

Season 1: How LLMs work

The transformer.

How embeddings, attention, and layers fit together into one model

One design behind most current LLMs

- **Why it won:** training processes a whole sequence at once, so it scales across huge amounts of hardware
- **Why calls cost what they do:** speed and price follow from its shape
- **Why models behave as they do:** fixed context, frozen knowledge, and no memory all start here

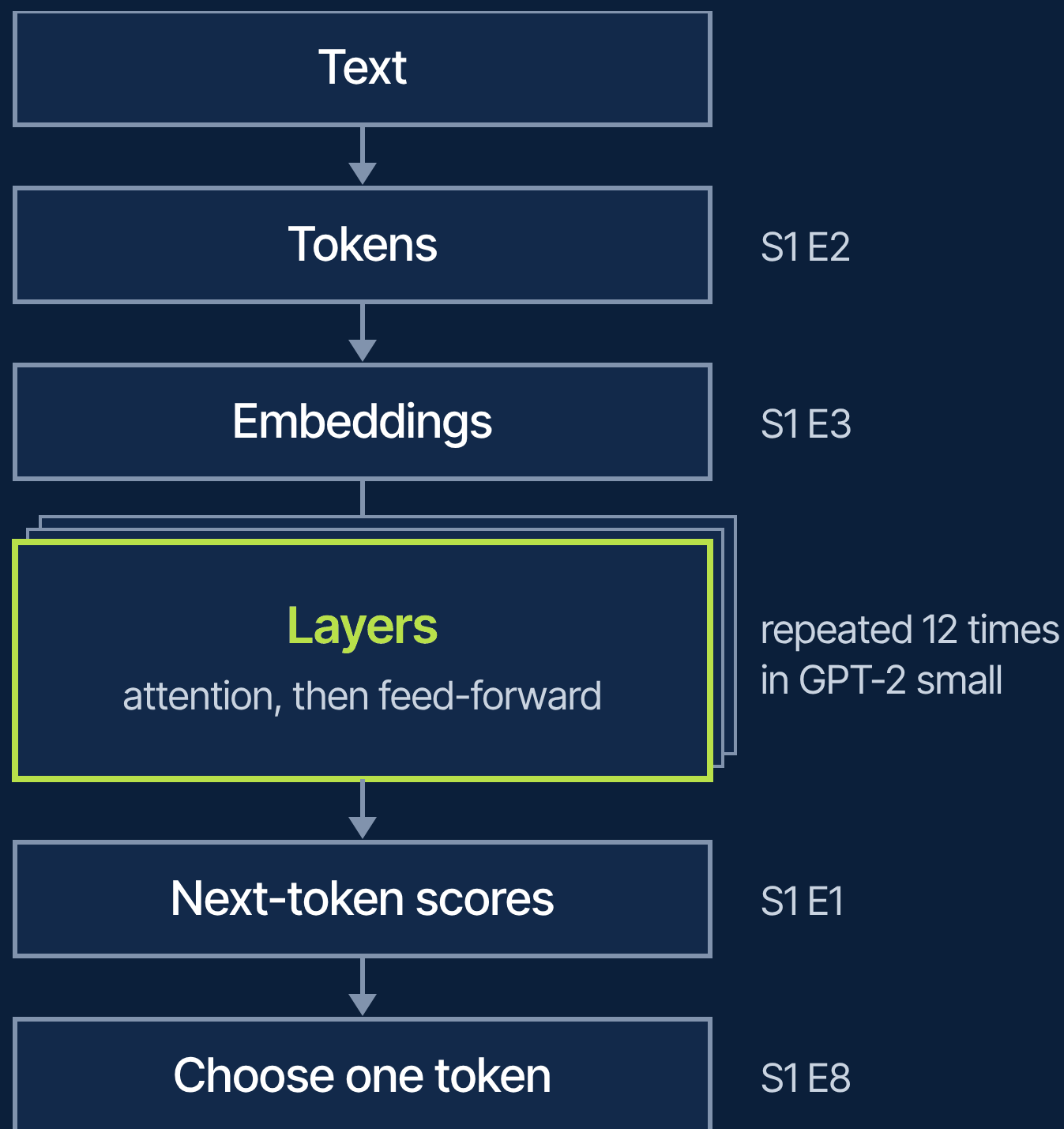
A stack of layers, each refining every token

Tokens go in as embeddings (S1 E3), pass through a stack of same-shaped layers, and leave as next-token scores (S1 E1).

Where it breaks

No single layer holds a fact. Knowledge is spread across the whole network, 124 million weights even in GPT-2 small, so it cannot simply be looked up, corrected, or deleted.

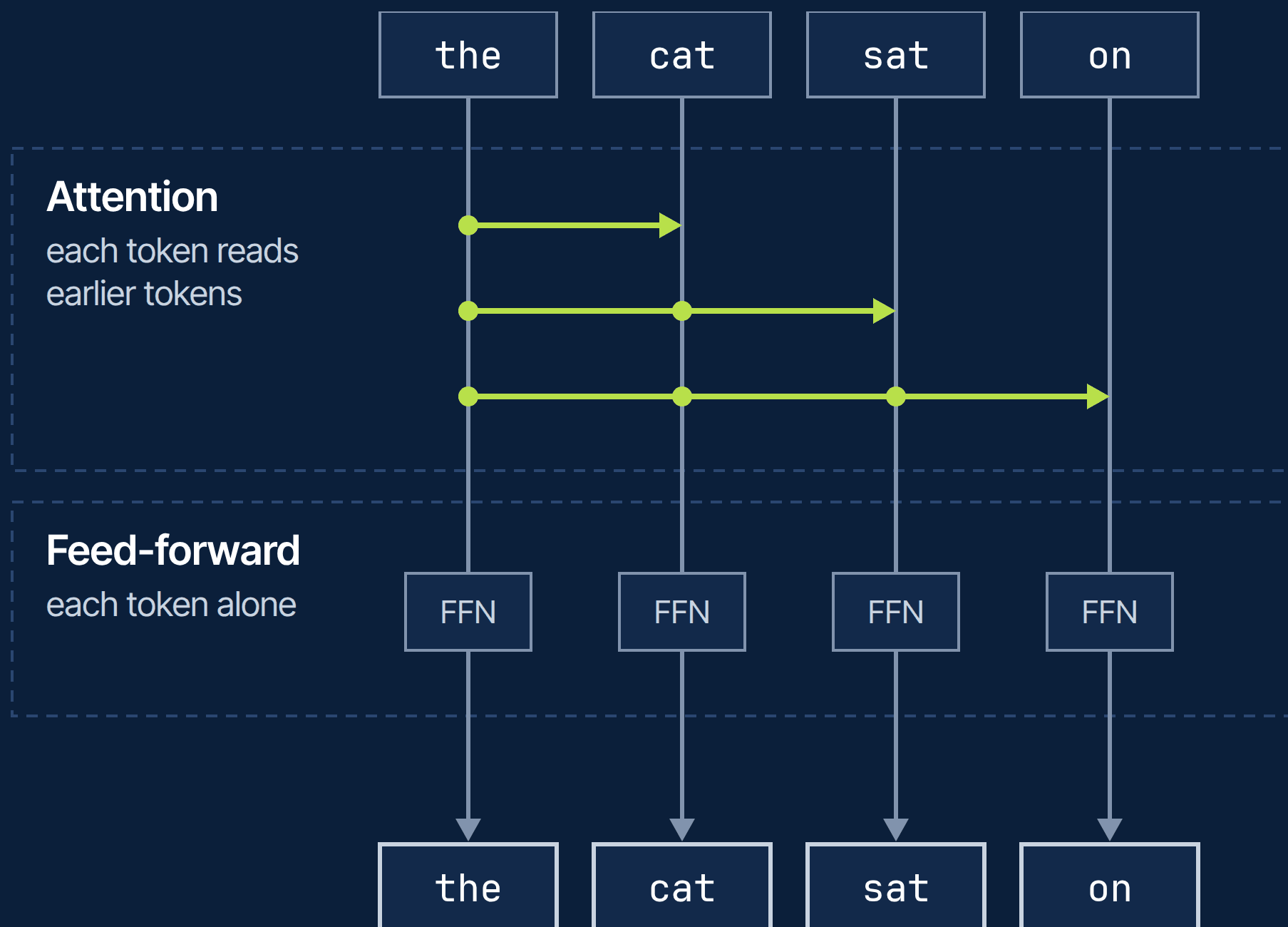
From text to the next token



Layer count from GPT-2 small, the model measured in S1 E4.

Inside one layer

Attention is the only place tokens exchange information.



Information flows forward only: "on" reads "the", "cat", and "sat", never the reverse

What each part does

Attention

Moves information between tokens, looking back only (S1 E4)

Feed-forward

Transforms each token on its own. Much of what a model learned is thought to be stored here

Residual connection

Each layer adds to a token's vector rather than replacing it, so earlier information carries through

Normalisation

Keeps the numbers in a stable range as dozens of layers stack up

GPT-2 small, by the numbers

Read from the loaded model, the same revision as S1 E4.

Layers	12 Refinement steps every token passes through, in order
Heads per layer	12 Attention patterns running side by side in each layer (S1 E4)
Vector size	768 numbers per token How each token is represented as it moves through (S1 E3)
Vocabulary	50,257 tokens Every possible next token; each step scores all of them (S1 E2)
Context	1,024 tokens Most it can see at once, prompt and answer together (S1 E6)
Parameters	124,439,808 Learned numbers; about 500 MB stored at 32-bit precision

Where the 124 million parameters live

Token embeddings	Attention	Feed-forward
38.6M, 31.0%	28.3M, 22.8%	56.7M, 45.5%

Shares computed from the layer shapes; they sum to the exact total. The last 0.7% is position embeddings and normalisation, and output weights share the embedding table. Released in 2019; frontier models are vastly larger, and most providers do not publish sizes.

Parallel to train, one token at a time to run

Training

Every position is predicted at once, because the right next tokens are already known

Reading a prompt

Input tokens are also processed together, in one pass

Generating

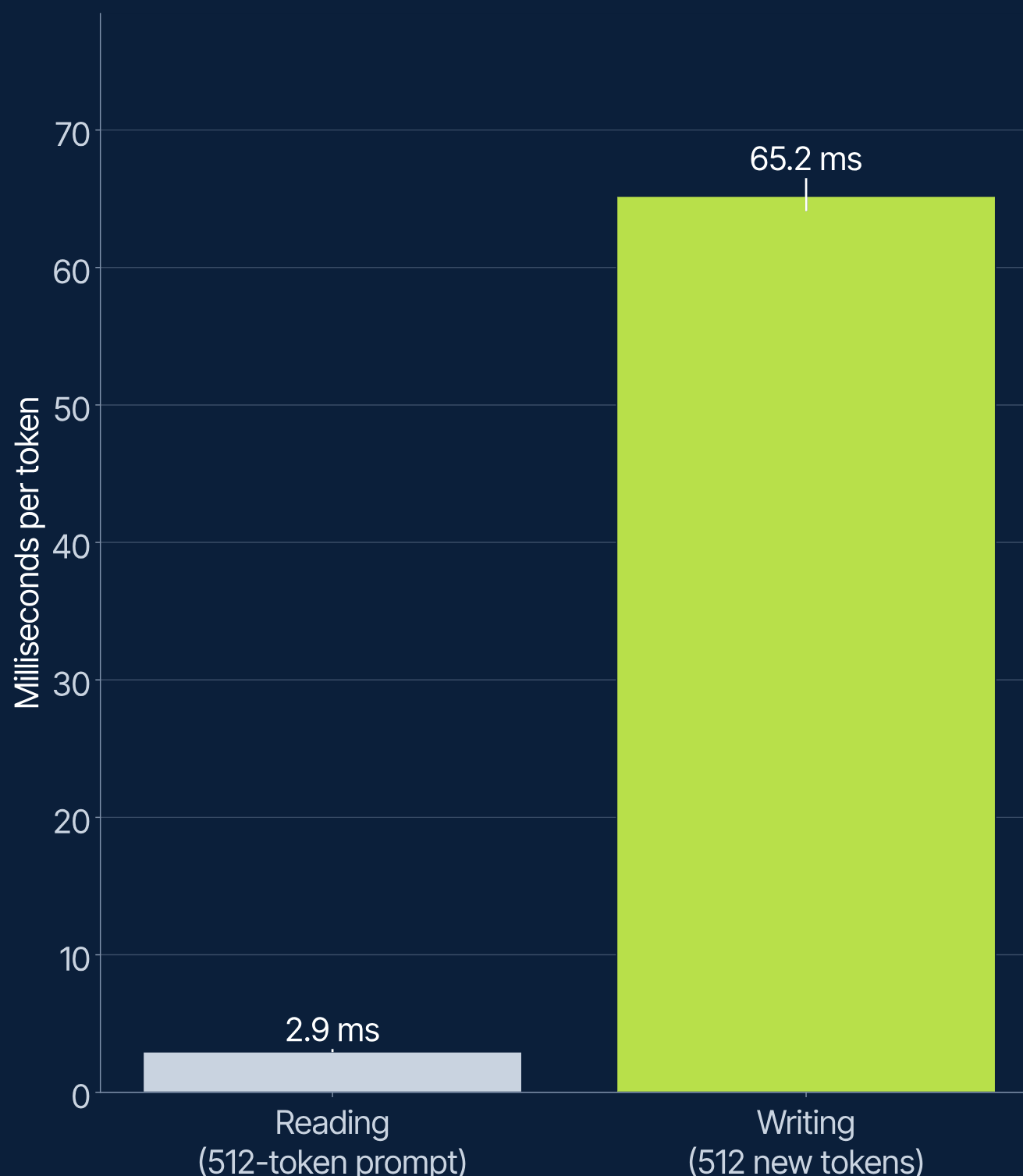
Each output token needs its own pass through every layer (S1 E1)

The consequence

Output tokens are slower to produce, and providers usually price them higher than input tokens.

Writing took 22 times as long per token as reading

512 tokens each way, median of 5 runs. Green marks the slower.



GPT-2 small on Intel Core i7-8565U @ 1.80GHz, 4 threads

Writing takes 22.1 times as long per token as reading.

Bars are medians; whiskers run from minimum to maximum.

Run it yourself

The same GPT-2 small as S1 E4, on your own machine.

```
tokenizer = AutoTokenizer.from_pretrained(
    "gpt2", revision=REVISION)
model = AutoModelForCausalLM.from_pretrained(
    "gpt2", revision=REVISION).eval()
long, short = prompt(tokenizer, 512), prompt(tokenizer, 8)
with torch.no_grad():
    start = time.perf_counter()
    model(**long, logits_to_keep=1)
    read = time.perf_counter() - start
    start = time.perf_counter()
    model.generate(**short, do_sample=False,
        min_new_tokens=512, max_new_tokens=512)
    write = time.perf_counter() - start
print(f"reading: {read / 512 * 1000:.1f} ms per token")
print(f"writing: {write / 512 * 1000:.1f} ms per token")
```

■ One read of 512 tokens, against 512 separate writes

[episodes/s1-e5-the-transformer/timing.py](#). The listing prints one cold run; the figures on these slides are medians of 5 runs.

Limits built into the design

- **Fixed context:** the model sees only what fits in its window (S1 E6)
- **Frozen knowledge:** what it knows stops when training stops
- **No memory:** nothing carries over between calls (S1 E1)
- **Cost grows with length:** attention compares every token with every earlier one (S1 E4)

What the shape means for your system

Latency

Long answers are slow answers: here, 33 seconds to write what took 1.5 seconds to read. Cap output length

Reuse within a call

Models keep earlier work (the KV cache) while generating. With it off, writing was about 12 times slower again

Reuse across calls

Providers can reuse a repeated prompt opening rather than recompute it (S1 E4)

Model size

More layers and wider vectors cost more per token. Test smaller models first

What breaks in production

- **Blaming prompt length for latency:** when most of the delay is the length of the answer
- **Largest model by default:** paying for capability the task never uses
- **Expecting current knowledge:** asking about events after the training cut-off
- **Treating open weights as explainable:** having every weight does not tell you why it answered as it did

Choosing a model is a cost and speed decision

Most current LLMs share this design at different sizes. Within a family, bigger usually means more capable, and slower and more expensive on every call.

Ask your team

Have we tested whether a smaller model meets our quality bar, and what does each option cost per task?

Decision rule

Choose the smallest model that passes your own tests, and budget latency on output length.

Next: S1 E6 Context windows

What fits in a model's view, and what happens at the edge.

Where this fits

The series map

S1 LLMs	S2 Retrieval	S3 Strategy
S4 Adapting	S5 Agents	S6 Economics
S7 Data	S8 Evaluation	S9 AI teams
S10 Reliability	S11 Security	S12 Delivery
S13 Performance	S14 Resilience	S15 Governance

☐ Leadership season

 Follow	linkedin.com/in/brendan-kelly-irl
 Episode guide	brendankellyai.github.io
 Code	github.com/BrendanKellyAI/applied-ai-lab