

Where AI Meets Code

Michael Feathers - R7K Research & Conveyance

- Not about tools
- Not about how to build AI into your products
- *About* AI and code

Black Box



Black Box

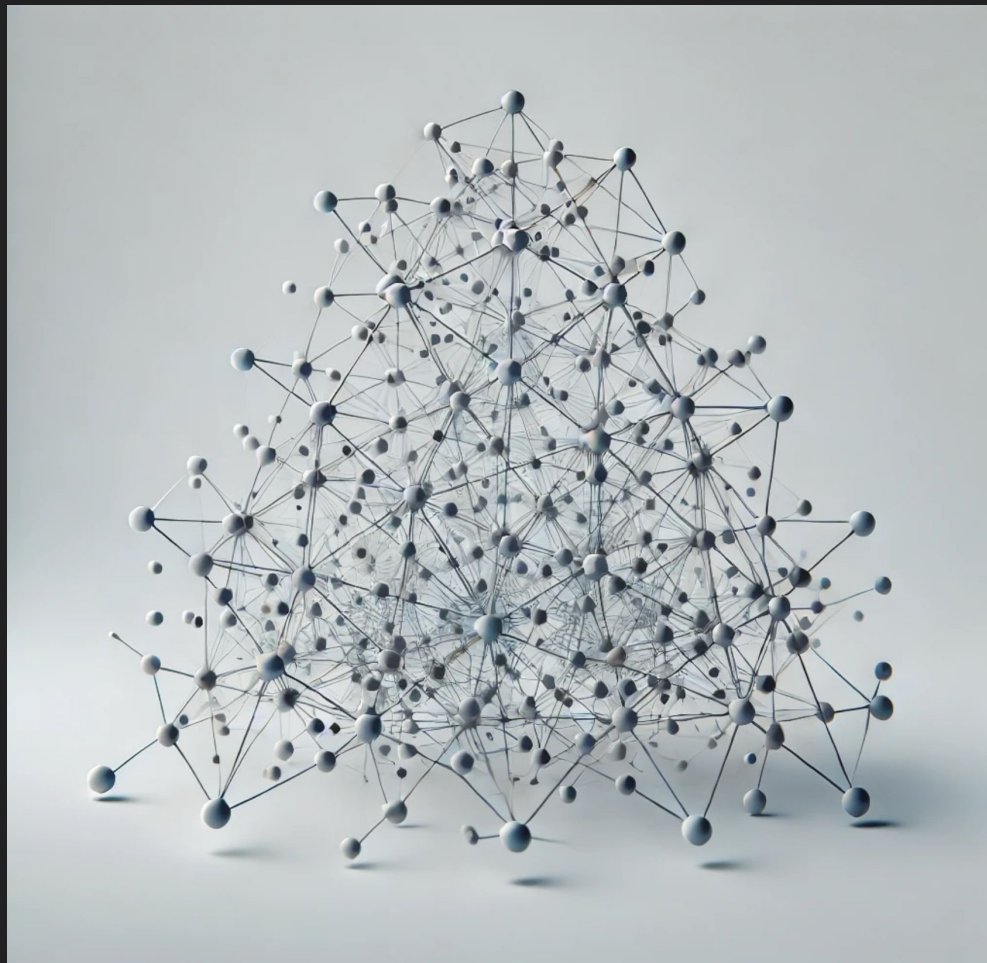


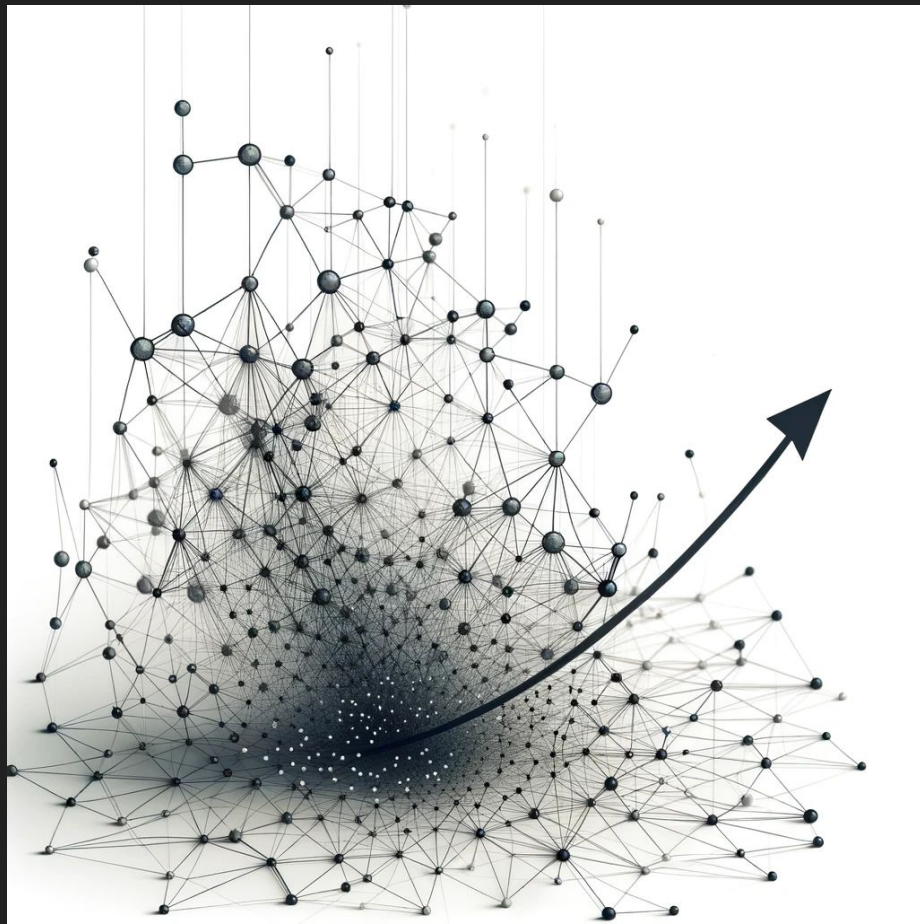
- *We have general* descriptions of LLMs*
- Tech is changing fast
- No transparency
- Tools are often “clamped”
- Specialized training
- Multi-agent systems, RAG, fine-tuning

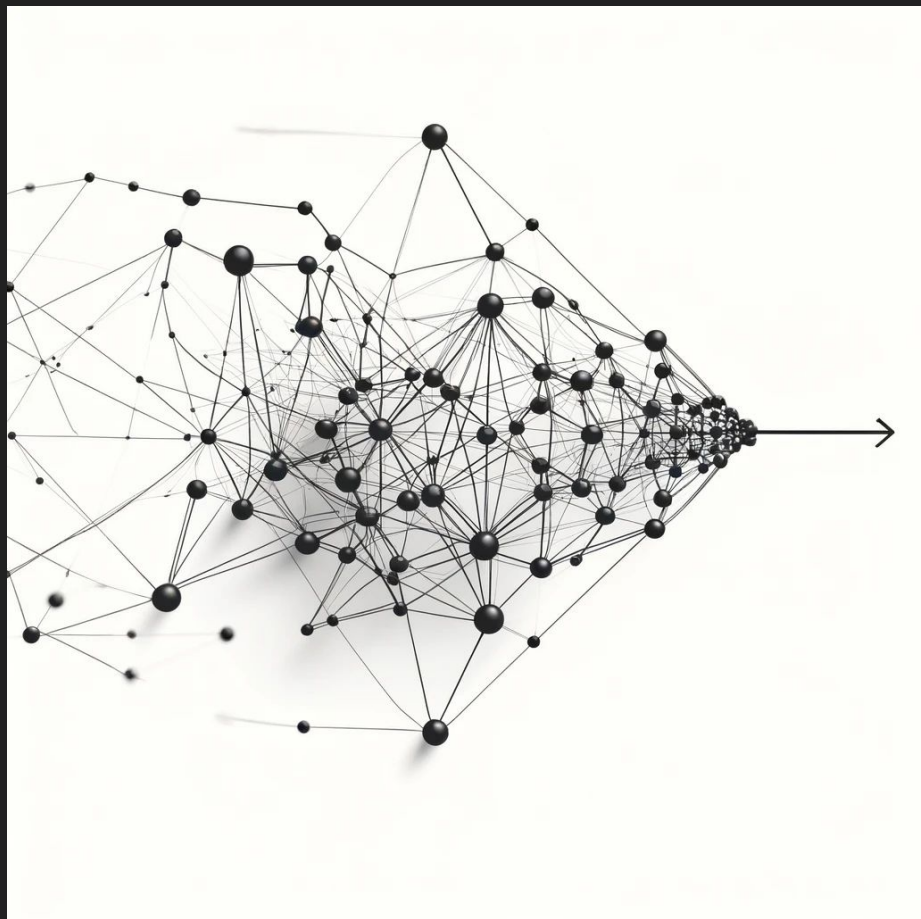
*Can we have a good
working model to guide us?*

**3Blue1Brown - Attention in transformers, visually explained*

The Surfacing Model





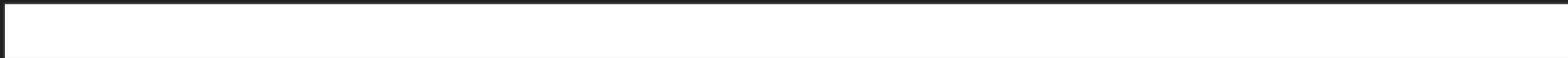






Context Window

Context Window



- The “working memory” for an LLM
 - How much can it be aware of at once

Context Window



4K → 128K → 1M → 2M

- The “working memory” for an LLM
 - How much can it be aware of at once

Lost in the Middle: How Language Models Use Long Contexts

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, Percy Liang

While recent language models have the ability to take long contexts as input, relatively little is known about how well they use longer context. We analyze the performance of language models on two tasks that require identifying relevant information in their input contexts: multi-document question answering and key-value retrieval. We find that performance can degrade significantly when changing the position of relevant information, indicating that current language models do not robustly make use of information in long input contexts. In particular, we observe that performance is often highest when relevant information occurs at the beginning or end of the input context, and significantly degrades when models must access relevant information in the middle of long contexts, even for explicitly long-context models. Our analysis provides a better understanding of how language models use their

Serial-position effect

🌐 15 languages ▾

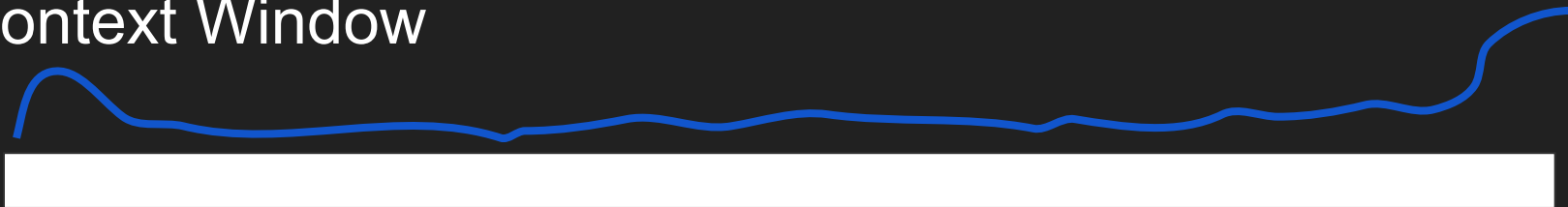
Article [Talk](#)

Tools ▾

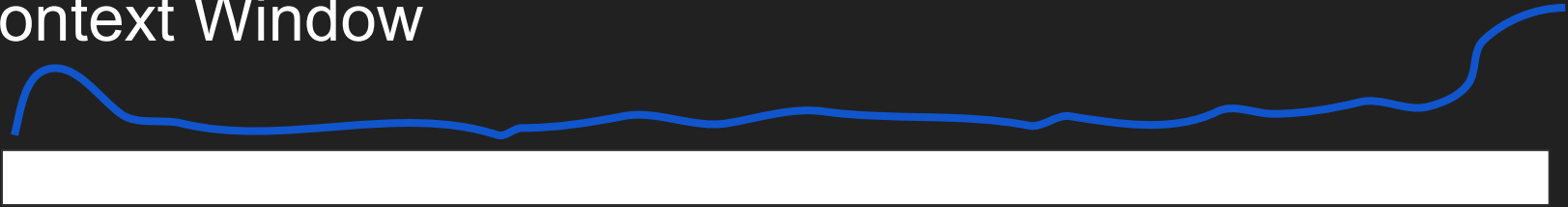
From Wikipedia, the free encyclopedia

Serial-position effect is the tendency of a person to recall the first and last items in a series best, and the middle items worst.^[1] The term was coined by [Hermann Ebbinghaus](#) through studies he performed on himself, and refers to the finding that [recall](#) accuracy varies as a function of an item's position within a study list.^[2] When asked to recall a list of items in any order ([free recall](#)), people tend to begin recall with the end of the list, recalling those items best (the **recency effect**). Among earlier list items, the first few items are recalled more frequently than the middle items (the **primacy effect**).^{[3][4]}

Context Window

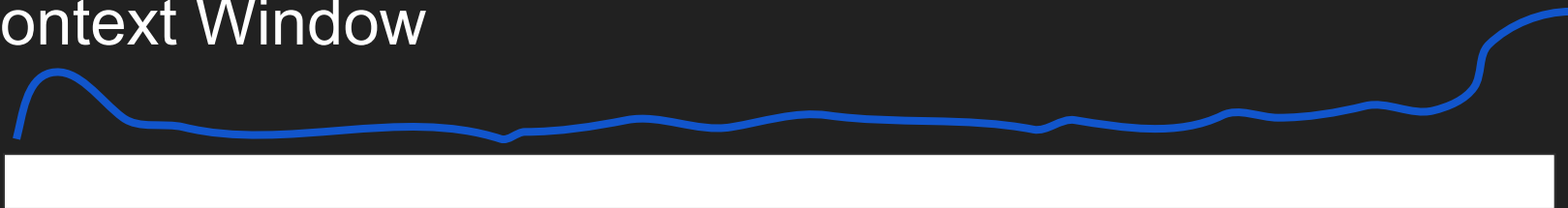


Context Window



Multi-step prompts often decrease retrieval performance

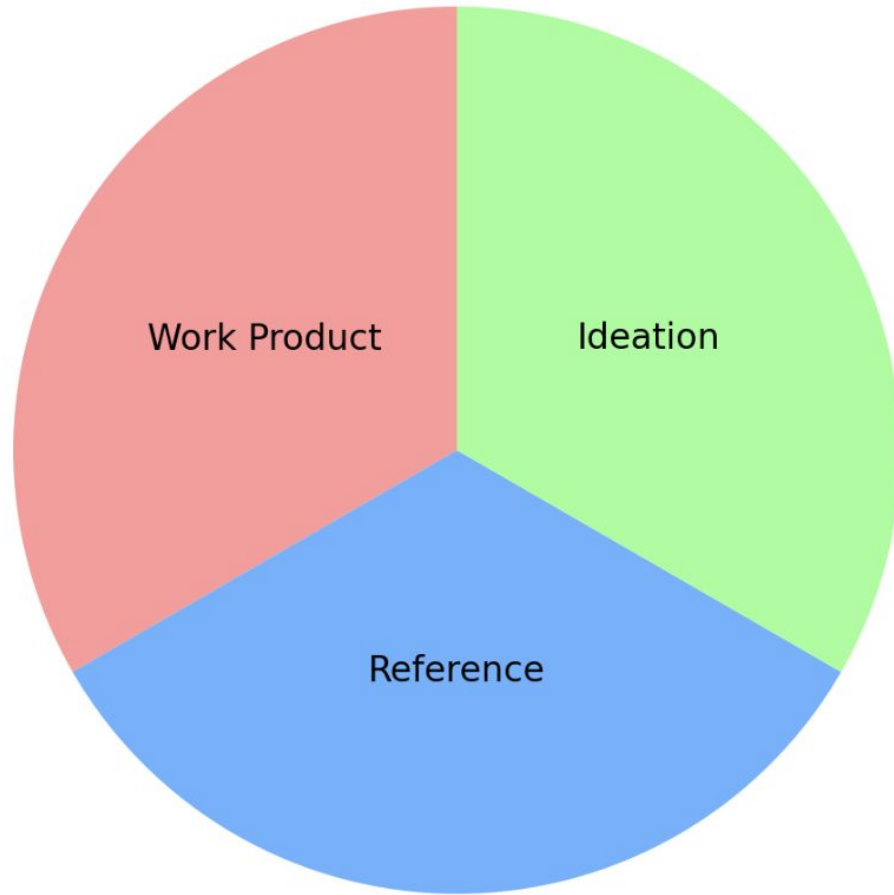
Context Window

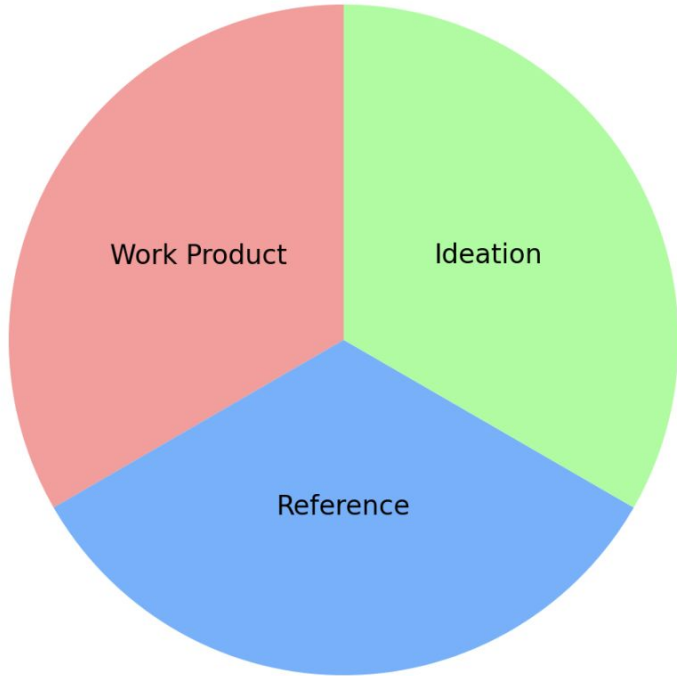


Maybe “logical attention” is smaller than the CW.

- *Maybe it can be overloaded*
- *Maybe it can change focus*
- *Maybe it dissipates over time*

Spectrum of Use

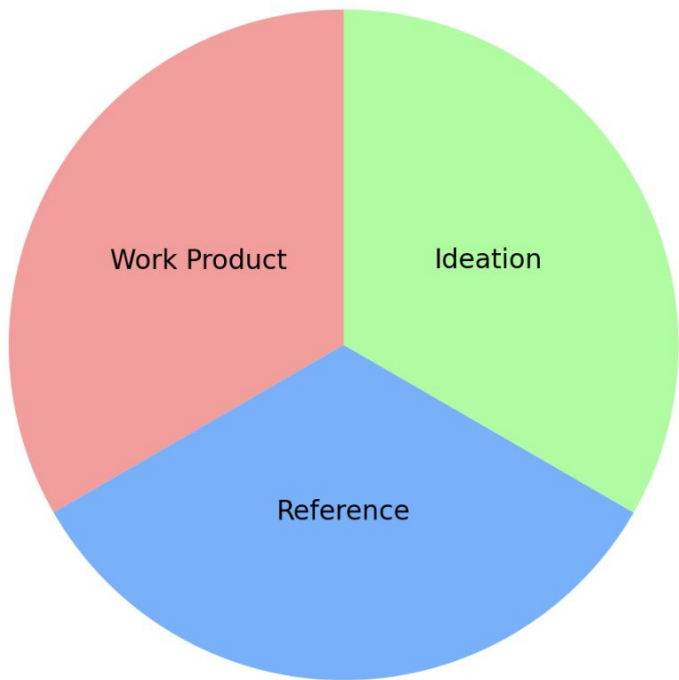




Work Product

- Generating or modifying things (code) that are part of our product
 - IP Considerations
 - Safety
 - Quality (req and impl)

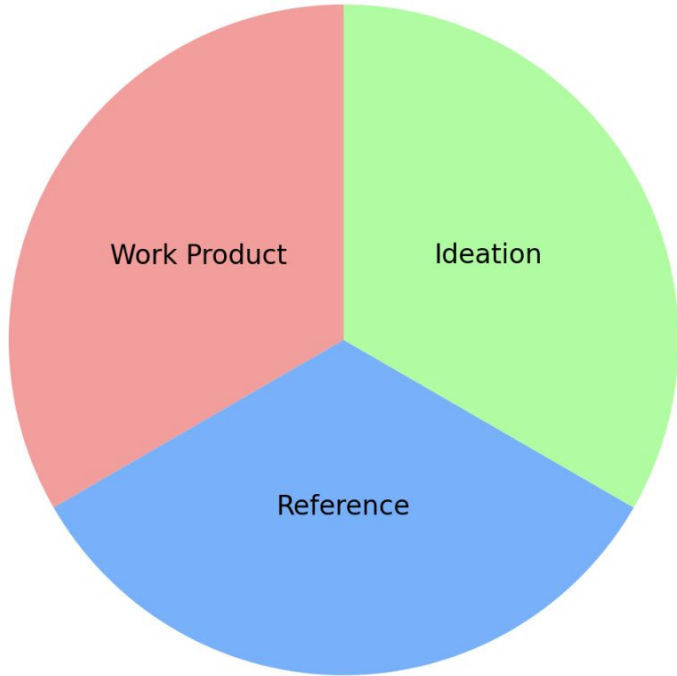
High-Validation (depending on risk)



Reference

- Finding answers (language, API, etc)
- Generating reference implementations
- Assistant-based research
 - Can be misled

Validation is “in the work”



Ideation

- Try things out
- Exploratory generation
- We can make the choice to never use these generated artifacts directly
 - Sandbox

Generating ideas, not deliverables

“It suggests, we decide”



Techniques

Make Projections

```
1  class Updater
2
3  def initialize io
4    @io = io
5  end
6
7  def run
8    actions = @io.read_actions.lines
9    updates = @io.read_updates.lines
10
11    due_updates = due(updates)
12    non_due_updates = non_due(updates)
13
14    new_actions = due_updates + actions
15    @io.write_actions(new_actions)
16
17    sorted_non_due = non_due_updates.sort_by do |line|
18      Day.from_text(line.split.first).date
19    end
20    @io.write_updates(sorted_non_due)
21  end
```

Let:

A = set of actions

U = set of updates

$D = \{u \in U \mid \text{date}(u) \leq \text{today}\}$ (due updates)

$N = U \setminus D$ (non-due updates)

Operations:

$A' = D \cup A$ (new actions)

$N' = \text{sort}(N)$ by date

Output:

Write A' to actions file

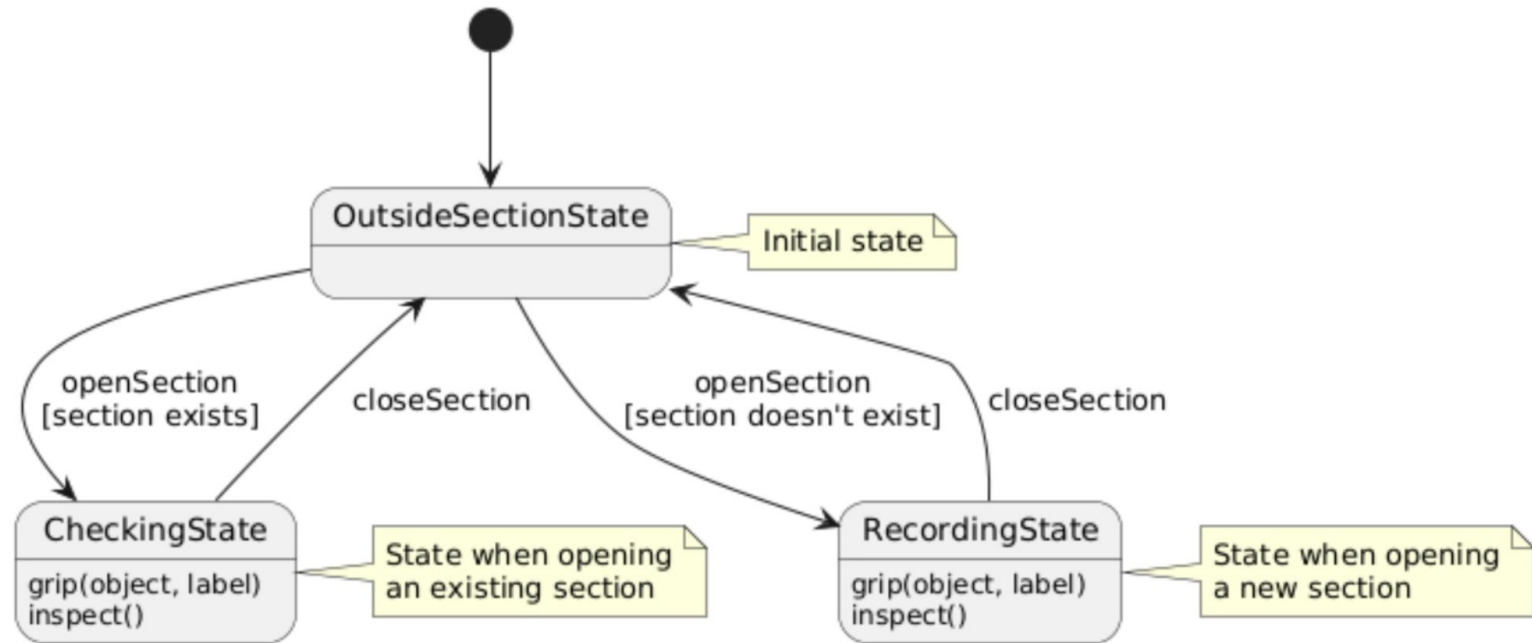
Write N' to updates file

Where:

$\text{date}(u)$ extracts the date from update u

$\text{sort}(N)$ arranges elements of N in ascending order of date

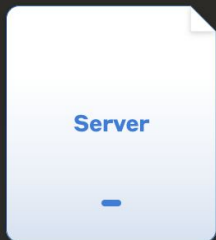
```
1  @startuml
2  [*] --> OutsideSectionState
3
4  state OutsideSectionState
5  state CheckingState
6  state RecordingState
7
8  OutsideSectionState --> CheckingState : openSection\n[sec\
9  tion exists]
10 OutsideSectionState --> RecordingState : openSection\n[se\
11 ction doesn't exist]
12
13 CheckingState --> OutsideSectionState : closeSection
14 RecordingState --> OutsideSectionState : closeSection
15
16 CheckingState : grip(object, label)
17 CheckingState : inspect()
```



There Are Many Possible Projections

- Project to Math
- Project to Diagrams
- Project to another Programming Language
 - Then Invert
- ...

Lensing



MF

List the 7 most important responsibilities in the Server class. Nofluff

Here are the 7 most important responsibilities of the Server class:

1. Parsing command line arguments to initialize the server configuration.
2. Reading the email list file into a Vector of InetAddress objects.
3. Connecting to the POP3 server to retrieve emails from the inbox.
4. Checking each email message in the inbox for unread status.
5. Creating a new email message with the original email's content and a predefined template.
6. Sending the new email message to the list of recipients using the SMTP server.
7. Deleting the processed emails from the inbox and closing the connection to the POP3 server.

From the point of view of the user, the 7 most important responsibilities of the Server class are:

1. Automatically checking for new emails at a specified interval (determined by the "checkPeriod" command line argument).
2. Retrieving unread emails from the user's inbox on the specified POP3 server.
3. Extracting relevant information from the original email, such as the subject, sent date, and content.
4. Creating a new email message with the original email's content and a predefined template, making it appear as if the user is sending the email on behalf of the original sender.
5. Sending the new email message to a list of recipients specified in the "emailListFile" command line argument, using the specified SMTP server.
6. Deleting the processed emails from the user's inbox to avoid duplicate processing in future iterations.
7. Running continuously in the background, allowing the user to receive and send emails automatically without manual intervention.

These responsibilities are important from the user's perspective as they automate the process of receiving, processing, and sending emails based on a predefined set of rules and templates, saving the user time and effort.

MF

What are the 4 most important from the pov of the user?

From the user's point of view, the 4 most important responsibilities of the Server class are:

1. Automatically checking for new emails at regular intervals, so the user doesn't have to manually check their inbox.
2. Sending out emails to a predefined list of recipients on behalf of the original sender, making it appear as if the user is forwarding the email.
3. Using a predefined email template to create new messages, ensuring consistency in the outgoing emails without requiring the user to draft each message manually.
4. Running continuously in the background, allowing the user to have their emails processed and sent out even when they are not actively using the application.

These four responsibilities cover the core functionality that the user would expect from an email forwarding and templating system, automating the most time-consuming and repetitive tasks while maintaining a consistent outgoing email format.



Call 1 the Scheduling responsibility. Extract it to another class.

To extract the scheduling responsibility from the Server class, we can create a new class called "EmailScheduler" that will handle the scheduling aspect of the application. Here's how you can refactor the code:

1. Create a new class named "EmailScheduler":

java

Copy code

```
public class EmailScheduler {
    private Server server;
    private int checkPeriod;

    public EmailScheduler(Server server, int checkPeriod) {
        this.server = server;
        this.checkPeriod = checkPeriod;
    }

    public void start() {
        while (true) {
            server.processEmails();
            try {
                Thread.sleep(checkPeriod * 1000 * 60);
            } catch (InterruptedException e) {
```

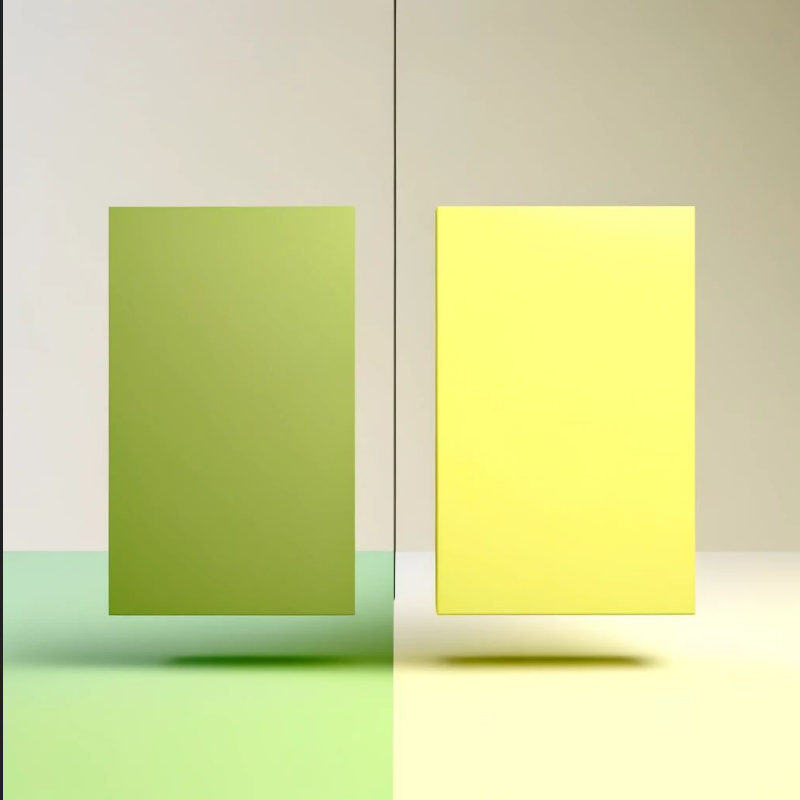
Generalization / Specialization in the Tool Space



Gwern - “The Scaling Hypothesis”

Do `Side By Sides`

Side By Side

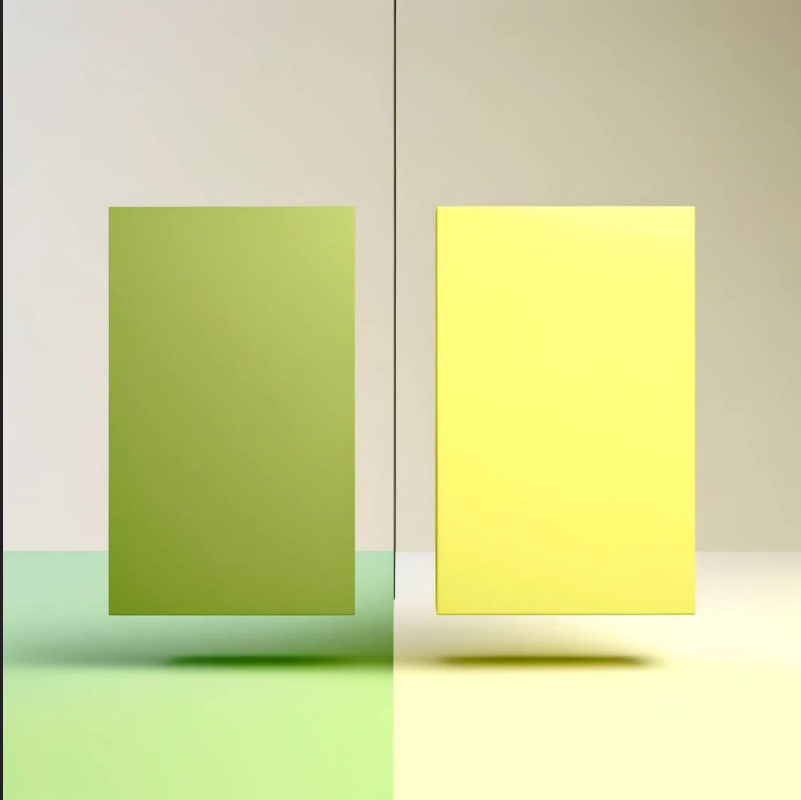


- **Offer Then Ask**

- Give the tool something you have made and then ask it for an alternative or critique
- Check to standards

Gold + Google, No. 10-11-12

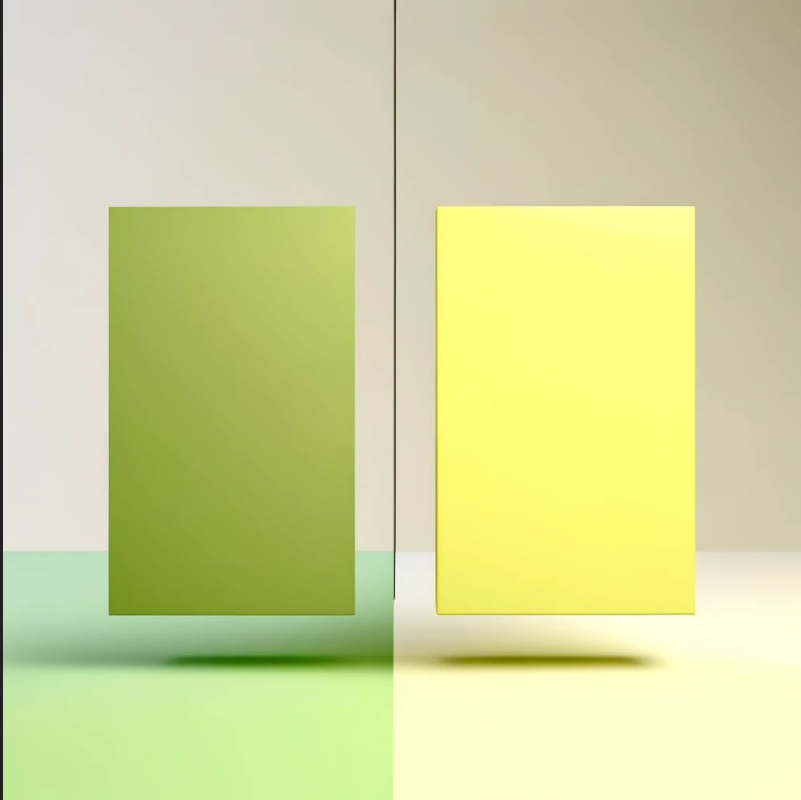
Side By Side



- **Solve Then Ask**
 - Solve a problem and then ask the tool to solve the same problem without seeing your solution.

Guides: [Google](#), [The City](#)

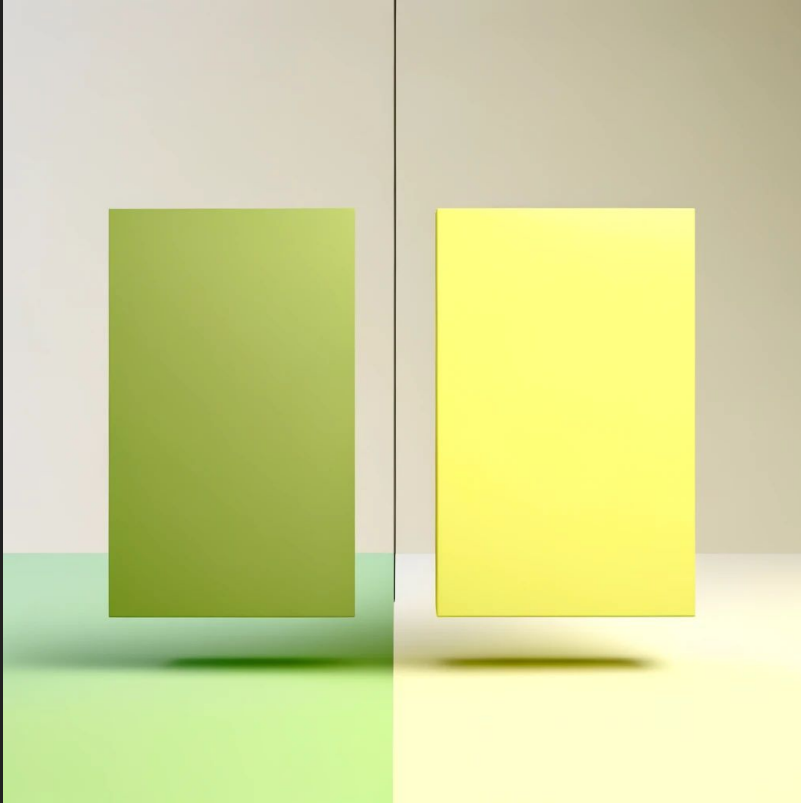
Side By Side



- **Ask Then Use**
 - Generate as solution and then use it.
 - We have to confront the quality issue directly.

Could it Google it for you?

Side By Side



- **Ask Then Reference**

- Generate a solution and then use it as a model for yours.

Guides: Google, LLM, or other

Create Pidgin Languages

```
13         ? ((MechanicsState)new CheckingState(store))
14         : ((MechanicsState)new RecordingState(store));
15     state.openSection(sectionName);
16 }
17
18 public boolean isRecording() {
19     return state.isRecording();
20 }
21
22 public void grip(Object object) {
23     state.grip(object, "");
24 }
25
26 public void grip(Object object, String label) {
27     state.grip(object, label);
28 }
```

```
13         ? ((MechanicsState)new CheckingState(store))
14         : ((MechanicsState)new RecordingState(store));
15     state.openSection(sectionName);
16 }
17
18 public boolean isRecording() {
19     return state.isRecording();
20 }
21
22 public void grip(Object object) {
23     state.grip(object, "");
24 }
25
26 public void grip(Object object, String label) {
27     state.grip(object, label);
28 }
```

test: grip o -> assert state.grip o

```
@Mock
private MechanicsState mockState;

private DefaultViseMechanics viseMechanics;

@Before
public void setUp() {
    MockitoAnnotations.initMocks(this);
    viseMechanics = new DefaultViseMechanics(mockStore);
    // Set the mocked state
    viseMechanics.state = mockState;
}

@Test
public void testGripCallsStateGripWithEmptyLabel() {
    Object testObject = new Object();

    viseMechanics.grip(testObject);

    verify(mockState).grip(testObject, "");
}
}
```

Make Waywords

“Convert all uses of Date to DateTime in this method but maintain the public interface. This will require converting all Date parameters to DateTime after they have been passed into the method and converting the return value to Date if necessary. If there are Date instance variables, do not convert them. Instead, convert DateTime to Date or Date to DateTime as needed to interact with them.”

“Do this to the *X* method also.”

“Do this to the *X* method also.”

“Intralate type *X* to type *Y* in method *M*.”

“Call the operation I just performed ‘intralate’. I want to be able to say: Intralate type X to type Y in method M.”

Generalizing

As an exercise, I tried to generalize *intralate* further. I wanted to be able to *intralate* a type in a method but also in a class, or a service.

In all of these cases, you need a notion of the *interface* that is being preserved. When you intralate methods, instance variables and method calls within the method are part of the interface – they interface with the rest of the class (and the rest of the world). In contrast, when you *intralate* a class, they are part of the “inside” – the types of instance variables and the types in internal method calls should always be converted.

More Extreme Waywording

```
1  def load_command_log
2    @log ||= @list.io
3      .read_log
4      .split($/)
5      .map { |line| line.split(',') }
6      .select { |items| items.size == 2 }
7      .map { |k, v| [k, v.to_i] }.to_h
8    @log.default = 0
9    @log
10  end
```

“I want to do this operation on many methods. Call this operation ‘floya.’ Perform it whenever I ask for the ‘floya’ of a method.”

Certainly, I understand that you want to define an operation called 'floya' that can be applied to Ruby methods. The 'floya' operation involves:

Merging contiguous sequences of statements with method chains that have no side effects into a single statement (or return expression). Separating any method chains that contain side effects into different statements. Not introducing new blocks. Not reusing local variables.

I'll keep this in mind, and whenever you ask for the 'floya' of a method, I'll apply these transformations. Just to confirm, here's a summary of what 'floya' does:

*Combines chains of pure operations into one line
Separates operations with side effects
Preserves overall functionality
Doesn't introduce new blocks or reuse local variables*

June 6, 2024

Extracting Concepts from GPT-4

We used new scalable methods to decompose GPT-4's internal representations into 16 million oft-interpretable patterns.

[Read paper ↗](#)

[Read the code ↗](#)

[Browse features ↗](#)

GPT-4 feature: phrases relating to things (especially humans) being flawed

[View full visualization ↗](#)

most people, it isn't. We all have wonderful days, glimpses of what we perceive to be perfection, but we can also all have truly shit-tastic ones, and I can assure you that you're not alone. So toddler of mine, and most other toddlers out there, remember; Don't be a

has warts. What system that is used to build real world software doesn't? I've built systems in a number of languages and frameworks and they all had warts and issues. How much research has the author done to find other solutions? The plea at the end seemed very lazywebish to me

System 1 and System 2

“The automatic operations of System 1 generate surprisingly complex patterns of ideas, but only the slower System 2 can construct thoughts in an orderly series of steps.”

– *Daniel Kahneman in Thinking, Fast and Slow*

System 1 and System 2

“The automatic operations of System 1 generate surprisingly complex patterns of ideas, but only the slower System 2 can construct thoughts in an orderly series of steps.”

– *Daniel Kahneman in Thinking, Fast and Slow*

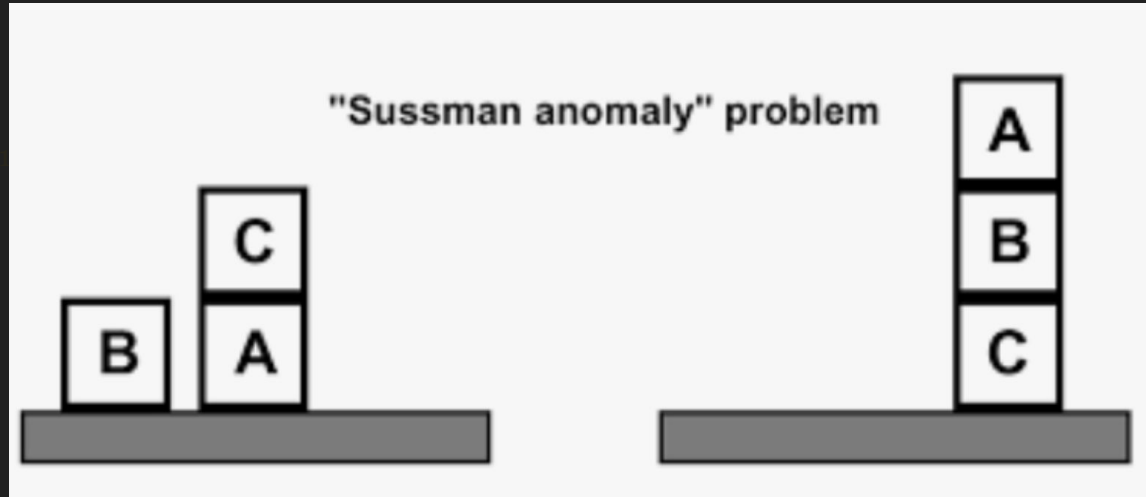
LLMs are great at System 1

Could've guessed the location

Could've guessed the location

Planning

- Can Large Language Models Reason and Plan? - Subbarao Kambhampati



I am playing with a set of blocks where I need to arrange the blocks into stacks. Here are the actions I can do

Pick up a block

Unstack a block from on top of another block

Put down a block

Stack a block on top of another block

I have the following restrictions on my actions:

I can only pick up or unstack one block at a time.

I can only pick up or unstack a block if my hand is empty.

I can only pick up a block if the block is on the table and the block is clear. A block is clear if the block has no other blocks on top of it and if the block is not picked up.

I can only unstack a block from on top of another block if the block I am unstacking was really on top of the other block.

I can only unstack a block from on top of another block if the block I am unstacking is clear.

Once I pick up or unstack a block, I am holding the block.

I can only put down a block that I am holding.

I can only stack a block on top of another block if I am holding the block being stacked.

I can only stack a block on top of another block if the block onto which I am stacking the block is clear.

Once I put down or stack a block, my hand becomes empty.

I am playing with a set of objects. Here are the actions I can do

Attack object

Feast object from another object

Succumb object

Overcome object from another object

I have the following restrictions on my actions:

To perform Attack action, the following facts need to be true: Province object, Planet object, Harmony

Once Attack action is performed the following facts will be true: Pain object

Once Attack action is performed the following facts will be false: Province object, Planet object, Harmony

To perform Succumb action, the following facts need to be true: Pain object

Once Succumb action is performed the following facts will be true: Province object, Planet object, Harmony

Once Succumb action is performed the following facts will be false: Pain object.

To perform Overcome action, the following needs to be true: Province other object, Pain object

Once Overcome action is performed the following will be true: Harmony, Province object, Object Craves other object

Once Overcome action is performed the following will be false: Province other object, Pain object

To perform Feast action, the following needs to be true: Object Craves other object, Province object, Harmony.

Once Feast action is performed the following will be true: Pain object, Province other object

Once Feast action is performed the following will be false:, Object Craves other object, Province object, Harmony

Planning

Blocksworld	One shot	206/600 (34.3%)	138/600 (23%)	289/600 (48.17%)	68/600 (11.3%)
	Zero shot	210/600 (34.6%)	241/600 (40.1%)	356/600 (59.3%)	3/600 (0.5%)
Mystery Blocksworld	One shot	26/600 (4.3%)	5/600 (0.83%)	8/600 (1.3%)	2/500 (0.4%)
	Zero shot	1/600 (0.16%)	1/600 (0.16%)	0/600 (0%)	0/500 (0%)

Caveats

- These are base LLMs
- LLM providers may be incorporating planning ability via modified architectures and integrations

Recursion and Levels

- Reports: It is hard to lead LLMs to solutions that involve recursion
- Experience: Testing *through* levels fails past a certain point

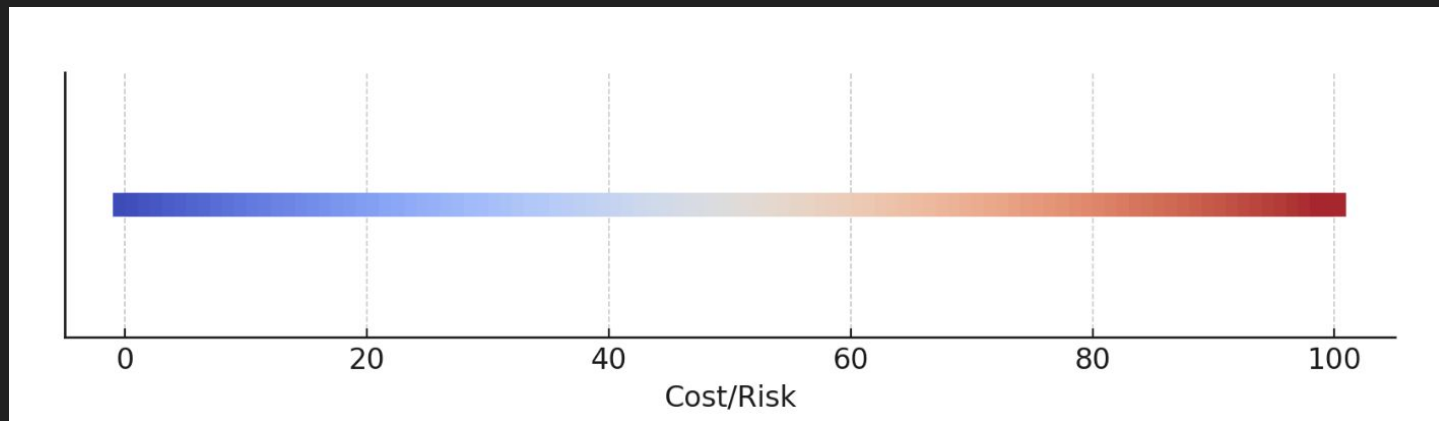
Build's Google ... No. 13-4829

More Techniques

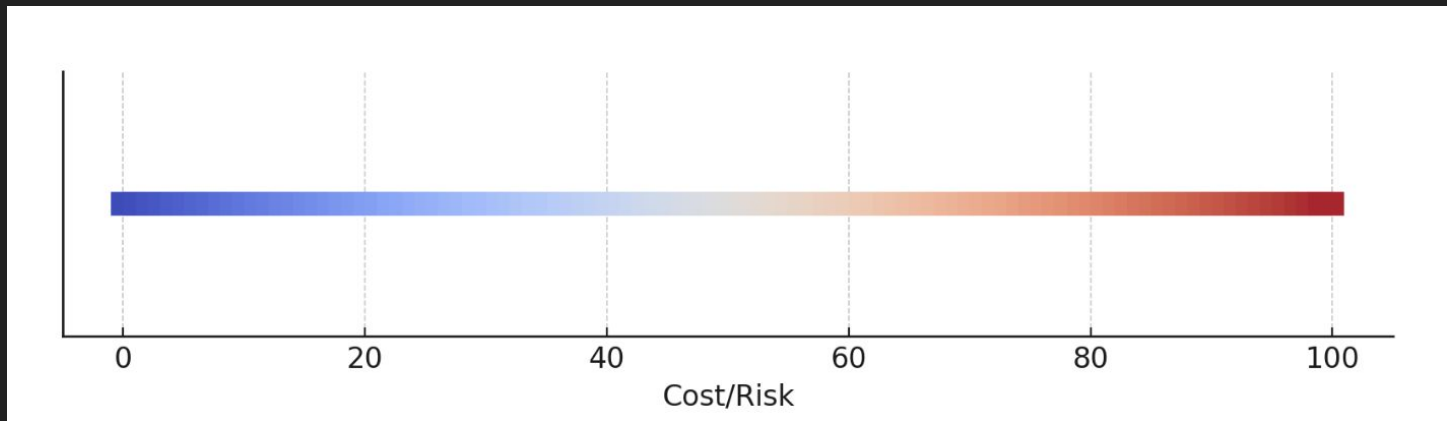
- Ask For N / Ask Again
- Notice Names
- Understand the Generated
- Step Check
- Completion Check
- Summary Prompting (Session Transport Problem)

Cost / Risk

Cost / Risk



Cost / Risk

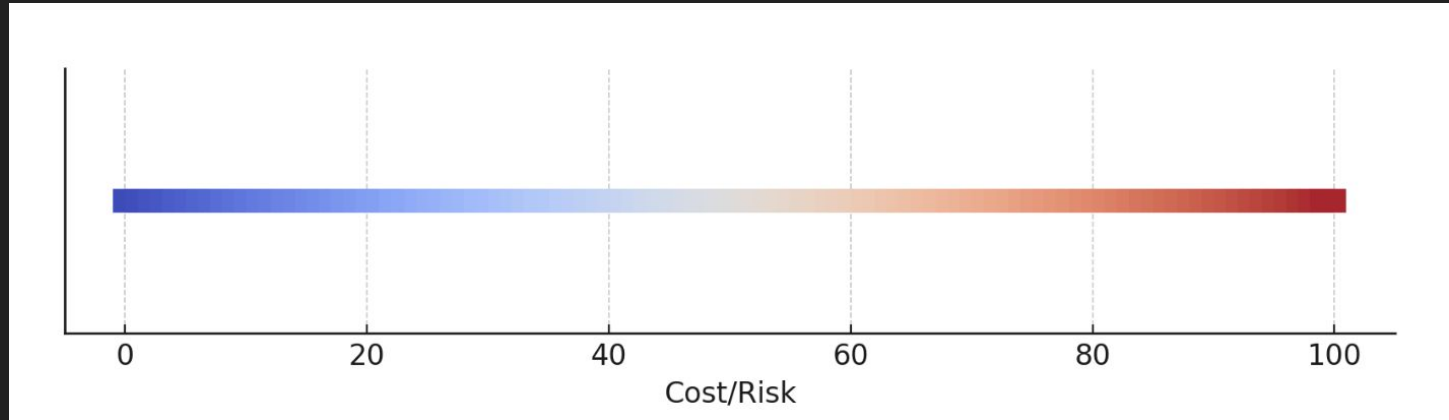


support code scaffolding consequential computation

internal tooling experiments key customer value features

disposable tests dynamic policy choices

Cost / Risk



gaming

entertainment

startups

regulated industries

mature market

life / fortune

creator tools

low switching cost

medical / law

Review Fatigue

Review Fatigue

(Attention Debt?)

Review is harder than the incremental learning that happens during development

You have to build up new context each time

AI has to pay for itself wrt to frustration and fatigue

We can and do discard tools that increase our
cognitive load

Work in Small Chunks

Get used to pivoting and discarding
things that don't work

(do this immediately)

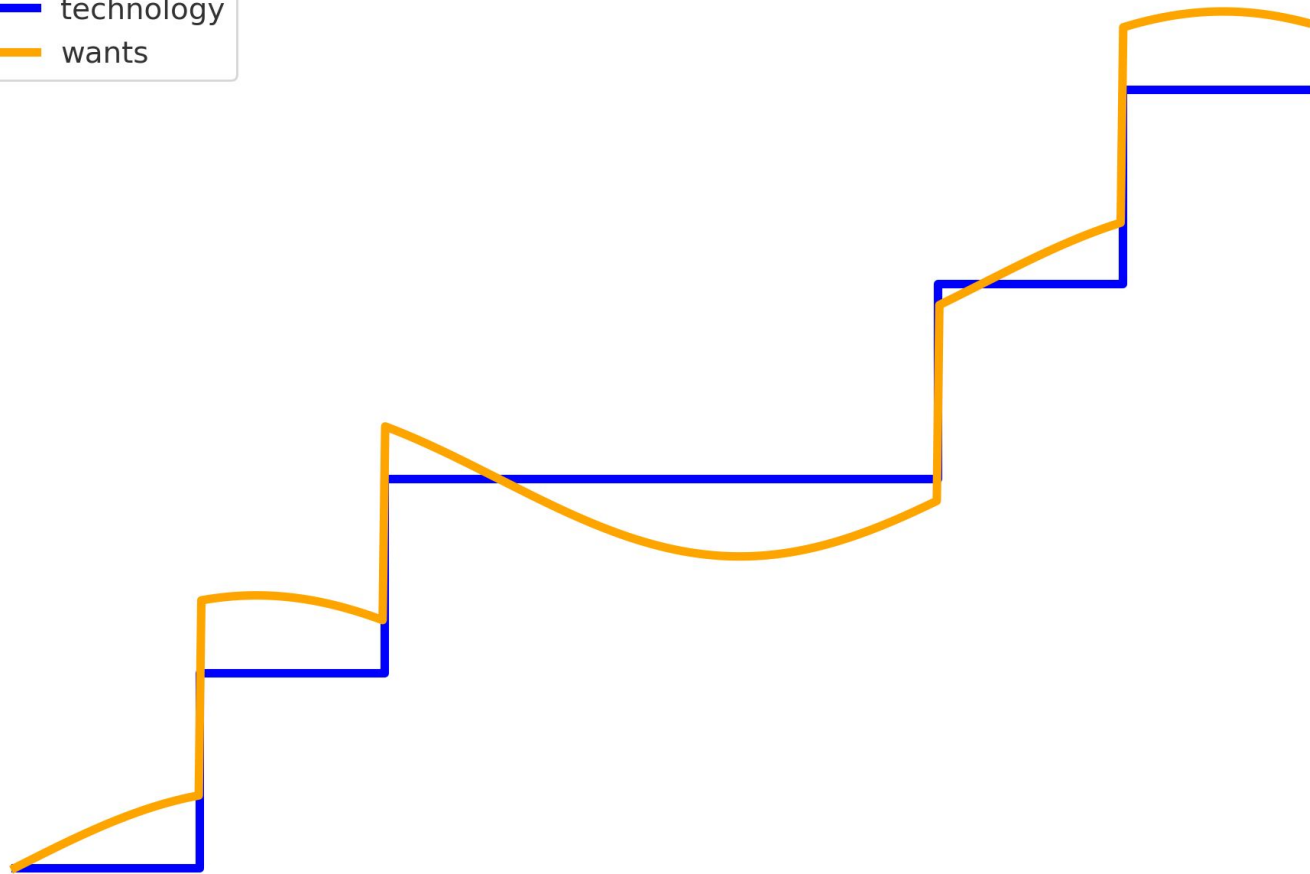
Automated Testing



Rigor



The Tech / Want Staircase

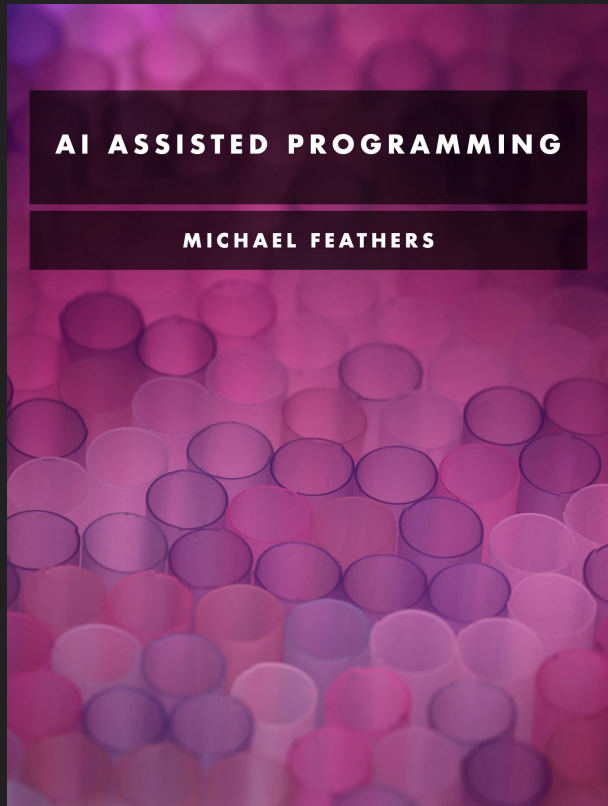


Our Trajectory

- We learn how to ask better questions
- The interaction is only as good as we are
- These tools make us better

This Is the Worst It Will Ever Be





On Leanpub

(in progress)