Efektivní správa kontextu v konverzační AI s Gemini: Perzistence napříč relacemi, optimalizace a integrace s MongoDB

1. Úvod: Přetrvávající výzva kontextu v konverzační AI

Imperativ kontextu

Kontext představuje základní kámen pro koherentní, smysluplné a poutavé konverzace mezi člověkem a počítačem. Umožňuje umělé inteligenci (AI) chápat nuance, pamatovat si předchozí výroky a poskytovat relevantní odpovědi, čímž se posouvá od jednoduchého odpovídání na otázky k opravdovému dialogu.[1] Bez efektivní správy kontextu se interakce stávají nesouvislými, opakujícími se a pro uživatele frustrujícími.[3]

Bezstavová povaha LLM a její důsledky

Velké jazykové modely (LLM), včetně Gemini, fungují v zásadě na bázi jednotlivých interakcí. Nemají inherentní dlouhodobou paměť, která by přetrvávala napříč oddělenými relacemi nebo konverzacemi.[5] Každá nová relace typicky začíná s "čistým štítem". Tato bezstavovost je často záměrným designovým rozhodnutím, částečně kvůli zvládání výpočetní složitosti a částečně kvůli ochraně soukromí uživatelů a bezpečnosti dat, jelikož informace nejsou samotným modelem po ukončení relace uchovávány.[5]

Tato charakteristika znamená, že odpovědnost za implementaci mechanismů pro perzistentní paměť spočívá na aplikační vrstvě. LLM jsou navrženy tak, aby neměly dlouhodobou paměť mezi relacemi, což je explicitně uváděno jako přínosné pro soukromí uživatelů a bezpečnost dat.[5] Aplikace spravující LLM je zodpovědná za perzistenci důležitých informací napříč relacemi pomocí externího úložiště.[6] Z toho vyplývá, že spoléhat se na to, že jádro LLM spontánně vyvine perzistentní paměť napříč relacemi, by mohlo být zavádějící. Bezstavová povaha, nabízející soukromí, je vlastností. Vývojáři tedy *musí* navrhovat řešení pro perzistentní paměť na aplikační úrovni. Nejedná se pouze o dočasné řešení, ale o zásadní architektonický aspekt pro budování stavové AI. Potřeba externích úložných řešení, jako je MongoDB, tedy není jen technickou nutností kvůli současným omezením LLM, ale je v souladu s filozofií designu, která upřednostňuje soukromí a dává vývojářům kontrolu nad tím, co se pamatuje.

Překlenutí mezery: Techniky a architektury

Tato zpráva prozkoumá strategie a architektury k překonání těchto inherentních omezení, umožňující chatbotům poháněným Gemini udržovat kontext napříč relacemi,

optimalizovat kvalitu a stručnost tohoto kontextu a využívat externí úložiště jako MongoDB pro perzistenci.

2. Porozumění kontextu v Google Gemini: Schopnosti a správa v rámci jedné relace

Velké kontextové okno Gemini: Změna paradigmatu?

Modely Gemini se pyšní výrazně velkými kontextovými okny, přičemž některé verze jako Gemini 1.5 nabízejí až 1 milion tokenů.[1] To představuje podstatný skok oproti starším modelům (např. 8K tokenů zmíněných v [7]). Toto rozsáhlé okno dokáže v rámci jedné interakce zpracovat značné množství informací, jako například 50 000 řádků kódu, celé romány nebo dlouhé transkripty podcastů.[7] To usnadňuje výkonné učení v kontextu (in-context learning), jak bylo demonstrováno schopností Gemini naučit se překládat z angličtiny do kalamangštiny pouze s materiály poskytnutými v kontextu.[7] Pro uživatele to znamená, že *v rámci jedné nepřetržité relace* dokáže Gemini udržet velmi dlouhou a detailní historii konverzace bez okamžitého zkracování.

Jak Gemini zpracovává kontext v jedné relaci

Kontextové okno funguje jako krátkodobá paměť.[7] Všechny tokeny (vstupní prompt, předchozí části konverzace) v tomto okně jsou modelem zvažovány při generování následující odpovědi.[1] Jak konverzace roste a blíží se limitu tokenů, starší části konverzace mohou být implicitně "zapomenuty" nebo "vytlačeny", aby uvolnily místo novějším interakcím, pokud nejsou explicitně spravovány aplikací.[6]

Nativní optimalizace: Kešování kontextu v Gemini API

Gemini API nabízí kešování kontextu jako primární optimalizaci při práci s dlouhými kontexty.[7] To může snížit náklady a latenci u opakovaných interakcí nebo když části kontextu zůstávají statické. Pull request na GitHubu pro kuchařku Gemini diskutuje implementaci perzistentního kešování kontextu pomocí JSON souboru pro ukládání a opětovné použití předchozích sumarizací a historie konverzace, čímž se snižuje počet redundantních volání API.[9] Ačkoliv je toto kešování přínosné, je často zaměřeno na optimalizaci opakovaných volání s podobnými *prefixy* kontextu v rámci potenciálně souvisejících, ale odlišných interakcí API, spíše než na plnou perzistenci stavu napříč relacemi pro unikátního uživatele v průběhu času.

Problém "jehly в kupce sena" a výkon

I přes velká kontextová okna existují výzvy. Větší okna mohou vnést šum a irelevantní informace, což potenciálně snižuje přesnost, pokud model zápasí s rozlišením kritických informací.[1] Toto je známé jako problém "jehly v kupce sena". Zpracování

větších kontextů také vyžaduje více výpočetního výkonu, času a způsobuje vyšší náklady.[1] Uvádí se, že modely Gemini jsou vysoce schopné extrahovat informace z rozsáhlých kontextů (až 99% přesnost v mnoha případech [7]), ale obecně je nejlepší se vyhnout předávání zbytečných tokenů.[7]

Existence velkých kontextových oken, jako u Gemini [7], posouvá, nikoli však zcela řeší, problém správy kontextu. Ačkoliv model dokáže pojmout velké množství dat, slepé vkládání veškeré historické konverzace je suboptimální. Velká okna mohou vést ke snížení přesnosti kvůli šumu a problému "jehly v kupce sena" [1], a jejich zpracování je výpočetně náročné a může zvyšovat latenci.[1] Doporučenou praxí je vyhýbat se předávání nepotřebných tokenů.[7] Výzva se tedy mění z "jak vměstnat kontext" na "jak efektivně a přesně zpřístupnit *správný* kontext v tomto velkém okně". Techniky pro sumarizaci kontextu, selekci a vyhledávání (jako RAG) tak zůstávají vysoce relevantní nejen pro modely s malými okny, ale také pro optimalizaci výkonu a relevance v rámci rozsáhlých oken Gemini. Důraz se přesouvá na *kvalitu* a *relevanci* kontextu, nejen na jeho kvantitu.

3. Strategie pro udržování kontextu napříč relacemi: Za hranice jedné interakce

Nepostradatelná role aplikační vrstvy

Jak již bylo uvedeno, LLM jako Gemini si inherentně neuchovávají paměť napříč relacemi.[5] Odpovědnost za správu a perzistenci konverzačního kontextu plně spočívá na aplikaci postavené kolem LLM.[6] To zahrnuje návrh mechanismů pro ukládání relevantních konverzačních dat na konci jedné relace a jejich opětovné načtení na začátku nové.

Přehled externích úložných řešení pro perzistenci

- **Databáze:** Relační nebo NoSQL databáze (jako MongoDB) jsou běžnou volbou pro ukládání historií konverzací, uživatelských profilů a sumarizovaného kontextu.[6]
- **Souborové systémy:** Jednoduché JSON soubory nebo jiné strukturované soubory lze použít pro méně komplexní scénáře nebo pro kešování, jak je vidět v příkladu kuchařky Gemini.[9]
- **Vektorové databáze:** Stále populárnější pro ukládání vektorových reprezentací konverzačních segmentů, umožňující sémantické vyhledávání relevantních minulých interakcí, často jako součást systému RAG.[8] MongoDB Atlas také nabízí schopnosti vektorového vyhledávání.[11]

Klíčová data k perzistenci

- Surová historie konverzace (uživatelské zprávy, odpovědi AI, časová razítka).
- Shrnutí minulých konverzací.
- Extrahované entity, záměry nebo uživatelské preference.
- Identifikátory relací a uživatelů pro propojení historie s konkrétními uživateli.

Udržování kontextu napříč relacemi znamená více než jen uchovávání historie; jde o zachycení vyvíjejícího se stavu uživatele. Cílem je "pamatovat si" [6], což zahrnuje nejen doslovný přepis, ale i vyvíjející se porozumění potřebám uživatele, jeho preferencím a celkovému stavu dialogu. Techniky jako sumarizace [3] a extrakce entit (implicitně zmíněné v RAG a obohacování kontextu) se snaží tento vyvíjející se stav destilovat. Ukládání tohoto "stavu" (např. v JSON v MongoDB) umožňuje AI obnovit konverzace s bohatším porozuměním, než jaké by poskytlo pouhé přehrání starých zpráv. To se podobá lidské paměti, která si nepamatuje jen slova, ale podstatu a klíčové body. Návrh schématu perzistentního úložiště by proto měl počítat se strukturovanými prvky stavu uživatele nad rámec pouhého seznamu zpráv. To činí vyhledávání cílenějším a reinjektáž kontextu efektivnější.

4. Optimalizace kontextu: Vyvažování stručnosti, kvality a relevance

Základní výzvou, jak zdůrazňuje uživatelský dotaz, je dosáhnout "co nejmenší délky, ale vysoké kvality" kontextu. Tato sekce se přímo zabývá strategiemi pro dosažení této rovnováhy.

A. Techniky pro kondenzaci a redukci kontextu

Sumarizace

- **Sumarizace pomocí LLM:** Využití LLM (i samotného Gemini nebo specializovaného modelu) k vytváření stručných shrnutí minulých segmentů konverzace.[3] Shrnutí nahrazuje starší zprávy, zachovává klíčové informace a zároveň snižuje počet tokenů.[3]
- **Princip fungování:** Prompt instruuje LLM, aby shrnul historii konverzace se zaměřením na klíčové informace, uživatelské preference, požadavky a rozhodnutí relevantní pro pokračování konverzace.[3] Cílem je, aby shrnutí nepřesáhlo požadovaný limit tokenů.
- **Nástroje a frameworky:** SummarizingTokenWindowChatMemory v LangChain4j (příklad v Javě, ale koncept je aplikovatelný) monitoruje počet tokenů a spouští sumarizaci.[3] Podobnou logiku lze vytvořit v Pythonu.
- **Výzkum:** Články z ArXiv diskutují LLM-sumarizátory konverzací pro sledování stavu dialogu [17] a vzájemně se posilující schopnosti syntézy dialogu a

sumarizace.[18] Hierarchické frameworky mohou segmentovat, kondenzovat a následně sumarizovat dlouhé dokumenty/konverzace.[19]
- **Přínos:** Výrazně snižuje zátěž tokenů při zachování podstaty minulých interakcí.

Selektivní kontext a komprese

- **Koncept:** Namísto sumarizace se tyto techniky snaží identifikovat a ponechat pouze nejvíce *informativní* nebo *kritické* tokeny či věty z kontextu a zbytek odstranit.[20]
- **Knihovna selective-context:** Balíček PyPI, který komprimuje prompty a kontext pomocí základního LM k výpočtu vlastní informace pro lexikální jednotky a vyhodnocení jejich informativnosti, což umožňuje LLM zpracovat přibližně 2x více obsahu.[21] Dokáže redukovat kontext na základě zadaného poměru.
- **Pokročilé techniky (bez nutnosti trénování):**
  - **Infinite Retrieval:** Zpracovává vstup po částech, používá skóre pozornosti k výběru top-K tokenů z aktuální části relevantních pro dotaz a ukládá do mezipaměti pouze ID tokenů celých vět obsahujících tyto kritické tokeny. Snižuje paměťové nároky tím, že neukládá plné KV stavy.[20]
  - **Cascading KV Cache:** Rozšiřuje posuvné okno pozornosti o vícevrstvou KV mezipaměť. Tokeny jsou uchovávány na základě historické důležitosti (EMA skóre pozornosti), přičemž nedávné tokeny jsou v horních vrstvách a starší tokeny s vysokou relevancí v hlubších vrstvách.[20]
- **Přínos:** Cílenější než sumarizace, potenciálně zachovává klíčové detaily, které by shrnutí mohlo abstrahovat. Redukuje šum.

Segmentace (Chunking)

- **Proces:** Rozdělení rozsáhlých dokumentů nebo dlouhých historií konverzací na menší, spravovatelné části (segmenty), které se vejdou do kontextových oken nebo jsou vhodné pro RAG.[13]
- **Osvědčené postupy:** Používání logických hranic (odstavce, sekce) nebo posuvných oken s přesahem pro zachování koherence.[13] Tutoriál ModelContextManager [23] demonstruje sémantickou segmentaci.
- **Přínos:** Nezbytné pro RAG a pro zpracování informací, které přesahují kontextové okno LLM, i když je velké. Umožňuje cílené zpracování relevantních segmentů.

B. Techniky pro obohacení kontextu a cílené vyhledávání

Generování rozšířené o vyhledávání (RAG)

- **Základní myšlenka:** Namísto vkládání všech informací do promptu RAG vyhledává relevantní úryvky z externí znalostní báze (která může zahrnovat minulé

sumarizované/segmentované konverzace uložené v MongoDB) a do promptu přidává pouze tyto úryvky.[7]
- **Jak pomáhá stručnosti a kvalitě:** Prompt zůstává stručný, protože obsahuje pouze nejrelevantnější vyhledané informace, přesto je odpověď LLM vysoce kvalitní, protože je informována těmito cílenými externími daty.
- **Proces:** Ukládání dat (např. segmentů konverzací, shrnutí) do vektorové databáze (MongoDB Atlas může fungovat jako jedna [11]). Když přijde nový dotaz, vyhledají se relevantní segmenty a připojí se k promptu.[8]
- **Gemini a RAG:** Zdroj [7] uvádí, že historicky bylo Q&A možné pouze s RAG kvůli omezenému kontextu a nízké faktické paměti. Ačkoliv je paměť Gemini lepší, RAG je stále cenný pro uzemnění odpovědí v konkrétních, aktuálních nebo proprietárních datech (jako jsou minulé interakce s uživatelem).
- **Přínos:** Výrazně snižuje množství informací přímo vkládaných do kontextového okna LLM, bojuje proti halucinacím a umožňuje přístup k obrovskému množství externích znalostí.

C. Inženýrství promptů pro optimální využití kontextu

- **Jasnost a stručnost:** Zajištění, aby prompty byly jasné a jednoznačné, aby efektivně vedly LLM, zejména u rozsáhlých kontextů.[1]
- **Strukturování promptů:** Používání šablon, jasných značek pro různé sekce (např. "Shrnutí minulé konverzace:", "Aktuální otázka:") a instruování modelu, jak používat poskytnutý kontext.[24]
- **Umístění dotazu:** U Gemini je často lepší umístit dotaz/otázku na *konec* promptu, za všechny ostatní kontextuální informace.[7]
- **Učení bez příkladu (Zero-shot), s jedním příkladem (One-shot), s několika příklady (Few-shot):** Poskytnutí příkladů v rámci promptu může vést LLM, zejména u specifických úkolů nebo formátů odpovědí.[22]
- **Promptování řetězcem myšlenek (Chain-of-Thought):** Rozdělení komplexního uvažování na kroky v rámci promptu může zlepšit kvalitu odpovědi.[22]
- **Přínos:** Maximalizuje užitečnost tokenů, které *jsou* zahrnuty v kontextovém okně.

Optimalizace kontextu není jednorázovým řešením, ale vyžaduje vícevrstvou strategii. Uživatel požaduje minimální délku a vysokou kvalitu kontextu. Různé zdroje představují rozmanité techniky: sumarizaci [3], selektivní kontext [21], RAG [8], segmentaci [13] a inženýrství promptů.[24] Žádná jednotlivá technika není univerzálně optimální. Sumarizace může ztratit detaily, RAG závisí na kvalitním vyhledávání a selektivní kontext potřebuje dobrá kritéria. Efektivní správa kontextu často zahrnuje kombinaci těchto přístupů: například segmentaci dlouhých konverzací [13], následnou sumarizaci těchto segmentů [3], jejich uložení do vektorové databáze [11] a použití RAG k vyhledání

relevantních shrnutí pro nový dotaz [8], to vše řízené kvalitním inženýrstvím promptů.[24] Problém "jehly v kupce sena" [1] zdůrazňuje, že i u velkých oken je klíčová relevance, což vyžaduje tyto vrstvy filtrování a vyhledávání. Vývojáři by proto měli zvážit pipeline přístup ke správě kontextu, kde se různé techniky aplikují v různých fázích (např. předzpracování historie, vyhledávání relevantních částí, konstrukce finálního promptu).

"Hodnota" tokenu je navíc kontextově závislá a dynamická. Techniky selektivního kontextu [20] se snaží identifikovat "kritické" nebo "informativní" tokeny. RAG [8] vyhledává "relevantní" informace. Sumarizace [3] kondenzuje na "klíčové informace". To naznačuje, že ne všechny tokeny v historii konverzace mají stejnou hodnotu pro *aktuální* část konverzace. Relevance minulých informací se může měnit s vývojem konverzace. Co bylo kritické před třemi tahy, může být nyní méně důležité. Techniky jako ModelContextManager [23] používají skórování založené na aktuálnosti, důležitosti a sémantické podobnosti k dynamickému hodnocení této hodnoty. Efektivní optimalizace kontextu tedy vyžaduje mechanismy pro dynamické hodnocení a prioritizaci prvků kontextu na základě jejich aktuální relevance k probíhající interakci, spíše než statické zkracování nebo jednoduché chronologické zahrnutí.

Následující tabulka poskytuje srovnání různých technik optimalizace kontextu:

Tabulka 1: Srovnání technik optimalizace kontextu

Technika	Popis	Výhody	Nevýhody	Vliv na délku kontextu	Vliv na kvalitu/relevanci kontextu	Klíčové zdroje/nástroje
Plná historie (naivní přístup)	Předání celé dosavadní historie konverzace.	Jednoduchost implementace; žádná ztráta informací (pokud se vejde do okna).	Rychle narazí na limit tokenů; vysoké náklady; potenciální šum a problém "jehly v kupce sena".[1]	Vysoká	Nízká (bez filtrace)	-

Zkracování posuvným oknem	Uchovává pouze N posledních zpráv/tokenů, starší zahazuje.	Jednoduchost; kontrola nad délkou kontextu.	Ztráta staršího, potenciálně relevantního kontextu; arbitrární odříznutí.[6]	Střední až Nízká	Střední	Základní implementace v mnoha chat systémech
Sumarizace konverzace	Použití LLM k vytvoření stručného shrnutí starších částí konverzace; shrnutí nahrazuje původní zprávy.[3]	Výrazné snížení počtu tokenů; zachování klíčových informací.	Možná ztráta detailů; náklady a latence na sumarizaci; kvalita závisí na sumarizačním modelu/promptu.[3]	Nízká	Střední až Vysoká	[3], LangChain ConversationSummaryMemory
Selektivní kontext / Komprese	Identifikace a uchování pouze nejinformativnějších tokenů/vět; zahození méně relevantních částí.[20]	Cílená redukce šumu; potenciální zachování kritických detailů; může být efektivnější než sumarizace.	Složitost implementace; kritéria pro selekci mohou být náročná na definici; možná ztráta kontextu.	Nízká až Střední	Vysoká (pokud je selekce přesná)	[21] (selective-context)[20]
Generování rozšířené o vyhledávání (RAG)	Vyhledání relevantních informací z externí báze znalostí (včetně	Přístup k rozsáhlým/aktuálním datům; snížení halucinací; kontext je vysoce	Složitost nastavení; závislost na kvalitě vyhledávání a znalostní báze;	Nízká (v promptu)	Vysoká	[8], LangChain, LlamaIndex

	minulé konverzace) a jejich přidání do promptu.[7]	relevantní a stručný.	latence vyhledávání.			

5. Implementace perzistentního kontextu pomocí JSON a MongoDB

A. Strukturování konverzačního kontextu jako JSON

JSON je lehký, lidsky čitelný a široce podporovaný formát pro výměnu dat. Dobře se hodí k NoSQL databázím jako MongoDB (která používá BSON, binární reprezentaci dokumentů podobných JSON) a je často formátem pro interakce API s LLM.[7] Gemini API lze nakonfigurovat pro strukturovaný výstup JSON poskytnutím schématu.[7] To je užitečné pro extrakci strukturovaných dat z konverzací nebo pro to, aby LLM formátoval svůj vlastní stav způsobem, který je snadno uložitelný. Existují dva způsoby: konfigurace responseSchema na modelu nebo poskytnutí schématu v textovém promptu [27]; klíčový je parametr response_mime_type: "application/json".

Při návrhu JSON schématu pro kontext chatu by mělo schéma zachycovat základní prvky pro rekonstrukci kontextu. Příklad polí: session_id, user_id, časová razítka, message_history (pole objektů zpráv, každý s rolí, obsahem, časovým razítkem), current_summary, extracted_entities, user_profile_hints. Zdroj [2626] pojednává o použití JSON Schema k získání konzistentních strukturovaných dat z LLM, což lze přizpůsobit pro ukládání kontextu. LlamaIndex SimpleChatStore může perzistovat historii chatu do JSON souboru (chat_store.persist(persist_path="chat_store.json")) nebo serializovat do/z JSON řetězce (chat_store.json(), SimpleChatStore.parse_raw(chat_store_string)) [12], což demonstruje jednoduchou JSON strukturu pro chatovací zprávy.

Následující tabulka ukazuje příklad JSON schématu pro ukládání konverzačního kontextu v MongoDB:

Tabulka 2: Příklad JSON schématu pro uložený konverzační kontext v MongoDB

JSON

```
{
  "session_id": "unique_session_string",
```

```
  "user_id": "unique_user_string",
  "created_at": "ISO_datetime_string",
  "last_updated_at": "ISO_datetime_string",
  "tags": ["topic_A", "product_inquiry"],
  "message_history": // Volitelné, pokud jsou zprávy segmentovány pro RAG
    }
    //... další zprávy
  ],
  "current_summary": { // Volitelné, pokud se používá sumarizace
    "summary_text": "Uživatel se dotazoval na produkt X. Asistent nabídl pomoc.",
    "generated_at": "ISO_datetime_string",
    "source_message_ids": ["unique_message_id_1", "unique_message_id_2"]
  },
  "extracted_entities": { // Volitelné
    "products": ["produkt X"],
    "intent": "podpora_produktu"
  },
  "user_preferences": { // Volitelné, naučené v průběhu času
    "preferred_language": "cs",
    "communication_style": "formal"
  },
  "vector_embedding_session_summary": [0.789,..., 0.101] // Volitelné, vektorová reprezentace
celkového shrnutí relace pro RAG
}
```

Toto schéma demonstruje, jak ukládat nejen surové zprávy [12], ale také shrnutí [3], entity a dokonce i vektorové reprezentace [11], což odráží myšlenku MongoDB jako "kontextového hubu". Poskytuje vývojáři výchozí bod pro vlastní modelování dat.

B. Využití MongoDB pro ukládání kontextu

MongoDB je vhodná díky flexibilnímu schématu (BSON), nativní práci s JSON-like dokumenty, škálovatelnosti a bohatým možnostem dotazování, včetně fulltextového vyhledávání [10] a klíčového vektorového vyhledávání pro RAG. JSON objekty definované výše lze přímo ukládat do kolekce MongoDB.

MongoDB Atlas nabízí vestavěné schopnosti vektorového vyhledávání [11], což umožňuje ukládat vektorové reprezentace segmentů/shrnutí konverzací vedle textu. Při zahájení nové relace nebo položení dotazu lze provést sémantické vyhledávání podobnosti proti těmto uloženým vektorovým reprezentacím k načtení nejrelevantnějšího minulého

kontextu. Zdroj [1111] poskytuje příklad v Rustu pro vytvoření systému AI paměti s Rig AI a MongoDB, včetně definice vektorového indexu pro pole embedding. LlamaIndex i LangChain mají integrace s MongoDB Atlas Vector Search.[7] LlamaIndex MongoDBAtlasVectorSearch vyžaduje definici indexu v MongoDB Atlas, např. na poli embedding se specifikovaným počtem dimenzí a metrikou podobnosti (např. kosinus).[15]

LangChain MongoDBChatMessageHistory třída umožňuje ukládat a načítat historii chatových zpráv přímo v MongoDB.[7] Vyžaduje connection_string, session_id, database_name a collection_name. Zprávy ukládá pod history_key (výchozí "History") klíčované pomocí session_id_key (výchozí "SessionId").[28]

C. Příklad pracovního postupu: Ukládání a načítání kontextu

1. **Zahájení relace:** Uživatel zahájí chat. Aplikace zkontroluje, zda existuje user_id. Pokud ano, načte nejnovější kontextový dokument(y) pro tohoto user_id z MongoDB. Může se jednat o posledních N zpráv, shrnutí nebo sémanticky relevantní segmenty prostřednictvím vektorového vyhledávání. Deserializuje JSON kontext a připraví jej pro vložení do promptu Gemini.
2. **Během relace:** Jak konverzace postupuje, nové zprávy (uživatele i AI) se připojují k reprezentaci kontextu v paměti. Periodicky (např. každých N tahů nebo při generování shrnutí) se aktualizuje dokument MongoDB novými zprávami nebo nejnovějším shrnutím. Příklad Rig AI [11] sumarizuje segmenty každé 3 tahy (6 zpráv).
3. **Ukončení relace (nebo neaktivita):** Provede se finální aktualizace MongoDB, aby byl zajištěn perzistentní nejnovější stav. To může zahrnovat generování finálního shrnutí relace.
4. **Prořezávání/archivace kontextu (dlouhodobě):** Pro velmi dlouhodobé ukládání je třeba zvážit strategie pro archivaci starších, méně relevantních částí historie nebo pro vytváření zhuštěnějších shrnutí v průběhu času za účelem správy nákladů na úložiště a efektivity načítání.

MongoDB může sloužit jako "kontextový hub", tedy více než jen záznamník zpráv. Zdroje ukazují, že MongoDB může ukládat historii chatu [14], surové dokumenty pro RAG [25] a vektorové reprezentace.[11] To naznačuje, že MongoDB může fungovat jako centrální repozitář pro různé formy zpracovaného kontextu: surovou historii, shrnutí, extrahované entity, uživatelské profily a vektorové reprezentace. Tento "kontextový hub" pak může sloužit různým potřebám: přímé vyvolání historie, vstup pro sumarizační modely, znalostní báze pro RAG atd. Dynamické schéma MongoDB [10] je pro tuto multifunkční roli obzvláště vhodné, jelikož struktura uloženého kontextu se může vyvíjet. Při návrhu schématu MongoDB je tedy vhodné přemýšlet široce o všech

způsobech, jakými může být kontext používán a zpracováván, nejen pro přehrávání konverzací. To umožňuje vytvořit všestrannější a výkonnější systém správy kontextu.

6. Využití frameworků a knihoven pro zrychlený vývoj

Implementace veškeré logiky správy kontextu od nuly může být složitá a časově náročná. Frameworky poskytují předpřipravené komponenty a abstrakce.

A. LlamaIndex

- **Chat Stores:** LlamaIndex nabízí různé implementace ChatStore pro správu historie chatu.[12]
  - SimpleChatStore: Úložiště v paměti, může perzistovat do/z JSON souborů na disku.[12] Užitečné pro jednoduché případy nebo pro začátek.
  - Databázová úložiště: Zahrnují RedisChatStore, DynamoDBChatStore, PostgresChatStore a potenciálně další, které by mohly být přizpůsobeny nebo vytvořeny pro MongoDB.[12] Tyto umožňují ukládat historii chatu vzdáleně.
  - ChatMemoryBuffer používá tato úložiště k poskytování paměti chatovacím enginům/agentům.[12]
- **Integrace MongoDB pro RAG:** MongoDBAtlasVectorSearch umožňuje používat MongoDB Atlas jako vektorové úložiště pro RAG, ukládání a dotazování vektorových reprezentací dokumentů/segmentů.[7] To je klíčové pro načítání relevantního minulého kontextu. Integrace zahrnuje nastavení klienta MongoDB, specifikaci názvů databáze/kolekce a definici indexu vektorového vyhledávání v Atlasu.[15]

B. LangChain

- **Chat Message Histories:** Třída MongoDBChatMessageHistory je speciálně navržena pro ukládání a načítání historie konverzací v MongoDB.[7] Spravuje relace pomocí session_id a ukládá zprávy.[28]
- **Document Loaders:** MongodbLoader může načítat dokumenty z MongoDB pro použití v RAG nebo jiném zpracování.[7]
- **Vector Stores:** Integrace MongoDBAtlasVectorSearch pro RAG.[7]
- **Memory Modules:** Různé paměťové moduly (např. BufferMemory, ConversationSummaryMemory), které mohou být zálohovány pomocí MongoDBChatMessageHistory.

C. Instructor

- **Strukturovaný výstup z Gemini:** Instructor pomáhá získávat strukturovaná data (např. JSON) z LLM jako Gemini pomocí Pydantic modelů k definici požadovaného

výstupního schématu.[31]
- **Relevance pro správu kontextu:** To je klíčové pro:
  - Zajištění, aby shrnutí generovaná Gemini dodržovala specifický formát JSON pro snadné uložení v MongoDB.
  - Extrakci klíčových entit nebo informací o stavu z odpovědí Gemini strukturovaným způsobem.
  - Samotné Gemini API podporuje konfiguraci výstupu JSON [7], a Instructor s tím může pracovat nebo přidat vrstvu validace a opakovaných pokusů.

D. Ostatní nástroje

- **Rig AI Framework:** Zdroj [1111] demonstruje budování systému dlouhodobé AI paměti pomocí Rig a MongoDB, včetně sumarizace konverzací a jejich ukládání/načítání na základě vektorové podobnosti.
- **Pieces for Developers / Pieces Copilot:** Nabízí funkce pro správu kontextu, včetně dlouhodobé paměti zachycováním materiálů pracovního postupu, používáním uložených úryvků kódu a odkazováním na složky. Může fungovat lokálně.[32] Ačkoliv se přímo netýká MongoDB, ilustruje správu kontextu na aplikační úrovni.

Ačkoliv frameworky jako LangChain a LlamaIndex [12] výrazně zrychlují vývoj poskytováním předpřipravených modulů, jejich efektivní využití vyžaduje pochopení základních konceptů. Například pro správnou volbu typu paměti, konfiguraci připojení k MongoDB nebo návrh RAG pipeline je stále nutné rozumět principům správy kontextu, sumarizace, vektorového vyhledávání atd., jak bylo popsáno v předchozích sekcích. Znalost *proč* je RAG užitečný (Sekce 4.B) pomáhá při jeho správné implementaci s komponentami RAG v LangChain. Tyto nástroje tedy nejsou "černými skříňkami"; solidní koncepční porozumění je životně důležité pro jejich přizpůsobení a řešení problémů.

Následující tabulka poskytuje přehled relevantních knihoven a frameworků pro správu kontextu:

Tabulka 3: Přehled relevantních knihoven/frameworků pro správu kontextu

Knihovna/Framework	Klíčové funkce pro správu kontextu	Možnosti integrace s MongoDB	Kompatibilita/Synergie s Gemini	Primární případ použití pro dotaz uživatele

LlamaIndex	Chat Stores (SimpleChatStore, databázové), integrace vektorových DB, paměťové moduly.	MongoDBAtlasVectorSearch pro RAG [15]; možnost vlastních Chat Stores.	Může využívat Gemini pro úlohy jako sumarizace; zpracovává výstup z Gemini.	Ukládání historie chatu, RAG s minulými konverzacemi, správa paměti pro agenty.
LangChain	Chat Message Histories (MongoDBChatMessageHistory), Document Loaders, Vector Stores, Memory Modules.	MongoDBChatMessageHistory [28], MongodbLoader [25], MongoDBAtlasVectorSearch.[14]	Může využívat Gemini jako LLM v řetězcích; zpracovává výstup z Gemini.	Ukládání historie chatu v MongoDB, RAG, komplexní konverzační řetězce.
Instructor	Parsování strukturovaného výstupu (JSON) z LLM pomocí Pydantic modelů, validace, opakované pokusy.	Nepřímá; zajišťuje, že data pro uložení do MongoDB jsou správně strukturovaná.	Přímo podporuje Gemini pro strukturovaný výstup [31]; užitečný pro formátování kontextu.	Zajištění strukturovaného kontextu (např. shrnutí, extrahované entity) pro ukládání a další zpracování.
Rig AI Framework	Budování AI paměťových systémů, sumarizace, vektorové vyhledávání.	Integrace s MongoDB pro ukládání a vyhledávání paměti.[11]	Může být použit s jakýmkoli LLM, včetně Gemini, pro generování a využívání paměti.	Implementace dlouhodobé paměti s ukládáním sumarizovaných konverzací v MongoDB.

7. Pokročilé úvahy: Protokol kontextu modelu (MCP) a orchestrace

Co je MCP?

Protokol kontextu modelu (MCP) je otevřený standard umožňující AI agentům/LLM bezpečně a plynule se připojovat k různým externím zdrojům dat, nástrojům a dalším modelům.[23] Definuje, jak může orchestrátor (např. klient poháněný modelem Gemma) objevovat dostupné nástroje/zdroje, požadovat akce, předávat kontext a přijímat výsledky.[33] Komponenty zahrnují Nástroje (funkce, které LLM mohou volat), Zdroje

(datové zdroje) a Prompty (šablony).[34]

MCP v ekosystému Gemini/Gemma

Blog Google Cloud [33] popisuje architekturu, kde Gemma funguje jako konverzační manažer a orchestrátor, využívající MCP k volání specializovaných služeb, jako je LLM pro překlad nebo Gemini Pro pro komplexní uvažování. Gemma zpracovává interakci s uživatelem a základní často kladené otázky, zatímco komplexní úkoly nebo ty, které vyžadují specifické znalosti (jako finanční analýza prostřednictvím Gemini Pro), jsou delegovány. Kontext (např. dotaz uživatele, historie konverzace) je předáván prostřednictvím MCP.

Relevance pro kontext napříč relacemi

Ačkoliv se primární zaměření MCP v příkladech často soustředí na použití nástrojů a orchestraci modelů v rámci jedné relace, koncept standardizovaného předávání kontextu lze rozšířit. Pokud by "služba dlouhodobé paměti" (např. postavená na MongoDB) vystavovala rozhraní kompatibilní s MCP, mohl by orchestrační LLM (jako Gemma) použít MCP k dotazování této služby na relevantní minulý kontext pro aktuálního uživatele.

Sémantická segmentace, dynamická správa tokenů a skórování relevance kontextu s MCP

Tutoriál na MarkTechPost [23] podrobně popisuje budování ModelContextManager, který implementuje sémantickou segmentaci, generování vektorových reprezentací a skórování relevance (na základě aktuálnosti, důležitosti, sémantické podobnosti) za účelem optimalizace kontextu pro LLM. Ačkoliv samotný tutoriál hluboce neintegruje "protokolovou" část MCP, demonstruje *druh* inteligentního zpracování kontextu, které by mohl provádět zdroj nebo nástroj s podporou MCP.

Standardizace výměny kontextu prostřednictvím MCP by mohla zjednodušit komplexní paměťové architektury. MCP poskytuje standardní rozhraní pro interakci LLM s externími nástroji a zdroji.[33] Systém perzistentní paměti (MongoDB + vyhledávací logika) lze považovat za takový externí zdroj. Pokud tento paměťový systém zpřístupní své schopnosti (např. "get_user_summary", "get_relevant_past_interactions") prostřednictvím MCP, pak by s ním mohl interagovat jakýkoli LLM agent kompatibilní s MCP bez nutnosti vlastního integračního kódu. To odděluje LLM agenta od specifických implementačních detailů paměťového systému a mohlo by zjednodušit budování komplexních, multi-agentních systémů, kde různí agenti mohou potřebovat přístup ke sdíleným nebo individuálním dlouhodobým pamětem. Ačkoliv je MCP stále ve vývoji, naznačuje budoucnost, kde by služby správy kontextu mohly být více

"plug-and-play", což by vývojářům umožnilo snadněji kombinovat nejlepší LLM se specializovanými paměťovými řešeními. Pro uživatele to znamená sledovat vývoj MCP, jelikož by mohl ovlivnit budoucí architektonické volby pro správu kontextu s Gemini a dalšími modely.

8. Osvědčené postupy, výzvy a etické aspekty

Bezpečnost dat a soukromí

Ukládání historie konverzací, zejména pokud obsahuje osobně identifikovatelné údaje (PII) nebo citlivá data, do externí databáze jako MongoDB, vyvolává obavy o bezpečnost a soukromí.[8]

- **Osvědčené postupy:** Implementace robustních kontrol přístupu, šifrování dat v klidu i při přenosu, anonymizace nebo pseudonymizace dat tam, kde je to možné, a dodržování příslušných předpisů o ochraně údajů (např. GDPR, CCPA). Jasně informovat uživatele o tom, jaká data jsou ukládána a na jak dlouho.

Správa syndromu degradace kontextu (CDS)

CDS označuje pokles koherence a užitečnosti v dlouhotrvajících konverzacích s LLM, dokonce i s velkými kontextovými okny.[4] Je způsoben limity kontextového okna (nakonec), akumulací šumu/nesprávných interpretací a nedostatkem "celkového obrazu" ze strany LLM.

- **Projevy:** Opakující se odpovědi, zapomínání zavedených faktů, přílišné zjednodušování.[4]
- **Zmírnění:** Diskutované techniky optimalizace kontextu (sumarizace, RAG, selektivní kontext) jsou klíčové pro zmírnění CDS tím, že zajišťují, aby kontext poskytovaný LLM zůstal relevantní, stručný a vysoce kvalitní. Pravidelné "resetovací" body nebo explicitní sumarizační prompty mohou pomoci.

Vyvažování nákladů, latence a kvality

Každý token odeslaný do/z Gemini API způsobuje náklady.[1] Ukládání a dotazování dat v MongoDB má také své náklady. Komplexní zpracování kontextu (např. sumarizace dalším voláním LLM, generování vektorových reprezentací, vektorové vyhledávání) přidává latenci.

- **Osvědčené postupy:** Neustálé monitorování nákladů a výkonu. Volba nejjednodušší efektivní strategie správy kontextu. Optimalizace databázových dotazů. Používání kešování (kešování kontextu Gemini API [7], kešování na aplikační úrovni pro načtený kontext). Experimentování s různými sumarizačními

modely/prompty pro vyvážení kvality a nákladů/latence.

Konzistence a spolehlivost výstupů LLM

LLM mohou být nekonzistentní nebo produkovat halucinace.[2] To ovlivňuje kvalitu generovaných shrnutí nebo extrahovaných entit uložených jako kontext.

- **Osvědčené postupy:** Používání technik inženýrství promptů pro konzistenci (šablony, značky, uvažování krok za krokem [24]). Používání nástrojů jako Instructor [31] pro validovaný strukturovaný výstup. Implementace zpětnovazebních smyček pro zdokonalování promptů a strategií generování kontextu.[2]

Škálovatelnost systému správy kontextu

Zajištění, aby schéma MongoDB, strategie indexování a logika načítání byly navrženy tak, aby škálovaly s rostoucím počtem uživatelů a objemem konverzací.

Efektivní správa kontextu je spíše nepřetržitý optimalizační proces než jednorázové nastavení. Výzvy jako CDS [4], kompromisy mezi náklady, latencí a kvalitou [1] a nekonzistence LLM [2] nejsou statické problémy. Chování uživatelů se může měnit, nové verze LLM (jako aktualizace Gemini) se mohou chovat odlišně a povaha konverzací se může vyvíjet. To znamená, že zvolené strategie správy kontextu (sumarizační prompty, prahové hodnoty pro vyhledávání RAG, metody segmentace) bude třeba v průběhu času monitorovat, vyhodnocovat a potenciálně upravovat. Zpětnovazební smyčky [2] jsou pro toto iterativní zdokonalování nezbytné. Vývojáři by proto měli zabudovat mechanismy pro hodnocení efektivity své správy kontextu (např. spokojenost uživatelů, míra dokončení úkolů, relevance načteného kontextu) a být připraveni iterovat na svém návrhu.

9. Závěr a budoucí výhled

Rekapitulace klíčových strategií pro Gemini

- Využití velkého kontextového okna Gemini pro bohaté interakce v rámci jedné relace, avšak s ohledem na optimalizaci.
- Implementace perzistence na aplikační úrovni pomocí MongoDB (s JSON) pro paměť napříč relacemi.
- Použití kombinace technik kondenzace kontextu (sumarizace, selektivní kontext) a obohacení (RAG) k vyvážení délky a kvality kontextu.
- Využití frameworků jako LlamaIndex, LangChain a Instructor k urychlení vývoje.

Vyvíjející se krajina

Kontextová okna LLM se pravděpodobně budou i nadále zvětšovat, ale principy

relevance a efektivity zůstanou prvořadé. Techniky pro automatickou optimalizaci kontextu (např. samokorigující RAG, adaptivní sumarizace) se stanou sofistikovanějšími. Standardy jako MCP mohou hrát větší roli v interoperabilní správě kontextu. Integrace strukturované paměti (databází) s nestrukturovanými interakcemi LLM bude i nadále klíčovou oblastí inovací.

Závěrečná myšlenka

Budování skutečně inteligentní a stavové konverzační AI s Gemini vyžaduje promyšlený přístup ke správě kontextu, přičemž se s ní zachází jako se základní architektonickou komponentou, nikoli jako s dodatečným prvkem. Kombinací schopností Gemini s robustními externími paměťovými řešeními a chytrými optimalizačními technikami mohou vývojáři vytvářet výrazně poutavější a efektivnější uživatelské zážitky.

Citovaná díla

1. LLM Context Windows: Basics, Examples & Prompting Best Practices, použito června 3, 2025, https://swimm.io/learn/large-language-models/llm-context-windows-basics-examples-and-prompting-best-practices
2. Using LLM Models to Build Smarter, Context-Aware AI Chatbots - FastBots.ai, použito června 3, 2025, https://fastbots.ai/blog/using-llm-models-to-build-smarter-context-aware-ai-chatbots
3. Enhancing LLM's Conversations with Efficient Summarization, použito června 3, 2025, https://foojay.io/today/summarizingtokenwindowchatmemory-enhancing-llms-conversations-with-efficient-summarization/
4. Context Degradation Syndrome: When Large Language Models ..., použito června 3, 2025, https://jameshoward.us/2024/11/26/context-degradation-syndrome-when-large-language-models-lose-the-plot
5. LLM Context History - Jaxon, použito června 3, 2025, https://jaxon.ai/llm-context-history/
6. How does an LLM retain conversation memory : r/ollama - Reddit, použito června 3, 2025, https://www.reddit.com/r/ollama/comments/1edan5c/how_does_an_llm_retain_conversation_memory/
7. Long context | Gemini API | Google AI for Developers, použito června 3, 2025, https://ai.google.dev/gemini-api/docs/long-context
8. LLM Limitations: Why Can't You Query Your Enterprise Knowledge ..., použito června 3, 2025, https://memgraph.com/blog/llm-limitations-query-enterprise-data
9. Batch Prediction with Long Context and Context Caching using ..., použito června

3, 2025, https://github.com/google-gemini/cookbook/pull/602
10. Seeking Advice on Memory Management for Multi-User LLM Agent ..., použito června 3, 2025, https://www.reddit.com/r/AI_Agents/comments/1jhub84/seeking_advice_on_memory_management_for_multiuser/
11. Creating AI Memories using Rig & MongoDB - DEV Community, použito června 3, 2025, https://dev.to/joshmo_dev/creating-ai-memories-using-rig-mongodb-2pg7
12. Chat Stores - LlamaIndex, použito června 3, 2025, https://docs.llamaindex.ai/en/stable/module_guides/storing/chat_stores/
13. How do I handle context window limitations when using semantic ..., použito června 3, 2025, https://milvus.io/ai-quick-reference/how-do-i-handle-context-window-limitations-when-using-semantic-search-with-llms
14. langchain-mongodb: 0.6.2 - Read the Docs, použito června 3, 2025, https://langchain-mongodb.readthedocs.io/en/latest/langchain_mongodb/api_docs.html
15. Mongodb - LlamaIndex, použito června 3, 2025, https://docs.llamaindex.ai/en/stable/api_reference/storage/vector_store/mongodb/
16. Get Started with the LlamaIndex Integration - Atlas - MongoDB Docs, použito června 3, 2025, https://www.mongodb.com/docs/atlas/atlas-vector-search/ai-integrations/llamaindex/
17. Effective and Efficient Conversation Retrieval for Dialogue State Tracking with Implicit Text Summaries - arXiv, použito června 3, 2025, https://arxiv.org/html/2402.13043v1
18. Mutual Reinforcement of LLM Dialogue Synthesis and Summarization Capabilities for Few-Shot Dialogue Summarization - arXiv, použito června 3, 2025, https://arxiv.org/html/2502.17328v1
19. A Novel LLM-based Two-stage Summarization Approach for Long Dialogues - arXiv, použito června 3, 2025, https://arxiv.org/html/2410.06520v1
20. Advancing Long-Context LLM Performance in 2025 – Peek Into Two ..., použito června 3, 2025, https://www.flow-ai.com/blog/advancing-long-context-llm-performance-in-2025
21. selective-context · PyPI, použito června 3, 2025, https://pypi.org/project/selective-context/
22. Comprehensive tactics for optimizing large language models for ..., použito června 3, 2025, https://blogs.oracle.com/ai-and-datascience/post/tactics-for-optimizing-large-language-models
23. A Coding Tutorial of Model Context Protocol Focusing on Semantic ..., použito června 3, 2025, https://www.marktechpost.com/2025/04/27/a-coding-tutorial-of-model-context-protocol-focusing-on-semantic-chunking-dynamic-token-management-and-context-relevance-scoring-for-efficient-llm-interactions/

24. Common LLM Prompt Engineering Challenges and Solutions - Ghost, použito června 3, 2025, https://latitude-blog.ghost.io/blog/common-llm-prompt-engineering-challenges-and-solutions/
25. MongoDB - LangChain, použito června 3, 2025, https://python.langchain.com/docs/integrations/document_loaders/mongodb/
26. Get consistent data from your LLM with JSON Schema - Thoughtbot, použito června 3, 2025, https://thoughtbot.com/blog/get-consistent-data-from-your-llm-with-json-schema
27. Structured output | Gemini API | Google AI for Developers, použito června 3, 2025, https://ai.google.dev/gemini-api/docs/structured-output
28. langchain_mongodb.chat_message_histories.MongoDBChatMessageHistory — LangChain 0.2.17, použito června 3, 2025, https://api.python.langchain.com/en/latest/chat_message_histories/langchain_mongodb.chat_message_histories.MongoDBChatMessageHistory.html
29. MongoDBChatMessageHistory - LangChain.js, použito června 3, 2025, https://v03.api.js.langchain.com/classes/_langchain_community.stores_message_mongodb.MongoDBChatMessageHistory.html
30. MongoDB Chat Memory - LangChain.js, použito června 3, 2025, https://js.langchain.com/docs/integrations/memory/mongodb/
31. Instructor - Python Library for Structured LLM Outputs | OpenAI ..., použito června 3, 2025, https://python.useinstructor.com/
32. Context length in LLMs: how to make the most out of it, použito června 3, 2025, https://pieces.app/blog/ai-context-making-the-most-out-of-your-llm-context-length
33. Build multilingual chatbots with Gemini, Gemma, and MCP | Google Cloud Blog, použito června 3, 2025, https://cloud.google.com/blog/products/ai-machine-learning/build-multilingual-chatbots-with-gemini-gemma-and-mcp
34. How to use Anthropic MCP Server with open LLMs, OpenAI or Google Gemini - Philschmid, použito června 3, 2025, https://www.philschmid.de/mcp-example-llama