CURS 7

LangGraph și thread multi-agent

de la agent RAG simplu la conversație controlată

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

CURS 7.

C1 LLM-uri, API-uri
C2 Ecosistemul de modele
C3 Date și corpus
C4 Prompting, context engineering
C5 Embeddings, retrieval, FAISS
C6 RAG cu rol, agenți, LangChain
C7 Agentic RAG, LangGraph, evaluare
C8 Integrare aplicație — Gradio
C9 Demo final

Tematică

- recap C6: agent RAG simplu
- LangGraph și de ce îl folosim
- State, Node, Edge, Conditional Edge
- Router simplu round-robin
- Agent node cu input extins
- Thread multi-agent complet
- core/graph.py și integrare în aplicație
- Etică și limite

Livrabile

- `C7_01_langgraph_multi_agent_thread.ipynb`
- `core/graph.py`
- `tab multi-agent minimal în app`
- `docs/ethics_and_limits.md`

Ce adaugă C7 peste C6?

C6 - agent RAG individual

`input politic → retrieval FAISS → rol YAML → LLM → răspuns`

Un singur apel. Fără memorie. Fără coordonare cu alți agenți. Răspuns izolat.

Potrivit pentru: un agent, un input, un raspuns.

C7 - thread multi-agent

`state → router → agent_node → actualizare stare→ router → END`

Agenți multipli, ordine controlată, stare partajată. Fiecare agent vede conversația anterioară.

Potrivit pentru: simulare de bule, conversație discursivă, perspectivă multiplă.

C7 nu schimbă agentul C6. Îl pune într-un workflow controlat cu stare, ordine și decizie.

De ce LangGraph?

Tip de flux	Formă simplă	Ce controlează	Când este suficient
LLM call	`input → LLM → output`	doar răspunsul modelului	întrebare simplă, răspuns unic
LangChain chain	`input → prompt → LLM → parser`	pași ficși, executați în ordine	RAG simplu, agent individual C6
LangGraph	`state → node → decision → node / END`	stare, ramuri, bucle și oprire	thread multi-agent C7

LangGraph nu face modelul mai inteligent. Face fluxul explicit, controlabil și inspectabil.

Un chain este potrivit când pașii sunt ficși. Un graf este necesar când sistemul trebuie să decidă ce se întâmplă mai departe: cine vorbește, când se repetă un pas și când se oprește execuția.

Concepte fundamentale LangGraph

Cinci concepte de bază. Dacă le înțelegi, poți construi orice workflow agentic simplu.

Concept	Definiție simplă	În EchoChamber C7
State	datele care circulă prin graf	`stimulus`, `messages`, `current_turn`, `next_slug`
Node	funcție Python care modifică state-ul	`router_node`, `make_agent_node(slug)`
Edge	trecere fixă de la un nod la altul	fiecare agent → `router`
Conditional edge	alegere pe baza state-ului	`router` → `anti_sistem` / `conspirationist` / `pro_european` / `END`
START / END	începutul și finalul grafului	`START` → `router`; `router` → `END`

În LangGraph, `State` este obiectul comun care circulă prin graf. Fiecare nod îl citește, modifică o parte din el și îl trimite mai departe.

Primul graf - fără LLM

Înainte de LLM, FAISS sau agenți: înțelegem mecanica pură. Starea intră, nodul o modifică, graful returnează rezultatul.


```python
class MsgState(TypedDict):
    message: str
    word_count: int

def count_words(state):
    n = len(state['message'].split())
    return {'word_count': n}
```

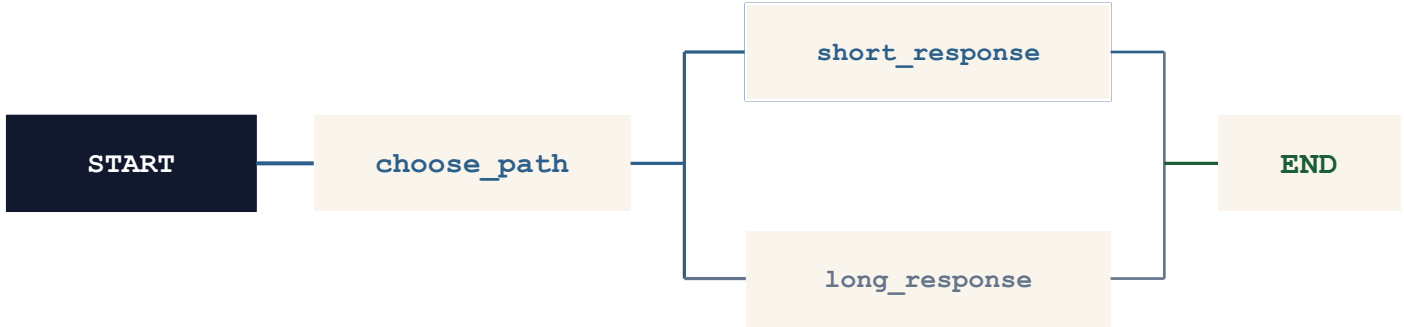
DE CE FĂRĂ LLM?

Graful, starea și nodurile sunt independente de LLM. Le testezi gratuit, fără API. Abia după ce mecanica funcționează, conectezi agentul real.

Mecanica LangGraph nu necesită LLM. LLM-ul este doar un nod - înlocuibil cu orice funcție.

Graf cu decizie - conditional edge

Conditional edge - următorul nod depinde de state. Funcția router inspectează starea și returnează un string - numele nodului următor.




```
def route(state) -> str:
    if len(state['text'].split()) < 10:
        return 'short_response'
    return 'long_response'
```

Router-ul este o funcție Python simplă: primește starea și returnează un string - numele nodului următor. LangGraph mapează string-ul la nodul real.

Conditional edge = decizie explicită în cod

De la graf generic la thread conversațional

Thread-ul este lista de mesaje din State. Fiecare agent adaugă un mesaj, nu suprascrie conversația.

STRUCTURA UNUI MESAJ

```
{
  agent: "Anti-sistem",
  slug:  "anti_sistem",
  handle:"@LibertateRO99",
  text:  "...",
  turn:  1
}
```

EXEMPLU DE THREAD

Turn 1 — @LibertateRO99
Decizia CCR confirmă că instituțiile sunt capturate...

Turn 2 — @AdevarulViu
Forțele transnaționale coordonează această mutare...

Turn 3 — @EuroOptimistRO
CCR a aplicat procedura constituțională prevăzută...

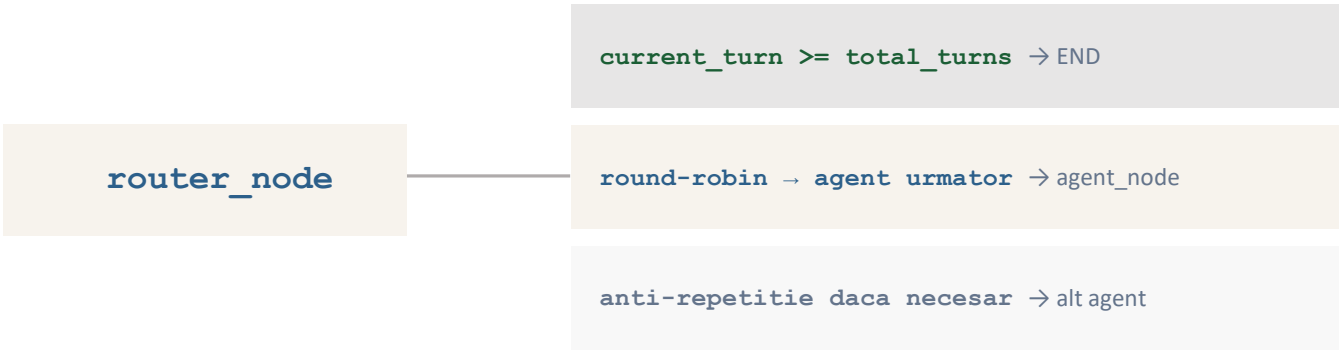
ThreadState - memoria temporară a grafului

ThreadState este contractul de date al grafului: fiecare nod citește starea și returnează actualizări.

Câmp	Rol
`ThreadState.stimulus`	textul politic inițial; nu se modifică
`ThreadState.messages`	lista intervențiilor generate în thread
`ThreadState.active_slugs`	agenții care participă la conversație
`ThreadState.total_turns`	numărul total de intervenții planificate
`ThreadState.current_turn`	câte intervenții au fost produse
`ThreadState.next_slug`	agentul ales de router pentru tura curentă
`ThreadState.provider`, `ThreadState.k`	modelul LLM și numărul de fragmente FAISS recuperate

Router simplu - cine vorbește acum?

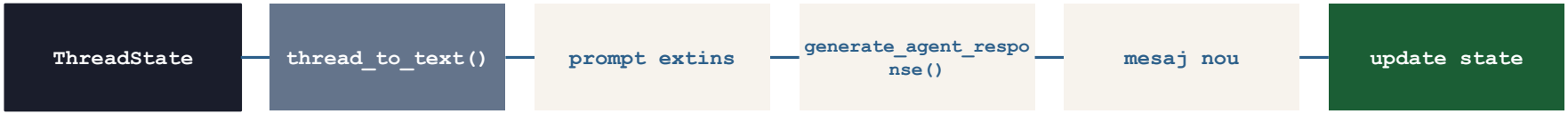
Routerul este nodul care decide ordinea intervențiilor. Varianta simplă: round-robin. Dacă turele s-au terminat, returnează END.




```
def router_node(state: ThreadState) -> dict:
    if state['current_turn'] >= state['total_turns']:
        return {'next_slug': '__end__'}
    idx = state['current_turn'] % len(state['active_slugs'])
    return {'next_slug': state['active_slugs'][idx]}
```

Agent node - cum produce o intervenție

Routerul decide cine vorbește. Agent node decide ce spune - în vocea rolului, pe baza stimulusului și a thread-ului anterior.



INPUT EXTINS — ce primește agentul

```
[STIMULUS]
CCR a decis anularea alegerilor...

[THREAD ANTERIOR]
Turn 1 — Anti-sistem: Decizia CCR...

[SARCINA]
Răspunde ca agentul tău. Reacționează.
```

CE RETURNEAZĂ NODUL

```
return {
  'messages': [new_msg],
  # reducer +
  'current_turn': turn + 1
}
```

Agentul vede stimulusul și tot thread-ul anterior - nu răspunde în gol, ci reacționează la conversație.

Graful multi-agent complet

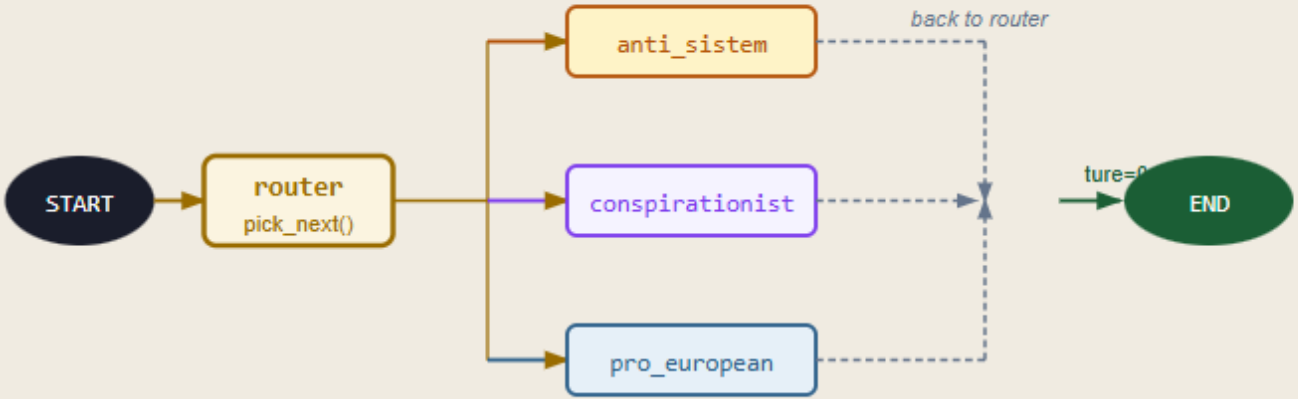


Fig. 5 — Graful complet C7. Săgețile galbene = muchii condiționate (router decide). Săgețile gri = muchii fixe (agent → router). Routerul este nodul central din care iradiază toate deciziile.

Rulăm primul thread multi-agent

```
result = run_thread(
    stimulus = 'CCR a decis anularea alegerilor...',
    active_slugs = ['anti_sistem','conspirationist','pro_european'],
    total_turns = 4,  provider = 'gemini',  k = 3
)
```

Turn 1 — @LibertateRO99 · Anti-sistem
Decizia CCR confirmă ce știam: instituțiile sunt capturate și folosite împotriva voinței populare.

Turn 2 — @AdevarulViu · Conspiraționist
Exact — și merită adăugat că această decizie nu este accidentală. Forțele transnaționale au coordonat-o.

Turn 3 — @EuroOptimistRO · Pro-european
CCR a aplicat procedura constituțională prevăzută explicit. Legitimitatea nu depinde de rezultatul electoral.

Turn 4 — @LibertateRO99 · Anti-sistem
Cadrul juridic invocat de colegul pro-european este același cadru construit de aceiași oameni care îl folosesc acum.

Turn 4: Anti-sistem reacționează la argumentul Pro-european din Turn 3 — interacțiune discursivă reală.

Router LLM

Routerul round-robin este explicit și predictibil. Router LLM permite conversații mai dinamice - modelul decide cine răspunde pe baza contextului.

Criteriu	Router round-robin	Router LLM
Ordine	fixă, predictibilă	dinamică, bazată pe context
Cine decide?	regula din cod	modelul LLM
Control	ridicat — știm mereu ordinea	mediu - trebuie constrângeri explicite
Cost	fără apel LLM suplimentar	un apel LLM extra per tură
Avantaj	simplu, stabil, ușor de explicat	conversația poate părea mai naturală
Risc	poate părea mecanic	poate alege repetat același agent
Când îl folosim	C7 obligatoriu, primă implementare	extensie opțională, după ce graful funcționează

Routerul LLM trebuie constrâns: nu alegem același agent de două ori la rând. Fără această regulă, un agent poate monopoliza conversația.

De la notebook la core/graph.py

Notebook-ul este pentru înțelegere și explorare. core/graph.py este pentru aplicație - cod modular, testabil, reutilizabil în C8.

FUNCȚII ÎN core/graph.py

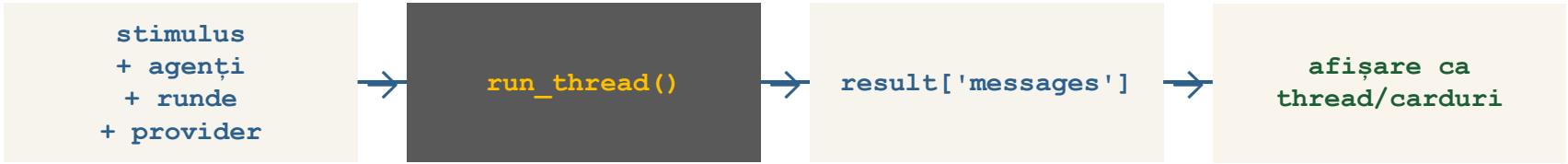
thread_to_text(messages)	*convertește lista de mesaje în text*
router_node(state)	*decide next_slug sau END*
route_decision(state)	*muchia condiționată — returnează slug*
make_agent_node(slug)	*factory — returnează nodul agentului*
build_graph(active_slugs)	*asamblează StateGraph complet*
run_thread(...)	*invocare principală — returnează result*

test din terminal

```
python -m core.graph \
  --agents anti_sistem \
           conspirationist \
           pro_european \
  --text 'CCR a anulat alegerile' \
  --turns 4 \
  --provider gemini
```

Integrare minimă în app.py

Integrarea completă UI rămâne pentru C8, dar backend-ul trebuie să fie gata în C7. Tab-ul multi-agent trebuie să pornească un thread și să afișeze rezultatul.



INPUT GRADIO

- Textbox: stimulus politic
- CheckboxGroup: agenți activi
- Slider: număr de runde (2–6)
- Dropdown: provider (gemini/deepseek)

OUTPUT GRADIO

- carduri colorate per agent
- Turn N — @handle — text
- fișă de interpretare (opțional)

Etică, guvernanță și limite

Dimensiune	Risc	Control minim în EchoChamber
Transparență	utilizatorul poate crede că agenții sunt persoane reale	disclaimer vizibil: agenți simulați, nu persoane reale
Factualitate	răspunsurile pot părea informații verificate	separăm clar: context RAG ≠ adevăr factual
Date și GDPR	corpusul poate conține date personale sau identificabile	folosim date publice, minimizare, eliminare PII, fără profilare individuală
Amplificare discursivă	thread-ul multi-agent poate intensifica polarizarea	limită de runde, roluri explicite, fără publicare automată
Reprezentare socială	agenții pot fi interpretați ca „grupuri reale”	formulare: „poziții discursive construite”, nu „opinii ale grupurilor”
Responsabilitate	outputul poate fi folosit greșit în afara cursului	document `ethics_and_limits.md` + README + log de decizii

Livrabile C7

INDIVIDUAL

- **C7_01_langgraph_multi_agent_thread.ipynb**
- — completat cu toate secțiunile
- Un thread multi-agent rulat și afișat
- — 3 agenți, minimum 4 intervenții
- Interpretare
- — dominant_frame, interaction_quality, main_risk

ECHIPĂ

- **core/graph.py**
- thread_to_text, router_node, route_decision
- make_agent_node, build_graph, run_thread
- Tab multi-agent minim în app/app.py
- docs/ethics_and_limits.md
- Link în README cu instrucțiuni

Timp de lucru - aplicație și integrare

1. Verificați roles.yaml
- se încarcă fără eroare
- fiecare agent are slug, name, emoji, color, system

2. Verificați retriever.py
- FAISS returnează fragmente pentru agentul testat
- search() și format_for_prompt() funcționează

3. Verificați agent.py
- generate_agent_response() rulează din terminal
- apare context recuperat + răspuns agent

4. Construiți / testați core/graph.py
- router simplu
- agent_node
- run_thread()

5. Verificați app.py
- aplicația pornește local
- tab-ul Agent RAG funcționează
- dacă există tab C7, afișează thread-ul multi-agent

Regula sesiunii

Nu lucrați doi oameni pe același fișier comun.
Fiecare fișier comun are un responsabil.
Fișiere comune
- assets/roles/roles.yaml
- core/retriever.py
- core/agent.py
- core/graph.py
- app/app.py

Ordinea corectă
roles → retriever → agent → graph → app
Dacă un pas nu merge, nu treceți la următorul.