LLM Fundamentals

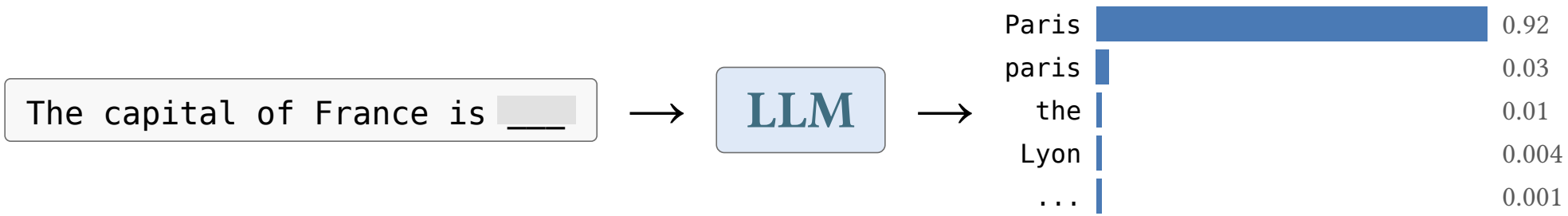
Agentic Security — Module 1

What Is an LLM?

Glorified Autocomplete

- LLMs exist to provide a single function: to **predict text**
- Given a set of text, they try to predict what comes next
- This leads to a surprising amount of intelligence

Predicting Tokens



- Each pass of inference tries to guess the next token
- Provides a range of possible tokens and their probability
- RNG is involved in token selection

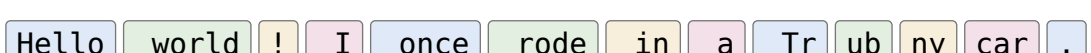
The Root of Hallucinations

- Why do models make things up?
  - Models do not access the real world
  - Weights are based merely on patterns in training data
  - “Predictable” does not mean “accurate”
- LLMs are permanently prone to hallucinations
  - Newer models are better, but it continues to happen

Tokens and Tokenizers

- Unlike humans, LLMs only see **tokens**
- **Tokens** are the basis of all LLM operations
- Rule of thumb: **4 characters = 1 token**
- LLMs use a **tokenizer** to convert text ⟶ tokens
  - Tokenizers are an active area of research

```
Hello world! I once rode in a Trubnycar.
```



Tokens: Why “How Many R’s In Strawberry?” Fails

user How many r’s are in “strawberry”?

what the model actually receives →

`How` `many` `r` `'s` `are` `in` `"` `st` `raw` `berry` `"?`

assistant There are **two** r’s in “strawberry”. ✗

LLMs only see **tokens**. They cannot look “inside” tokens.

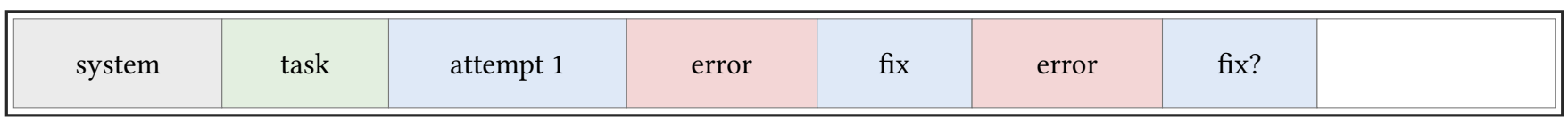
Context Window

(AKA “context limit”)

- Models can only operate up to a maximum number of tokens
- Minimizing token usage is extremely important
  - Sometimes called “Context engineering”
- There are several interesting techniques to bypass the context window

Context Rot

Long sessions degrade before they hit the limit:



- Every token in context comes with a **performance penalty**
- Exclude **everything possible**

Token Proxies

rtk: “CLI proxy that reduces LLM token consumption by 60-90% on common dev commands.”

git push 15 lines

```
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 412 bytes
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2)
To github.com:you/repo.git
   a1b2c3d..e4f5g6h  main -> main
...
```

rtk git push 1 line

```
ok main
```

Models were trained on regular `git` output… how do they handle this?

Compaction

We can just **summarize!**

- Most agent harnesses (e.g. Claude Code, Pi) support compaction
- Can be triggered manually or automatically

40 turns of history 48k tokens

```
user: the build is failing
assistant: what's the error?
user: <pastes 200 lines of stack trace>
assistant: try clearing the cache
user: still failing, same error
assistant: <reads 3 files>
... 34 more turns ...
```

after compaction 600 tokens

```
Summary of prior conversation:
Build fails with ImportError in
utils.py. Cache clear did not help.
Root cause: circular import between
utils.py and models.py, introduced
in commit a1b2c3d.
Next step: extract shared types.
```

Compaction is a lossy process.

Inside the Model

Weights and Parameters

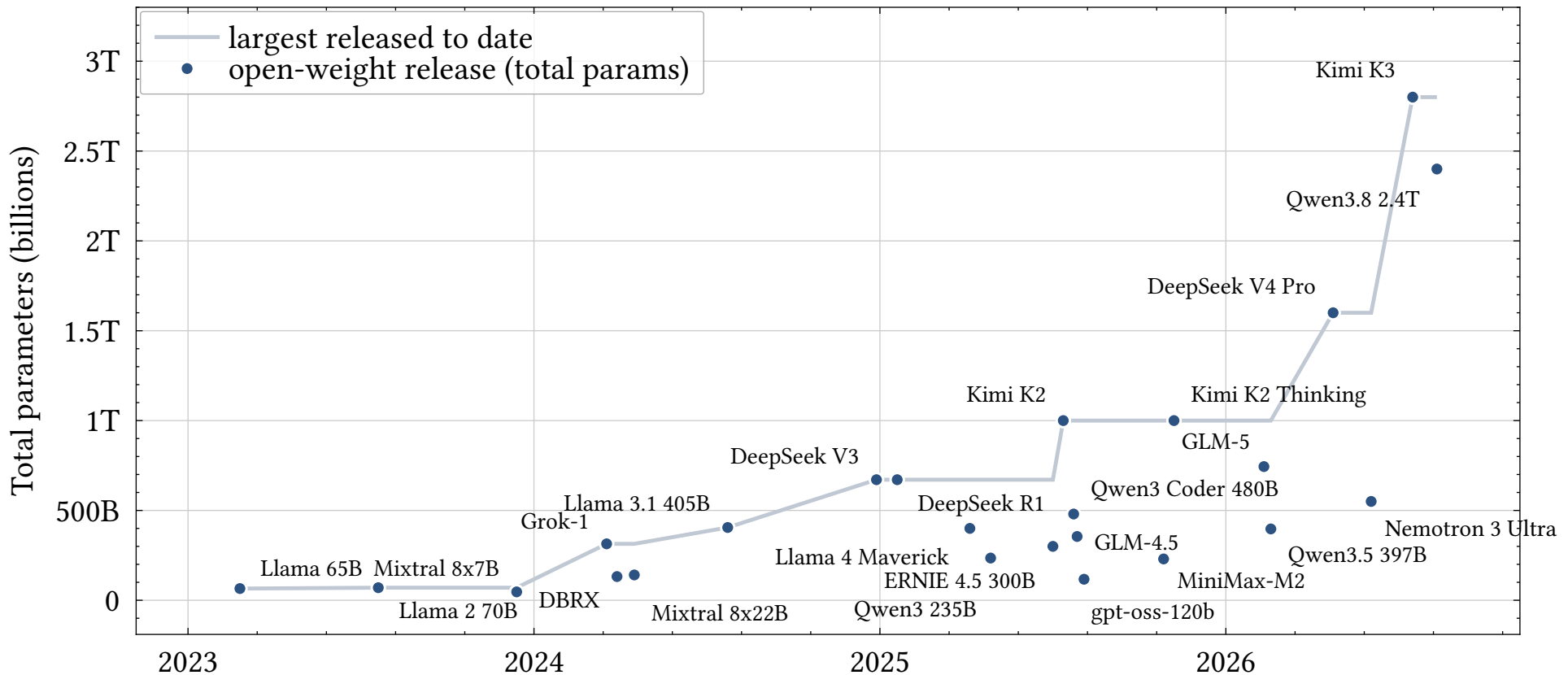
All LLMs are fundamentally a large collection of **weights** (or **parameters**)

- Weights are just numbers
- “70B” = 70 billion parameters
- When you **increase model size**…
  - **Intelligence** increases
  - **Resource usage** increases
  - **Cost** increases
  - **Speed** decreases

Parameter Count

- Parameter count is a **rough** capability signal
  - Kimi K3 is **2.8T** parameters
  - Anthropic's Mythos is rumored to have **10T** parameters

Open-Weight Models Are Getting Bigger

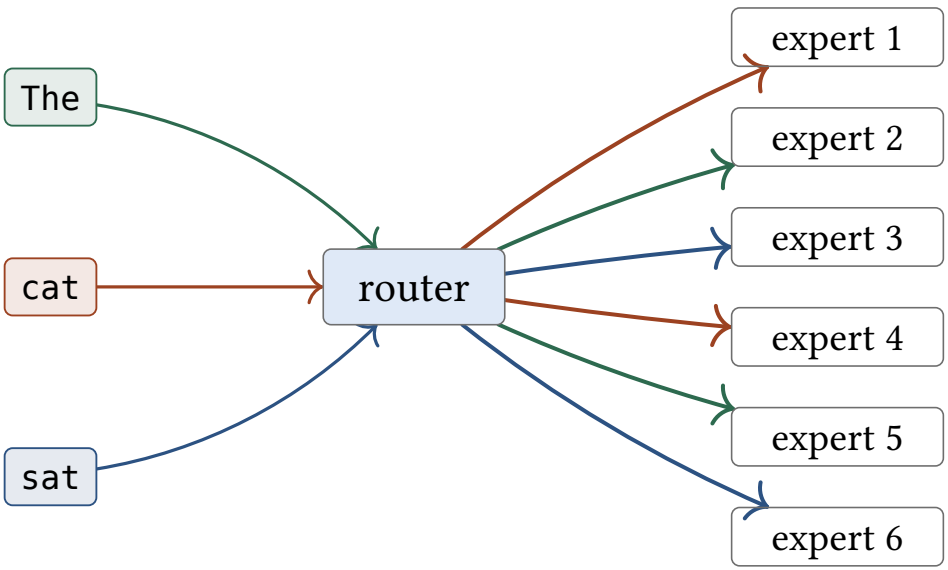


Total (not active) parameters as published by each lab. Release dates: Artificial Analysis https://artificialanalysis.ai/

Dense vs. MoE

- Traditionally, an LLM uses its **full network** of weights during inference
  - This is slow and costly
  - **Dense** models still work this way
- What if we partitioned sections of weights into “experts”?
  - Small overhead cost
  - Faster to execute each task
  - Think about a human team

Mixture of Experts (MoE)



- Every token is routed **independently**
- Large total parameter count, small **active** count
- You'll see something like "671B total, 37B active"

Quantization

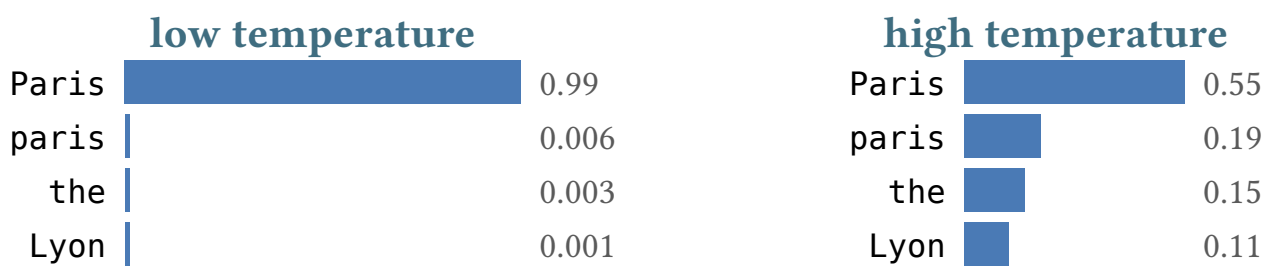
Weights are just numbers. What if we rounded them?

precision	one weight, stored	size
16-bit	`0.34179687500`	2 bytes
8-bit	`0.3418`	1 byte
4-bit	`0.34`	0.5 bytes

- 16-bit ⟶ 8-bit ⟶ 4-bit: smaller, faster, cheaper
- Costs some quality; degradation grows as precision drops
- How large open models fit on small hardware

Sampling and Temperature

1. LLM outputs probabilities
2. **Temperature** influences probabilities
3. **Sampling** decides which token wins
4. Repeat



- Low temperature -> more deterministic
- High temperature -> more random and unpredictable

The Inference API

The OpenAI API Spec

```
{"model": "small", "messages": [
  {"role": "system", "content": "You are a professional developer."},
  {"role": "user", "content": "Write this feature..."}]}
```

```
{"model": "small", "messages": [
  {"role": "system", "content": "You are a professional developer."},
  {"role": "user", "content": "Write this feature..."}]}
  {"role": "assistant", "content": "Okay, here's the plan: ..."}]}
  {"role": "user", "content": "Let's adjust the plan..."}]}
  {"role": "assistant", "content": "Building now..."}]}
```

The OpenAI API Spec

- Most common inference API spec
- **Roles**: `system`, `user`, `assistant` (+ tool messages, later)
- **Stateless**: the `messages` array is the whole conversation
- Supports **streaming**

What a Transcript Actually Is

`system`	“You are a security analyst.”
`user`	“Summarize this finding…”
`assistant`	“Improper session invalidation in the logout flow…”
`user`	“What’s the severity?”
`assistant`	“High — an attacker with a captured token can…”

Prompt Caching

- Earlier, we said inference is stateless…
  - **Except** for prompt caching.
  - Stores **key-value (KV) states**
- Using cached context is 10x cheaper
- Expires when…
  - You modify history
  - TTL expires

The System Prompt

- It's the first message in the array, with `role: "system"`
- Informally privileged context

user

Is this login code safe?

system

You are a helpful coding assistant.

assistant

Looks good! One small note: you could extract the validation into a helper to keep things tidy.

system

You are a security auditor. Assume every input is hostile.

assistant

The password is compared with `==`, which leaks timing. Session IDs come from `random`, not `secrets`.

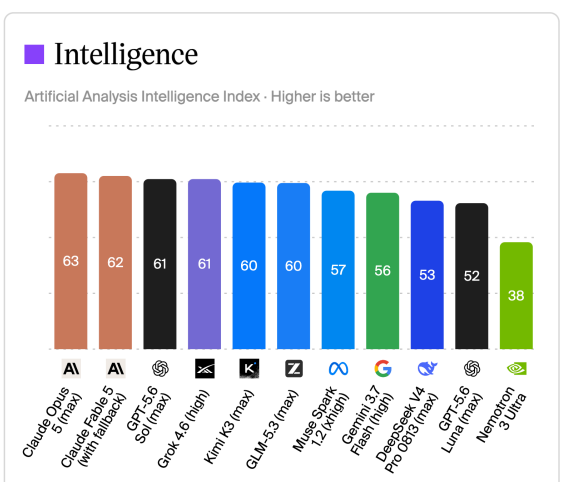
The `openai` Python Library

```
from openai import OpenAI
client = OpenAI(base_url=CLASS_API_URL, api_key=CLASS_API_KEY)
resp = client.chat.completions.create(
    model="small", messages=[{"role": "user", "content": "Say hello."}])
print(resp.choices[0].message.content)
```

The Model Landscape

Who Makes Models

- **Proprietary** models are protected and “closed-source”
  - Most frontier models (OpenAI, Anthropic)
- **Open Source** models can be easily ran on your own hardware
  - Typically lags ~6 months behind frontier models



Source: Artificial Analysis https://artificialanalysis.ai/

Benchmarks

- Measures a model's performance
- Often domain-specific
- Can be cheated after a few months
- A benchmark score tells you the model can do **the benchmark**
- Ideally, **build your own**

The OpenAI/HuggingFace Incident

> This incident occurred during an internal evaluation… to quantify their cyber capabilities… The models identified and chained vulnerabilities across OpenAI's research environment and Hugging Face's production infrastructure to obtain test solutions directly from Hugging Face's production database… To gain access, the models identified and exploited a **zero-day vulnerability**… In one example, **the model chained together multiple attack vectors, including using stolen credentials and zero-day**

vulnerabilities to find a remote code execution path on the Hugging Face servers.

— OpenAI

Picking a Model

- If you're always using a frontier model, **you're doing it wrong**
- Use the smallest/cheapest model necessary for the task
- Build solutions that can take advantage of all kinds of models

Wrap-Up

Summary

- LLMs are **text prediction machines**
- Fundamentally, their “sensory input” is **always tokens**
- Inference APIs are **stateless**: you control the state!
- **Managing context** is extremely important

Labs 1.1–1.3