

# Software, incertezza e GenAI: verso nuove pratiche di sviluppo

Enrico Zimuel, [www.zimuel.it](http://www.zimuel.it)

Tech Lead & Principal Software Engineer presso Elastic (USA)

Prof. a.c. di Machine Learning presso Univ. Torino

Prof. a.c. di Architetture RAG e GenAI presso Univ. Roma Tre



# Sommario

- GenAI e modelli probabilistici
- Test dei modelli tramite benchmark
- Generazione di codice con LLM
- Utilizzo di modelli GenAI nel software
- Testing di software con Agentic AI
- Monitoraggio di soluzioni GenAI
- Ci saranno ancora programmatori in futuro?

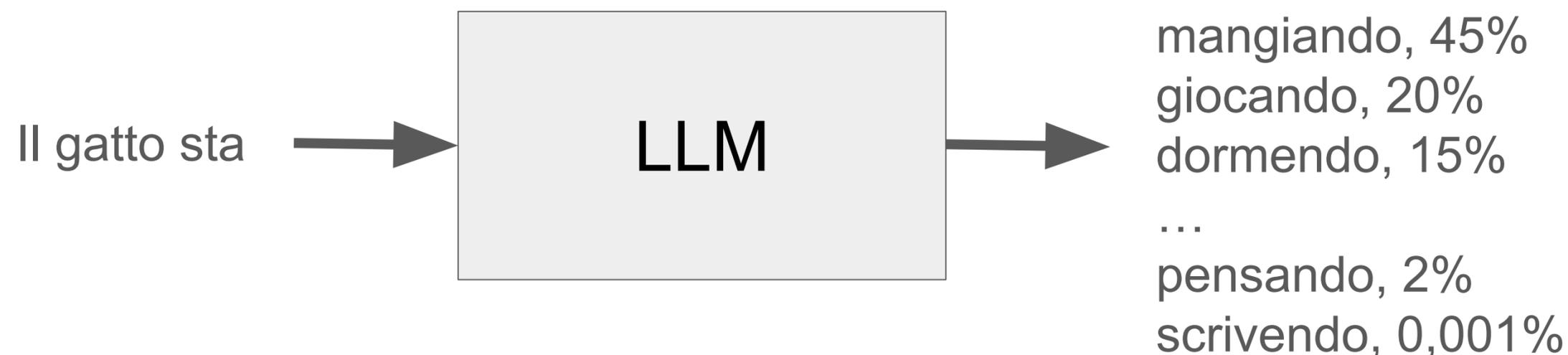


Image generated using Perplexity.ai

# GenAI e probabilità

- I modelli di Generative AI sono **modelli probabilistici**
- Un modello probabilistico descrive un fenomeno o un insieme di dati tramite una **distribuzione di probabilità**
- In altre parole, invece di fornire un unico risultato deterministico, il modello assegna una **probabilità a ciascun possibile esito**
- Nel caso dei LLM la probabilità è associata ai **token** (parole)
- Una frase in input è un insieme di token, un LLM completa la frase scegliendo tra i token più probabili (top-k)

# Probabilità condizionata



- $P(x_t | x_1, x_2, \dots, x_{t-1})$  è la probabilità che l'output sia  $x_t$  dato l'input  $x_1, x_2, \dots, x_{t-1}$

- Esempio:

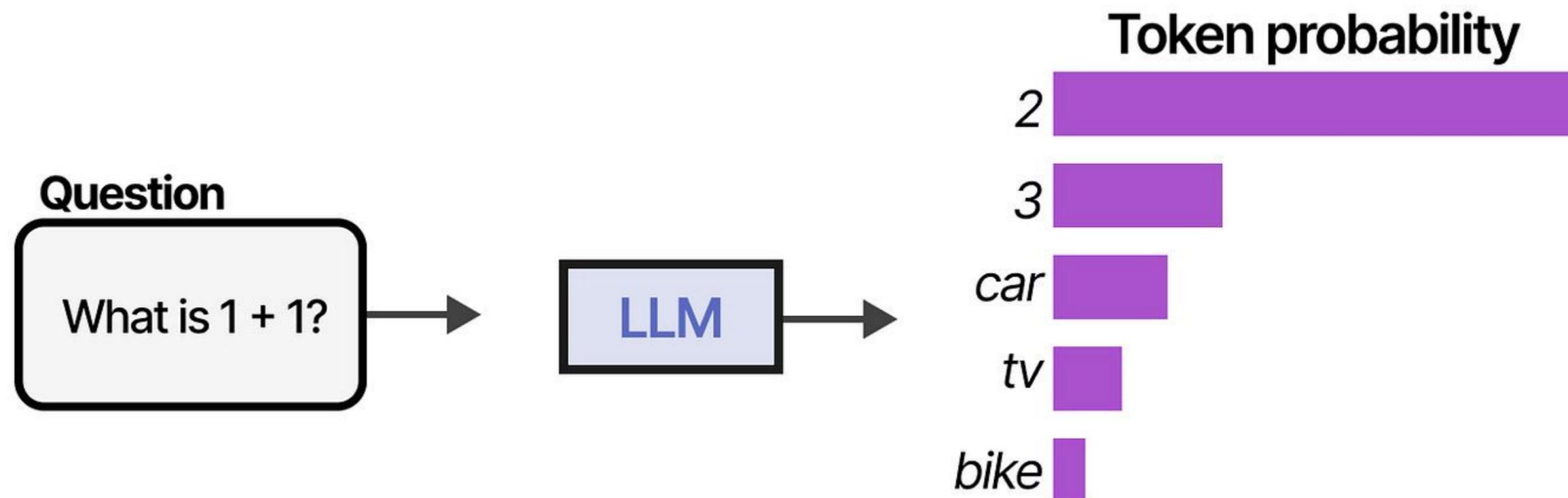
$$P(\text{"mangiando"} | \text{"Il gatto sta"}) = 0,45$$

$$P(\text{"giocando"} | \text{"Il gatto sta"}) = 0,20$$

$$P(\text{"dormendo"} | \text{"Il gatto sta"}) = 0,15$$

...

# L'errore è intrinseco nel modello



$P("3" | "What is 1 + 1?") \neq 0$   $P("car" | "What is 1 + 1?") \neq 0$

...

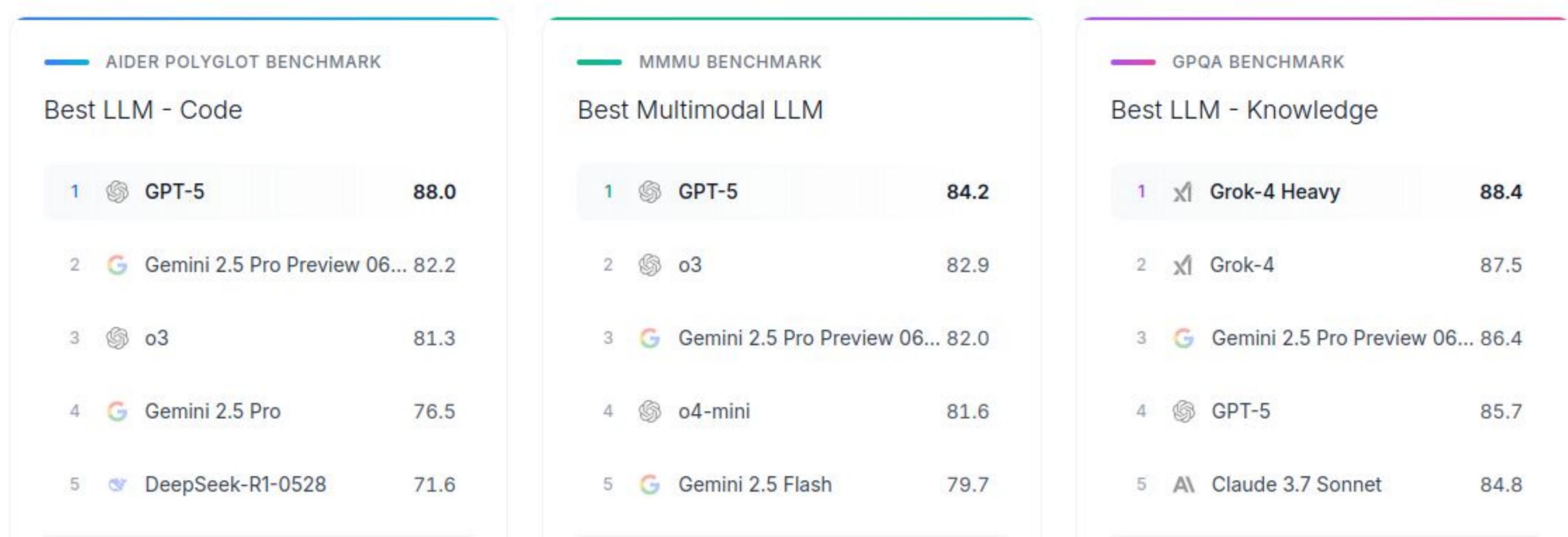
# Come stimare la probabilità di errore?

- In un modello predittivo si può definire la precisione (**accuracy**), ossia la percentuale di predizioni corrette sul totale
- Nella maggior parte dei casi un LLM produce frasi, come possiamo stimare una predizione (una frase) corretta?
- Alcuni approcci:
  - Se l'output è un programma, possiamo provare ad eseguirlo
  - Possiamo calcolare la distanza tra gli embedding, ossia quanto la risposta si discosta semanticamente dall'input (dalla domanda)
  - Possiamo utilizzare un altro LLM giudice che valuti la bontà della risposta data, utilizzando un insieme di domande e risposte preconfezionate (benchmark)

$$\text{Accuracy} = \frac{\text{Correct predictions}}{\text{All predictions}}$$

# Benchmarks

- Esistono tantissimi benchmark per testare le capacità dei LLM
- Esempio: <https://llm-stats.com/>

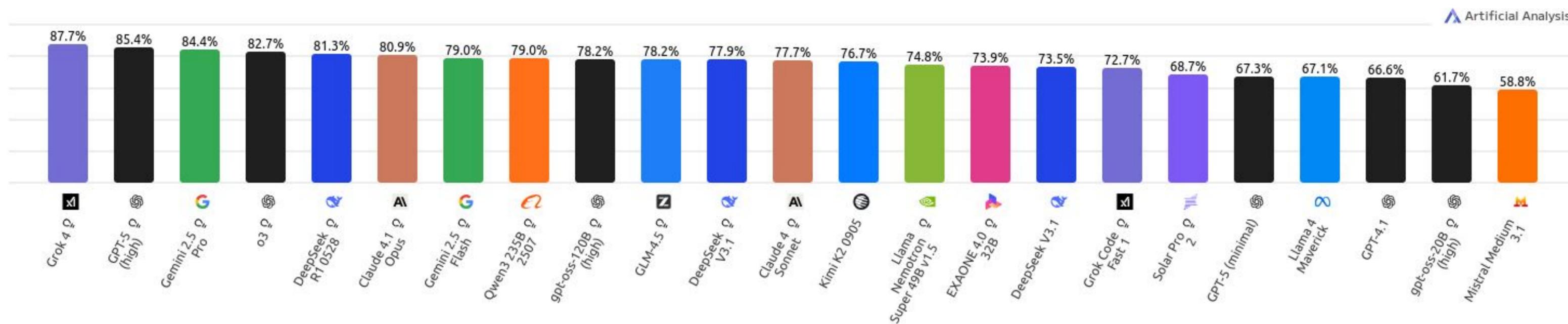


# GPQA Benchmark

- GPQA: Graduate-Level Google-Proof Q&A Benchmark è un benchmark di 448 domande a risposta multipla preparate da esperti di biologia, fisica e chimica
- Esperti con PhD nei rispettivi settori riescono a rispondere ad arrivare al 65% di accuratezza nelle risposte. Non esperti, avendo a disposizione Google per 30 minuti, riescono ad arrivare al 34%
- Fonte: <https://arxiv.org/pdf/2311.12022>

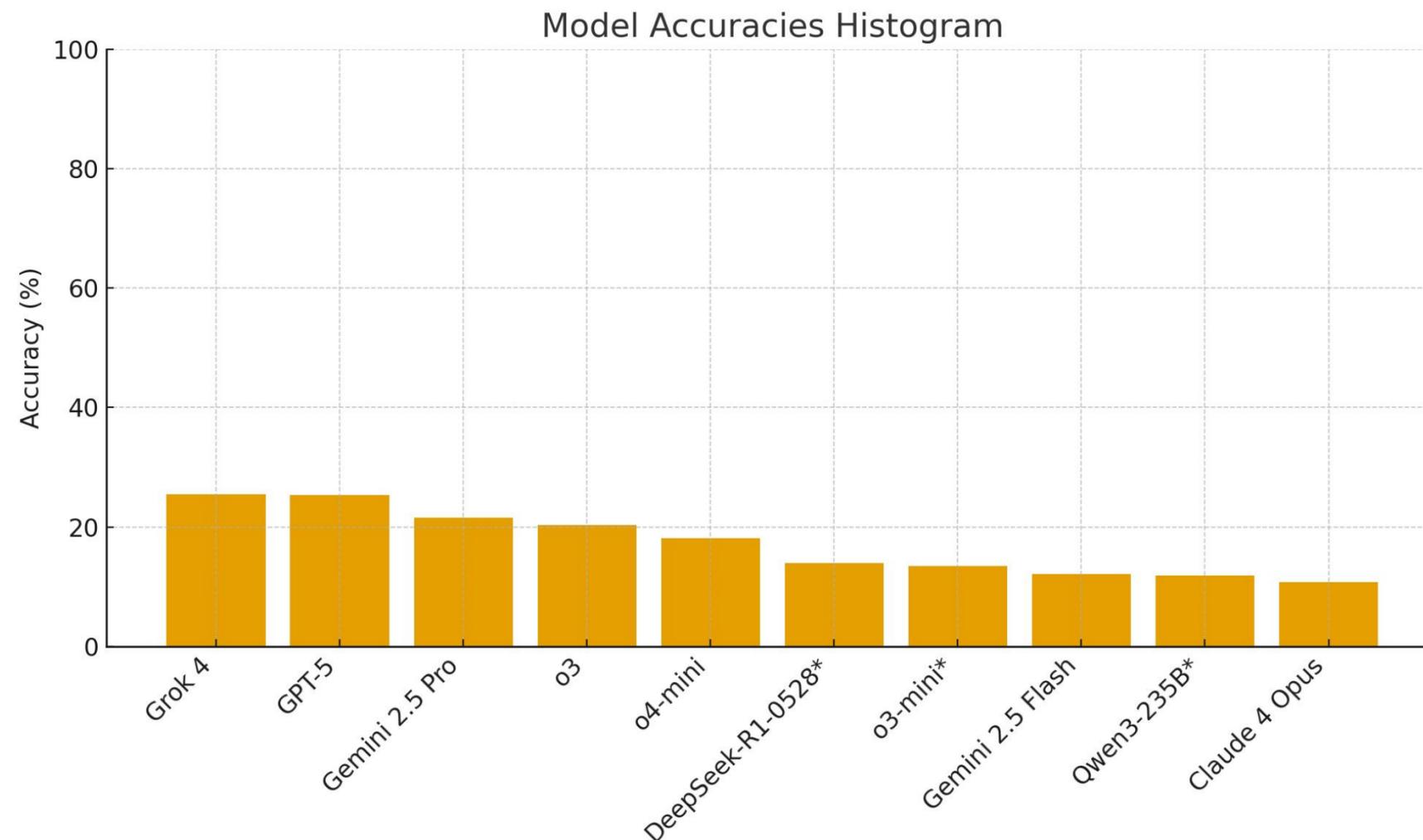
# GPQA Diamond Benchmark Leaderboard

- Diamond benchmark considera le 198 domande più difficili di GPQA
- Fonte: <https://artificialanalysis.ai/evaluations/gpqa-diamond>



# Benchmark: Humanity's Last Exam

- Benchmark multimodale di 2500 domande realizzate da esperti del settore
- Fonte: <https://agi.safe.ai/>

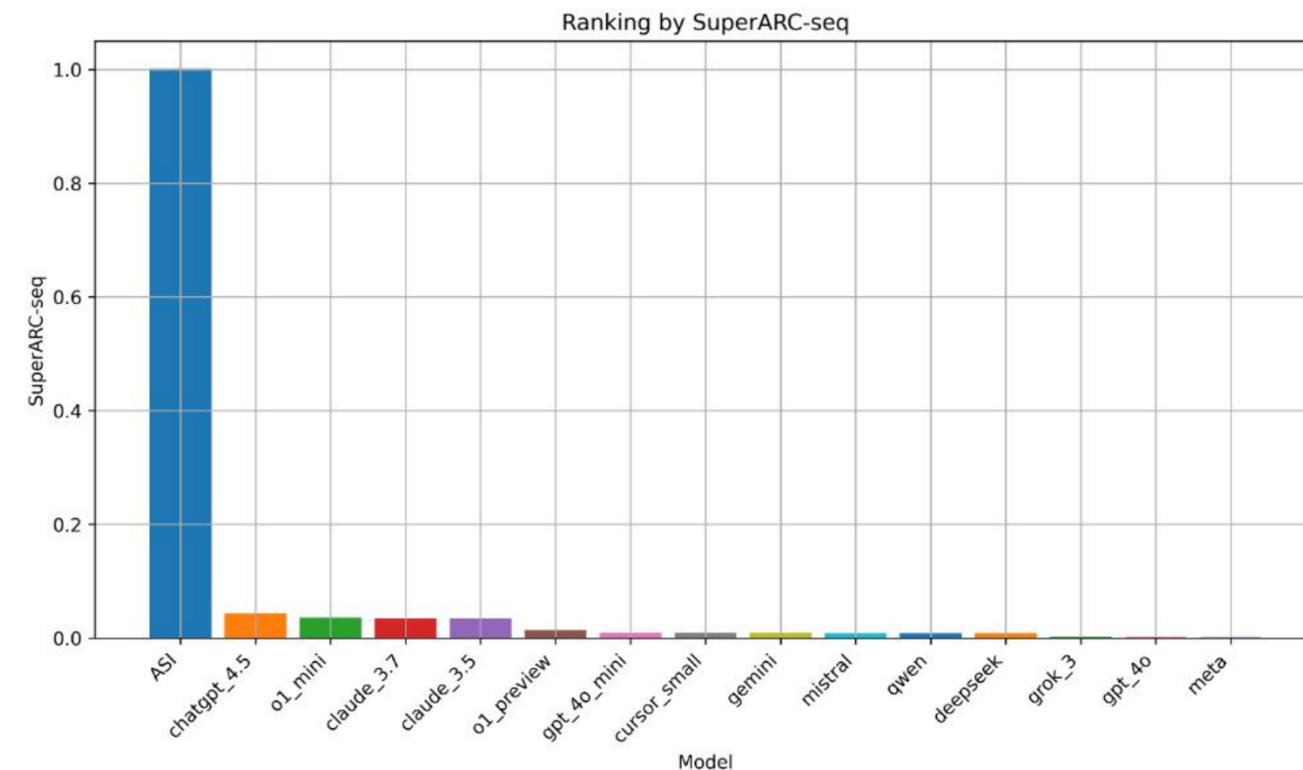
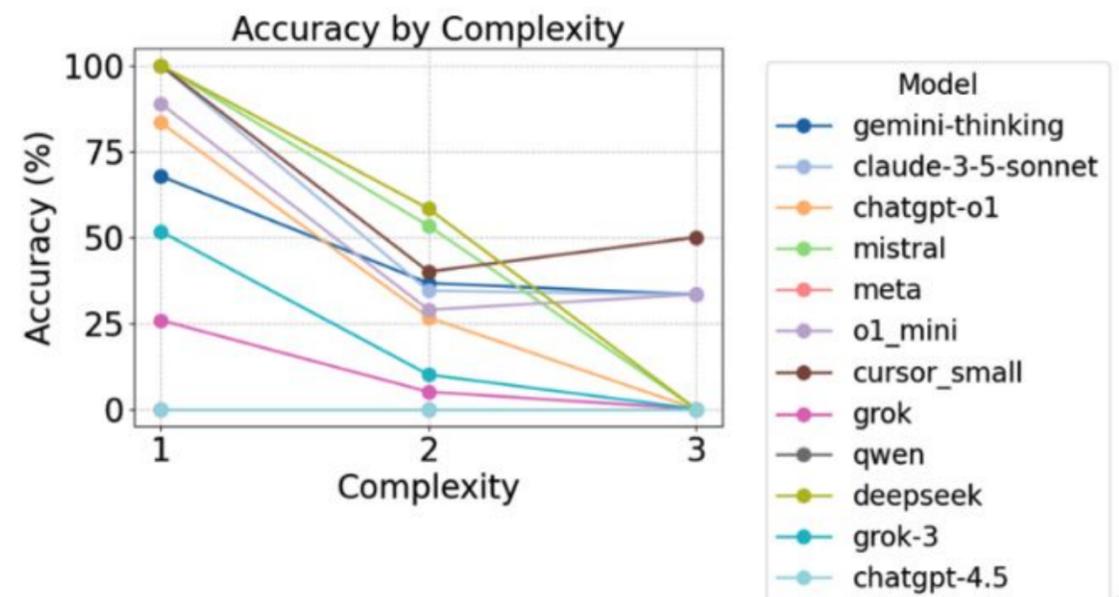


Judge Model: o3-mini | Dataset Updated: April 3rd, 2025

Model	Accuracy (%) ↑	Calibration Error (%) ↓
Grok 4	25.4	
GPT-5	25.3	50.0
Gemini 2.5 Pro	21.6	72.0
o3	20.3	34.0
o4-mini	18.1	57.0
DeepSeek-R1-0528*	14.0	78.0
o3-mini*	13.4	80.0
Gemini 2.5 Flash	12.1	80.0
Qwen3-235B*	11.8	74.0
Claude 4 Opus	10.7	73.0

# Benchmark: SuperARC

- Benchmark per determinare capacità di AGI e ASI utilizzando un approccio diverso ossia la capacità di generare nuova conoscenza dall'osservazione
- “Our findings strengthen the suspicion regarding the fundamental limitations of LLMs, exposing them as systems optimised for the **perception of mastery** over human language”
- Fonte: <https://arxiv.org/pdf/2503.16743v2>



# LLM e generazione del codice

# Strumenti per la generazione del codice

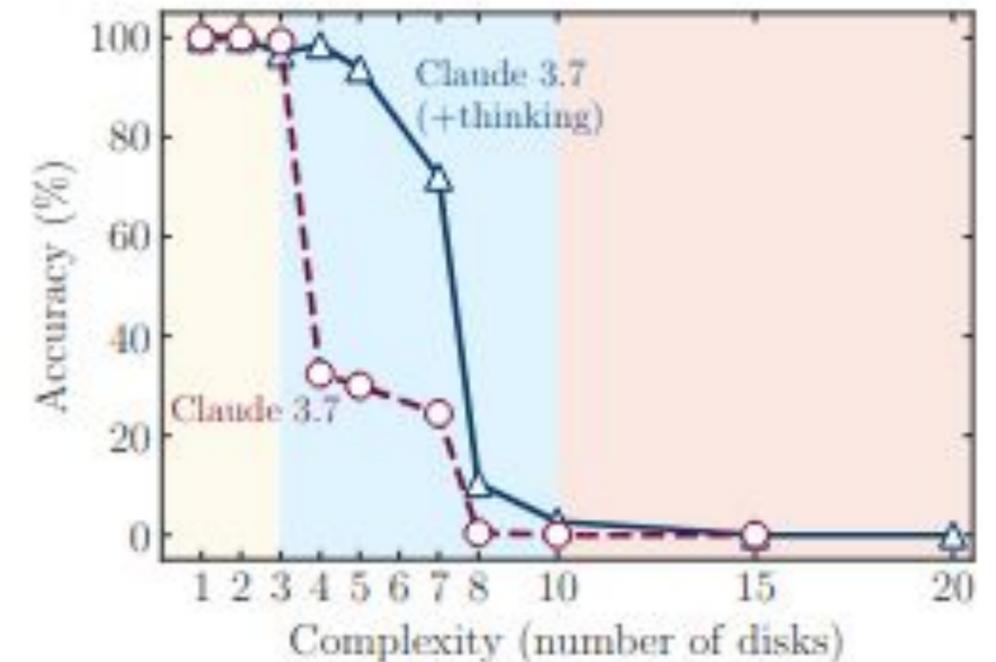
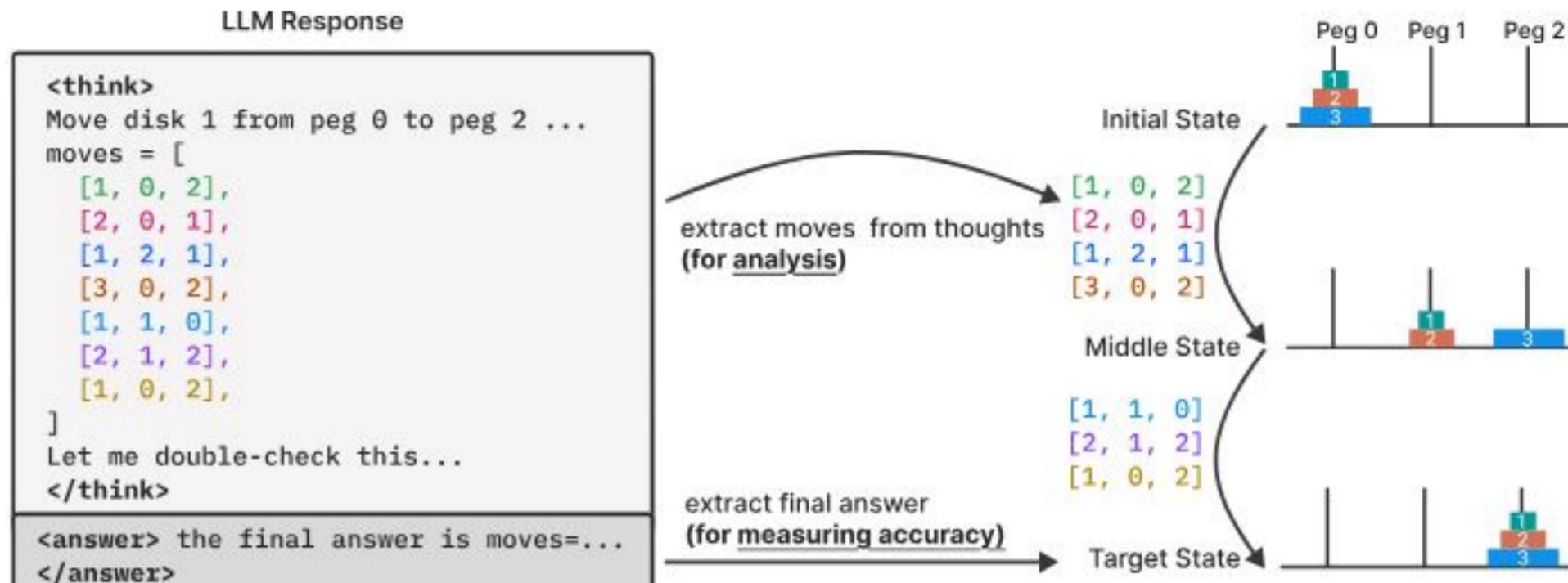
- [Github Copilot](#) integrato in Visual Studio Code
- [Warp](#)
- [Cursor](#)
- [OpenAI Codex](#)
- [Mistral Code](#)
- [Gemini Code Assistant](#)
- [DeepSeek Coder](#)
- [Amazon Q](#)
- [Kiro](#)
- ...

# Large Reasoning Model (LRM)

- I modelli LRM sono un'evoluzione dei LLM progettati per simulare un processo di ragionamento (reasoning tasks)
- Fondamentalmente, si cerca di guidare un LLM all'elaborazione di un compito suddividendolo in più task
- Si addestrano questi modelli con catene di pensiero (chain-of-thought)
- Incoraggiati a scomporre un problema complesso in sottoproblemi più semplici
- Esempi di questi modelli: OpenAI o1/o3, GPT-5 reasoning, DeepSeek R1, Claude 3.7 Sonnet Thinking, Gemini thinking, etc

# L'illusione del ragionamento

- Sembra che i modelli LLM e LRM diano solo un'illusione di ragionamento
- Questa capacità di “ragionare” sembra svanire all'aumentare della complessità
- Fonte: Parshin Shojaee et al., [The Illusion of Thinking](#), Apple



# Aider's polyglot benchmark

- Un benchmark per testare la capacità di programmazione dei LLM su un insieme di 225 esercizi in diversi linguaggi: C++, Go, Java, JavaScript, Python, e Rust.
- Fonte: <https://aider.chat/docs/leaderboards/>

Model	Percent correct	Cost	Command	Correct edit format	Edit Format
gpt-5 (high)	88.0%	\$29.08	<code>aider --model openai/gpt-5</code>	91.6%	diff
gpt-5 (medium)	86.7%	\$17.69	<code>aider --model openai/gpt-5</code>	88.4%	diff
o3-pro (high)	84.9%	\$146.32	<code>aider --model o3-pro</code>	97.8%	diff
gemini-2.5-pro-preview-06-05 (32k think)	83.1%	\$49.88	<code>aider --model gemini/gemini-2.5-pro-preview-06-05 --thinking-tokens 32k</code>	99.6%	diff-fenced
gpt-5 (low)	81.3%	\$10.37	<code>aider --model openai/gpt-5</code>	86.7%	diff
o3 (high)	81.3%	\$21.23	<code>aider --model o3 --reasoning-effort high</code>	94.7%	diff
grok-4 (high)	79.6%	\$59.62	<code>aider --model openrouter/x-ai/grok-4</code>	97.3%	diff
gemini-2.5-pro-preview-06-05 (default think)	79.1%	\$45.6	<code>aider --model gemini/gemini-2.5-pro-preview-06-05</code>	100.0%	diff-fenced
o3 (high) + gpt-4.1	78.2%	\$17.55	<code>aider --model o3</code>	100.0%	architect

# Vibe coding

- Il termine **vibe coding** è stato proposto da [Andrej Karpathy](#), esperto di IA nel febbraio 2025
- Per vibe coding si intende una pratica di programmazione che utilizza un modello automatico di generazione del codice
- Un **vibe coder** non scrive righe di codice ma chiede al modello di generare del codice utilizzando uno o più prompt
- In caso di errori, il vibe coder chiede direttamente al modello di sistemarli
- E' un **processo iterativo** che consente a chi ha poca o nessuna esperienza di programmazione di generare del codice che (apparentemente) funziona

# Critiche al vibe coding

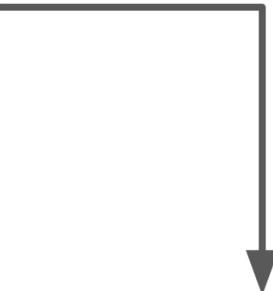
- Il **vibe coding** è uno strumento che può essere utile per generare POC
- Il codice generato attraverso un processo di vibe coding tende ad avere una notevole complessità architetturale (spaghetti code)
- La manutenzione di un codice di questo tipo può risultare complicata
- I test del codice, soprattutto quelli funzionali, possono avere bias se fatti generare dallo stesso modello
- Mettere in produzione codice generato da un processo di vibe coding è sicuramente rischioso

# Controllo (umano) del codice

- Caso d'uso: quando un commento nel codice può ingannare un LLM
- Fonte: [Our first outage from LLM-written code](#)

```
for {
    repos, repoResp, err := ghClient.Apps.ListUserRepos(ctx, *installation.ID, repoOpt)
    if err != nil {
        // Log error but continue with other installations
        log.Printf("Error fetching repositories for installation %d: %v", *installation.ID, err)
        break
    }
    // ...
}
```

LLM review



```
for {
    repos, repoResp, err := ghClient.ListUserRepos(ctx, *installation.ID, repoOpt)
    if err != nil {
        // Log error but continue with other installations
        log.Printf("Error fetching repositories for installation %d: %v", *installation.ID, err)
        continue
    }
    // ...
}
```

# Utilizzo di GenAI in un'architettura software

# Cambio di paradigma

- Per utilizzare con successo un modello di GenAI come componente software è necessario un approccio diverso (cambio di paradigma)
- Da un processo deterministico tradizionale (programmazione) a un processo di tipo probabilistico (errore intrinseco)
- Se l'errore stimato attraverso test e benchmark è accettabile per il caso d'uso allora il modello di GenAI può essere utilizzato
- Il riconoscimento di un errore generato da un modello di GenAI non è banale e deve essere affrontato caso per caso

# Agenti “intelligenti”

- Un agente è un software che stabilisce in maniera autonoma l’invocazione di altri componenti software
- L’autonomia decisionale è affidata in parte o in tutto a un LLM
- Un elemento chiave è la capacità dei LLM di decidere se eseguire un componente software
- Utilizzo di più agenti (software) che sfruttano le proprietà emergenti dei LLM:
  - Decomposizione in più task (decomposition task)
  - Invocazione di tool (tool invocation)
  - Autocorrezione basata su feedback (self-correction based on feedback)
- Nelle architetture ad agenti i LLM funzionano come “**motori di ragionamento**”

# Invocazione di tool

- Una delle proprietà emergenti dei LLM è quella di **“riconoscere l’esigenza di eseguire un componente software (tool) esterno”**
- Gli LLM sono in grado di capire se nella domanda (prompt) è presente una richiesta che richieda l’invocazione di un tool:
  - il LLM prepara la sintassi per l’invocazione del codice
  - Il codice viene eseguito da un software deterministico
  - Il risultato dell’esecuzione viene passato nel contesto del prompt al LLM
  - Il LLM risponde avendo il contesto (i dati)

# Esempio di tool con OpenAI

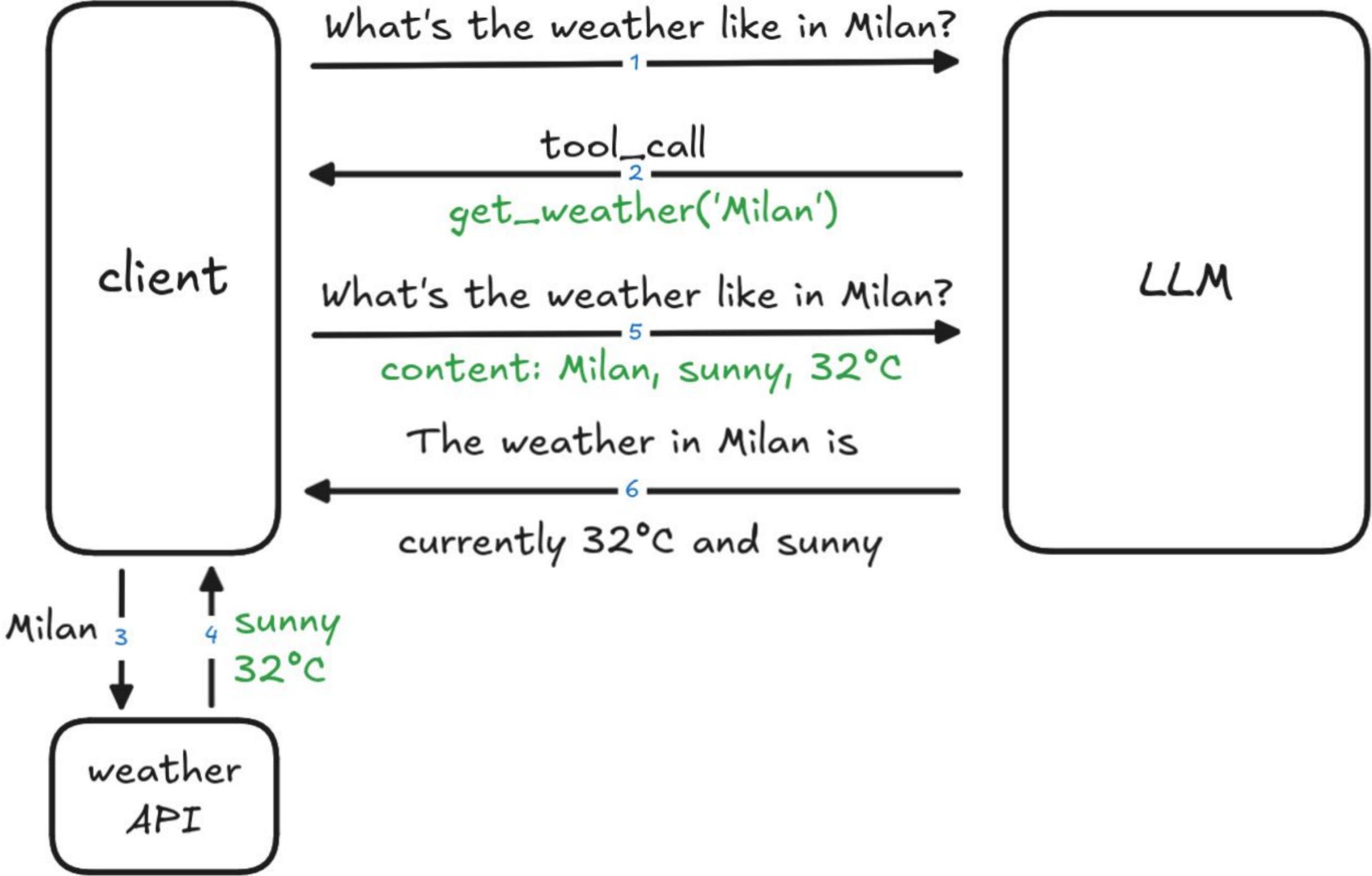
POST https://api.openai.com/v1/chat/completions

```
{
  "model": "gpt-4.1",
  "messages": [
    {
      "role": "user",
      "content": "What is the weather like in Milan today?"
    }
  ],
  "tools": [
    {
      "type": "function",
      "function": {
        "name": "get_weather",
        "description": "Get current temperature for a given location.",
        "parameters": {
          "type": "object",
          "properties": {
            "location": {
              "type": "string",
              "description": "City and country e.g. Rome, Italy"
            }
          },
          "required": [
            "location"
          ],
          "additionalProperties": false
        },
        "strict": true
      }
    }
  ]
}
```



```
[{
  "id": "call_12345xyz",
  "type": "function",
  "function": {
    "name": "get_weather",
    "arguments": "{\"location\": \"Milan, Italy\"}"
  }
}]
```

# Che tempo fa a Milano?



# Strumenti per il testing

- Recentemente stanno uscendo diversi tool per il testing di applicazioni GenAI
- [Ragas](#), un framework e un insieme di tool per il testing e il monitoring di un'applicazione GenAI
- [DeepEval](#), un framework per la valutazione di LLM e applicazioni che utilizzano LLM
- [LangFuse](#), una piattaforma per la valutazione, il monitoring e il debugging di applicazioni LLM
- [LangSmith](#), una piattaforma di valutazione, monitoring e debugging di applicazioni LLM

# Context Engineering

- Una nuova figura professionale, il **Context engineer**
- Un programmatore in grado di recuperare il contesto (i dati) da fornire ad un LLM per l'elaborazione
- Richiede competenze di:
  - Information retrieval (motori di ricerca, database)
  - Analisi e manipolazione dei dati (web scraping, data science)
  - Programmazione per l'implementazione di agenti (web, protocolli HTTP, MCP, A2A)
  - Prompt engineering e conoscenze di LLM e NLP
  - Testing, benchmarking e monitoring (sistemi cloud)

# Riassumendo

- I modelli di GenAI sono probabilistici
- L'errore nei modelli di GenAI è intrinseco
- La generazione di codice dei LLM grazie ai Reasoning Model può dare un contributo notevole allo sviluppo software
- Il controllo umano del codice e dei test prodotti è fondamentale
- I test sono la parte più importante in un'architettura software con GenAI
- Il monitoring è un altro aspetto fondamentale perchè l'affidabilità di questi modelli può variare a seconda dell'utilizzo (input)

“Abbiamo bisogno di più sviluppatori nell'era dell'AI, non di meno”

PhD Shalini Kurapati, CEO di [Clearbox AI](#)

# Riferimenti

- P. Shojaee et al., [The Illusion of Thinking: Understanding the Strengths and Limitations of Reasoning Models via the Lens of Problem Complexity](#), Apple, June 2025
- Horace He et al., [Defeating Nondeterminism in LLM Inference](#), Thinking Machines, September 2025
- Marc Brooker, [LLMs as Parts of Systems](#), Marc's Blog, August 2025
- [The Coding Personalities of Leading LLMs](#), Report by Sonar, August 2025
- Yihong Dong et al., [A Survey on Code Generation with LLM-based Agents](#), July 2025
- Mark Chen et al., [Evaluating Large Language Models Trained on Code](#), OpenAI, 2021
- Enrico Zimuel, [Tool calling in agentic AI](#), June 2025
- Philipp Schmid, [The New Skill in AI is Not Prompting, It's Context Engineering](#), Google DeepMind, June 2025
- Chengshuai Zhao et al., [Is Chain-of-Thought Reasoning of LLMs a Mirage? A Data Distribution Lens](#), August 2025
- Alberto Hernández-Espinosa et al., [SuperARC: An Agnostic Test for Narrow, General, and Super Intelligence Based On the Principles of Recursive Compression and Algorithmic Probability](#), April 2025
- Maarten Grootendorst, [A Visual guide to LLM Agents](#), March 2025

# Grazie!

Contatti: enrico (at) zimuel.it

Linkedin: <https://www.linkedin.com/in/ezimuel/>

