### **Jak tworzyć skuteczne prompty do ekstrakcji danych z tekstów historycznych**

Celem jest przekształcenie za pomocą dużych modeli językowych (LLM) nieustrukturyzowanych danych tekstowych (źródeł, opracowań, inwentarzy historycznych) w ustrukturyzowane formaty, takie jak JSON czy CSV, gotowe do importu do baz danych, systemów GIS czy baz wiedzy. Poniższe zasady pomogą w tworzeniu precyzyjnych i efektywnych promptów.

#### **1.** Podstawy**:** r**ola,** k**ontekst**,z**adanie**

Każdy prompt powinien zaczynać się od jasnego zdefiniowania ram działania modelu.

* **Określenie roli (“persona”):** Warto zacząć od nadania modelowi konkretnej roli. To ustawia jego "stan umysłu" i aktywuje odpowiednie obszary jego "wiedzy".
  + *Przykład:* "Jesteś ekspertem w dziedzinie historii gospodarczej Polski w XIX wieku i specjalizujesz się w analizie i kategoryzacji danych ze Słownika Geograficznego Królestwa Polskiego."\* lub "Działaj jako archiwista-asystent, którego zadaniem jest precyzyjna ekstrakcja danych biograficznych z rękopisów."
* **Opis materiału wejściowego:** Szczegółowy opis tekstu, który model będzie analizował. Pewne terminy i pojęcia zmieniały swoje znaczenie w czasie, wskazanie epoki z której analizowany tekst pochodzi może mieć znaczenie dla modelu. Im więcej kontekstu, tym mniej błędów interpretacyjnych.
  + *Przykład:* `"Otrzymasz fragmenty Słownika Geograficznego Królestwa Polskiego z końca XIX wieku. Tekst jest napisany w archaicznym języku polskim, zawiera liczne skróty i specyficzną strukturę haseł. Każde hasło dotyczące miejscowości jest oddzielone od następnego."
* **Precyzyjne zdefiniowanie zadania:** Wskazanie modelowi, co jest celem nadrzędnym.
  + *Przykład:* `"Twoim zadaniem jest zidentyfikowanie i wydobycie informacji o obiektach przemysłowych i rzemieślniczych opisanych w tekście. Zignoruj informacje o rolnictwie, demografii i strukturze własnościowej, chyba że są bezpośrednio związane z opisywanym obiektem przemysłowym."
* **Jakość tekstu**: Warto pamiętać że jakość wyników - danych przygotowanych przez model zależy także od jakości tekstu wejściowego. Jeżeli analizowany tekst był skanowany i rozpoznawany, kluczowa jest jakość OCR, przed użyciem promptów dobrze jest oczyścić tekst (np. usunąć łamanie wierszy, poprawić typowe błędy OCR)

#### **2. Definiowanie** s**truktury** w**yjściowej**

To kluczowy element, model językowy musi wiedzieć nie tylko **co** znaleźć, ale także **jak** to zapisać.

* **Model danych (schemat JSON):** Należy zdefiniować dokładną strukturę wyjściową, najlepiej w formacie JSON Schema lub poprzez opisany przykład. Struktura składa się z listy: nazw pól i typu pola (typu danych), których oczekujemy (typu np. string - tekst, integer - liczba całkowita, boolean - prawda/fałsz, lista informacji np. lista nazw).
  + *Przykład:*  
    {  
     "nazwa\_miejscowosci": "string", // nazwa miejscowości w mianowniku  
     "typ\_obiektu": "string", // np. 'wieś', 'miasto', 'folwark'  
     "data\_powstania": "integer | null", // Rok jako liczba lub null, jeśli brak danych  
     "wlasciciel": "string | null", // nazwa właściciela lub właścicieli  
     "liczba\_mieszkańców": "integer | null", // liczba mieszkańców  
     "informacje\_dodatkowe": "string" // dodatkowe informacje   
    }
* **Obsługa brakujących danych:** Warto wyraźnie poinstruować model, co ma zrobić, gdy nie znajdzie jakiejś informacji. Pozwala to, w jakimś stopniu, uniknąć "halucynowania" danych.
  + *Przykład:* `"Jeśli w tekście nie ma informacji o dacie powstania lub liczbie pracowników, użyj wartości 'null' dla odpowiednich pól. Nie pomijaj klucza w wynikowym JSONie.", lub
  + “Jeżeli tekście nie ma informacji o dacie powstania miejscowości, pomiń pole w wynikowej strukturze danych JSON”

#### **3. Zaawansowane** i**nstrukcje i** r**eguły**

Diabeł tkwi w szczegółach, zwłaszcza w tekstach historycznych.

* **Dostarczanie przykładów (few-shot prompting):** To jedna z najskuteczniejszych technik. Wskazane jest podanie 1-3 przykładów fragmentu tekstu wejściowego i odpowiadającego mu, idealnie sformatowanego wyniku JSON. Przykłady powinny obejmować zarówno typowe przypadki, jak i te problematyczne (np. z niejednoznacznym zapisem).
* **Reguły specjalne, logika warunkowa:** specyficzne dla źródła zasady interpretacji mogą być bardzo istotne dla osiągnięcia wyników dobrej jakości.
  + *Przykład:* `"W tekście haseł autor notatki często jest oznaczony inicjałami na końcu hasła (np. 'B. Ch.'), niekiedy jest to inicjał i nazwisko. Zidentyfikuj autora i przypisz go do pola 'autor\_notatki', inicjały w środku tekstu dotyczą zwykle innych osób, takie dane zignoruj."
* **Kategorie pozytywne i negatywne:** Aby uniknąć błędnej kategoryzacji, dobrym pomysłem jest podawanie przykładów tego, co **należy**, a czego **nie należy** zaliczać do danej kategorii.
  + *Przykład:* `"Do kategorii 'przemysł' zaliczaj: fabryki, huty, gorzelnie, browary, cegielnie, tartaki. Do tej kategorii NIE zaliczaj: młynów, wiatraków, kuźni rzemieślniczych - dla nich użyj osobnych kategorii ‘młyny’ i 'rzemiosło'."
* **Słownik skrótów i synonimów:** Jeśli źródło używa niestandardowych skrótów lub archaicznych nazw, należy dostarczyć modelowi listę takich skrótów.
  + *Przykład:* `"Poniżej znajduje się lista skrótów używanych w tekście: 'mr.' oznacza 'morgi', 'fol.' oznacza 'folwark', albo też prościej 'mr.=morgi, fol.=folwark'"
* **Wielojęzyczność**: teksty historyczne często zawierają sformułowania i fragmenty w innych niż tekst główny języku, aczkolwiek współczesne modele językowe radzą sobie dobrze z wieloma językami (szczególnie spośród tych kilkudziesięciu najpopularniejszych) warto w instrukcjach dla modelu wskazać, że będzie miał do czynienia z tego typu tekstem
* **Niepewność**: można wskazać modelowi jak ma postępować w przypadku informacji niejasnych lub niepewnych - czy takie dane pomijać, czy zamiast wartości z tekstu podawać wartość ‘niepewne’, można też wskazać by opcjonalnie model dodawał w takim przypadku dodatkowe pole ‘uwagi’ gdzie mógłby napisać dlaczego jakaś informacja jest niejasna lub niepewna.
* **Dodatkowe informacje**: wydobywając z tekstu informacje, które będą służyły do dalszego przetwarzania warto dać modelowi możliwość zapisania informacji dodatkowych (tworząc dodatkowe opcjonalne pole), które będą mogły posłużyć np. do bardziej precyzyjnej identyfikacji danych. Np. jeżeli oczekujemy informacji o miejscu urodzenia jakiejś osoby, to poza nazwą miejscowości np. Stara Wieś istotna może być informacja ‘koło Kalisza’.
* **Dane testowe**: dobrym nawykiem jest przygotowanie zestawu testowego tekstów do analizy i wzorcowego ręcznie przygotowanego wyniku. Można wówczas porównywać wyniki np. różnych modeli, oraz wyniki różnych wariantów promptu.

#### **4. Sterowanie** p**rocesem** a**nalizy**

Można spróbować "zmusić" model do bardziej metodycznej pracy.

* **Instrukcje krok po kroku (chain of thought):** Można poprosić model, aby analizował tekst w określonej kolejności. To spowalnia jego działanie, ale drastycznie zwiększa precyzję (**uwaga**: nowsze modele jak o3, GPT-5, Gemini Pro domyślnie same przeprowadzają taki proces).
  + *Przykład:* `"Przeanalizuj tekst krok po kroku. Krok 1: Zidentyfikuj główną nazwę miejscowości, której dotyczy hasło. Krok 2: Przeczytaj uważnie cały opis w poszukiwaniu słów kluczowych związanych z przemysłem. Krok 3: Dla każdego znalezionego obiektu, wydobądź szczegółowe informacje zgodnie ze schematem. Krok 4: Przed wygenerowaniem ostatecznego JSONa, zweryfikuj, czy wszystkie informacje pochodzą bezpośrednio z tekstu."\*
* **Dekompozycja złożonych zadań:** Jeśli zadanie jest bardzo skomplikowane (np. wymaga wydobycia 20 różnych pól z zawiłego tekstu), nie należy tworzyć jednego "super-promptu". Warto podzielić pracę na kilka prostszych promptów (np. jeden do danych geograficznych, drugi do demograficznych, trzeci do gospodarczych) i połączyć wyniki w kolejnym kroku. Model poradzi sobie znacznie lepiej z kilkoma prostszymi zadaniami niż z jednym ekstremalnie złożonym.

#### **5. Optymalizacja**

Pierwszy prompt rzadko jest idealny, dlatego ważne jest jego doskonalenie.

* **Zacznij prosto, potem komplikuj:** można rozpocząć pracę od prostego promptu i podstawowego zadania. Przetestować go na kilku przykładach, przeanalizować błędy i stopniowo dodawać kolejne reguły, przykłady i ograniczenia, aby je wyeliminować.
* **Kontrola "kreatywności" (parametr “temperature”):** Do zadań ekstrakcji danych, gdzie liczy się precyzja, a nie kreatywność, warto używać niskich ustawień parametru temperatura (np. 0.0 lub 0.1). Zapewnia to bardziej powtarzalne i deterministyczne wyniki. W typowych interfejsach typu “chat” nie ma jednak takiej możliwości, można to zrobić w [aistudio.google.com](http://aistudio.google.com/), czy w aplikacji OpenWebUI oraz oczywiście w przypadku obsługi modeli przez API za pomocą skryptów.

#### 6. Uruchamianie promptów

Każda z popularnych aplikacji typu ‘chat’ np. ChatGPT czy Gemini pozwoli na uruchomienie przygotowanego promptu. Tekst do analizy można załączyć jako dodatkowy plik. Obecne modele pozwalają przetworzyć jednorazowo kilkaset tysięcy czy nawet milion tokenów (token to fragmenty wyrazów lub krótkie wyrazy, na które model dzieli tekst podczas przetwarzania). W praktyce plik tekstowy (\*.txt) o objętości 1 MB (czyli ok. miliona znaków) może zostać odczytany, ale modele nie są (obecnie) w stanie zwracać jednorazowo dłuższych odpowiedzi a dane w formacie JSON mogą mieć sporą objętość. Dlatego pliki z tekstem do analizy przez model dobrze jest podzielić na mniejsze - kilkadziesiąt - 100 tys. znaków i przetwarzać osobno a następnie scalić wyniki. Jeżeli przetwarzane jest źródło w rodzaju słownika, które dzieli się na hasła, dobrze jest przygotować wcześniej tekst w podziale na hasła z unikalnymi identyfikatorami, co umożliwi później łatwe scalanie wyników z wielu procesów przetwarzania. Narzędziem do scalania wielu plików JSON może być np. jq, JSON Merge CLI lub skrypty Python.

#### 7. Dodatkowe pliki

prompt\_gemini.txt - tekst przykładowego promptu, który wyszukuje dane podstawowe dla miejscowości - nazwy, przynależność administracyjna, parafie itd.)

sgkp\_test\_data.txt - dane do testów, 25 przykładowych haseł z SGKP

sgkp\_gemini\_result.json - przykład wyniku przygotowanego przez model Gemini Pro 2.5