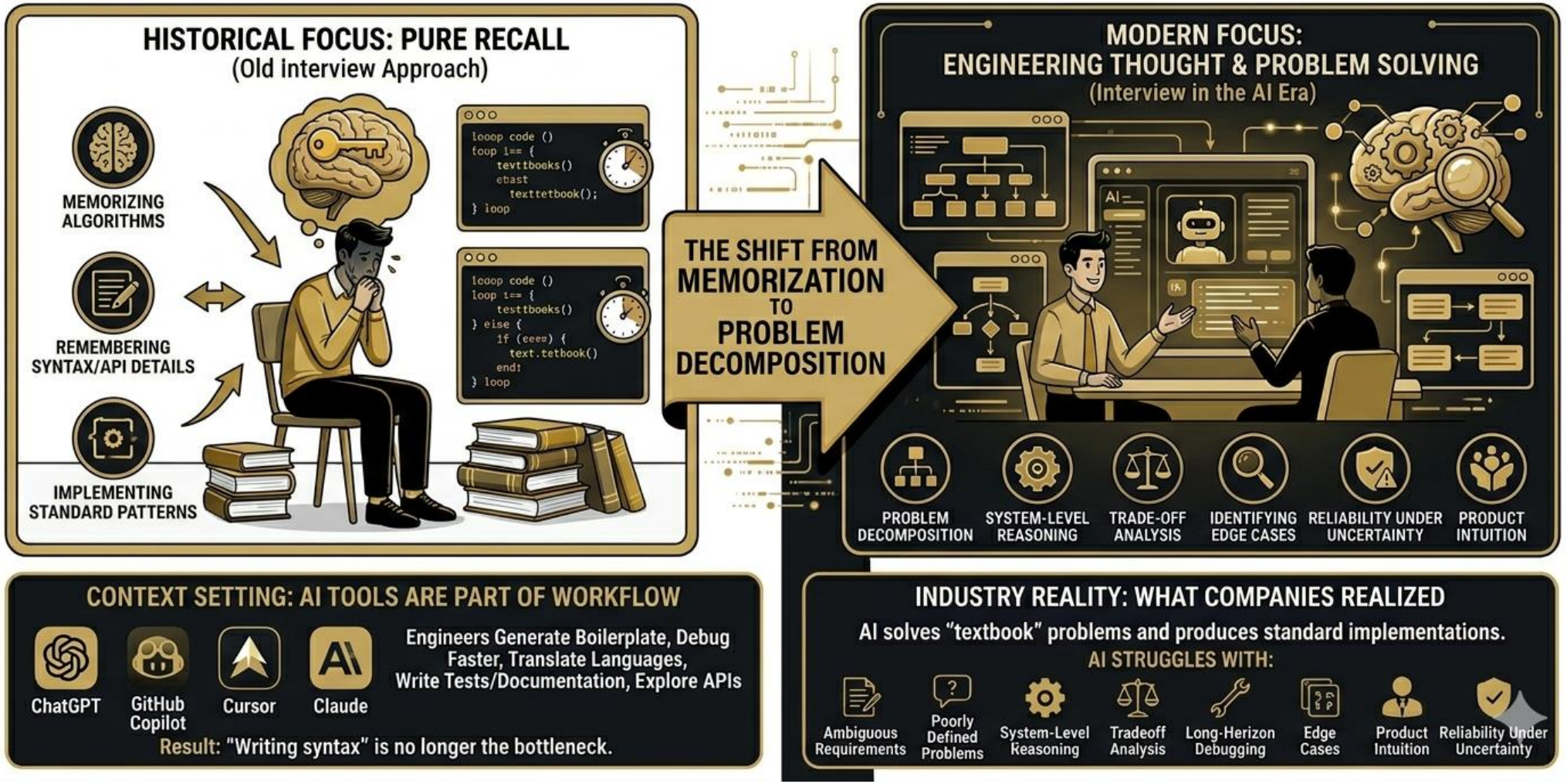
Job Interview in the AI-Era: Coding, Systems, Agents

AI Assisted Coding and Coding Interview Platforms

Gaurav Patel

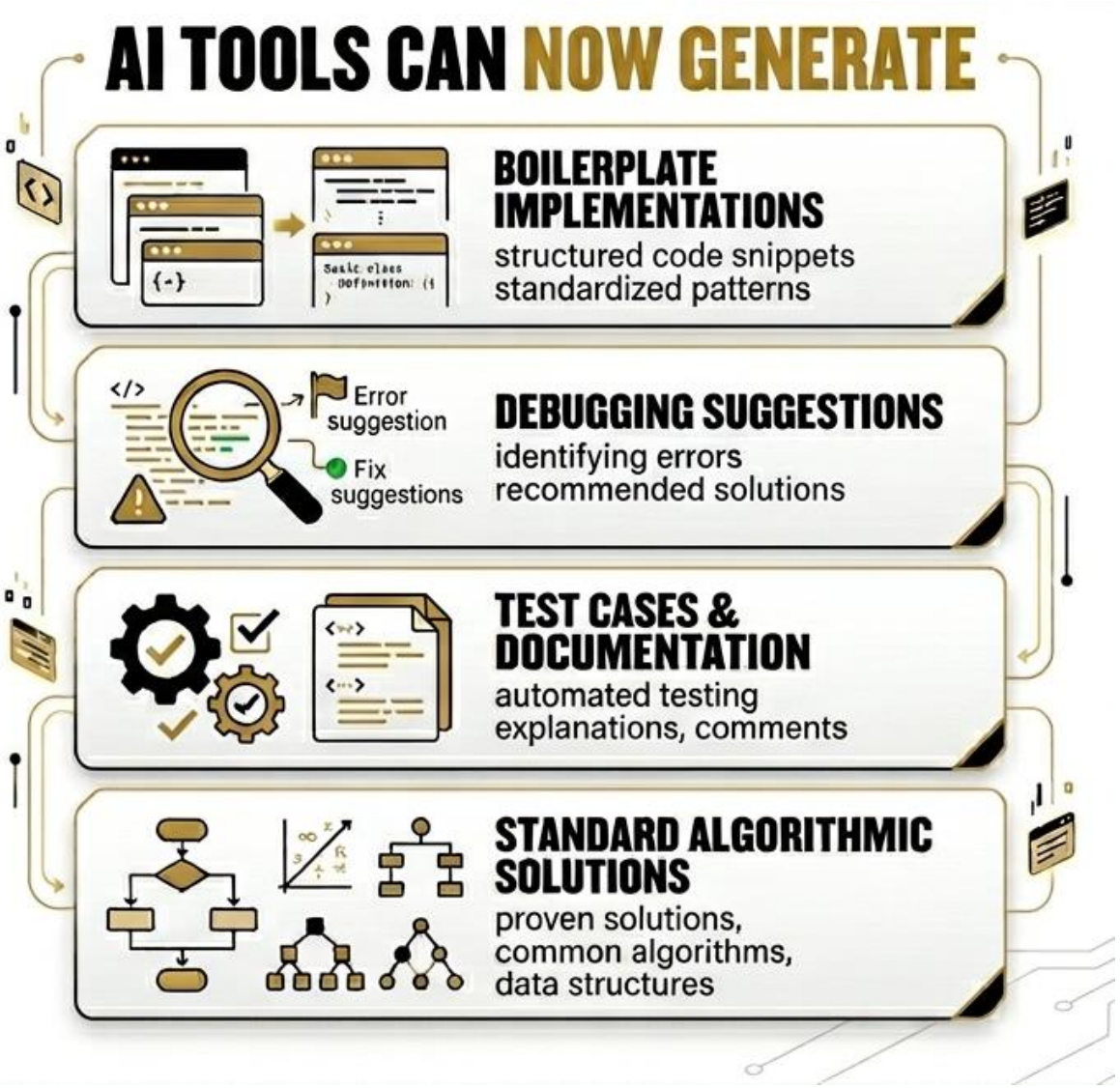
PURDUE UNIVERSITY

The New Interview Landscape in the Age of AI-Assisted Coding

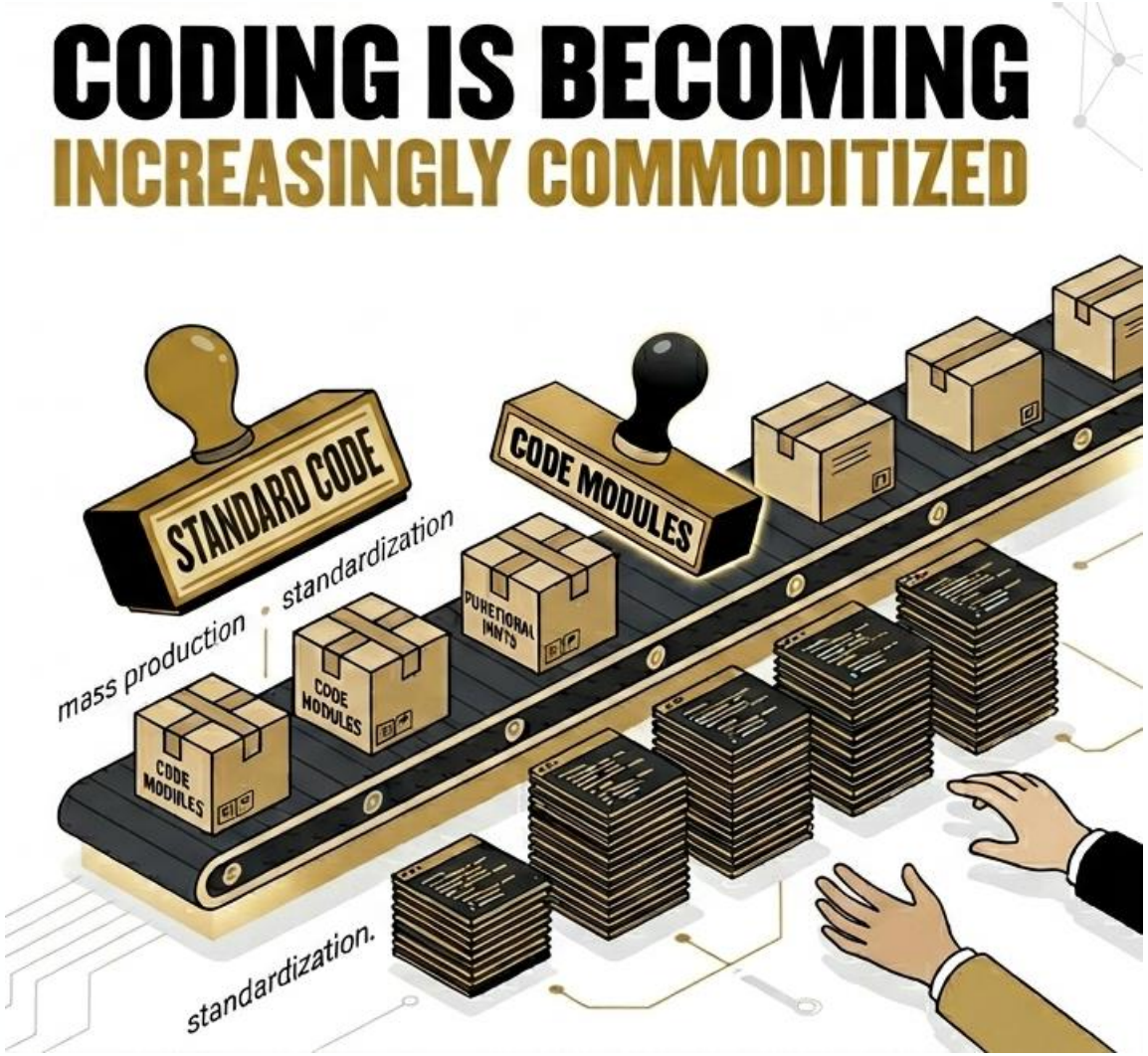


PURDUE UNIVERSITY

Capabilities of AI tools

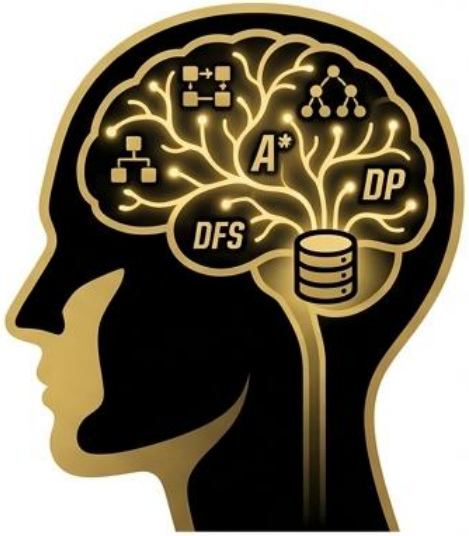


Code is now a Commodity



PURDUE UNIVERSITY

What Typical Coding Interview Test?

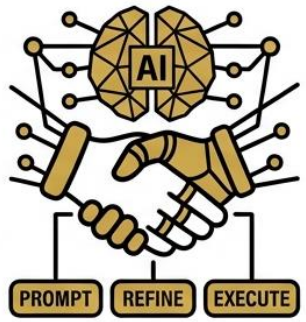


MEMORIZATION:
ALGORITHMS & PATTERNS

SYNTAX FLUENCY

SPEED-BASED CODING

What Interviews Increasingly Test Now after AI?



ABILITY TO WORK EFFECTIVELY WITH AI TOOLS

PROBLEM DECOMPOSITION

REASONING UNDER AMBIGUITY

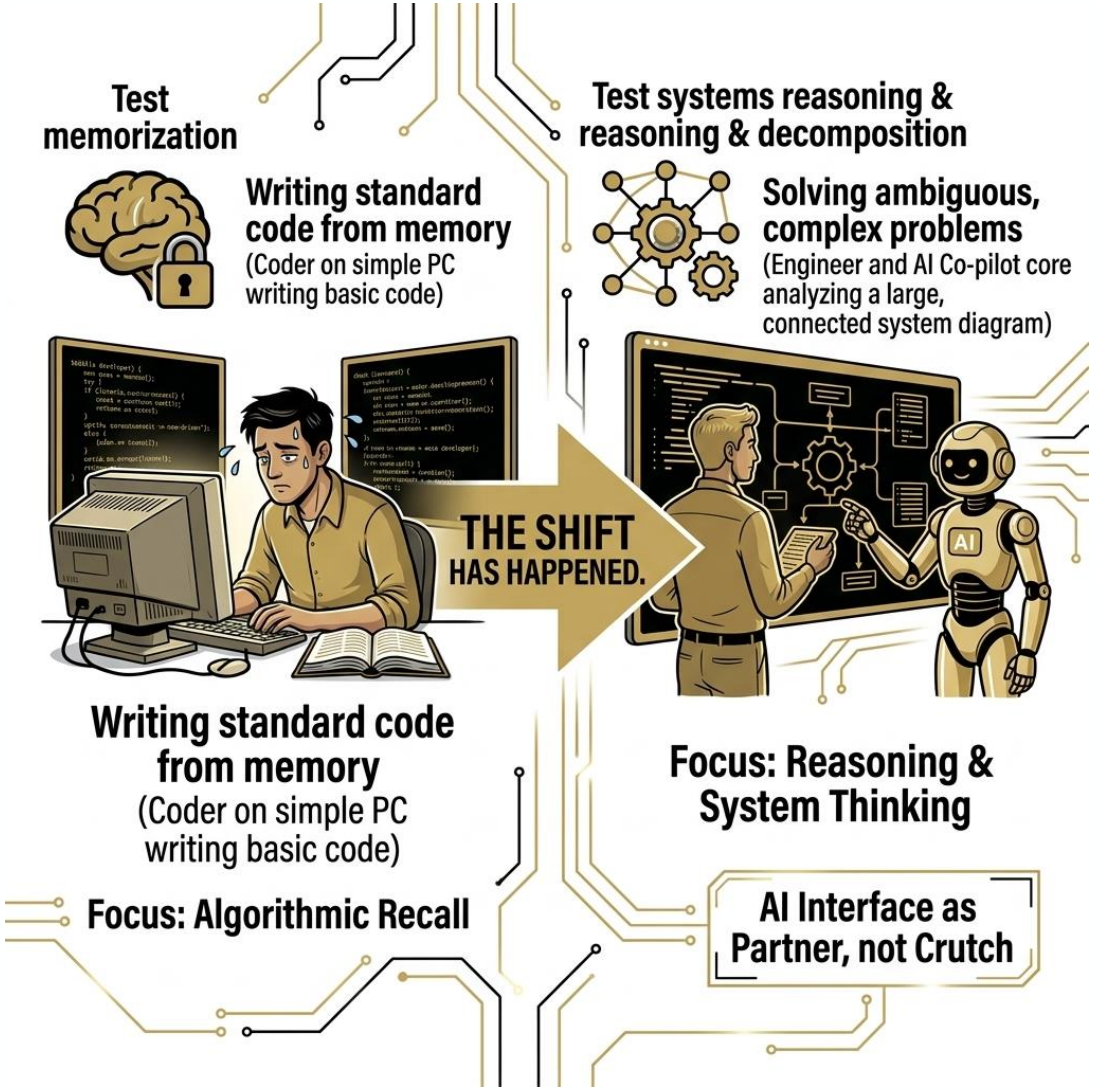
SYSTEM & TRADEOFF THINKING

DEBUGGING AND VALIDATION

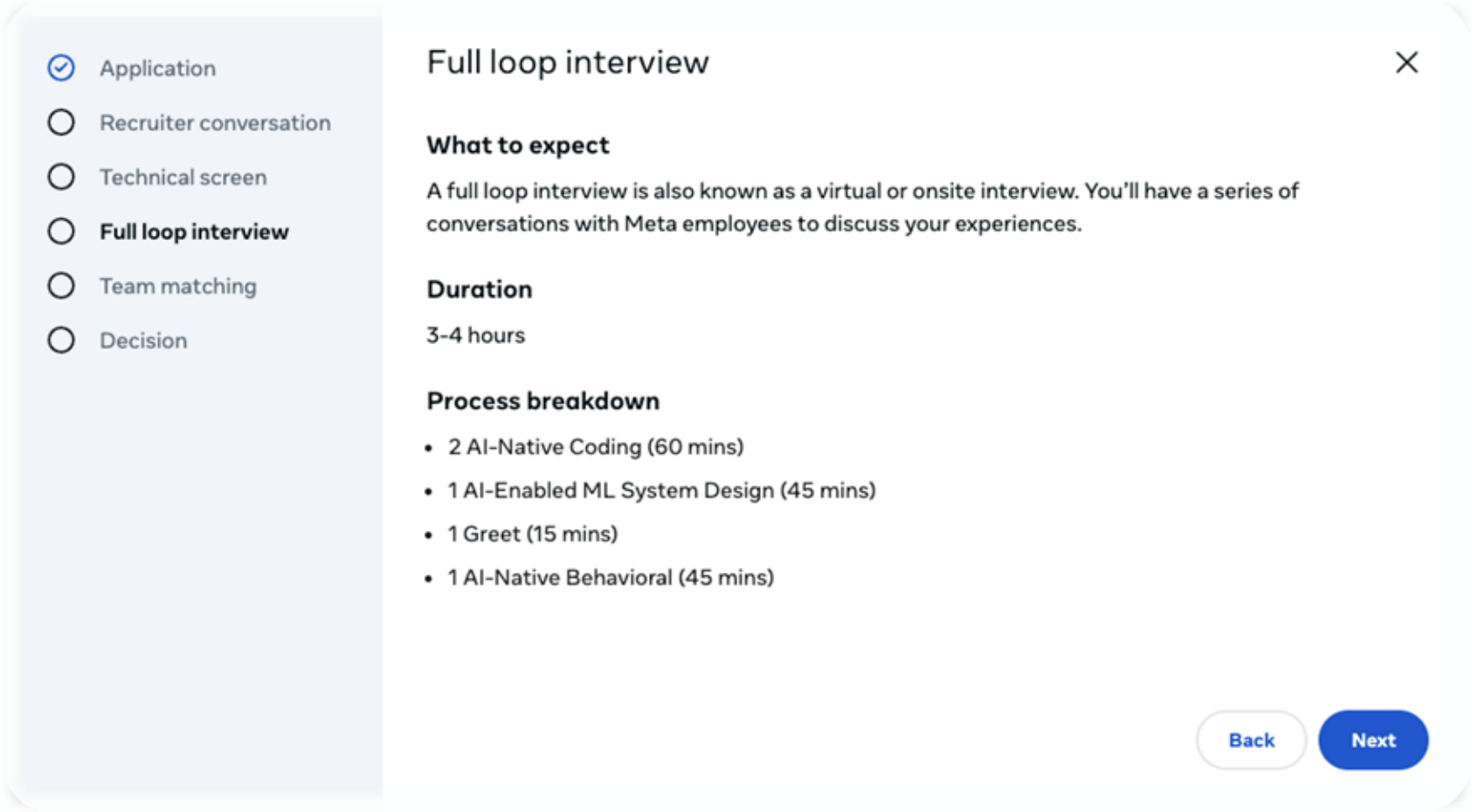
PURDUE UNIVERSITY

Key Shift

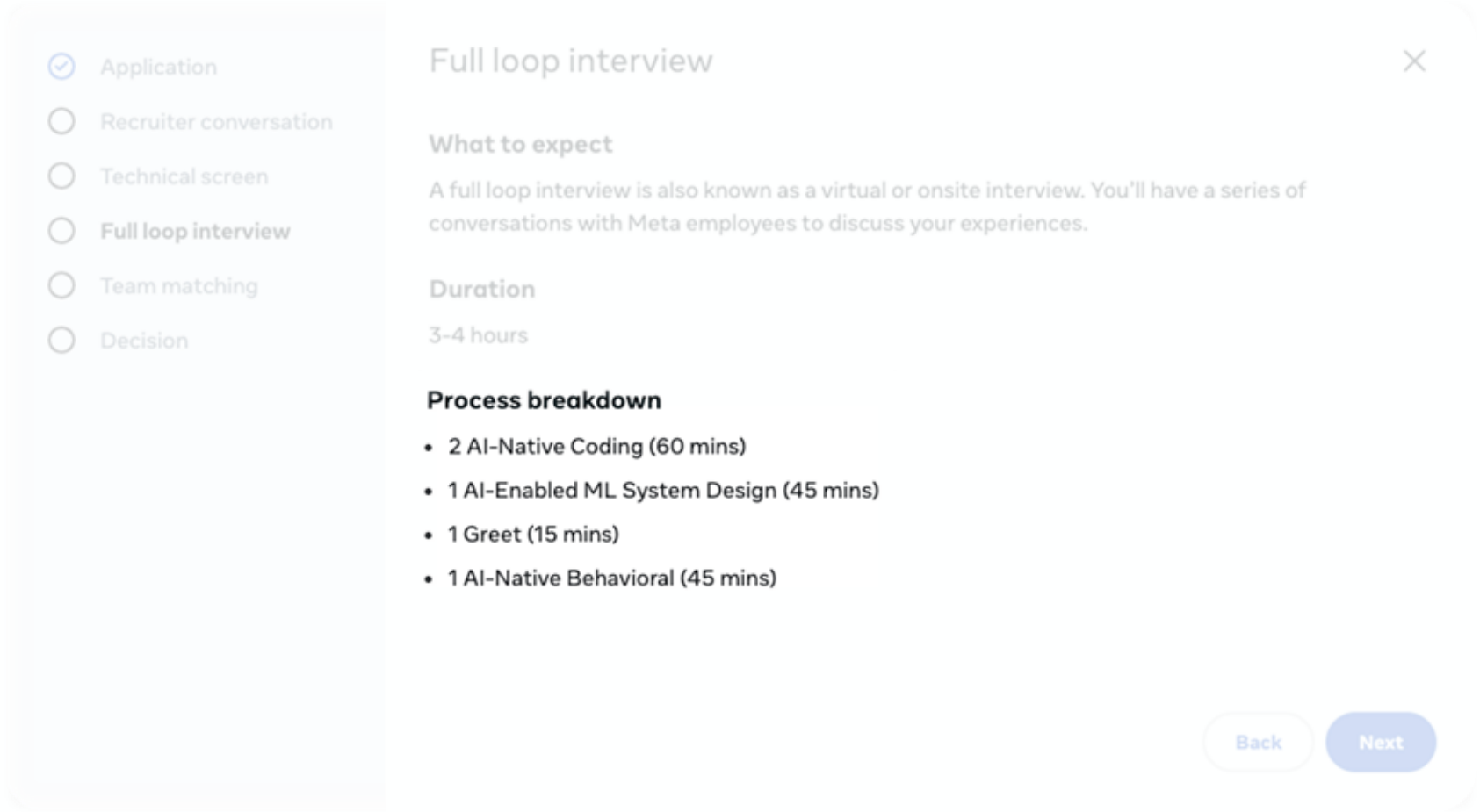
- The question is no longer:
  - “Can you write code from memory?”
- It is increasingly:
  - “Can you **think clearly**, **guide AI effectively**, and **validate solutions**?”



Example from Meta!



Example from Meta!



PURDUE UNIVERSITY

But, Things That Still Matter

STRONG FUNDAMENTALS

COMMUNICATION

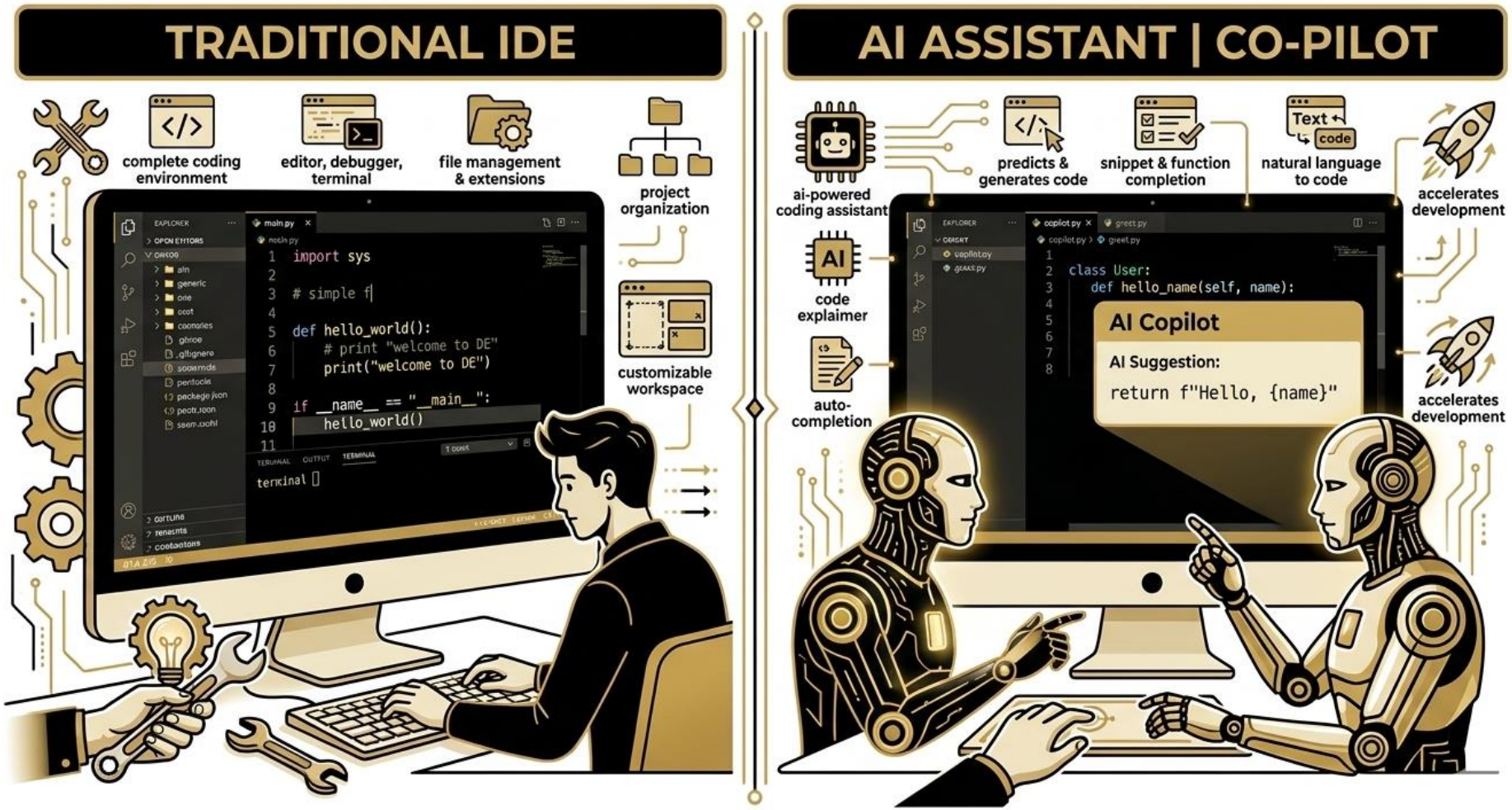
ENGINEERING JUDGEMENT

ADAPTABILITY

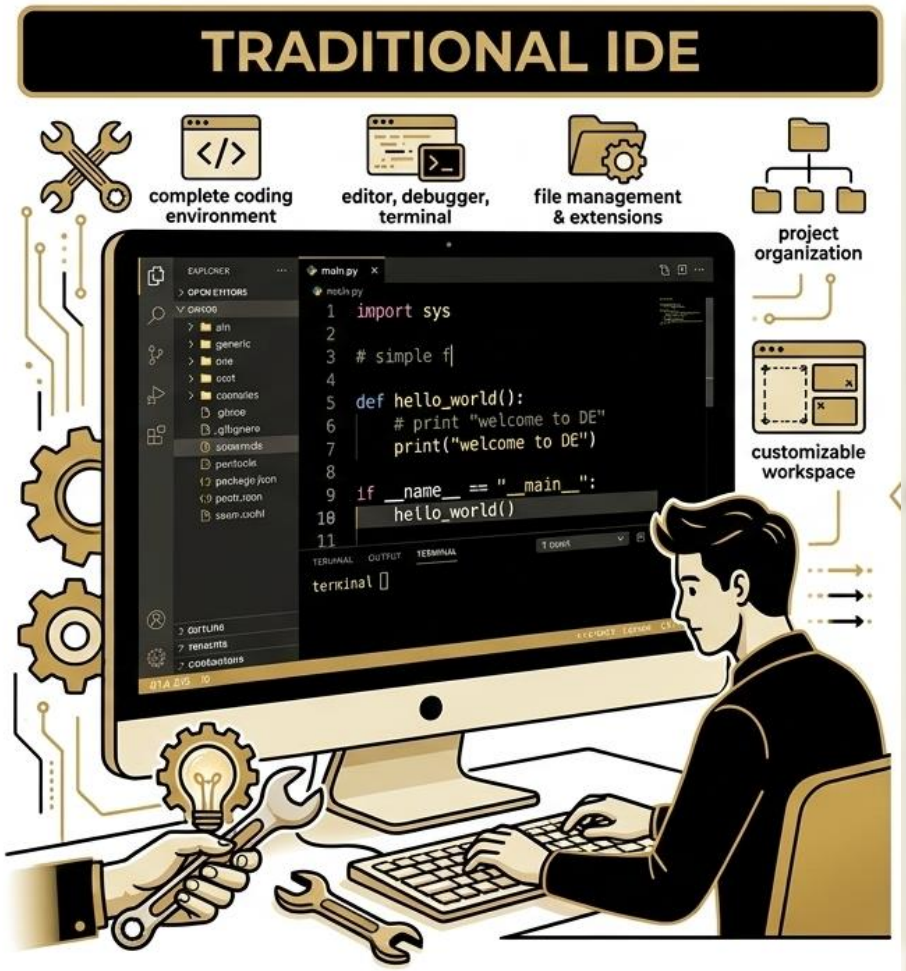
DEEP UNDERSTANDING OVER SURFACE-LEVEL PROMPTING

PURDUE UNIVERSITY

Traditional vs. AI Assisted Coding



Traditional IDE



PURDUE UNIVERSITY

Traditional IDE – the native VS Code

A free, open-source code editor by Microsoft — the most widely used IDE in the world.

Syntax Highlighting & IntelliSense

Understands your language: auto-completes variables, functions, and imports as you type. Works across 50+ languages out of the box.

Extension Marketplace

One-click installs for linters, debuggers, themes, and AI tools. Over 50,000 extensions — Copilot, Prettier, ESLint, GitLens, and more.

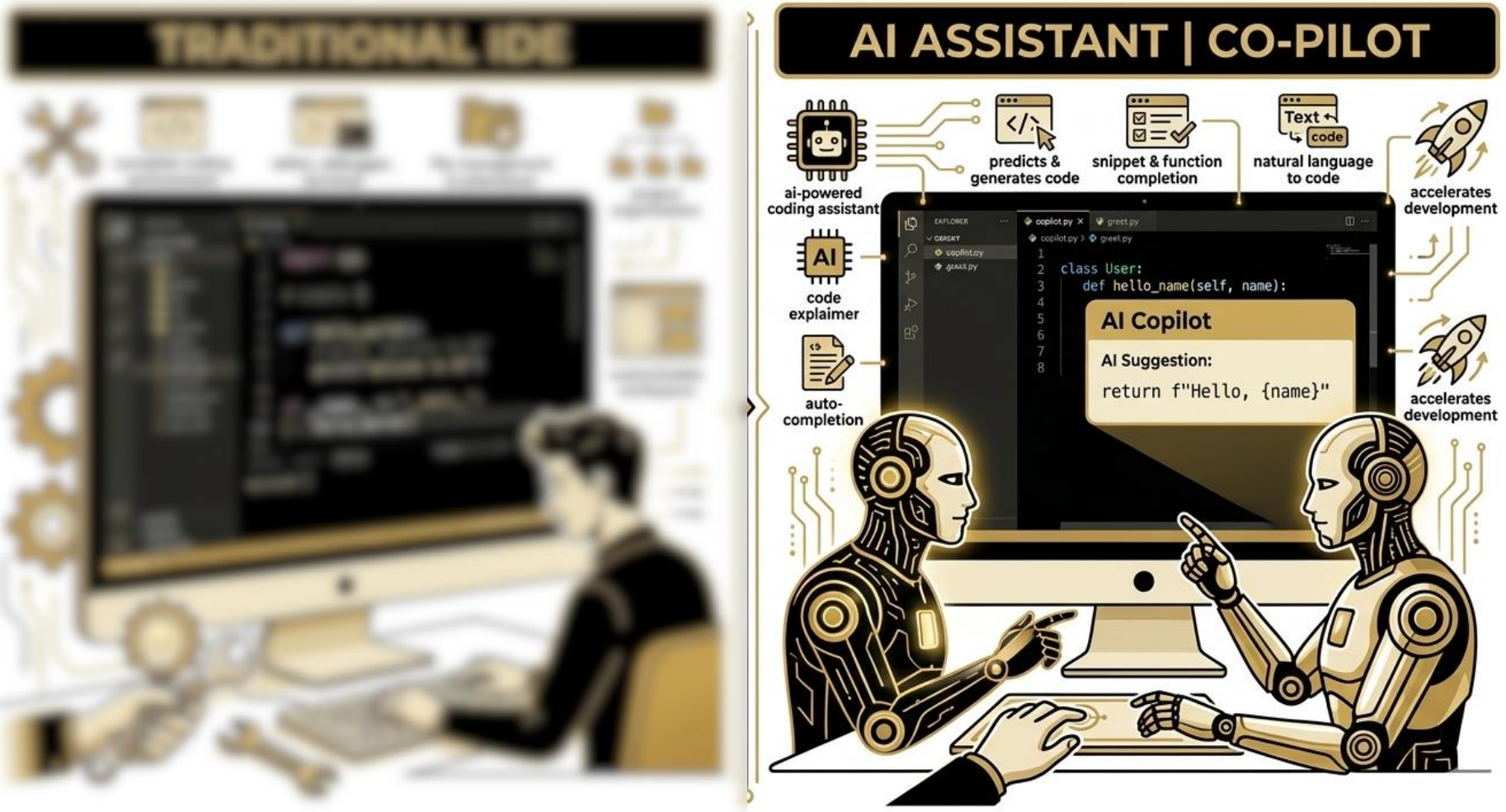
Integrated Terminal

Run your code, git commands, and build scripts without leaving the editor. Supports bash, zsh, PowerShell, and more alongside your files.

Built-in Debugger

Set breakpoints, inspect variables, and step through code for any language. Debug Python, JS, Java, and more — no external tool needed.

AI Assisted Coding



AI Assistants: Tools of the Trade

Three tools every candidate should know — and know how to use.

ChatGPT

The one everyone knows

- Great starting point for drafting, debugging & ideation
- Strong at natural language → code translation
- Conversational: easy to iterate with follow-up questions
- Most people already have muscle memory here

GitHub Copilot

Two modes, one tool

- Inline autocomplete — ghost text as you type in your IDE (VS Code)
- Chat mode — ask questions, refactor, explain code in context
- Inline is fastest; chat is better for understanding & rewrites
- Tight IDE integration makes it feel like a smart pair programmer

Claude

For long reasoning chains

- Handles very long context — entire codebases, long specs
- Stronger at multi-step reasoning and nuanced explanation
- Use when you need to think through system design or trade-offs
- Less guessing, more structured analysis than alternatives

GitHub Copilot

An AI coding assistant trained on billions of lines of code — lives inside your editor, not a browser tab.

Inline Autocomplete

Ghost text · Appears as you type

- Predicts the next line — or the next 10 lines — as gray ghost text
- Press Tab to accept, Esc to dismiss, or just keep typing to ignore
- Shines on boilerplate: loops, error handlers, test stubs, regex
- Context-aware: reads your file and open tabs to match your style
- Zero friction — no shortcut needed, it just appears

Chat Mode

Conversational · Ask anything about your code

- Open a panel: ask "explain this", "refactor this", "why is this slow"
- Highlight any code block, right-click → "Ask Copilot" for instant context
- Better for understanding trade-offs and design choices than autocomplete
- Generates full functions or files from a natural language description
- Slower but deeper — use when you need to think, not just type

PURDUE UNIVERSITY

What is the Interviewer Looking to Test?

- **Problem decomposition over memorization**
  - Can you break an ambiguous problem into manageable steps before prompting or coding?
- **Prompting quality and technical communication**
  - How clearly can you instruct AI tools with constraints, edge cases, and desired outputs?
- **Verification and critical thinking**
  - Do you validate correctness, complexity, corner cases, and hidden failure modes instead of blindly trusting AI?

What is the Interviewer Looking to Test?

- **Iteration and debugging workflow**
  - How effectively do you refine prompts, inspect outputs, and recover from incorrect generations?
- **Collaboration with AI as a productivity tool**
  - Are using AI to accelerate thinking and implementation, not replace reasoning?

Vague vs. Structured Prompt

Same task. Very different results. A Data Science example — building a classification model.

Vague Prompt

Prompt sent to AI:

"Help me build a machine learning model for classification."

What the AI doesn't know:

- Which language or framework to use
- What dataset or problem you have
- What libraries are available to you
- How to format the output
- Your skill level or team constraints

⚠ What you likely get:

A generic sklearn Decision Tree example that uses toy data, no comments, no evaluation, and may not even run in your environment.

Vague vs. Structured Prompt

Same task. Very different results. A Data Science example — building a classification model.

Structured Prompt

Prompt sent to AI:

"Act as a senior data scientist. [Role]
I have a Pandas DataFrame with 10k rows, 15 features, and a binary churn target — class imbalance ~80/20. Using Python 3.11 and sklearn. [Context]
Use only sklearn, pandas, and numpy. Add inline comments and avoid data leakage. [Constraint]
Return a clean Python code block, then 3 bullet points explaining key decisions. [Output Format]"

What the AI now knows:

- Expertise level → professional-grade code
- Dataset shape, imbalance, and goal
- Exact libraries, Python version
- Format: code + reasoning bullets

What you actually get:

A production-ready pipeline with train/test split, SMOTE or class_weight handling, cross-validation, and a clear explanation of every decision.

PURDUE UNIVERSITY

The 4-Part Prompt Framework

Most AI responses are only as good as the prompt. Structure yours and you'll get code that's usable.

1

Role

Tell the AI who to be

Set the expertise level and perspective. Without a role, the AI defaults to a generalist — too broad for technical work.

"Act as a senior data scientist with expertise in scikit-learn and model evaluation."

2

Context

Describe the exact situation

State your dataset, tools, language, and goal. Context prevents generic answers that don't fit your stack.

"I have a Pandas DataFrame with 10k rows, 15 features, and a binary classification target. Using Python 3.11 and sklearn."

3

Constraint

Set the guardrails

Specify what to avoid: no external libraries, keep it readable, no global variables, match a style guide.

"Use only sklearn, numpy, and pandas. No external libraries. Add inline comments explaining each step."

4

Output Format

Say exactly what you want back

Specify format: code block only, markdown, a table, step-by-step with explanations, or a before/after comparison.

"Return a single Python code block. Below it, add a 3-bullet explanation of the key decisions made."

PURDUE UNIVERSITY

Prompting Strategies: Quick Reference

Apply these patterns to any coding or data science task — and watch the output quality jump.

Start with the role

✓ *"Act as a data scientist"*

✓ *"You are a Python expert"*

✗ *Leaving role blank → generic output*

Be specific about your stack

✓ *"Python 3.11, pandas, sklearn"*

✓ *"Jupyter notebook environment"*

✗ *"Use Python" → may suggest wrong version or libs*

Name your constraints up front

✓ *"No loops — use vectorised ops"*

✓ *"Max 30 lines, well-commented"*

✗ *Fixing bad code after the fact wastes time*

Specify the output format

✓ *"Code block only, no prose"*

✓ *"Table of model metrics"*

✗ *Vague format → mixed code + prose = messy*

Iterate, don't restart

✓ *"That works — now add cross-validation"*

✓ *"Rewrite using a Pipeline object"*

✗ *Rewriting the full prompt every time*

Ask for reasoning too

✓ *"Explain why you chose Random Forest"*

✓ *"List trade-offs vs. Logistic Regression"*

✗ *Code without understanding = a liability*

The goal isn't a perfect first prompt — it's knowing how to steer. Role → Context → Constraint → Output Format is your starting scaffold; iteration is the real skill.

The 3 Big Pitfalls of AI-Assisted Coding

Know these failure modes before they cost you in an interview or on the job.

01

Hallucinated APIs

The function looks real. It isn't.

AI invents plausible-looking class names, method signatures, and parameters that do not exist in the library. It writes polished, formatted code using them — and won't warn you.

Example

Copilot generates a random uses a random class names SmartEncoder() — does not exist in any version of sklearn.

02

Confidently Wrong Logic

It runs. It's still wrong.

Code executes without errors but produces incorrect results — data leakage across train/test splits, wrong metric for the problem type, or silent off-by-one bugs.

Example

StandardScaler fitted on the full dataset before splitting — inflates validation accuracy, fails in production.

03

Over-Reliance

You shipped it. Can you explain it?

Copy-pasting AI output without understanding creates a liability. Works until an interviewer asks 'why did you use this?' — or a bug appears and you have no idea where to look.

Example

"Copilot suggested GridSearchCV." An interviewer hears: 'I don't understand my own submission.'

PURDUE UNIVERSITY

Pitfall #1 — Hallucinated APIs

AI invents plausible-sounding methods with total confidence. Your job is to verify before you ship.

✗ AI-Generated — Will Break at Runtime

Copilot wrote this without any warning. It throws an ImportError the moment you run it.

```
# Copilot suggestion — looks reasonable
from sklearn.preprocessing import SmartEncoder

enc =
SmartEncoder(
    handle_unknown='auto', sparse_output=False
)

X_enc = enc.fit_transform_safe(X_train)

# ImportError: cannot import 'SmartEncoder'
```

Why it happens

AI is trained on code patterns, not live APIs. It extrapolates what a function 'should' be named — plausibly, confidently, and often incorrectly.

✓ Verified Against the Actual Docs

The real class is OrdinalEncoder — different name, different arguments. Always check the sklearn docs.

```
# Correct sklearn API (verified in docs)
from sklearn.preprocessing import OrdinalEncoder

enc = OrdinalEncoder(
    handle_unknown='use_encoded_value',
    unknown_value=-1
)

X_enc = enc.fit_transform(X_train)

# Works. Verified. Arguments are correct.
```

How to catch it

Search the exact class name in the official docs. If it's not there, it doesn't exist. Run on 5 rows before trusting any import.

PURDUE UNIVERSITY

Pitfall #2 — Confidently Wrong Logic

No error is thrown. The model trains. Results look reasonable. The logic is still wrong.

✗ Runs Fine — Results Are Wrong

Data leakage: scaler sees test data during training, inflating scores. The model will underperform in production.

```
# ⚠  Leakage — scaler sees ALL data
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)   # entire dataset!

X_train, X_test = train_test_split(
    X_scaled, y, test_size=0.2
)

# Test stats leaked into training → invalid eval
```

Why it's dangerous

Accuracy looks great in dev, fails in production. Silent bugs are worse than noisy ones because they're easy to miss.

✓ Correct Order — No Leakage

Split first. Fit the scaler only on X_train. Transform X_test using those same parameters — never refit.

```
# ✓  Split first — scaler never sees test data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)   # train only
X_test  = scaler.transform(X_test)    # apply same
params

# Test evaluation is now truly held-out
```

The rule

Fit only on training data. This applies to every transformer: scalers, imputers, and encoders alike.

PURDUE UNIVERSITY

Pitfall #3 — Over-Reliance

The code works. But if you can't explain it in a debrief, you don't own it — and that's a problem.

Owning the Output in a Debrief

"Walk me through your modeling pipeline."

✗ *"I used Copilot for most of it."*

✓ "I used Copilot for the boilerplate, then changed the metric to F1 because of class imbalance. I can walk through each stage and explain every decision."

"Why GridSearchCV over RandomizedSearchCV?"

✗ *"Copilot suggested it, so I went with it."*

✓ "The search space was small — 3×3 = 9 combinations — so exhaustive was fast. For larger spaces I'd use RandomizedSearchCV to control runtime."

"Did AI tools help you here?"

✗ *"Yes, Copilot wrote most of it."*

✓ "Yes — for structure and boilerplate. I corrected the evaluation logic and rewrote a leakage issue it introduced. I can explain every line."

How to Review

Keep asking these question while generating code and verifying them.

The 4-Step Review: Before You Submit

Does it actually run?

Execute on a 5-row sample first. AI code often has import errors, wrong argument names, or missing variables that only appear at runtime.

Is the logic sound?

Trace every step manually. Check the order of operations, metric choice for your problem type, and watch for data leakage patterns.

Can you explain every line?

Cover the code and narrate it aloud to yourself. If you stumble on a line, that's your study target before any debrief or submission.

Do the results make sense?

98% accuracy on imbalanced data is a red flag. Check the confusion matrix, F1, and precision/recall for all classes before calling it done.

Best Practices: AI-Assisted Coding

The engineers who use AI best aren't those who use it most — they review it hardest.

Red Flags — Stop and Verify

- ⚑ A library or method you don't recognise — check the docs before using it
- ⚑ Accuracy > 95% on first attempt — likely data leakage or a trivial split
- ⚑ Hardcoded magic numbers with no comments — ask the AI to explain them
- ⚑ No error handling in production output — AI skips this unless explicitly asked
- ⚑ Model fitted before train/test split — classic leakage pattern, check the order

Run Before You Trust

- Execute on a 10-row sample — catch import errors fast
- Print shapes, dtypes, value_counts at each pipeline step
- One assert per stage: no NaNs, correct shape, expected classes

Run on 5 rows. Print X_train.shape. Write one assert. Catch errors before they hide deeper in the pipeline.

Ask for Explanations Too

- After every code block: "Explain each section in plain English"
- Push further: "What are the trade-offs vs. the alternative?"
- Use Chat to stress-test your understanding before a debrief

"Explain why SMOTE here instead of class_weight. What are the risks of oversampling before splitting, and when would you avoid it?"

Own Every Line You Ship/Finalize

- Read every line aloud, stumbling reveals gaps to close
- Delete code you can't explain; simple owned code beats clever borrowed code
- In interviews: be ready to say what you'd do without AI

"I chose [X] because [reason]. I considered [Y] but ruled it out because [trade-off]. I can walk you through every decision here."

PURDUE UNIVERSITY

Interview Platforms

Different platforms serve different stages of the hiring funnel — each with its own AI policy. Know which is which.

1. CoderPad
2. HackerRank
3. CodeSignal
4. Codility
5. Karat
6. Interviewing.io

PURDUE UNIVERSITY

CoderPad: What to Expect (Click Here)

The most common live-coding environment in industry. Know the interface before interview day.

The CoderPad Interface

Code Editor (left pane)

Shared in real time — both you and your interviewer type here. Every keystroke is visible to them live.

Run Button

Executes in a sandboxed environment. Output in the console below. Run as often as you like — not graded on number of runs.

Notes Panel (right pane)

Scratch work, pseudocode, or your approach before coding. Interviewers see this too — use it to think out loud.

Whiteboard Mode

Some companies enable a drawing canvas for diagrams. Ask your interviewer before assuming it's available.

AI Tools

Copilot and ChatGPT blocked by default. Most companies use the default. Treat every CoderPad session as AI-free. However, AI tools are enabled for AI assisted coding rounds.

Interview Strategy

Before you type anything

- Read the prompt fully — twice. Paraphrase it back to the interviewer
- Write your approach in the notes pane as pseudocode first
- State edge cases out loud: empty input, single element, large n

While you're coding

- Narrate every decision: 'I'm using a dict here for O(1) lookup'
- Run early and often — don't wait for a perfect solution
- If stuck, say so and describe what you've tried. Silence loses points, thinking out loud gains them

After you get output

- Trace through a test case manually — don't just trust the run
- State the time and space complexity unprompted
- Suggest what you'd improve: 'With more time I'd handle...'

PURDUE UNIVERSITY

HackerRank & CodeSignal – Screening Rounds

These platforms power the first automated filter before any human sees your application. Know what you're walking into.

HackerRank

Timed, auto-graded, broadly used

What the test looks like

- 2–4 problems, 60–90 min, auto-graded with partial credit
- Spans easy strings/arrays up to medium DS&A topics
- Some DS roles add SQL, regex, or a notebook task

Proctoring & AI policy

- Tab-switching detected — don't open AI tools
- Webcam + copy-paste detection active at many companies
- Copilot/ChatGPT prohibited — treat as a closed-book exam

How to prepare

- Use HackerRank's free Interview Preparation Kit
- Time yourself strictly — clock pressure is real
- Read all problems first; start with what you know

CodeSignal

Portable GCA score · Used by Uber, Figma, OpenAI

The GCA (General Coding Assessment)

- 4 problems, increasing difficulty, 70-minute limit
- Score 300–600 — portable across companies, valid 6 months

What makes it different

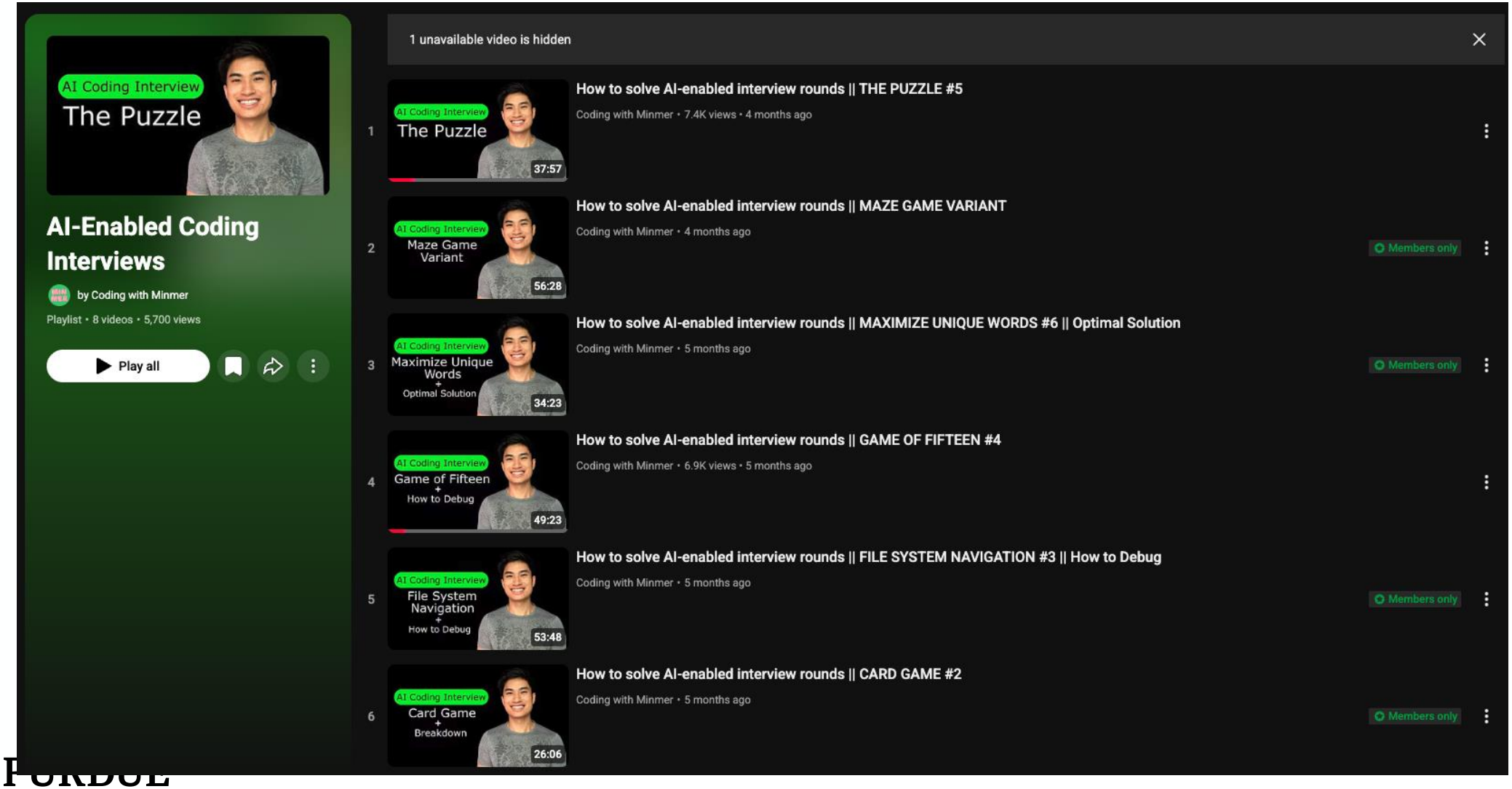
- Keystroke analysis — measures how you solve, not just if you pass
- Large copy-pastes flagged; Copilot triggers anomaly detection
- Strictly proctored with web-cam and microphone.

Strategy for CodeSignal

- Solve in order — early problems weigh more
- Clean, readable code is rewarded by style analysis
- A brute-force that passes beats clever code that doesn't

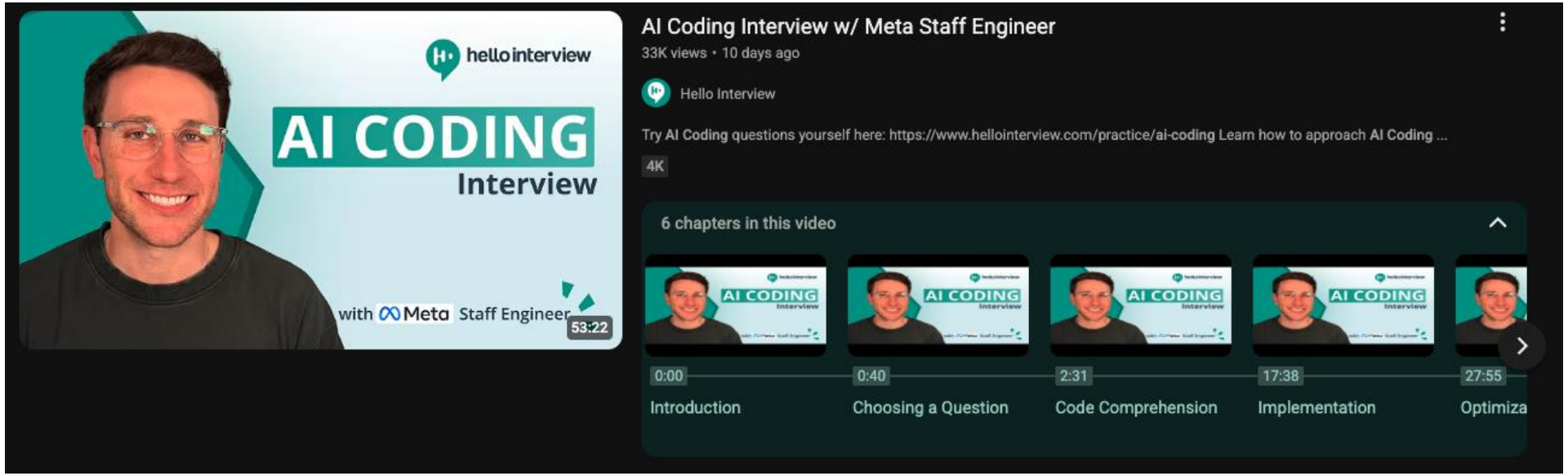
PURDUE UNIVERSITY

AI Assisted Interview Resource (Click Here)



PURDUE UNIVERSITY

AI Assisted Interview Resource (Click Here)



PURDUE UNIVERSITY

Thank You

PURDUE UNIVERSITY