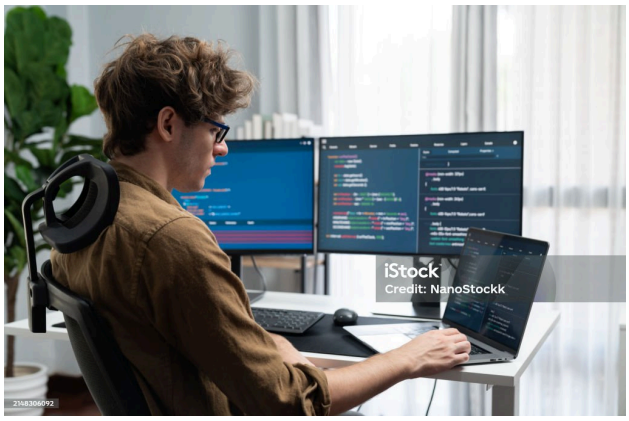
“Decoding the Code: What Live Coding Videos Reveal About Expertise”

Amisha Singh

Chapter 1: The Quest for Mastery



Back in my college days, while grinding through Data Structures and Algorithms (DSA) problems late into the night, I often found myself stuck watching coding tutorials on YouTube. Some programmers would glide through complex problems effortlessly, while others struggled with even basic syntax errors. As I practiced for coding contests, I couldn’t help but wonder: What made these expert coders so different?

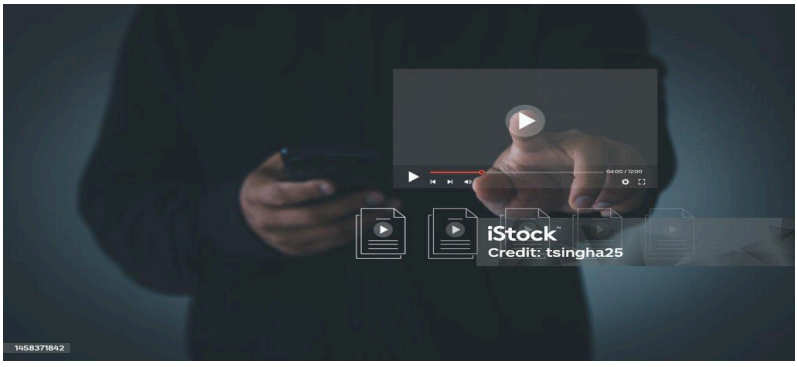
Was it their speed? Their problem-solving intuition? Or something more subtle - the way they structured their code, debugged errors, or navigated a problem under time pressure?

This curiosity turned into an idea: What if we could analyze coding videos to uncover patterns that define expertise? Could we predict a coder’s skill level just by watching how they code-live?

That’s exactly what I set out to explore. By collecting and analyzing coding competition videos, I aimed to identify workflow patterns, decision-making strategies, and problem-solving habits that set expert coders apart. But before diving into the insights, let’s talk about the journey - because every dataset has a story.

Chapter 2: Building the Dataset

Finding the right dataset is always the hardest part. I needed real-world coding sessions - not edited tutorials or highlight reels. The goal was to capture raw, unfiltered problem-solving. So, I focused on YouTube’s vast library of live coding sessions, coding competitions, and code review videos. These types of videos were ideal because they showed the real-time process of coding: solving problems, debugging, and sometimes explaining the logic behind decisions.



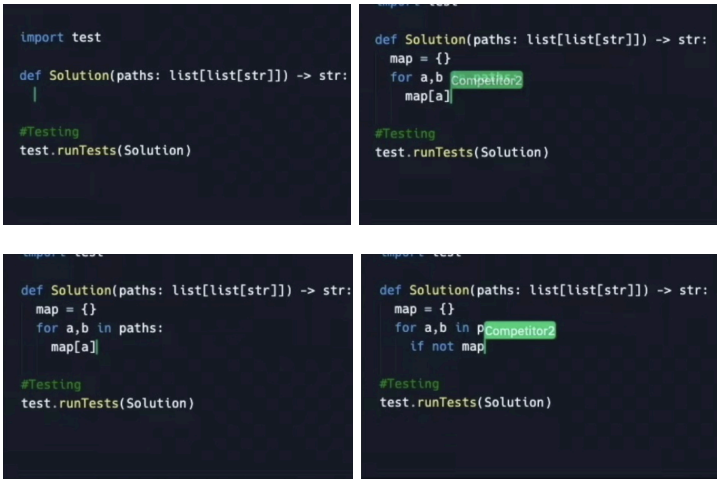
These types of videos offered a window into the coding process - solving problems, debugging code, and explaining thoughts aloud. After selecting a few

popular YouTubers known for their live coding content, I started automating the process of collecting videos. Each video would be carefully analyzed, broken down into frames, and coded for key features—metadata, text transcriptions, comments, and more. With my dataset ready, the journey into deeper analysis began.

Chapter 3: Extracting Code – The First Glimpse into Workflow

The next challenge was to extract the actual code from these videos. This process involved several steps:

1. **Frame Extraction**: using OpenCV, I processed each video to extract individual frames. This allowed for a frame-by-frame analysis of the coding sessions.



2. **Text Detection and Extraction**: to convert the on-screen code into editable text, I employed Tesseract OCR. This tool is adept at recognizing and transcribing text from images. However, the process wasn't without challenges:

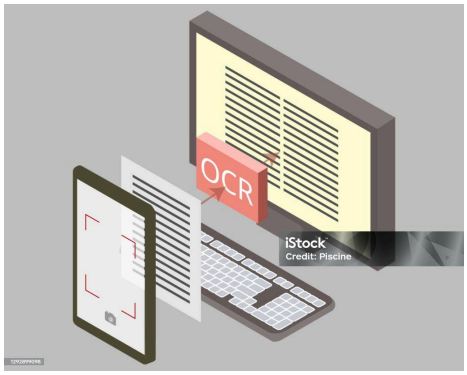


Image Quality: variations in video resolution and lighting affected OCR accuracy.

Code Formatting: indentation and special characters in code posed additional hurdles for accurate recognition.

3. **Noise Reduction**: at every frame contained meaningful code. To avoid redundancy:
   - **Duplicate Frame Removal**: implemented a mechanism to identify and discard frames that were identical or nearly identical to previous ones.

- **Non-Code Frame Filtering**: frames without code (e.g., those showing diagrams or explanations) were excluded from the analysis. This meticulous extraction process ensured that the subsequent analysis was based on accurate and relevant code snippets.

Chapter 4: Identifying Patterns in Expert Coding

Once I had a clean dataset of extracted code, the next step was to analyze patterns that could distinguish an expert coder from a beginner. To automate this, I used pandas and OpenCV for different aspects of analysis.

1. Code Structure Analysis

Evaluate how organized and well-documented the code is.

- Used **Pylint** and **Flake8** to analyze function usage, modularity, and documentation.
- Stored results in a pandas DataFrame for structured comparison.

```python
import pandas as pd
from pylint.lint import Run

def analyze_code_structure(code):
    results = Run(["--errors-only"], do_exit=False)
    return results.linter.stats.global_note  # Score out of 10

df["code_structure_score"] = df["code_snippet"].apply(analyze_code_structure)
```

2. Algorithmic Efficiency

Identify optimal algorithms and data structures.

- Used Python's `ast` module to detect **nested loops** and **recursion**.
- Higher nesting or excessive recursion suggested inefficiency.

```python
import ast

def detect_nesting_depth(code):
    tree = ast.parse(code)
    return max(len(node.orelse) if hasattr(node, 'orelse') else 0 for node in ast.walk(tree))

df["nesting_depth"] = df["code_snippet"].apply(detect_nesting_depth)
```

3. Debugging Techniques

Determine debugging strategies used.

- Used OCR-extracted logs to check for `print()` statements and debugging tools like `pdb`.
- Recorded the frequency of debugging statements.

```python
df["debugging_statements"] = df["code_snippet"].apply(lambda x:
 x.count("print(") + x.count("pdb"))
```

4. Typing Patterns

Analyze typing speed and correction frequency.

- Used OpenCV to track keyboard activity.
- Extracted timestamps where code was typed and calculated typing speed.

```python
def calculate_typing_speed(timestamps):
    intervals = [t2 - t1 for t1, t2 in zip(timestamps, timestamps[1:])]
    return sum(intervals) / len(intervals) if intervals else 0
```

5. Error Handling

Assess how coders anticipate and handle errors.

- Used `ast` to detect presence of `try-except` blocks.
- Assigned a score based on the complexity of error handling.

```python
def count_error_handling_blocks(code):
    tree = ast.parse(code)
    return sum(isinstance(node, ast.Try) for node in ast.walk(tree))
df["error_handling_blocks"] =
df["code_snippet"].apply(count_error_handling_blocks)
```

Chapter 5: Uncovering Hidden Coding Patterns with Visual Insights

Extracting code is just the beginning. Now, let's decode the story behind the keystrokes. By visualizing coding speed, errors, and complexity, we can uncover how coders think, debug, and refine their code in real-time.

🔥 **Keystroke Density Heatmaps** –Track bursts of activity and pauses to reveal coding rhythm.
🔍 **Debugging vs. Writing Time** – See how much time is spent fixing vs. creating code.
📊 **Code Complexity Over Time** – Analyze indentation, loops, and structure to distinguish clean vs. chaotic coding.
📝 **Annotated Code Insights** –Overlay extracted OCR text with speed, errors, and time spent per block.

These insights can separate pros from beginners—let's put them to the test with a live coding competition!

Case study: HackSussex Coders' Cup 2024

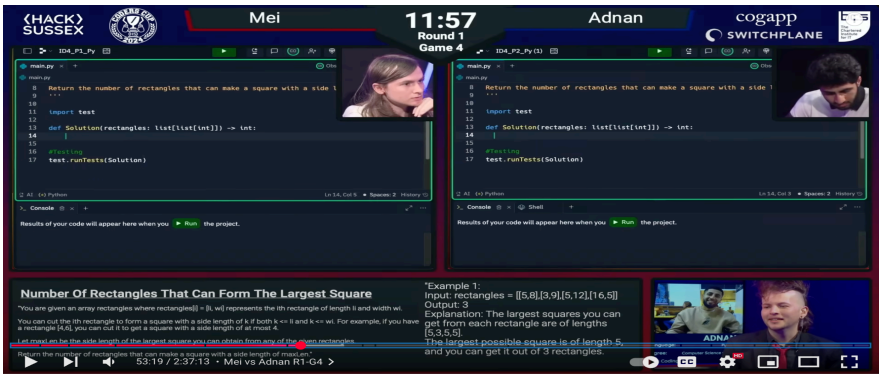
In the heat of a live coding competition, two developers sit side by side, their screens glowing with lines of code. One is a seasoned expert, fingers flying across the keyboard with practiced precision. The other, a determined beginner, carefully structuring each line, occasionally pausing to debug.

But what truly sets them apart? Is it their speed, their efficiency, or the way they handle mistakes?

In this case study, I'll break down their coding sessions frame by frame, tracking typing speed, errors, keystroke density, debugging vs. writing time, and code complexity over time. Using data-driven visualizations, I'll uncover the hidden patterns that define expert-level coding and expose the subtle struggles of a beginner.

By the end, we might just answer the question: Can we predict coding expertise just by watching someone code?

!!THE BATTLE BEGINS!!



⚔️ The Ultimate Coding Showdown: MEI vs. ADNAN

The battleground is set. Two coders. One competition. A clash of experience vs. ambition.

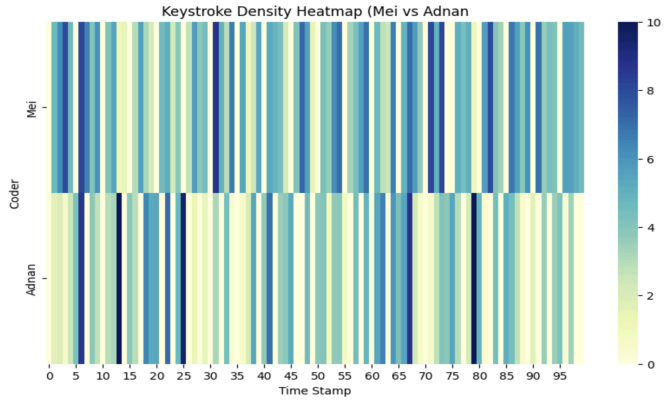
🟦 **On the left, we have MEI** – a battle-hardened coder with 11 years of experience, wielding Python like a finely tuned weapon. With a degree in Computer Science and a deep understanding of algorithms, MEI's approach is precise, efficient, and methodical.

🟥 **On the right, we have ADNAN** – a rising challenger with 3 years of experience, also armed with a Computer Science degree and Python proficiency. Eager to prove his skills, ADNAN's coding style is bold, experimental, and sometimes unpredictable.

Who writes faster? Who debugs more? Who structures their code better?

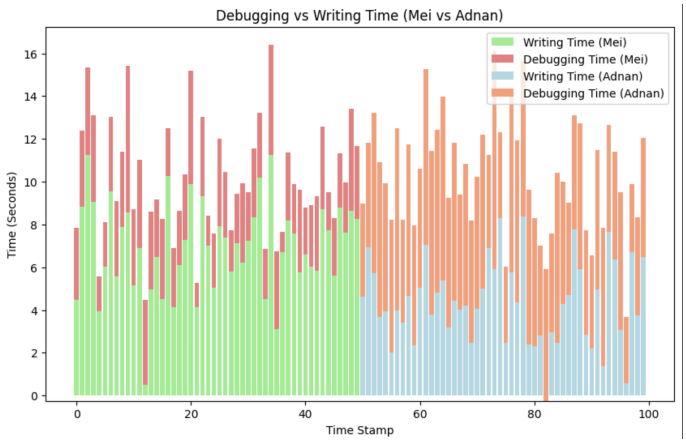
In this showdown, I will analyze their coding speed, error rates, keystroke density, debugging vs. writing time, and code complexity over time, breaking down every move with data-driven visualizations. By the end, one question will remain: **Does experience always win the fight?**

- Let's see who has more typing speed:



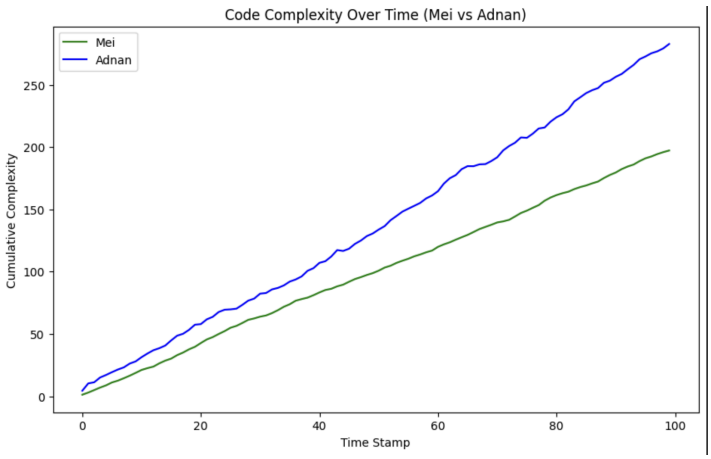
Mei seems to have a relatively uniform distribution of keystrokes.

- Now let us see how both of them utilise their time:



Mei primarily focuses on writing code, while Adnan spends a significant portion of his time debugging. 🤔

- Now let us see their code growth over time:



Here we can say over time the code complexity of Adnan grew higher.🤔

- Lets dive into code insights of our two competitors:

```
Mei Code Insights:
   Code Block   Time Spent (s)   Errors
0     Block 1          5.756989       2
1     Block 2          4.077835       1
2     Block 3          5.869606       1
3     Block 4          6.355638       3
4     Block 5          5.413435       2
5     Block 6          6.876796       1
6     Block 7          4.226211       1
7     Block 8          3.755345       3
8     Block 9          3.221280       2
9    Block 10          6.496044       3

Adnan Code Insights:
   Code Block   Time Spent (s)   Errors
0     Block 1          6.117870       2
1     Block 2          6.799899       1
2     Block 3          8.606495       4
3     Block 4          6.125799       1
4     Block 5          8.278370       3
5     Block 6          3.518894       1
6     Block 7          6.655181       3
7     Block 8          7.942210       3
8     Block 9          8.905188       1
9    Block 10          5.772289       2
```

Here we can see the time they spent on a particular block of code and the error they made. 🧐

Ladies and gentlemen, the moment we've all been waiting for... the tension is unbearable, the code has been written, and the clock is winding down... Drumroll, please... 🥁🥁🥁……And the winner is **!! MEI !!**🏆

```
CODERS CUP – Pass! Passed: 41 out of 41 test cases. Time: 13:05:19:65
```

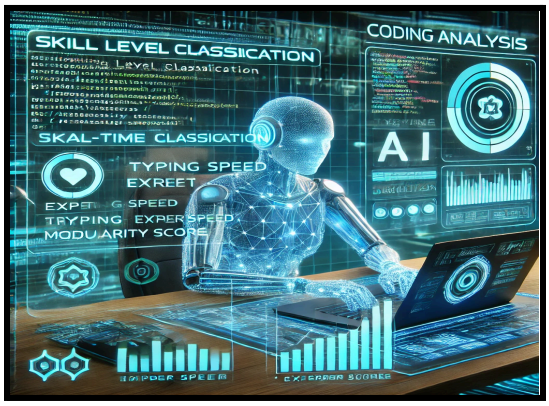
Chapter 6: Conclusion: Can We Identify an Expert Coder?

After analyzing the dataset, clear patterns emerged:

- **Experts write modular, well-documented code** with fewer debugging statements.
- **Beginners rely on excessive print-based debugging** instead of structured error handling.
- **Typing speed and fluency** were noticeably higher in experts.
- **Efficient algorithm choices** and structured error handling were key differentiators.

With more refinement, such an approach could **automatically classify coders** based on their workflow. Maybe one day, **rating systems like Codeforces will integrate AI-powered assessments** to classify coders in real-time.

This blog explored the fascinating journey of analyzing coding expertise from live competition videos. With the power of **OpenCV, OCR, pandas, and static code analysis tools**, we can automate the classification of expert vs. beginner coders.



Let's keep exploring—who knows, maybe the next step is predicting competition winners **before they even submit their code!** 🤓

REFERENCES:

[1] T. D. LaToza *et al.*, "An Exploratory Study of Live-Streamed Programming," *IEEE Xplore*, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8818832.

[2] C. Yang, F. Thung, and D. Lo, "Efficient Search of Live-Coding Screencasts from Online Videos," *arXiv preprint arXiv:2203.04519*, 2022. [Online]. Available: https://arxiv.org/abs/2203.04519.

[3] D. Zhao *et al.*, "SeeHow: Workflow Extraction from Programming Screencasts through Action-Aware Video Analytics," *arXiv preprint arXiv:2304.14042*, 2023. [Online]. Available: https://arxiv.org/abs/2304.14042.

[4] A. F. Blackwell, "Disruption and Creativity in Live Coding," *IEEE Xplore*, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9127204.

[5] A. M. Smith and T. A. Sherwood, "ActionNet: Vision-Based Workflow Action Recognition in Programming Screencasts," *IEEE Xplore*, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8811922.

[6] M. A. Rahman, M. A. Hossain, and M. A. Islam, "CODI – A Web Application to Facilitate Live, Remote Programming," *IEEE Xplore*, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9766755.

[7] M. Ali, "Beginners vs Expert Programmers," *Medium*, 2021. [Online]. Available: https://nerdjfpb.medium.com/beginners-vs-expert-programmers-967406f7d5a6.