C U R S 1

Introducere

Planul cursului și modele AI

Ingineria Ai – Aplicații cu agenți inteligenți
Analiza Datelor Complexe, Facultatea de Sociologie și Asistență Socială, Universitatea Babeș Bolyai

CURS 1.

C1 LLM-uri, API-uri - ce construim și cum pornim
C2 Ecosistemul de modele - alegem modelul de bază
C3 Date și corpus - colectare, curățare, metadate
C4 Prompting, adnotare, context engineering
C5 Embeddings, retrieval, RAG
C6 RAG, agenți, LangChain, LangGraph
C7 Agenți orchestrați - LangGraph,metrici,etică
C8 Integrare aplicație - Gradio
C9 Demo final - prezentări și feedback

Tematică

- Introducere în curs: ce construim, cum lucrăm, cum suntem evaluați
- Ce este un LLM și cum îl folosim în practică
- Cum accesăm un model: API vs local
- Cum funcționează un apel la model
- Tokeni, context și cost
- Parametrii de generare: temperature, top-p, max tokens, seed

Activități practice

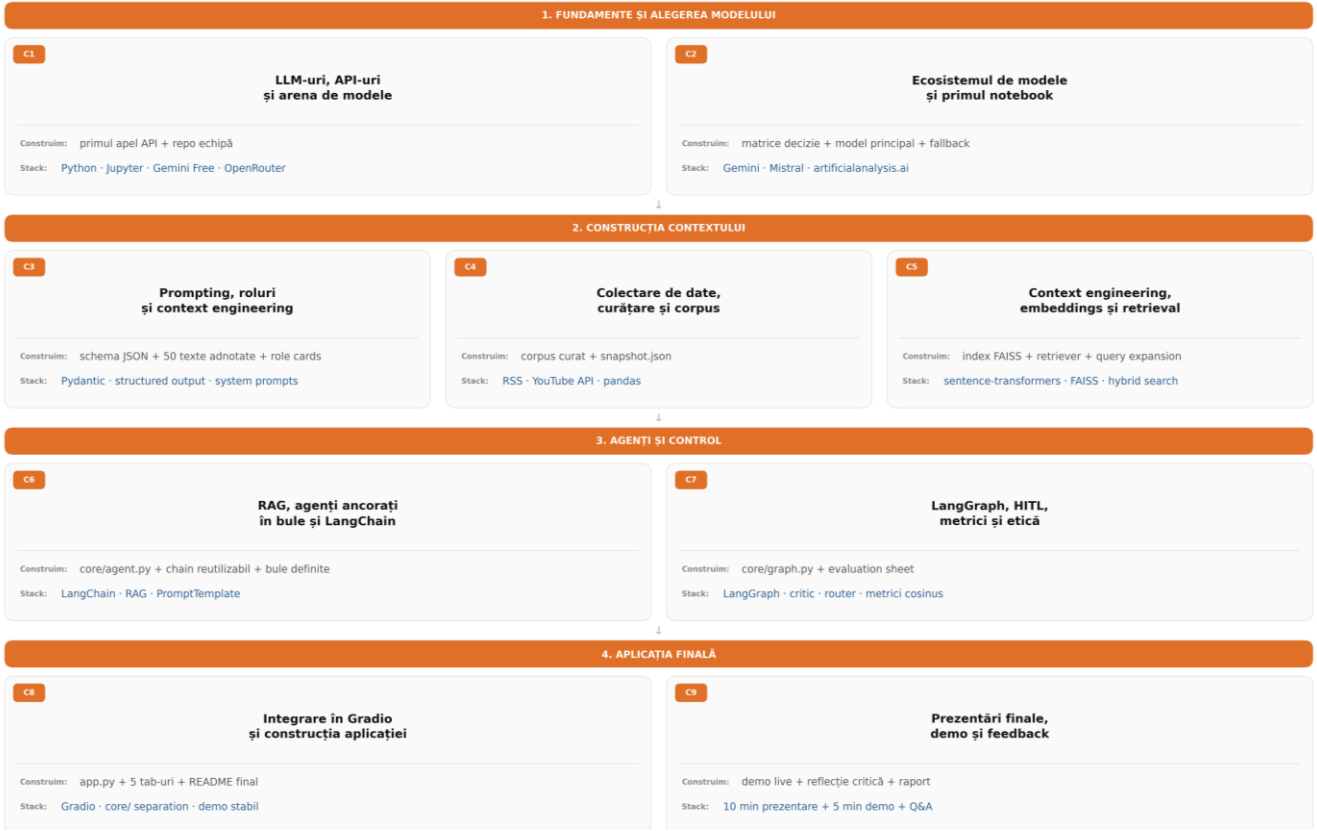
- Setup API key și configurare .env
- Primul apel la model din notebook
- Exercițiu: rezumat neutru al unei știri scurte
- Experiment pe temperatură: același prompt, valori diferite
- Formare echipe
- Inițializare repo de echipă

Livrabile

- Notebook funcțional cu primul apel
- Experiment documentat pe temperatură
- Un rezumat neutru reușit
- Repo de echipă cu README, .env.example, notebook introductiv

→ La finalul C1 trebuie să știi să apelezi un model, să controlezi răspunsul și să salvezi un experiment simplu, reproductibil.

Planul cursului



Competențe dezvoltate

Temă	Practică în curs	Aplicații profesionale
Integrare AI în aplicații	API-uri, modele LLM, aplicații simple	aplicații interne, automatizare, instrumente digitale
Lucru cu date text	colectare, curățare, structurare, metadate	data analyst, research analyst, AI data specialist
Prompt engineering aplicat	instrucțiuni, formate, consistență, testare	configurare chatboturi, suport, operațiuni digitale
RAG și căutare semantică	răspunsuri pe documente și surse verificabile	document QA, knowledge management, legal, HR
Agenți AI și fluxuri automate	pași, reguli, verificări, instrumente	automatizare de procese, workflow design
Evaluare și control al calității	erori, limite, comparații, validare umană	QA pentru AI, evaluare modele, audit intern
Prototipare rapidă	notebook-uri, aplicație Gradio, demo final	portofoliu, freelancing, proiecte organizaționale
Guvernanță AI	bias, date personale, trasabilitate, control uman	conformitate, audit, responsabilitate digitală

STACK-UL CURSULUI

NUCLEUL OBLIGATORIU

Acces la modele: Python, Jupyter, Gemini Free sau OpenRouter Free
RAG: sentence-transformers și FAISS
Orchestrare: LangChain și LangGraph
Interfață: Gradio

OPȚIONAL / DEMO

Ollama — alternativă locală pentru modele
Chroma — alternativă explicată pentru vector store
n8n — demo pentru automatizare și fluxuri externe
MCP — demo conceptual pentru tool access și standardizare

EVALUARE FINALĂ

EchoChamber

STRUCTURA NOTEI

1p Oficiu
1p Prezență la seminarii – punct bonus
1p Activitate seminar
3p Teme individuale (3 × 1 punct)
5p Proiect de echipă
___
11p Total

PROIECT - 5 puncte

3p CONTRIBUȚIE INDIVIDUALĂ

10 issues closed cu commit → 3p
7 – 9 → 2p
4 – 6 → 1p
sub 4 → 0p
+ mesaje clare, activitate regulată
⚠ 0 commits = 0p indiferent de echipă

1p DEMO LIVE

· agentul tău rulează live
· explici role card-ul și corpusul
· comentezi un rezultat din metrici

1p CALITATEA PRODUSULUI (notă comună)

· agenți funcționali, corpus + bule construite, vectorstore rulează, UI, inovație

Proiect final
Aplicație interactivă – EchoChamber

CE ESTE

- Aplicație interactivă cu agenți AI
- Simulează bule discursive pe comentarii reale
- Fiecare agent răspunde dintr-o perspectivă diferită

CE POȚI FACE ÎN APLICAȚIE

- vezi cum răspund toți agenții la același stimulus
- discuți cu un agent (chat)
- rulezi dezbatere multi-agent (runde)

DE CE O CONSTRUIM

- să înțelegem cum influențează contextul răspunsul
- să vedem diferențe între perspective
- să construim un sistem AI complet, nu doar prompturi

CUM FUNCȚIONEAZĂ

Stimulus
știre sau întrebare

↓

Retrieval
comentarii similare din corpus

↓

Agent
rol + stil discursiv

↓

Răspuns
generat cu context real

CE COMPONENTE ARE

Corpus
comentarii reale colectate și curățate

Vector store
căutare semantică cu FAISS

Retriever
selectează contextul relevant

Agenți
roluri discursive diferite

UI — Gradio
interfață demonstrabilă local

EchoChamber Studio

discursive bubble simulation · romanian political comments · each agent speaks from its community's perspective

LLM Model

DeepSeek Chat

Agents | News | Debate

All agents read the **same stimulus** and respond independently. **Metrics** compares their responses to a neutral baseline.

STIMULUS

Daca mai traia Ceausescu, nu mai existau bule discursive nici social media!

▶ RUN AGENTS

⚙ METRICS

🙏 PERSONALIST-SALVATOR

TEXTBOX

Doamne ajută că avem azi un om precum domnul Călin Georgescu, care vede limpede unde ne-au dus 35 de ani de trădare națională, nu să ne întoarcem la vremuri trecute, ci să construim o Românie suverană, curată și demnă, așa cum o merită poporul adevărat.

😤 ANTI-SISTEM

TEXTBOX

Asta e vorba aia, acum fură în numele democrației și libertății de exprimare, pe-atunci măcar făceau și ceva pe lângă furat. Ne-au dat internet să urlăm în gol, în timp ce ei își fac veacurile prin vile și pensii speciale.

🕵️ CONSPIRAȚIONIST

TEXTBOX

Exact, acum se vede clar că niciun sistem de control nu e întâmplător, fie că e vorba de cenzura comunistă sau de algoritmii care șterg discret vocile incomode, doar că unii preferă să creadă că libertatea asta digitală e reală, când de fapt cineva trage sforile din umbră.

🔍 PRO-EUROPEAN

TEXTBOX

Comparația asta cu Ceaușescu e o fugă de realitate. În România de azi, există instituții care funcționează, presă liberă și reguli clare garantate de statul de drept, tocmai pentru că am ales o cale europeană. A regreta un regim care interzicea orice dezbatere publică înseamnă a ignora lecțiile istoriei și angajamentele noastre democratice.

CONCEPT

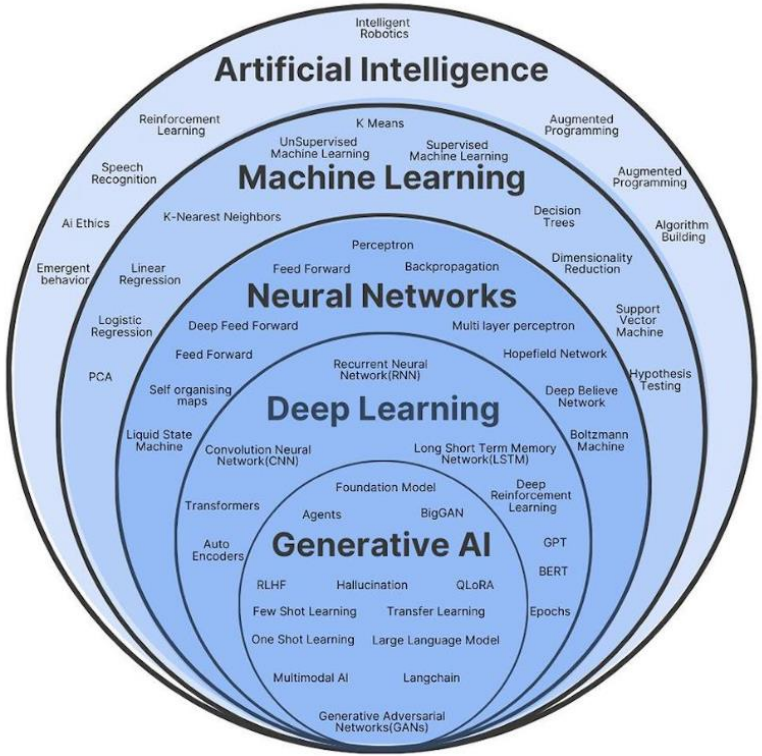
Ce este Generative AI (GenAI)?

Generative AI - sisteme de inteligență artificială care învață tipare și structura statistică a datelor și apoi produc ieșiri noi - text, imagini, audio, video sau cod - compatibile cu acele tipare învățate (Stanford HAI, 2026; OECD, 2026).

Ideea de bază este: învață distribuția / regularitățile datelor → generează noi instanțe plauzibile din aceeași familie de date, nu doar recunoaște sau etichetează exemple existente (Stanford HAI, 2026).

Cele mai importante familii tehnice asociate cu GenAI sunt **Transformers, diffusion models, GANs și VAEs** (Stanford HAI, 2026).

Față de învățarea automată tradițională, care este de regulă optimizată pentru sarcini specifice precum clasificare, predicție, detecție sau recomandare, GenAI este proiectat în primul rând pentru producerea de conținut nou, dar poate fi folosit și pentru sarcini analitice (Stanford HAI, 2026).



Ce putem face cu Generative AI

Generare de conținut nou: redactare de texte, rezumate, titluri, imagini, cod, materiale didactice sau date sintetice. GenAI este definit tocmai prin capacitatea de a produce ieșiri noi pe baza tiparelor învățate din date.

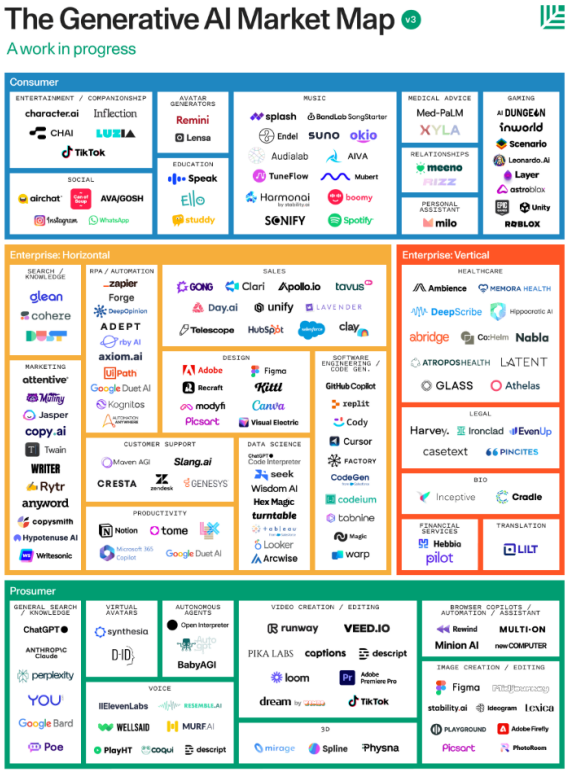
Transformare de text: sumarizare, rescriere, traducere și adaptare de stil, de exemplu transformarea unui articol într-un rezumat neutru sau într-un text accesibil pentru un public non-tehnic. Aceste utilizări rămân generative, chiar dacă scopul este mai degrabă funcțional decât creativ.

Clasificare și adnotare: etichetare de comentarii după poziție (stance), ton, temă, țintă(target) sau alte categorii. Aici modelul este generativ ca arhitectură, dar este folosit operațional ca instrument de clasificare.

Extracție structurată de informații: extragere de actori, afirmații, date, instituții, locații sau relații și returnarea lor în format JSON sau tabel- „generative information extraction”.

Întrebări și răspunsuri pe documente (RAG): căutare de fragmente relevante într-un corpus și generarea unui răspuns bazat pe surse - chatboți pe PDF-uri, arhive, rapoarte și proceduri interne.

Agenți cu tool-uri și memorie: sisteme care folosesc contextul conversației, instrumente externe și pași multipli de lucru. Memoria permite păstrarea informației între interacțiuni și susține procese mai complexe.



https://sequoiacap.com/article/generative-ai-act-two/

Cum funcționează Generative AI

FLUX: PRETRAINING → ÎNVĂȚAREA REPREZENTĂRILOR → GENERARE → ADAPTARE → ALINIERE

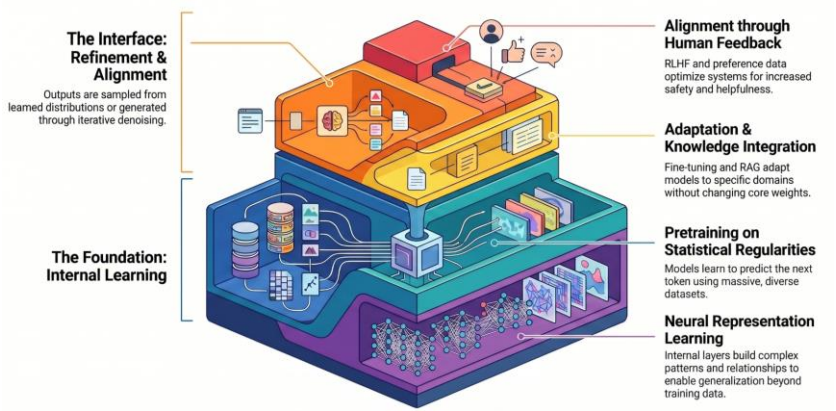
Pretraining: modelul este antrenat pe seturi foarte mari de date pentru a învăța regularități statistice; pentru majoritatea LLM-urilor, asta înseamnă să învețe să prezică **următorul token** dintr-o secvență.

Învățarea reprezentărilor: prin straturile rețelei neuronale, modelul construiește reprezentări interne ale tiparelor, contextului și relațiilor, astfel încât poate generaliza dincolo de exemplele exacte văzute în antrenare.

Generare (inferență): un prompt condiționează iesirea (output), iar modelul generează prin eșantionare din distribuțiile de probabilitate învățate; în text acest lucru se face de obicei token cu token

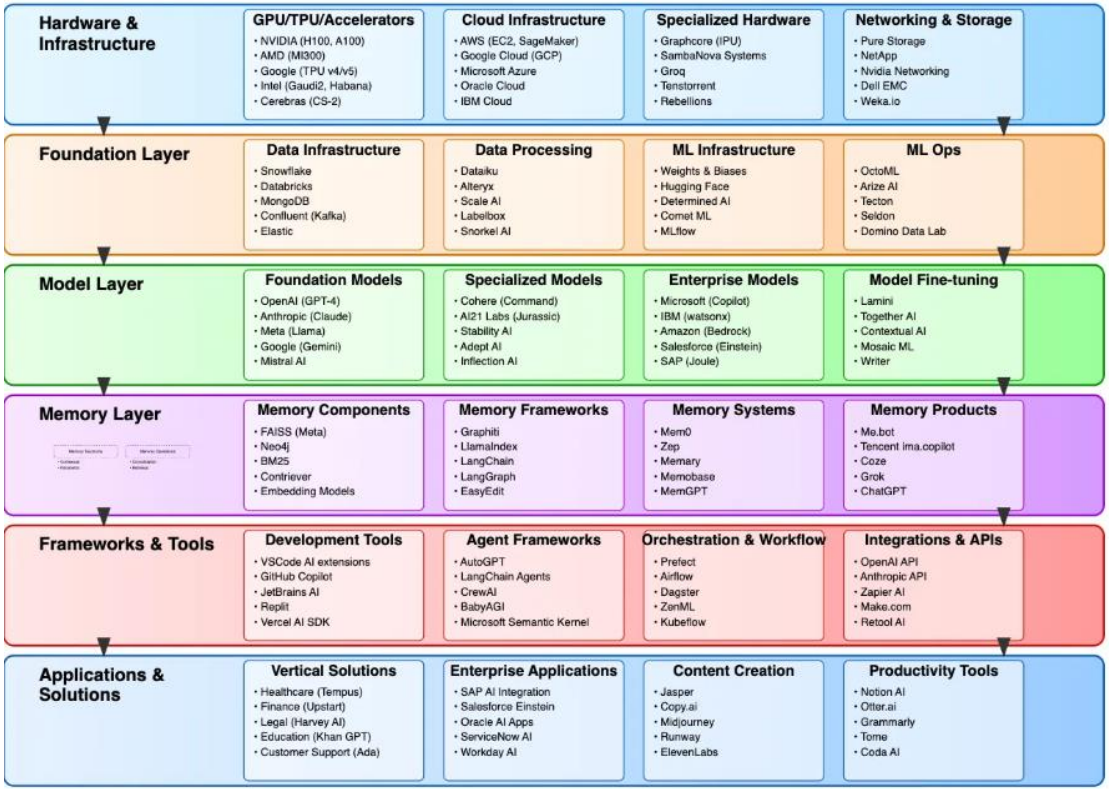
Adaptare: modelele preantrenate pot fi adaptate prin **instruction tuning / supervised fine-tuning** pe seturi de date mai mici și mai specializate, iar **RAG** adaugă cunoaștere externă la inferență fără a modifica greutățile modelului.

Aliniere / feedback: multe modele conversaționale sunt optimizate suplimentar cu **date de preferință umană**; în RLHF (reinforcement Learning from Human Feedback) , judecățile umane ajută la antrenarea unui model de recompensă, iar reinforcement learning împinge sistemul spre răspunsuri mai sigure.



The Complete AI Market Ecosystem: From Hardware to Applications

Analysis by Gennaro Cuofano



"The AI stack has evolved from infrastructure to applications, with memory serving as the critical bridge connecting models to real-world use cases."

https://businessengineer.ai/p/the-complete-ai-market-ecosystem

CONCEPT

Nivelele de autonomie AI

Haase & Pokutta (2025) — cadrul didactic al cursului

L0 **Tool**
Modelul răspunde la o întrebare. Nicio autonomie.

L1 **Role**
Modelul are o identitate definită și răspunde consistent în personaj.

L2 **Agent-like**
Poate folosi tools, caută documente, ia decizii simple.

L3 **Fully Agentic**
Planifică singur pașii, decide ce tool să folosească, iterează.

L4 **Multi-Agent**
Mai mulți agenți cu roluri diferite interacționează și se coordonează.

L5 **Dinamici emergente**
Comportamente neprogramate apar din interacțiunile repetate.

Haase, J. & Pokutta, S. (2025). Beyond Static Responses: Multi-Agent LLM Systems. arXiv:2506.01839

LLM, Agenți AI, Agentic AI

LLM

- Model de limbaj antrenat pe cantități mari de text; generează răspunsuri token cu token prin predicție probabilistică.
- Primește un prompt și produce un rezultat (output) - nu memorează, nu acționează, nu are acces la informații în timp real.
- Poate rezuma, clasifica, extrage informații, traduce, genera cod sau răspunde la întrebări.

AGENȚI AI

- Sisteme construite pe un LLM conectat la instrumente (tools) , memorie sau date externe - LLM-ul rămâne motorul de raționament.
- Pot executa acțiuni: căutare web, interogare baze de date, apeluri API, generare de cod.
- Exemplu: chatbot cu search, sistem RAG, agent de calendar.

AGENTIC AI

- Sisteme cu autonomie extinsă: planifică pași, aleg singuri instrumentele și iterează până ating obiectivul - fără intervenție umană la fiecare pas.
- Pot coordona mai mulți agenți și corecta rezultate intermediare. Exemplu: research agent, fact-checking automat, simulare multi-agent.

LIMITĂRI COMUNE

Halucinații · cunoaștere înghețată · context finit · bias din date · comportament greu de auditat la sisteme autonome

→ LLM = motorul · agentul = sistemul care îl folosește · Agentic AI = autonomie orientată spre obiective

Transformer Models

Transformer-ul original combină encoder + decoder și a introdus mecanismul de self-attention, care permite modelului să „cântărească” ce părți din secvență sunt relevante pentru fiecare token.

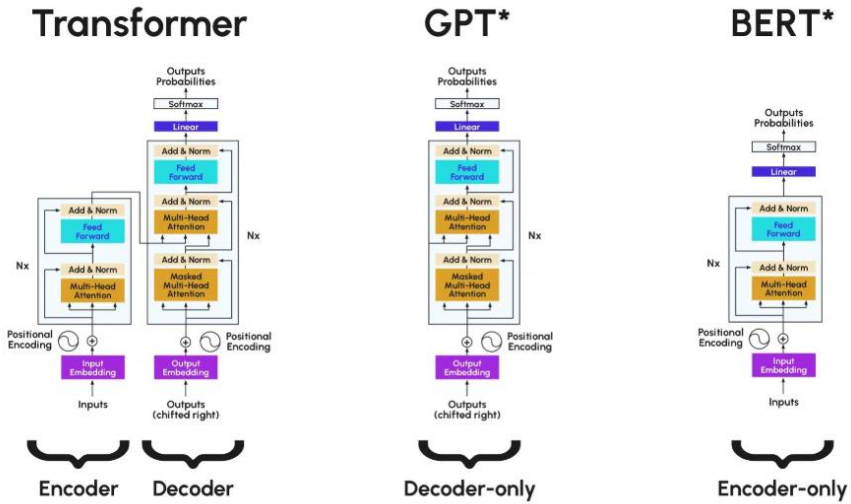
GPT folosește doar partea de decoder și este optimizat pentru next-token prediction; de aceea este foarte bun la generare de text, dialog, completare și agenți conversaționali.

BERT folosește doar partea de encoder și citește contextul bidirecțional; de aceea este foarte bun la înțelegere, clasificare, căutare semantică și extragere de informații.

GPT produce, BERT interpretează, iar Transformer-ul este arhitectura de bază din care au derivat ambele.

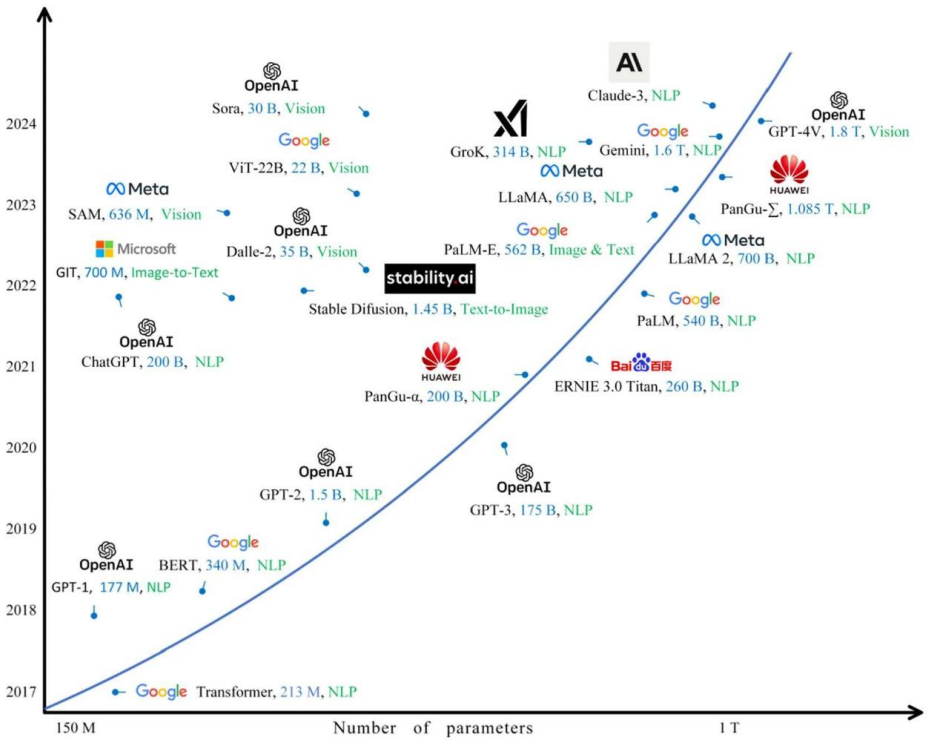
Pentru cursul nostru
Când folosim chatboturi, RAG sau agenți, lucrăm de regulă cu modele de tip GPT / decoder-based LLMs.

Când facem clasificare, adnotare, reprezentări (embeddings) sau recuperare (retrieval) , logica este mai apropiată de zona encoder / understanding.



https://liora.io/en/all-about-transformers-models

Parametrii modelului LLM



An overview of large AI models and their applications
DOI: 10.1007/s44267-024-00065-8

Parametrii sunt valorile interne pe care modelul le învață în timpul antrenării. (weights + biases)
În general, mai mulți parametri = capacitate mai mare de a surprinde relații complexe și de a urma instrucțiuni mai bine.

Modelele mai mari cer și mai multă memorie, timp de răspuns mai mare și cost mai mare.

Mai mare nu înseamnă automat mai bun: contează și datele de antrenare, arhitectura, fine-tuning-ul și sarcina (task) concretă.

Cum mă ajută în alegerea modelului

- Dacă vreau viteză, cost mic și rulare ușoară, aleg un model mai mic.
- Dacă vreau calitate mai bună pe sarcini complexe (raționare, instrucțiuni lungi, analiză mai fină), aleg un model mai mare.
- Alegerea bună nu este „cel mai mare model”, ci cel mai potrivit model pentru task + buget + hardware.

Exemplu practic

- Model mai mic (ex. 7B–8B) → bun pentru clasificare simplă, prototipuri, cost redus.
- Model mai mare (ex. 70B+) → mai bun pentru reasoning, generare mai coerentă și taskuri mai dificile, dar mai scump și mai greu de rulat.

CONCEPT

Tokeni, context și cost

https://tokencalculator.ai/
https://platform.openai.com/tokenizer

CE SUNT TOKENII

- Token ≠ cuvânt - un token poate fi o silabă, un cuvânt, semn de punctuație sau mai mult
- ~¾ dintr-un cuvânt în medie pentru engleză, mai mult pentru română
- Tokenization = procesul prin care textul este împărțit în tokeni
- LLM-ul citește și generează token cu token
- Context window = suma tuturor tokenilor din conversație
- Mai mult context nu înseamnă automat răspuns mai bun

COST ȘI VITEZĂ

- Costul depinde de (input tokens + output tokens)
- Gemini Flash 2.0: gratuit, 1.500 apeluri/zi
- Prompturile mai lungi sunt de regulă mai lente și mai scumpe
- Latența = timp până la primul token (TTFT)
- Limita de context se aplică totalului: prompt + istoric + output

→ **Tokenii sunt moneda sistemului - costul, viteza și limitele se exprimă în tokeni**

Referință: platform.openai.com/tokenizer · ai.google.dev/gemini-api/docs/models

Parametrii de generare

Un LLM e controlat prin parametri + context + format - nu doar prin prompt

`temperature`

ca termostat al creativității

Controlează cât de previzibil sau de surprinzător este răspunsul. La 0, modelul alege mereu cel mai probabil cuvânt următor. La 1.5 - foarte variat, creativ, dar mai instabil

0 → răspuns identic la fiecare rulare | 1.5 → variat, creativ, uneori imprecis

`top-p`

ca pâlnie care filtrează cuvintele

Modelul calculează probabilități pentru toate tokenurile posibile, apoi păstrează cele mai probabile tokenuri până când suma lor ajunge la p. La p = 0.9, modelul alege doar din tokenurile care acoperă împreună 90% din probabilitate; restul sunt eliminate.

0.1 → vocabular foarte restrâns | 1.0 → aproape fără filtrare top-p

`max_output_tokens`

ca foarfecă pe un text lung

Setează lungimea maximă a răspunsului. Dacă modelul nu a terminat fraza, aceasta este tăiată brusc. Util ca protecție împotriva răspunsurilor uriașe și costisitoare.

256 → răspuns scurt, economic | 4096 → documente lungi, cod complet

`context window`

ca fereastră pe o masă de lucru

Tot textul pe care modelul îl poate citi simultan - prompt, conversație, documente. Ce iese din fereastră nu mai este disponibil direct în acel apel. Nu este memorie pe termen lung.

8k tokens ≈ ~6 pagini A4 | 2M tokens ≈ o carte întreagă

`seed`

ca rețetă cu ingrediente fixe

Un număr „secret” care ancorează aleatoriul modelului. Același seed cu același prompt produce mereu același răspuns. Esențial pentru teste și evaluări comparative.

fără seed → răspuns diferit la fiecare rulare | seed fix → mai reproductibil

`structured output`

ca matriță care impune forma exactă

Forțează modelul să răspundă exclusiv în formatul JSON specificat. Fără introduceri, fără text în plus - doar structura exactă cerută. Indispensabil în pipeline-uri automate.

False → răspuns narativ liber | True → JSON valid garantat

`frequency_penalty`

ca taxă pe repetiție

Penalizează cuvintele care au apărut deja în răspuns. Cu cât un cuvânt a fost folosit mai des, cu atât e mai puțin probabil să apară din nou. Încurajează modelul să varieze vocabularul

0 → repetițiile sunt permise liber | 2.0 → vocabular forțat variat

`presence_penalty`

ca taxă pe prezență

Penalizează orice cuvânt care a apărut cel puțin o dată, indiferent de frecvență. Împinge modelul spre teme și idei noi, nu spre variații ale aceluiași subiect.

0 → modelul poate reveni la aceleași teme | 2.0 → forțează diversitate

`stop`

ca semafor roșu în text

Definești un șir de caractere (ex. '###' sau 'END') la care modelul se oprește imediat. Util în pipeline-uri unde știi exact unde se termină răspunsul util.

ex. stop=["#"] → taie la primul # | fără stop → modelul scrie până la max_tokens

Ai Playgrounds

Google AI Studio - https://aistudio.google.com/
https://ai.google.dev/gemini-api/docs/models - lista de modele
playground oficial pentru Gemini; are **Run settings** pentru parametrii modelului, safety settings și tool-uri precum structured output, function calling, code execution și grounding.

HuggingFace Playground – https://huggingface.co/playground
Interfață de test pentru modele disponibile pe HuggingFace

OpenAI Playground - https://platform.openai.com/playground
playground pentru testarea prompturilor și a setărilor de generare; documentația OpenAI tratează explicit corespondența dintre Playground și API prin parametri precum `temperature`, `top_p`, `max_tokens`.

Anthropic Workbench — https://console.anthropic.com/workbench
interfață de testare pentru prompturi în Console; poți seta modelul, `temperature` și `max tokens`, apoi exporta promptul ca exemplu de cod.

Mistral Studio - https://console.mistral.ai/
studio-ul Mistral pentru API și playground; documentația îl separă clar de Le Chat și spune explicit că poți testa modele în playground, ajusta parametri și compara outputuri.

Cohere Playground - https://dashboard.cohere.com/playground
developer playground pentru **Chat** și **Embed**; are `System message`, `JSON Mode`, function/tool use, `temperature`, `seed` și chiar reasoning în panoul de parametri.

Github Playground https://github.com/marketplace/models
marketplace + playground pentru testarea mai multor modele

Exercitiu de clasa

aistudio.google.com/prompts/new_chat

1. Deschide AI Studio si scrie promptul de test

Mergi la link-ul de sus. In campul de chat scrie exact: "Descrie un apus de soare in 2 propozitii." Modelul selectat este Gemini 3 Flash Preview.

2. Testeaza Temperature = 0 (raspuns inghetat)

In panoul Run settings (dreapta), trage sliderul Temperature pana la 0. Apasa Run de 3 ori. Raspunsul este identic la fiecare rulare — modelul nu improvizeaza.

3. Creste Temperature la 1.5 (raspuns liber)

Muta sliderul Temperature la 1.5. Apasa Run de 3 ori. Fiecare raspuns e diferit — uneori poetic, uneori bizar. Aceasta e creativitatea modelului.

4. Limiteaza raspunsul din Advanced settings

Deruleaza in jos in Run settings si deschide Advanced settings. Seteaza Output token limit la 20. Raspunsul este taiat brusc

5. Activeaza Structured outputs

In sectiunea Tools din Run settings, activeaza toggle-ul Structured outputs si apasa Edit. Adauga schema: { "culori": string[], "emotie": string }. Modelul returneaza JSON pur.

Ce sa urmaresti

Toate setarile sunt in panoul Run settings, in dreapta ecranului.

`Temperature slider`
Merge de la 0 (stabil) la 2 (haos). Il gasesti direct in Run settings, imediat vizibil.

`Structured outputs`
Toggle in sectiunea Tools. Apasa Edit pentru a defini schema JSON dorita.

`Advanced settings`
Sectiune pliabila la baza panoului Run settings. Contine Output token limit.

`System instructions`
Casuta separata in Run settings. Aici dai rolul sau contextul modelului.

`concluzie`
Parametrii schimba comportamentul fara a modifica promptul — putere mare.

Cum accesăm un model - API vs local

CLOUD / API — RECOMANDAT PENTRU CURS

- Accesezi modelul printr-un endpoint REST
- Fără setup hardware, ușor de integrat în Python
- Gratuit cu limite zilnice: Gemini Free, OpenRouter
- Recomandat: Gemini Flash 2.0 sau orice model prin OpenRouter

LOCAL — OPȚIONAL, PENTRU EXPERIMENTARE

- Rulezi modelul pe hardware propriu cu Ollama sau LM Studio
- Necesită GPU sau CPU puternic (8+ GB RAM)
- Fără costuri per cerere, fără quota
- Recomandat doar dacă ai hardware - altfel rămâi pe API

https://medium.com/@georgije.stanisic00/local-vs-cloud-llms-whats-the-best-choice-317a9cbdfd21

→ **Dacă poți rula un apel API în Python și primești răspuns — ești gata**

Resurse: ai.google.dev · openrouter.ai · ollama.com · github.com/cheahjs/free-llm-api-resources,

Function calling

Function calling este mecanismul prin care un LLM poate decide că are nevoie de un tool extern, nu doar de text, și returnează un apel structurat cu numele funcției + argumentele necesare.

Modelul nu execută singur funcția. Aplicația ta sau backend-ul execută instrumentul, apoi trimite rezultatul înapoi modelului, iar modelul formulează răspunsul final pentru utilizator.

Nu este încă un agent complet; este o capabilitate de „tool use" care stă la baza multor sisteme agentice. Un agent apare când adaugi pași multipli, decizie, stare sau memorie.

Exemple Gemini

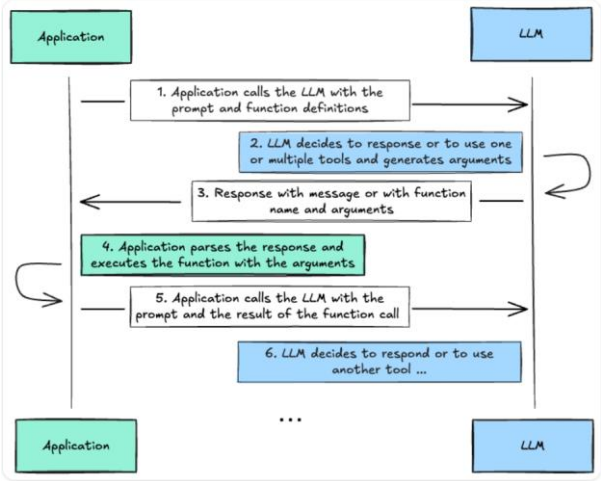
- **Augment knowledge**: acces la baze de date, API-uri, knowledge bases.
- **Extend capabilities**: calculator, chart-uri, code execution.
- **Take actions**: programare întâlniri, trimitere emailuri, control smart home.

Exemplu simplu

- User: „What is the weather in Cluj-Napoca?"
- Model: propune `getWeather(city="Cluj-Napoca")`
- Aplicația execută funcția
- Modelul transformă rezultatul într-un răspuns natural pentru utilizator.

Ideea-cheie

→ **Function calling = modelul decide ce tool trebuie folosit; aplicația execută acțiunea.**



https://huggingface.co/docs/hugs/guides/function-calling

Structured output

- Un LLM poate genera răspunsuri într-un format bine definit, de exemplu **JSON, XML sau câmpuri fixe**.
- Este util atunci când outputul trebuie să fie **citit de alte sisteme**, nu doar de oameni.
- Crește **predictibilitatea și consistența** răspunsurilor, mai ales în pipeline-uri automate.

De ce contează

- LLM-urile sunt probabilistice și pot varia formularea răspunsului.
- Outputul structurat reduce variația inutilă și face rezultatele mai ușor de validat.
- Este important pentru **automatizare, extracție de date, function calling și stocare în baze de date**.

Exemplu de prompt-based formatting.
„Returnează vremea pentru Cluj-Napoca în format JSON, cu câmpurile `city`, `temperature`, `unit` și `condition`.”

Exemplu de răspuns

```
{
  "city": "New York",
  "temperature": 22,
  "unit": "C",
  "condition": "sunny"
}
```

```
{
 "type": "object",
 "properties": {
  "tema": {
   "type": "string"
  },
  "ton": {
   "type": "string"
  },
  "incredere_in_institutii": {
   "type": "string",
   "enum": [
    "ridicata",
    "medie",
    "scazuta"
   ]
  }
 },
 "required": [
  "tema",
  "ton",
  "incredere_in_institutii"
 ],
 "propertyOrdering": [
  "tema",
  "ton",
  "incredere_in_institutii"
 ]
}
```

Open-source LLMs nu sunt „gratuite”

Some simulations on how costs can spiral out of control.

- Descărcarea modelului poate fi gratuită, dar costurile reale apar după aceea: infrastructură, integrare în produs, monitorizare, actualizări și suport tehnic.
- Chiar și pentru un sistem intern relativ mic, costurile anuale pot deveni mari, deoarece ai nevoie de GPU-uri, storage, loguri, rețea și oameni care să țină sistemul funcțional.
- O mare parte din cost nu vine din model, ci din echipă: ML engineers, MLOps, software engineers, evaluare și uneori experți de domeniu pentru validarea rezultatelor.
- Open-source oferă control mai mare, flexibilitate și uneori mai multă confidențialitate, dar cere mai multă muncă de operare, debugging, securizare și mentenanță.

https://machine-learning-made-simple.medium.com/the-costly-open-source-llm-lie-f83fdc5d5701

API-uri cloud disponibile

- OpenRouter - 20+ modele gratuite: Llama 3.3 70B, Gemma 3, Mistral Small
- Google AI Studio - Gemini Flash 2.0 · 1.500 cereri/zi gratuit
- Groq - inferență ultra-rapidă, Llama 4, Qwen
- Cerebras - Llama 3.3 70B și Qwen 3 cu latență mică
- GitHub Models - 40+ modele: GPT, Claude, DeepSeek-R1
- Cloudflare Workers AI - 60+ modele, 10.000 neuroni/zi
- Mistral (La Plateforme) - modele proprietare și open-source
- Cohere - Command models, 1.000 cereri/lună

Resurse gratuite

github.com/cheahjs/free-llm-api-resources

Anatomia unui apel la model

CELE 4 COMPONENTE ALE ORICĂRUI APEL

```
model       = 'gemini-2.0-flash'
messages    = [{'role':'user','content':'...'}]
temperature = 0.3
max_tokens  = 512
```

Componentă	Ce face	Unde îl controlezi
model	Alegi ce motor rulează	parametru în apel
messages	Instrucțiuni + întrebare + istoric	system / user / assistant
parametri	Controlezi comportamentul generării	temperature, max_tokens, seed
output	Text / JSON / usage stats	.content, .usage.total_tokens

→ **Orice apel are exact aceste 4 componente — indiferent de model sau provider**

Greșeli tipice în primul notebook

Greșeală	Simptom	Fix
Cheia API în cod	Cheie vizibilă în repo / commit	Citește din os.environ sau .env
Nume model greșit	AuthenticationError sau 404	Copiază exact din documentația modelului
Ratelimit 429	Eroare după câteva apeluri	sleep(1) între apeluri, max 60/min
Prompt bun, parametri greșiți	Output tăiat sau prea variat	Verifică max_tokens și temperature
Output prea lung — taiat silențios	Răspuns incomplet fără avertisment	Mărește max_tokens sau scurtează inputul
.env fără .gitignore	Cheia ajunge pe GitHub	Adaugă .env în .gitignore înainte de orice commit

→ **Cel mai comun motiv de blocare la laborator este cheia API greșit configurată**

Primul notebook, Github Teams

- Creare cheie API — Gemini (ai.google.dev) sau OpenRouter (openrouter.ai)
- Setup notebook: citim cheia din .env, alegem modelul și facem primul apel
- Exercițiu: rezumat neutru al unei știri scurte (max. 60 de cuvinte)
- Experiment pe temperatură - același prompt, valori diferite (0 → 1.5)
- Formăm echipele
- Inițializăm repo-ul: `README, .env.example, notebook introductiv`

→ **Dacă la final ai un apel API funcțional și un repo — C1 e complet**

Ce luăm cu noi în C2

Știm acum să apelăm și să controlăm minimal un model.
Următoarea problemă este: cu ce model lucrăm de fapt?

Ce știm după C1	**Ce nu știm încă**
Cum facem un apel API	Care model e cel mai bun pentru proiect
Ce înseamnă temperature, tokens	Cum comparăm sistematic modelele
API vs local - când folosim ce	Cum documentăm alegerea metodologic
Ce este un output bun	Cum alegem un fallback

→ **C2: ecosistemul de modele**