Politechnika Śląska

Sprawozdanie z modułu nr 2

WEBAPPSEC 2023/2024

Bezpieczeństwo webaplikacji

Kierunek: Informatyka

Członkowie zespołu:
Grzegorz Koperwas

Gliwice, 2023/2024

Spis treści

1. Wprowadzenie

W sprawozdaniu omówimy sposoby przeciwdziałania podatności `API4:2023 Unrestricted Resource Consumption`. Będziemy omawiać je w kontekście aplikacji wykorzystujących rozwiązania oparte na LLM.

1.1. Założenia dot. zadania laboratoryjnego

LLM, z racji ogromnej ilości parametrów, wymagają dużych ilości pamięci. Są również nie przewidywalne, podatne na ataki *prompt injection*[2] oraz są nowym trendem, przez co nie koniecznie mamy zawsze do czynienia z przemyślanym rozwiązaniem.

Z tego powodu będziemy się skupiać na rozwiązaniach, które nie tylko ograniczają prawdopodobieństwo zablokowania się systemu, ale również ograniczających jego podatności na ataki *prompt injection*.

Będziemy ewaluowali rozwiązania w ramach ich przydatności do zabezpieczenia przykładowej aplikacji realizującej *Retrieval Augmented Generation*, w skrócie `RAG`.

Aplikacje te wykorzystują wektorowe bazy danych, gdzie przechowywane są dane. W ramach zapytania baza taka potrafi dobrać informacje, które pomagają odpowiedzieć `LLM` na zadane pytania. Korzystają one z wyspecjalizowanych sieci zwanych *embeddings*, które zamieniają tekst na wektor. Mając nasz tekst w formie wektorowej, możemy łatwo porównywać dystans między różnymi elementami, czyli pośrednio ich podobieństwo.

2. Rozwinięcie

Podstawową linią obrony przeciwko zbyt dużemu zużyciu zasobów jest ograniczenie tego, ile pojedyncze zapytanie do systemu może wywoływać modele `LLM`. Innym czynnikiem, który musimy kontrolować, jest *kontekst* oraz jego rozmiar. Modele `LLM` operują nie na nieskończonej ilości danych, tylko na pewnym wycinku tekstu, jaki jest im zadany jako argument.

Te przysłowiowe okienko, przez które na problem spoglądają `LLM` nazywamy kontekstem, nic poza nim nie ma wpływu na sieć. Jego rozmiar jest mierzony w *tokenach*, które mniej-więcej odpowiadają słowom lub częściom słów. Każdy model ma określony rozmiar kontekstu, przykładowo dla modeli `LLama` mamy do czynienia z kontekstem wielkości 2048 tokenów[4], a niektóre z modeli `GPT` mają konteksty rozmiaru nawet 128 tysięcy tokenów, dla modelu `GPT-4 Turbo`.

Jednak musimy pamiętać, że często koszt użycia modelu jest wyznaczany na podstawie ilości tokenów, które zostają przekazana jako kontekst, jak i ilości wygenerowanych tokenów.

2.1. Kontrola rozmiaru kontekstu

Podczas budowania naszej aplikacji wykorzystującej `RAG` możemy chcieć dać modelowi dostęp do całej naszej bazy danych. Jednak przekazanie w kontekście wszystkiego jak leci, nie zadziała z następujących powodów:

- Będzie to kosztowne, ponieważ płacimy za każdy token[3].
- Będzie to nieefektywne, ponieważ nasze dane nie zmieszczą się w kontekście.

W celu rozwiązania tego problemu możemy skorzystać z biblioteki *langchain*, która pomaga nam zarządzać kontekstem.

Możemy skorzystać z jej pomocy w celu integracji z wektorową bazą danych, na przykład *chromadb*. Pozwoli nam to do naszego kontekstu wprowadzać tylko dokumenty (lub ich fragmenty) faktycznie związane z naszym problemem.

Z użyciem takiej bazy wektorowej możemy ograniczyć przetwarzane informacje tylko do konkretnych dokumentów lub ich fragmentów, co pozwala nam używać mniejszych kontekstów, co ogranicza koszty pojedynczego zapytania.

Dodatkowo możemy zastosować prosty limit ilości tokenów, które może model wygenerować. Inną opcją kontroli wyników modeli jest, na przykład

w narzędziu `llama.cpp`[1], jest zadawanie gramatyki, jaką musi spełniać odpowiedź.

Przykładowo, zamiast ograniczać ilość tokenów, możemy modelowi uniemożliwić wygenerowanie więcej niż dwóch zmian. Możemy również nakazać modelowi generowanie odpowiedzi w konkretnym formacie, na przykład `yaml`, zamiast opisywać mu w kontekście, jakie warunki ma spełniać jego odpowiedź.

2.2. Kontrola zapytań

Dodatkowo możemy stosować tradycyjne metody kontroli ilości zapytań, na przykład:

- Ograniczenia ilości zapytań dla klienta.
- Ograniczenia rozmiaru zapytania.

Jednak te metody nie są specyficzne dla modeli `LLM`, więc nie chcę ich szczegółowo omawiać w tym referacie.

3. Podsumowanie i wnioski

- *Podsumowanie* - Modele `LLM` mogą korzystać zarówno z tradycyjnych metod zabezpieczania `API`, jak i z innych technik pozwalających optymalizować ich działanie.
- *Wnioski* - Koszt wykorzystania modeli `LLM` jest zależny od rozmiaru kontekstu, dlatego musimy kontrolować jego rozmiar. Jest to podobne do tego, czemu stosujemy paginację.

4. Spis literatury

[1] Georgi Gerganov. *llama.cpp*. 2023. URL: https://github.com/ggerganov/llama.cpp (term. wiz. 30. 11. 2023).

[2] Kai Greshake i in. *Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection*. 2023. arXiv: 2302.12173 [cs.CR].

[3] OpenAI. *Pricing*. 2023. URL: https://openai.com/pricing (term. wiz. 30. 11. 2023).

[4] Hugo Touvron i in. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL].