syncro soft

Open4Tech Summer School 2026

From Explainable AI to Explainable Software

Alexandra Udristoiu

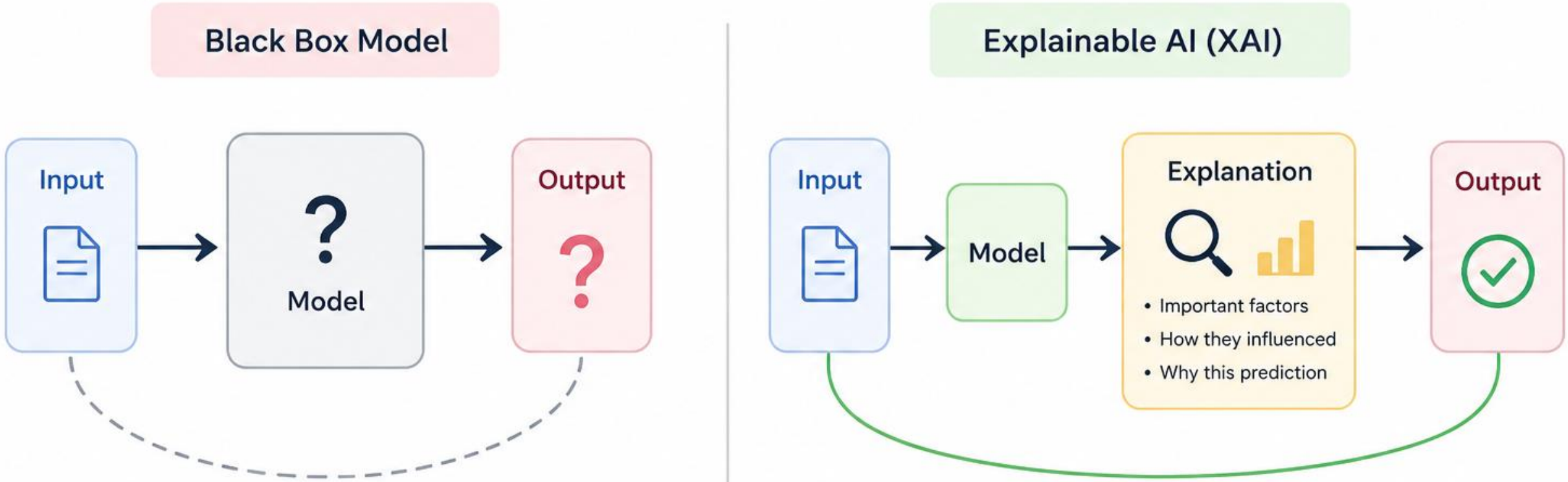
Software Developer, Syncro Soft

What is XAI?

- As AI systems become more complex, understanding how they make decisions becomes increasingly difficult.
- Explainable AI provides methods for understanding and interpreting AI predictions and decisions.
- XAI helps identify which inputs and patterns influenced a model's output.
- XAI surfaces biases, errors, and unexpected behaviours.
- XAI is essential for trustworthy, transparent, and accountable AI systems.

Explainable AI (XAI)

From predictions to understanding



Core Principles

- **Transparency** – Make AI decisions visible and understandable.
- **Interpretability** – Explain why a prediction was made.
- **Trustworthiness** – Enable informed trust in AI systems.
- **Fairness** – Detect and mitigate bias.
- **Accountability** – Support responsibility for decisions.
- **Robustness** – Provide reliable and consistent explanations.

Why it matters?

Healthcare: disease diagnosis, medical image analysis

Finance: loan approval, fraud detection

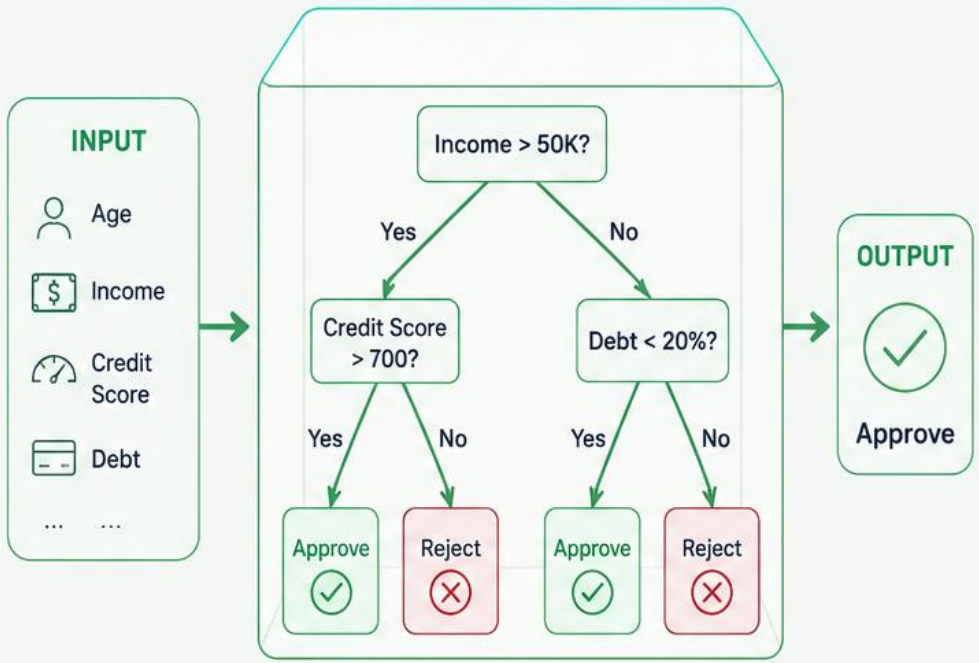
Autonomous vehicles: object detection, route planning, driving decisions

Recruitment: CV screening, candidate ranking

The higher the impact of an AI decision on people's lives, the more important it becomes to understand and explain that decision.

WHITE BOX MODEL

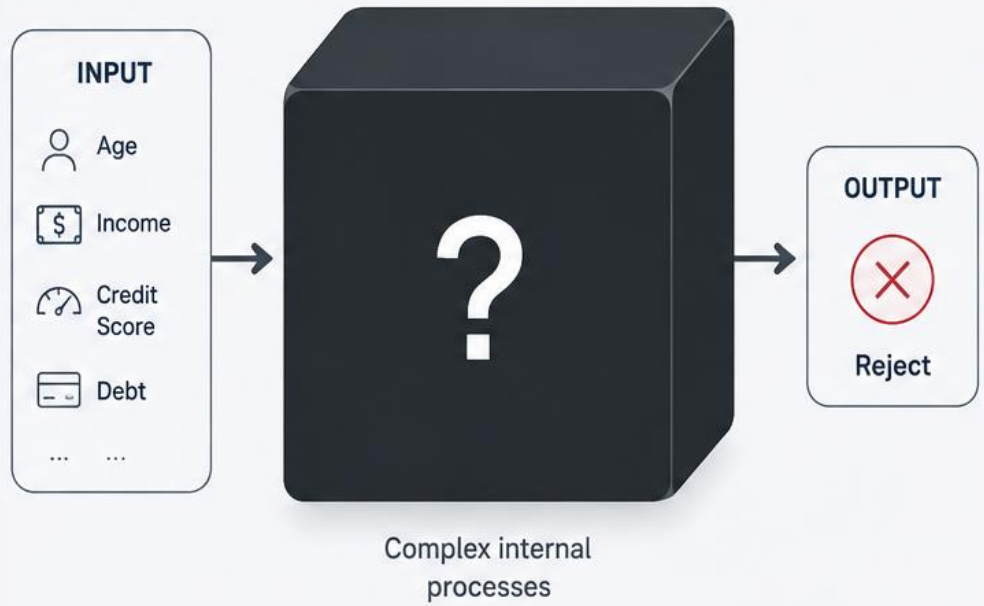
(Interpretable)



- You can see the logic and rules
- Easy to understand and explain
- Easier to debug and trust

BLACK BOX MODEL

(Not Interpretable)



Complex internal processes

- You cannot see how it works inside
- Difficult to understand or explain
- Harder to debug and trust

Feature importance

- Feature importance measures how much each input contributes to a model's predictions.
- It helps identify which factors have the greatest influence on the model's behaviour.
- Feature importance provides a **global explanation** of the model rather than explaining individual predictions.
- It can reveal unexpected dependencies, biases, or data quality issues.

LIME

- Explains a single prediction locally.
- Tests small input changes to see how outputs respond.
- Fast, intuitive, and works across text, images, and tabular data.
- Limitation: Uses random sampling, so explanations may vary slightly between runs.

SHAP

- Detailed explanation for one prediction.
- Fairly distributes feature contributions (game theory).
- Shows each feature's impact on the outcome.
- Reliable and consistent.
- Computationally expensive; often approximated.

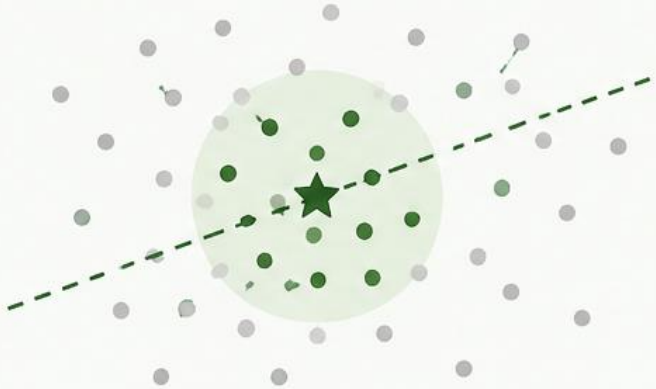
LIME vs SHAP

LIME

Explains this prediction locally

Predicted price:
$450,000

Focus: This house only



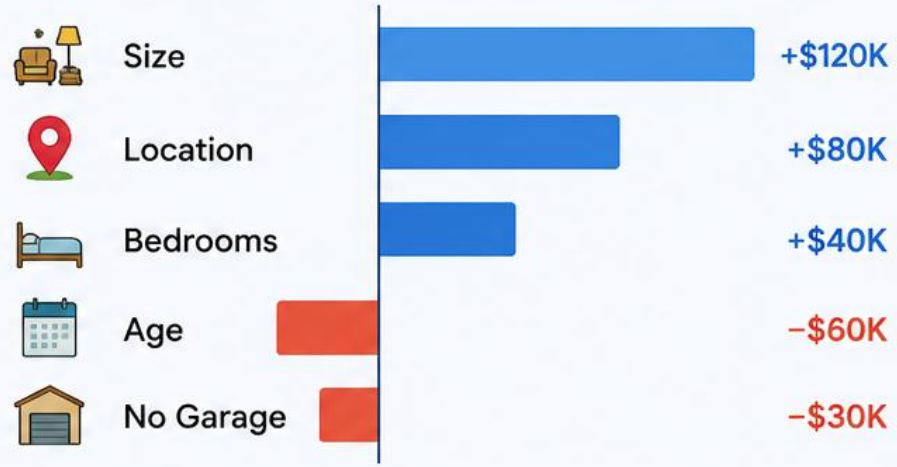
Looks at similar houses near this one to explain the prediction.

SHAP

Explains this prediction using all data

Predicted price:
$450,000

Focus: All features (global + local)



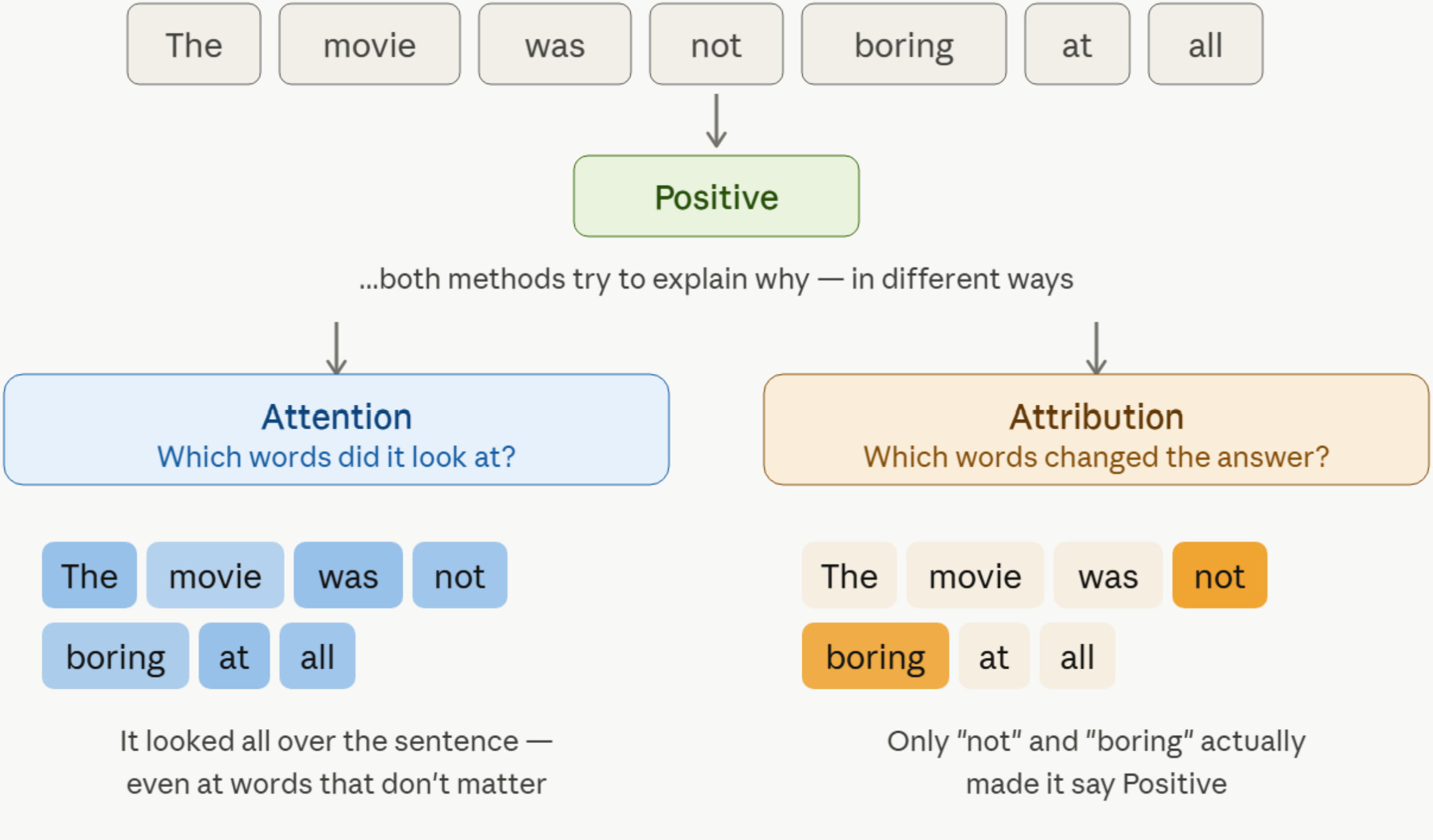
Shows how each feature pushes the price up or down.

Grad-CAM heatmaps

- Grad-CAM (Gradient-weighted Class Activation Mapping) visualizes which regions of an image contributed most to a model's prediction.
- It highlights the areas the model focused on when identifying an object or class.
- Grad-CAM provides a **local explanation** for a specific prediction.
- It helps verify whether the model is using meaningful visual features.

Explainability in LLMs

- LLMs make decisions based on billions of parameters and complex interactions between tokens.
- Unlike traditional ML models, there are often no clear features that directly explain a prediction.
- Understanding why a particular response was generated remains an active research challenge.
- Explanations often focus on what information influenced the output rather than the model's internal reasoning.
- XAI is used for trust, safety, debugging, and detecting hallucinations.


The movie was not boring at all
Positive
...both methods try to explain why — in different ways
Attention
Which words did it look at?
Attribution
Which words changed the answer?
The movie was not boring at all
The movie was not boring at all
It looked all over the sentence — even at words that don't matter
Only "not" and "boring" actually made it say Positive

Attention visualisation

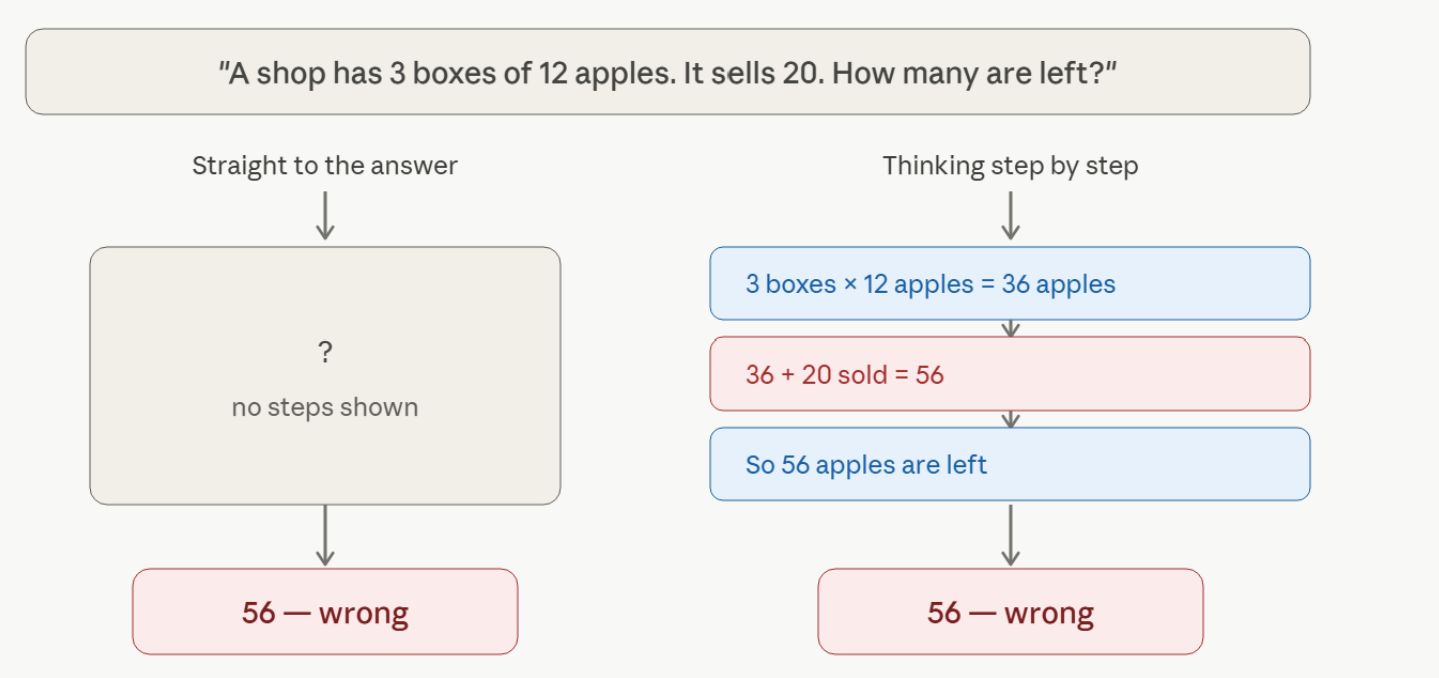
- Shows which tokens the model *looked at* for each part of its output.
- Read straight from the model's internal weights: cheap and already there.
- Gives an intuitive heatmap over the input.
- Catch: attention isn't influence. A token can be looked at heavily and still not change the answer.

Token attribution

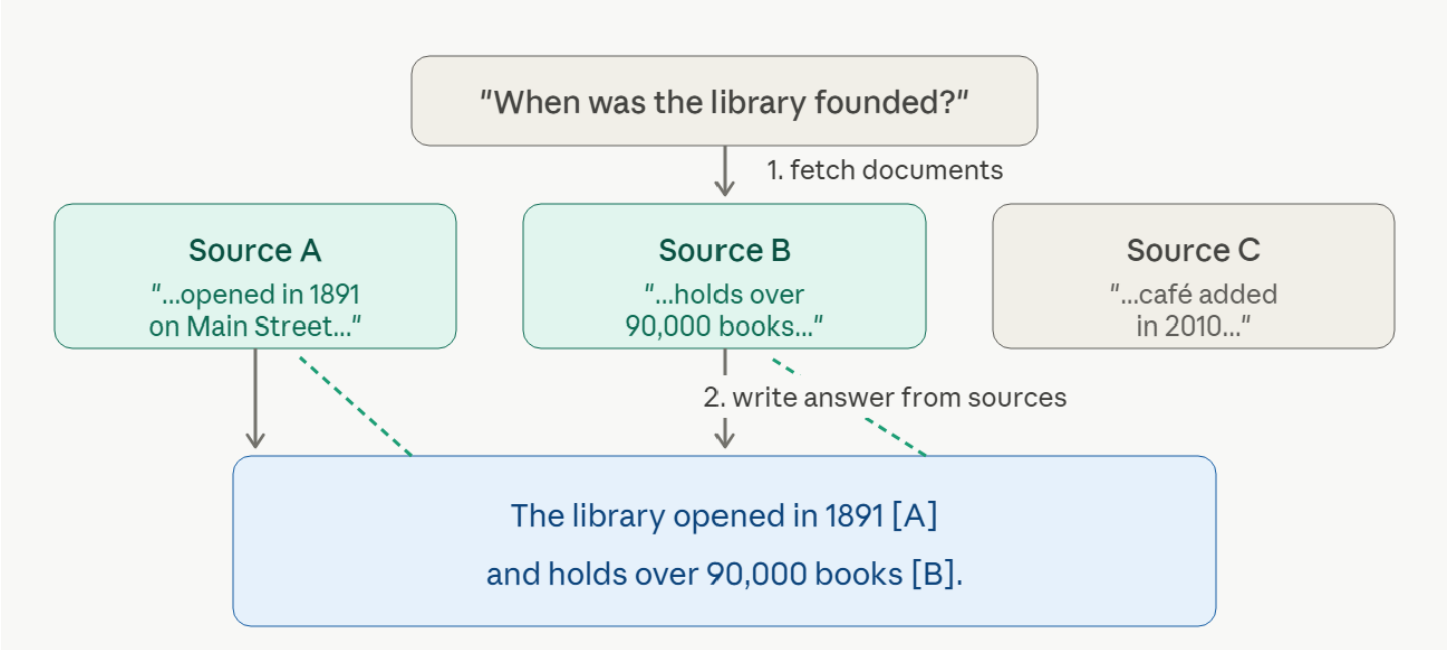
- Shows which tokens actually *changed the output*.
- Works by perturbing tokens or measuring gradients and watching the prediction shift.
- Targets causal influence: what made the model say this, not just what it saw.
- Catch: more expensive, and different methods can disagree on the same input.

Chain-of-Thought Reasoning

- Model "thinks step by step"
- Writes out intermediate reasoning
- Makes logic legible and checkable
- Easy to spot where reasoning breaks
- Caveat: stated steps may not match real reasoning



RAG source inspection



- Model fetches documents, then answers
- Inspect which sources were retrieved
- Trace each claim to its passage
- Most practical, verifiable transparency
- Powers AI search citation features

We demand explanations from AI. Why is "it works" often enough for software?

I want to build a simple assignment submission system for a university course.

Students should be able to:

View available assignments
Submit an assignment before the deadline
Update their submission before the deadline
See whether their assignment has been submitted
View their grade and feedback once graded

Instructors should be able to:

Create assignments
Set assignment deadlines
View student submissions
Grade submissions
Leave written feedback

Please build a complete working web application with a frontend and a small backend.

Use:

HTML, CSS, and JavaScript for the frontend
Node.js and Express for the backend
Any simple storage approach you think is appropriate

Your Assignments

Reading Response: Chapter 4

Missed

A short reflection on the assigned reading.

Due: Jun 16, 2026, 12:02 PM · 3 days ago Points: 20

Deadline passed

Problem Set 3: Recursion

Not submitted

Solve problems 1–8. Submit a link to your repository or paste your code.

Due: Jun 19, 2026, 12:02 PM · 45 min left Points: 10

Submit assignment

Essay 1: Causes of World War I

Not submitted

Write a 1,000-word essay analysing the main causes of WWI. Cite at least three sources.

Due: Jun 23, 2026, 12:02 PM · 4 days left Points: 100

Submit assignment

Can we trust user identity?

- User test: Enter any name and select Instructor, access Instructor functionality.
- The application accepts the claimed identity
- AI intentionally omitted authentication because it was not part of the requirements.
- Implementing authentication requires securing every request, role, and permission.

A "working" system can still create invalid states

- User test: Grade a student 90 / 100. Change the assigment to 50 points.
- Result: The grade is now 90 / 50.
- Individual features work correctly. The system as a whole does not protect its own consistency.

What Happens When Multiple Users Act at the Same Time?

```
const db = readDB();
// modify data
writeDB(db);
```

- Instructor A grades Alice. Instructor B grades Bob. Both save at nearly the same time.
- Real systems must handle concurrent updates safely.
- Some problems only appear when many users interact with the system together.

The Same Rule Exists in Multiple Places

Frontend:

JavaScript

```
if (!a.pastDeadline) {
  actionBtn = ...
}
```

Backend:

JavaScript

```
if (isPastDeadline(assignment)) {
  return res.status(403)
}
```

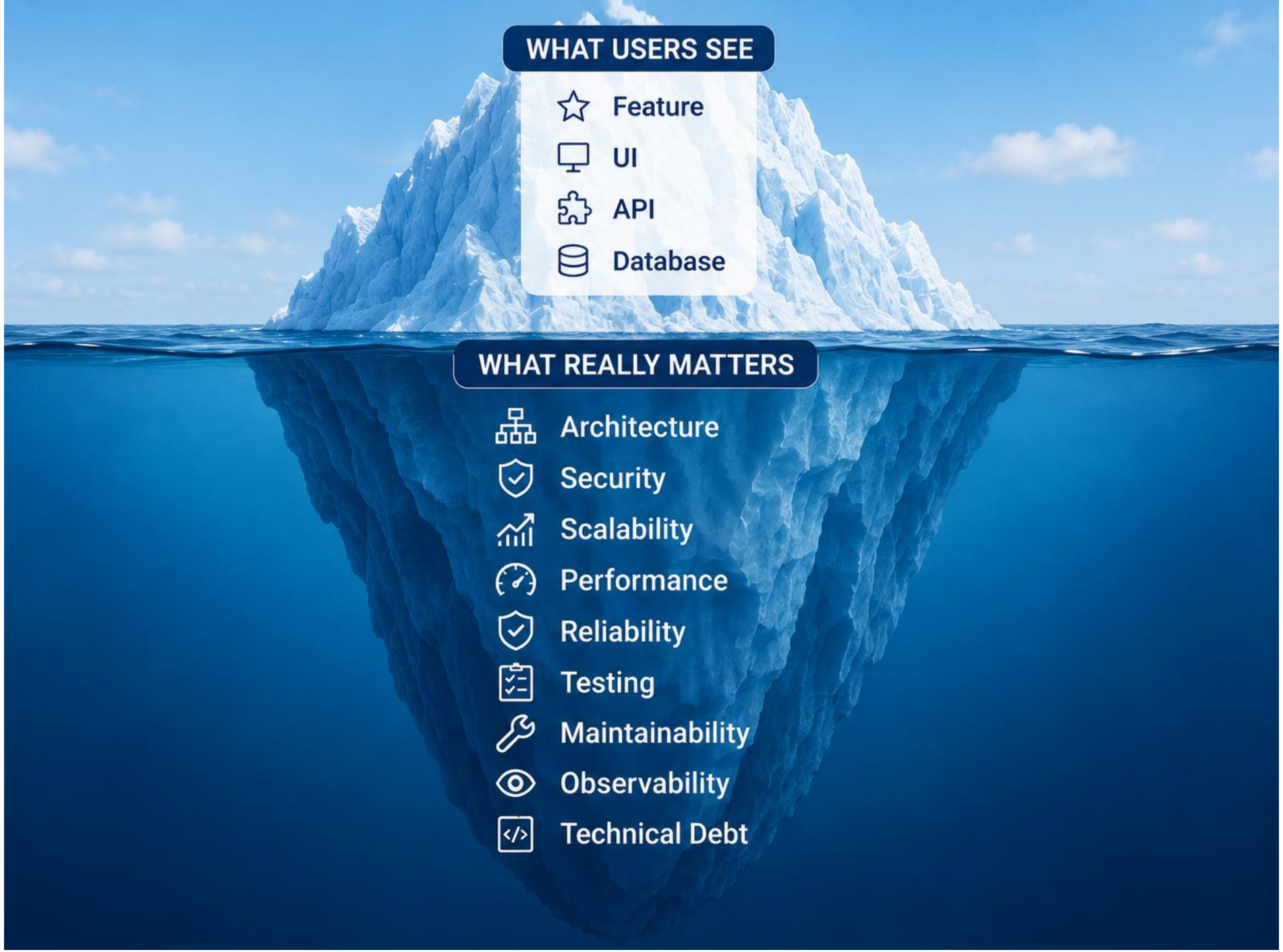
- Requirement: Student may edit until the deadline. The same rule is implemented multiple times.
- Future changes must update every copy to ensure consistent behaviour.

How it relates to XAI?

- That prompt had no single answer. What did the AI decide for you?
- The application works.
- The decisions behind it remain hidden.
- Understanding them later may be far more expensive than making them consciously today.

How it relates to XAI?

- Is the code easy to read?
- Is the logic in the right place?
- Will it be easy to change later?
- What happens when the app gets more users?
- What happens when two users act at the same time?
- Is private data protected?
- What trade-offs did the AI choose for you?


WHAT USERS SEE
Feature
UI
API
Database
WHAT REALLY MATTERS
Architecture
Security
Scalability
Performance
Reliability
Testing
Maintainability
Observability
Technical Debt

Are we asking the wrong questions?

- AI needs more and more files to answer a question -> How do we give it more context?
- AI takes longer and longer to understand a task -> How do we make it faster?
- AI proposes changes across 50 files -> How to accept code we don't actually review?
- **What is this telling us about the way we write code?**

What AI is not so great at

- AI tends to replicate existing patterns, including the bad ones.
- AI does not improve architecture or system design on its own.
- AI does not automatically reduce technical debt or complexity.
- AI struggles to evaluate long-term tradeoffs and business priorities.
- AI cannot take ownership of quality, reliability, or maintainability.
- AI amplifies existing weaknesses just as easily as existing strengths.

What AI is great at

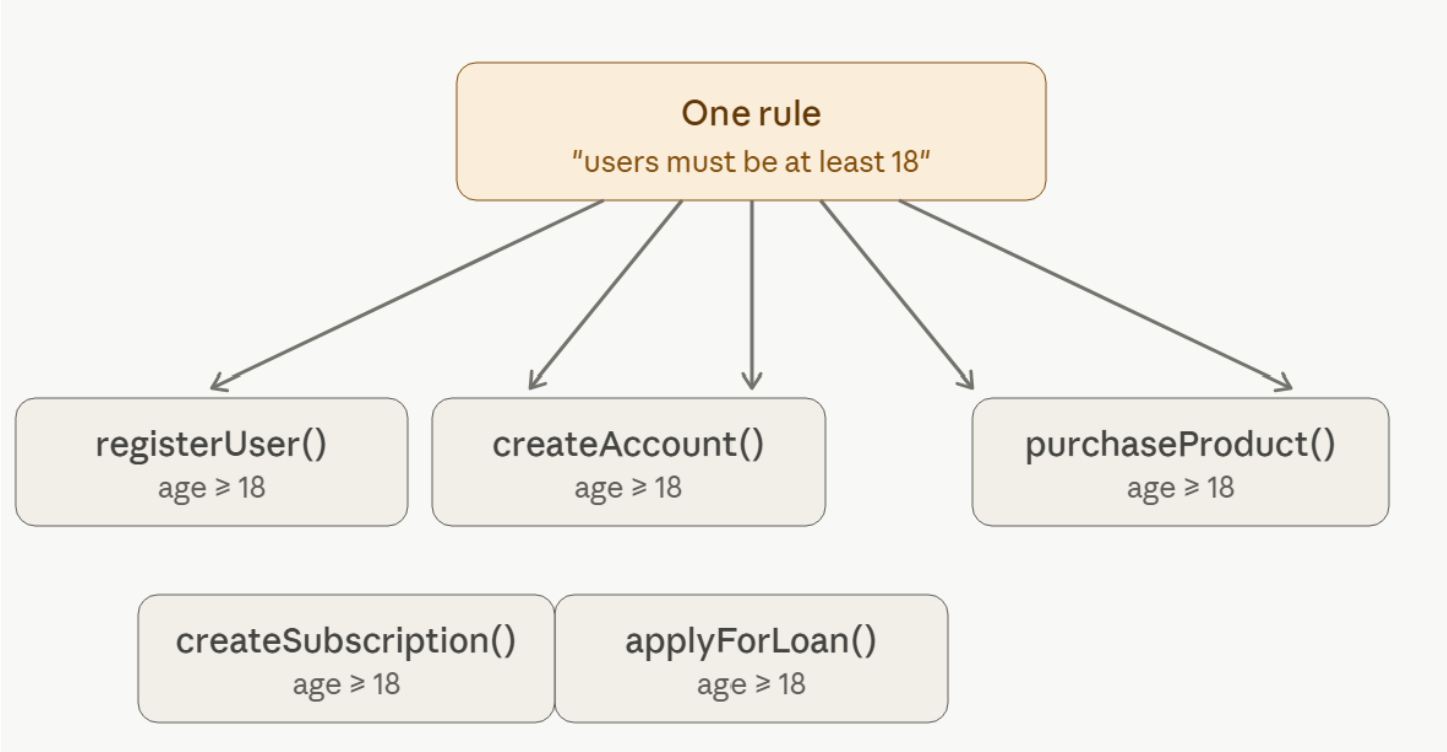
- Accelerates implementation and reduces repetitive work, especially within an architecture that already holds
- Makes unfamiliar technologies and frameworks easy to explore on your terms
- Point it at a hard question and it surfaces trade-offs and options fast; deciding which matter stays your call
- It can prototype three approaches in the time you'd hand-write one, turning architecture questions into things you test rather than guess
- Amplifies good engineering practices and established patterns; the judgment about what to ask is still yours

What you need to be great at

- **Critical Thinking:** Question assumptions, assess risks, and weigh tradeoffs.
- **Learning & Adaptability:** Focus on concepts and patterns; AI accelerates learning new tools and technologies.
- **Simplicity:** As AI can quickly create complexity, reducing it becomes a key advantage.
- **AI opens many doors when you give it structure to build in; without it, those same doors quietly close.**
- **Architecture & Standards:** Strong foundations help systems evolve; AI works best within clear structures.

Back to the duplicated logic problem..

- A new requirement arrives that changes age to 21 years. How many places need to change?
- Solution: Ask AI to change all 5 of them???



Back to the duplicated logic problem..

- How do you know there are still only 5 copies?
- How do you know AI found them all?
- How do you test that they still behave consistently?
- What happens when a 6th copy is added?
- If AI gives you 20 modified files, do you accept them?
- **How can you trust such a system?**
- **What happens when the system evolves even further?**

A short tale of two teams

Same tools, same starting line. How they build decides where they end up.

Beginning

Team A

- Before adding something, they think about how it fits the parts already there.
- They agree on a few shared conventions, so the code stays consistent.
- They keep an eye on how parts depend on each other.
- May feel a little slower at first.

Team B

- They focus on getting each feature working and move on to the next.
- Everyone solves problems independently, using whatever approach seems best at the time.
- They optimize for getting features delivered quickly.
- They ship things almost immediately.

After a while...

Team A

- A new feature usually fits where it belongs, touching one part of the code.
- The code follows the same patterns throughout, so it reads predictably.
- One change rarely affects anything far away.
- When something breaks, the cause is usually close to where it shows.
- A new teammate finds their way around fairly quickly.

Team B

- A new feature means edits scattered across many unrelated parts.
- Every corner was written differently, so there's no pattern to rely on.
- One change can break something far away that seemed unconnected.
- The same rule is copied in several places and has to be kept in sync by hand.
- Old features start breaking in ways that are hard to explain.
- Only the person who wrote a piece really understands how it works.

The product was thriving. The seeds of decay had already been planted.

AI: the same tool, applied to both teams

Team A: clean structure

clear patterns, low coupling

- AI sees what it needs in one place.
- AI follows the existing structure.
- Changes stay small and local.
- The team understands the result.

Team B: tangled structure

no patterns, hidden dependencies

- AI needs to see everything, but can't.
- AI misses scattered copies and links.
- One prompt changes a dozen places.
- The team accepts what it can't fully check.

The team celebrated what AI could build. Nobody noticed what the system could no longer become.

Eventually

Team A

- New features still slot in without a fight.
- The structure still guides where things go.
- Releases rarely break what already worked.
- The team still understands the system and changes it with confidence.
- AI gives Team A room to start the project Team B is still stuck in.

Team B

- What used to take a day now takes multiple weeks.
- Bugs spread faster than the team can fix them.
- The app slows down and nobody can find why.
- Fixing one thing quietly breaks another.
- The code is too tangled for anyone to hold in their head.
- The people who understood it start to leave, taking what they knew.

The world is changing faster than ever. One system is evolving with it. The other survived until it couldn't.

A short conclusion

- This was never a story about good and bad engineers, perfect and imperfect teams.
- It was a story about choices, trade-offs, and the questions that were asked, or no longer asked. One system kept creating options. The other gradually ran out of them.
- AI rewards the path we choose to build.