozwoju

Projekt z powodzeniem zademonstrował, że integracja Dużych Modeli Językowych z dedykowanymi narzędziami programistycznymi stanowi obiecujący kierunek w usprawnianiu pracy z modelami BPMN. Kluczowym wnioskiem jest stwierdzenie, że system osiąga najwyższą wartość jako zaawansowany asystent (co-pilot), a nie w pełni autonomiczny ekspert. Jego siła leży w precyzyjnym wykonywaniu poleceń i sugerowaniu rozwiązań w dobrze zdefiniowanym kontekście, pozostawiając ostateczną, strategiczną decyzję w rękach człowieka. Konieczność stosowania precyzyjnych instrukcji systemowych (*prompt engineering*) oraz starannego doboru narzędzi okazała się kluczowa dla skuteczności agenta

System posiada znaczny potencjał do dalszego rozwoju. Główne kierunki to:

1. **Zaawansowane techniki promptingu:** Eksploracja technik takich jak Few-Shot Prompting czy Chain-of-Thought w celu poprawy zdolności rozumowania modelu w bardziej złożonych scenariuszach.
2. **Implementacja mechanizmu RAG (Retrieval-Augmented Generation):** Wzbogacenie wiedzy modelu o obszerną bazę danych zawierającą oficjalną specyfikację BPMN 2.0, wzorce projektowe i najlepsze praktyki. Pozwoliłoby to na generowanie bardziej precyzyjnych i ugruntowanych w standardzie rekomendacji.
3. **Dostrajanie modelu (Fine-tuning):** Wytrenowanie specjalizowanego modelu na dużym, kuratorowanym zbiorze par (błędny model, poprawny model), co mogłoby znacząco poprawić jego zdolności autonomicznej korekty.
4. **Rozbudowa zestawu narzędzi:** Dodanie bardziej zaawansowanych narzędzi analitycznych, np. do wykrywania pętli, analizy kosztów/czasu procesu czy symulacji wykonania.
5. **Integracja z repozytoriami procesów:** Umożliwienie bezpośredniej pracy na modelach przechowywanych w systemach takich jak Camunda, Signavio czy innych korporacyjnych repozytoriach.

# Podsumowanie

Stworzony system jest innowacyjnym narzędziem, które skutecznie łączy moc obliczeniową Dużych Modeli Językowych z potrzebami współczesnej analizy biznesowej. Poprzez automatyzację wykrywania błędów i interaktywną modyfikację modeli BPMN, projekt nie tylko usprawnia pracę analityków, ale także otwiera nowe perspektywy dla zastosowania sztucznej inteligencji w cyklu życia oprogramowania i procesów biznesowych. Stanowi on solidną podstawę do dalszych badań nad tworzeniem inteligentnych systemów wspierających złożone zadania inżynierskie.