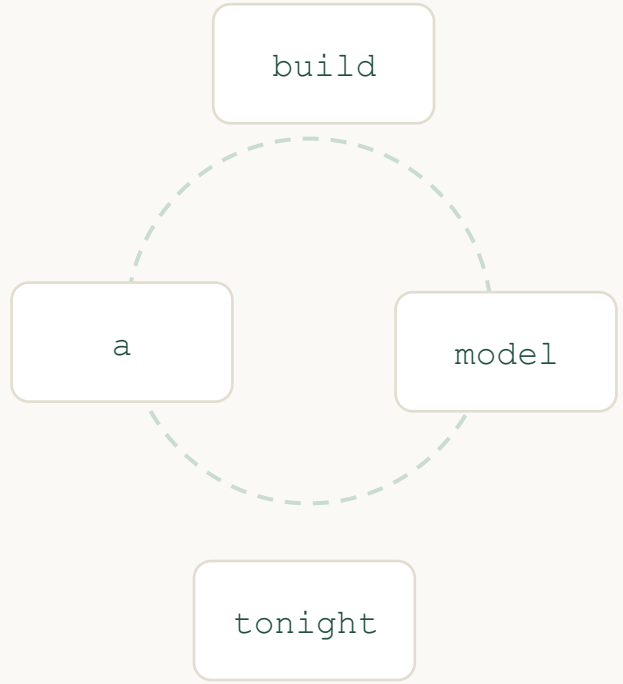
INDUSTRIAL AI — WEEK 2

Language models, from the inside.



Tonight you build one from an empty file and watch it learn to write — then price three real ones against each other, to the dollar.

Lecture 2 of 24 · Industrial AI · August 2026 · Zoom + Arfa Tower

WHERE WE ARE

Week 1 was the map. Tonight we open the engine.

Month 1 builds one capability per week. Tonight is the substrate everything else stands on.

W1	The landscape — workflows vs agents, grounding, the trust ladder	
W2	**Language models — tokens, the loop, the economics**	*tonight*
W3	Retrieval — grounding the model in your documents	
W4	Agents — letting it act, safely	

The promise: understand tonight, and nothing later in this course is magic — only engineering.

TONIGHT'S CLIENT STORY

Two failures, one root cause

> *“A bank's assistant confidently quotes a policy clause that does not exist.”*

Fluent, formatted, cited — and invented. By 8:00 you can explain why, in one paragraph.

> *“The per-token bill is already line two of the budget.”*

Nobody priced a token before the pilot. By 9:00 you can price any workload before writing code.

One mechanism explains both. That mechanism is the next 40 minutes.

TONIGHT'S PLAN

Two hours, five acts

I	**What the model sees**	— tokens are the unit of cost — measured live on your language	7:08
II	**How it writes**	— the loop, the dice — and why hallucination is the mechanism	7:26
III	**What training buys**	— base → instruct → reasoning · the economics · the licences	7:35
IV	**Build one**	— a GPT from an empty file, training live on this laptop	7:48
V	**Price three**	— the lab with the TAs — quality × latency × cost per 1,000 calls	8:25

One rule tonight: nothing is asserted that isn't then measured, trained, or priced in front of you.

What this week is worth

Worth	Nothing on its own. Everything as the substrate — every AI product you will ever evaluate sits on one of these.
Ask your team	“What does one user-session cost at our volume — and what is the licence on the base model?”
Bad build tell	A per-token cost nobody has calculated. Or “it's open source” said of a model with an acceptable-use policy.

The tokenizer playground

```
$ python 01_tokenizer_playground.py --live

> The boiler pressure is high
  [The][ boiler][ pressure][ is][ high]
  5 tokens  · 5.6 chars/token

> مصنوعی ذہانت صنعتی پیداوار کے معیار…
16   tokens  · 3.3 chars/token
  1,000 calls, input side → $0.048

> def alarm(t): return 'CRIT' if t>90
  19 tokens  · 2.9 chars/token
```

Type, don't paste

an English sentence
the same sentence in Urdu
a line of Python

Watch the count.
Read the session cost line aloud.

What you just watched

A token is not a word, and not a character.

It is a frequent chunk of bytes, learned from the vendor's corpus — and it is the unit of cost.

The corpus was mostly English.

So Urdu pays ~2.7× per character. Measured live, five minutes ago, on this laptop.

The model cannot see inside a token.

Counting the r's means opening a unit it cannot open. Not a bug. Not fixed by scale.

`strawberry`

3 tokens. *Zero letters.*

“How many r's in strawberry?” — the model literally cannot look.

Byte-pair encoding, complete

```
1  every byte is a token        vocab = 256
2  count adjacent pairs
3  merge the most frequent pair
4  repeat, to vocab_size

   ('t','h') seen 640×  →  'th' = token 257
```

That's it.

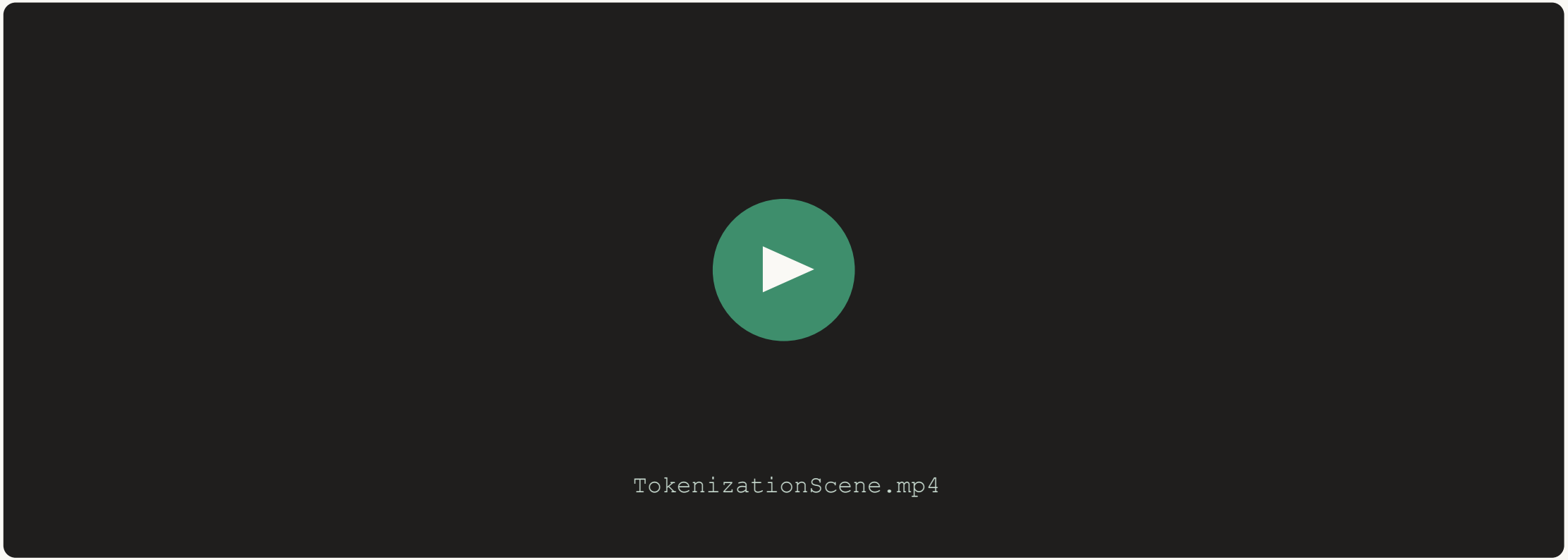
Compression, learned from data.
Everything else is engineering.

Theirs: 200,000 merges on terabytes.
Ours tonight: 80 merges on Shakespeare,
in 60 lines you can read.

"the machine learning model" →

ACT I · ANIMATION · 40 SECONDS

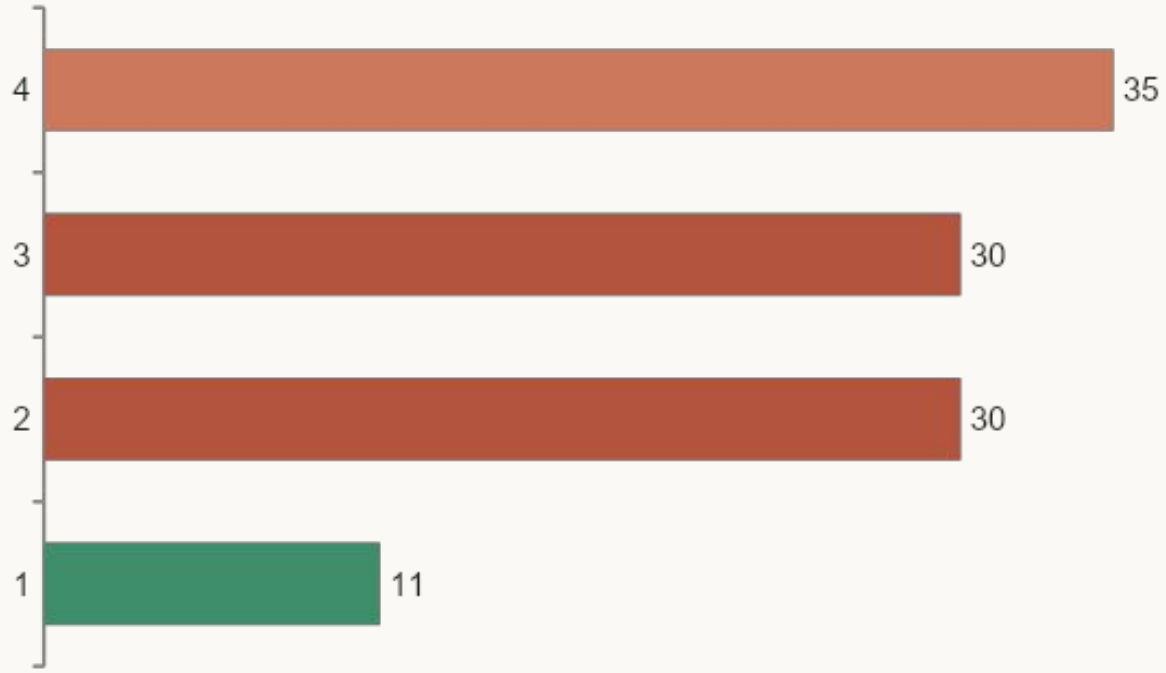
What the model actually sees



After 3Blue1Brown's “But what is a GPT?” — our scene code ships in animations/, reusable every week.

Reading the bill

Tokens per 100 characters — the numbers Demo 1 just measured on this laptop.



Every price sheet has two numbers

`input`	**~$5 / Mtok**
`output`	**~$25 / Mtok**

Output is 3–5× input — and reasoning models burn the dear kind.

Act I done: you can price a workload. Act II — why the thing that costs this much sometimes lies.

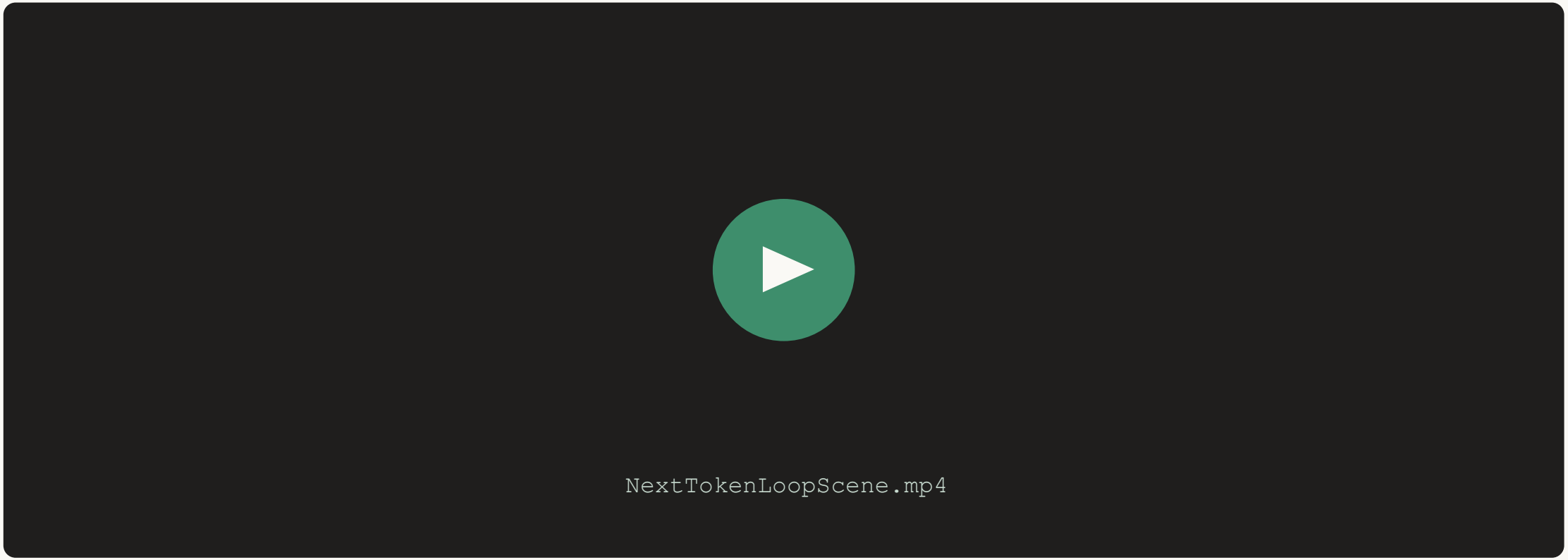
Next-token prediction is the entire product

```
logits = model(context)          # a score for every token in the vocabulary
probs  = softmax(logits / T)     # scores → probabilities
next   = sample(probs)           # roll the dice
context.append(next)             # and again. forever.
```

There is no step two. *Chat, reasoning, agents — packaging around this loop.*

ACT II · ANIMATION · 45 SECONDS

One token at a time



Watch the sampled token get appended: the model reads its own words back. Hold that thought.

Temperature: loading the dice

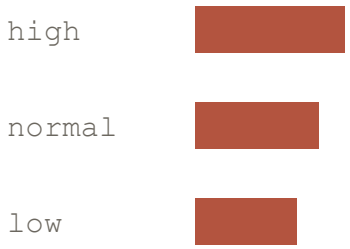
T → 0

greedy — the argmax, every time. Deterministic, repetitive, safe.

T = 0.7

the production default. Coherent, with variety.

T = 1.4



flattened — rare tokens get real probability. Chaos.

Same model. Same weights. Different dice.

▸ `TemperatureScene.mp4`

Hallucination is the mechanism working correctly.

The model never knows. It scores. When the true answer never appeared in training, something else scores higher — and gets said with identical confidence, because confidence is not a thing the loop has.

ACT II · BACK TO THE BANK

So what happened with Clause 14.2?

“

“Clause 14.2” was simply the highest-probability continuation of a question about clauses.

Fluent. Formatted. Confident. Never true.

EXIT · THE FIX

You cannot patch this with a prompt. You ground it with retrieval — the model answers from documents you put in front of it, with citations.

→ Week 3

Say it in every meeting: “the model didn’t lie — nothing in its context knew better.” Act III — what training does fix.

Base, instruct, reasoning

`base`

continues text — that is all pretraining teaches.

"What is 2+2?" → it asks you three more questions

`instruct`

post-trained to follow the chat template and answer.

"What is 2+2?" → "4."

`reasoning`

spends extra output tokens thinking first.

slower · the 5×-priced tokens · pays off on multi-step work only

Tonight's model is a base model. Ask it anything — it will Shakespeare at you regardless. Now you know why.

“Instruct” is a formatting convention, trained in

```
<|system|> You are a maintenance triage assistant
<|user|> Pump P-104 tripped twice on overload…
<|assistant|> {"equipment": "P-104", …

a base model has never seen these tags.
that is the whole secret of instruct models.
```

Post-training taught it one habit: after user: comes a helpful assistant:. Nothing deeper. Week 12, you train one yourself.

The context window is priced per token, per call

```
system prompt — stable, cacheable at ~10% price
```

```
retrieved documents & history — quietly dominates the bill
```

```
the user's question — the only genuinely new tokens
```

…and every call resends all of it.

Four cost facts

output ≈ 3–5× the input price

long context: slower AND dearer — mechanism in Week 9

a cached prefix ≈ 10% price — prompt order matters

attention sags in the middle of a long window

A prompt is a specification. Treat it like one.

Do

- zero-shot, then few-shot, then role design — measured, in that order
- structured output through a schema the code checks
- one job per call
- version-controlled and reviewed, like any spec

Smell

- “respond ONLY with JSON” — pleaded, three times
- one mega-prompt doing five jobs
- prompt_v_final_FINAL2 in a notepad
- expecting a prompt to add knowledge the model never had

What prompting cannot fix: knowledge it never had → retrieval, W3. Behaviour drilled in → fine-tuning, W12.

“Open source” is doing a lot of work in that sentence

OSI licence — Apache-2.0, MIT	`Qwen · OLMo · DeepSeek-R1`	*ship it. Read the NOTICE anyway.*
open weights, custom terms	`Llama (naming + AUP) · Gemma (AUP)`	*usually — obligations travel with derivatives*
non-commercial · research	`assorted “research licences”`	*no. Whatever the benchmark says.*
closed API	`the frontier providers`	*you ship a dependency — and a residency question*

Opened tonight: the register, row zero. That closes Act III — now we stop describing models and build one.

Now we build one

1 · character tokenizer

vocab = 65 distinct characters. BPE's baby sibling.

2 · bigram baseline

next char from current char alone. 30 seconds. Gloriously dumb.

3 · add attention

tokens talk to the past — a real, tiny GPT. Same shape as the big ones.

4 · train, sample, watch

1.1M characters of Shakespeare. A sample every ~250 steps.

After Andrej Karpathy's nanoGPT and “Zero to Hero”. Same architecture as the frontier models, minus the zeros. His 2-hour derivation is your watching this week.

Watch it learn to write

```
step 0 — loss 4.36 (= guessing)
dFbHdt3BCwxq$wR,cdImWT,j'ltuK gjgeSsk;BPP
```

```
step 500 — loss 2.31
Taghe oul.
Here p berealeareakes thicherenos tmy benter
```

```
step 3000 — loss 1.74
CLIFFORD:
Now, the with new in the gone, that not brother.
Thou so see that fall of and arm your by branch
```

Narrate the loss.

4.17 = pure guessing, ln(65). Watch it fall.

Read every sample aloud.

Noise → words → verse, in about two minutes, on a laptop.

Everything in the file, and whose week it is

`CharTokenizer`	encode / decode — ten lines. You already know this.	*tonight*
`get_batch`	targets = the same text shifted one right — the whole trick of training data	*tonight*
`CausalSelfAttention`	tokens read the past; one masked line bans the future	*derived in Week 9*
`generate()`	logits → ÷T → softmax → sample → append. Demo 1's loop, verbatim.	*tonight*
`the training loop`	forward · loss · backward · step	*owned by Week 7*

Nothing in the file is magic. Some of it is just early. Act V — now we price the grown-up versions.

ACT IV · THE REVEAL

A frontier model is the same loop — with a million times more of everything

0.83M parameters · vocab of 65 characters · **two minutes on a laptop** · loss **4.17 → 1.74**

Scale buys grammar, then facts, then instruction-following, then reasoning.

Scale does not buy truth. It is still next-token prediction — hallucination comes standard.

```
$ python 03_sampling_knobs.py --temps 0.0 0.7 1.4     # demo 3 · recorded
T=0 loops politely · 0.7 writes · 1.4 raves
— on the model you just watched being born
```

The lab: one task, three models, one table

`local`	open weights on your laptop — Ollama	$0 per call · 67% quality tonight
`hosted`	open weights behind a cheap API	fast · ≈ $0.01 per 1,000 calls
`frontier`	a frontier API	best quality · ≈ $2.50 per 1,000 calls

```
$ python 04_model_comparison.py --offline
  zero keys · zero wifi · full table

  local 67%   hosted 94%   frontier 100%
```

Scored, not vibed. 18 deterministic checks — the right answers were written down before any model was called. That habit is Week 8, starting now.

ACT V · THE GRADED ARTIFACT — DUE THURSDAY 23:59

What you submit

1 **your trained tiny LM** — checkpoint + a 200-token sample at T=0.7 + one paragraph on the arc you watched

2 **the comparison table** — offline mode grades identically — plus one failure, classified fails-safe or fails-dangerous

3 **cost sheet, page one** — your product idea, your measured tokens, your estimated traffic

4 **licence register, row zero** — every model tonight touched — ours included; its corpus counts

Stretch: retrain on your own corpus — `--data your.txt` — Urdu poetry, your code, plant logs. Best sample gets shown next week.

“This model is ‘open source.’ What do you check before you believe that — and what can still stop you shipping it?”

Answers in the Discord thread by Thursday. Best one gets dissected next week — that's a promise, not a threat.

Next week — the fix for Clause 14.2. Retrieval, embeddings, answers with citations. Pre-work drops tonight.