MVC vs LLM app

Dr. Cristian Kevorchian

Facultatea de Matematică și Informatică

OOP vs SOA

	OOP (Programare Orientată pe Obiecte)	SOA (Arhitectură Orientată pe Servicii)
Structura	Bazată pe obiecte care combină date și funcționalități programate	Bazată pe servicii independente care comunică prin mesaje
Modularitate	Obiectele sunt module reutilizabile	Serviciile sunt module reutilizabile și independente
Scalabilitate	Scalabilitate limitată datorită dependenței de obiecte	Scalabilitate ridicată prin adăugarea de noi servicii
Interacțiune	Obiectele interacționează direct prin metode	Serviciile interacționează prin contracte bine definite
Flexibilitate	Mai puțin flexibilă în integrarea cu alte sisteme	Foarte flexibilă, permite integrarea ușoară cu alte sisteme
Aplicații Web Clasice	Utilizate pentru aplicații monolitice și sisteme integrate	Utilizate pentru aplicații distribuite și microservicii
Aplicații Moderne cu IA	Mai puțin utilizate, dar pot fi integrate în sisteme OOP	Foarte utilizate pentru integrarea serviciilor de IA și API-uri
Gestionarea Datelor	Datele și comportamentul sunt combinate în obiecte	Datele și comportamentul sunt separate în servicii
Reutilizare	Reutilizarea codului prin moștenire și polimorfism	Reutilizarea serviciilor prin expunerea API-urilor
Complexitate	Complexitate crescută în gestionarea obiectelor și relațiilor	Complexitate gestionată prin servicii autonome și independente

Similarități între polimorfism și mecanismul de atenție din LLM

Adaptabilitate la context:

- În polimorfism, un obiect sau metodă își schimbă comportamentul în funcție de context.
- În mecanismul de atenție, distribuția atenției se adaptează dinamic în funcție de contextul frazei sau secvenței analizate, punând accent pe părțile relevante ale intrării.

Reutilizarea aceleiași structuri:

- Polimorfismul permite reutilizarea unei interfețe comune pentru mai multe implementări.
- În LLM-uri, același mecanism de atenție este utilizat în mod repetat pe fiecare strat al rețelei, dar rezultatele variază în funcție de semnificația intrării.

Comportament dinamic:

- Polimorfismul dinamic permite unei metode să aleagă o implementare specifică în timpul execuției.
- Mecanismul de atenție evaluează relațiile dintre cuvinte și își ajustează ponderile pentru a evidenția cele mai relevante părți ale secvenței în timp real.

Diversitatea răspunsurilor:

- În polimorfism, aceeași metodă poate produce rezultate diferite pe obiecte diferite.
- În mecanismul de atenție, același algoritm produce rezultate diferite pentru fiecare intrare, deoarece depinde de structura și semnificația contextului.

	Aplicație Web MVC	Aplicație LLM
Arhitectură	Model-View-Controller	Large Language Model(LLM)
Cazuri de utilizare	Site-uri de e-commerce, CMS-uri, aplicații enterprise	Chatbots, asistenți virtuali, generare de conținut
Dezvoltare	Limbaje server-side (C#, Java, PHP)	Integrare API-uri LLM, ajustare modele
Performanță	Randare pe server, bună pentru SEO	Consum mare de resurse, necesită putere de calcul
Arhitectura pe nivele	Clară (Model, View, Controller)	Integrare complexă a modelului de limbaj
Interactivitate	Mai puțin interactiv, dar stabil	Interacțiuni în timp real, generare de text uman-like

Datele în MVC

Model: În MVC, modelul reprezintă datele și logica de business a aplicației. Modelul gestionează accesul la date, validarea și manipularea acestora. Datele sunt de obicei stocate în baze de date relaționale sau alte forme de stocare persistentă.

View: Vederea este responsabilă pentru prezentarea datelor utilizatorului. Aceasta preia datele din model și le afișează într-un format prietenos pentru utilizator, cum ar fi paginile web sau interfețele grafice.

Controller: Controlerul acționează ca un intermediar între model și view. Acesta gestionează cererile utilizatorilor, manipulează datele prin model și actualizează expunerea datelor.

Datele în LLM

Antrenarea: LLM-urile sunt antrenate pe seturi masive de date textuale. Aceste date provin din diverse surse, cum ar fi articole, cărți, site-uri web și alte texte disponibile public. Scopul este de a ”învăța” structura și regulile limbajului natural.

Pre-antrenare și Fine-tuning: LLM-urile trec prin două etape principale de antrenare:

- Pre-antrenare: Modelul este expus la un volum mare de date textuale pentru a învăța reprezentări generale ale limbajului.
- Fine-tuning: Modelul este ajustat pentru sarcini specifice folosind seturi de date suplimentare și parametri ajustați.

Utilizare: După antrenare, LLM-urile pot genera text, traduce text , răspunde la întrebări și efectua alte sarcini de procesare a limbajului natural. Datele utilizate pentru aceste sarcini sunt procesate în timp real, iar modelul generează răspunsuri bazate pe cunoștințele acumulate în timpul antrenării.

Diferențe cheie

Scop: MVC este o arhitectură de software utilizată pentru a structura aplicațiile web și desktop, în timp ce LLM-urile sunt modele de inteligență artificială destinate procesării și generării limbajului natural.

Gestionarea datelor: În MVC, datele sunt gestionate printr-un model centralizat care interacționează cu baza de date. În LLM, datele sunt utilizate pentru antrenarea modelului și nu sunt stocate în mod tradițional, ci sunt încorporate în parametrii modelului.

Interacțiunea cu utilizatorul: MVC se concentrează pe interacțiunea directă cu utilizatorul prin interfețe grafice, în timp ce LLM-urile interacționează prin generarea de text și răspunsuri automate.

Middleware as a Service un framwork de deployment al Modelelor IA

Middleware as a Service (MaaS) este un concept care facilitează integrarea și implementarea modelelor de inteligență artificială (IA) în diverse aplicații și infrastructuri.

Acesta oferă un cadru pentru dezvoltarea, implementarea și gestionarea modelelor IA, asigurând conectivitatea și interoperabilitatea între diferite componente ale sistemului.

Caracteristici

Conectivitate: MaaS asigură o conexiune eficientă între sursele de date, modelele IA și sistemele de implementare. Utilizează API-uri, SDK-uri și protocoale de internet pentru a livra datele către modelul IA și pentru a distribui rezultatele către sistemele care le utilizează.

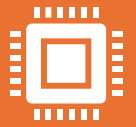
Scalabilitate: Permite scalarea ușoară a aplicațiilor IA prin adăugarea de noi servicii și resurse, asigurând astfel o performanță optimă în funcție de cerințele aplicației.

Flexibilitate: Oferă flexibilitate în integrarea cu alte sisteme și tehnologii, permițând dezvoltatorilor să utilizeze atât soluții open-source, cât și soluții proprietare pentru a construi și implementa modele IA.

Automatizare: Facilitează automatizarea proceselor de antrenare, implementare și monitorizare a modelelor IA, reducând astfel complexitatea și timpul necesar pentru dezvoltarea soluțiilor IA.

Securitate: Asigură securitatea datelor și a modelelor prin implementarea de măsuri de protecție și conformitate cu standardele de securitate

Exemple de Framework-uri și Instrumente



IBM Watsonx.ai: Oferă o selecție de modele și servicii middleware pentru dezvoltarea, implementarea și monitorizarea agenților IA ca seturi de API-uri gata de implementare în producție

Azure Semantic Kernel: Un framework care oferă modele abstracte și integrări pre-constituite pentru dezvoltarea rapidă a aplicațiilor IA

Numurus NEPI: O platformă open-source pentru dezvoltarea soluțiilor de automatizare robotică și IA la nivel edge computing, oferind interfețe utilizator, drivere hardware și conectivitate IoT

DEVOPS vs MLOPS

	DevOps	MLOps
Definiție	Practică ce integrează dezvoltarea software (Dev) și operațiunile IT (Ops) pentru livrare continuă.	Practică ce integrează dezvoltarea modelelor de ML, operațiunile și gestionarea ciclului de viață ML.
Scop	Automatizarea și eficientizarea procesului de dezvoltare, testare și implementare a aplicațiilor software.	Automatizarea și eficientizarea procesului de creare, antrenare, implementare și monitorizare a modelelor ML.
Flux de lucru	Include gestionarea codului, testarea, CI/CD (Continuous Integration/Continuous Deployment).	Include gestionarea datelor, antrenarea modelelor, validarea, CI/CD și monitorizarea modelelor.
Componenta principală	Codul aplicației.	Codul + datele + modelul antrenat.
Ciclul de viață	Codul sursă → Build → Testare → Implementare → Monitorizare.	Date → Preprocesare → Antrenare model → Validare → Implementare → Monitorizare.
Instrumente comune	Jenkins, Git, Docker, Kubernetes, Ansible, Terraform.	MLflow, TensorFlow Extended (TFX), Kubeflow, Airflow, DVC, Seldon Core.
Monitorizare	Se concentrează pe sănătatea aplicației (uptime, performanță, erori).	Include monitorizarea performanței modelului (drift-ul datelor, acuratețea modelului).
Provocări	- Gestionarea complexității aplicațiilor. - Menținerea unui ciclu CI/CD continuu.	- Drift-ul datelor și al conceptelor. - Reantrenarea automată a modelelor pe noi seturi de date.
Echipe implicate	Dezvoltatori, ingineri DevOps, administratori de sistem.	Cercetători ML, ingineri de date, ingineri MLOps.
Focalizare pe date	Nu include prelucrarea sau gestionarea datelor.	Include procesarea, etichetarea și gestionarea datelor.
Versiuni	Gestionează versiuni ale codului și aplicațiilor.	Gestionează versiuni pentru cod, date și modele.
Automatizare	CI/CD pentru aplicații software.	CI/CD + CT (Continuous Training) pentru modele ML.
Complexitate	Mai puțin complex, deoarece fluxurile se concentrează pe cod.	Mai complex, implicând interacțiunea dintre date, cod și modele.

DEMO