Towards AI-Native Software Engineering (SE 3.0)

Ahmed E. Hassan

(in collab with CSE Team@Huawei)

Mustafa Prize Laureate for Intelligent Software Engineering

Fellow of ACM/IEEE/Steacie/AAIA

How to cite this session?

```
@misc{Hassan2024AIwareTutorialSE3,
author = {Ahmed E. Hassan and Gustavo A. Oliva and Dayi Lin and Boyuan Chen and Zhen Ming and Jiang},
title = {Towards AI-Native Software Engineering (SE 3.0)},
howpublished = {Tutorial presented at the AIware Leadership Bootcamp 2024},
month = {November},
year = {2024},
address = {Toronto, Canada},
note = {Part of the AIware Leadership Bootcamp series.},
url = {https://aiwarebootcamp.io/slides/2024_aiwarebootcamp_hassan_towards_ai_native_se.pdf }}
```

Check this paper for more information about this session

```
@article{Hassan2024SE3,
  title={Towards AI-Native Software Engineering (SE 3.0): A Vision and a Challenge Roadmap},
  author={Ahmed E. Hassan and Gustavo A. Oliva and Dayi Lin and Boyuan Chen and Zhen Ming and Jiang},
  journal={arXiv preprint arXiv:2410.06107},
  year={2024}
}
```

Overview of the session

- ❑**Rethinking Software Engineering**
  - ❑Software eras
  - ❑Software engineering eras
  - ❑Towards AI-native SE: our vision and technology stack
- ❑**Where are we today?**
  - ❑**AI4SE:** A very shallow perspective on the ROI of AI4SE
  - ❑**SE4AI:** Too focused on AI models and prompting instead of AI systems
- ❑**Conclusion**

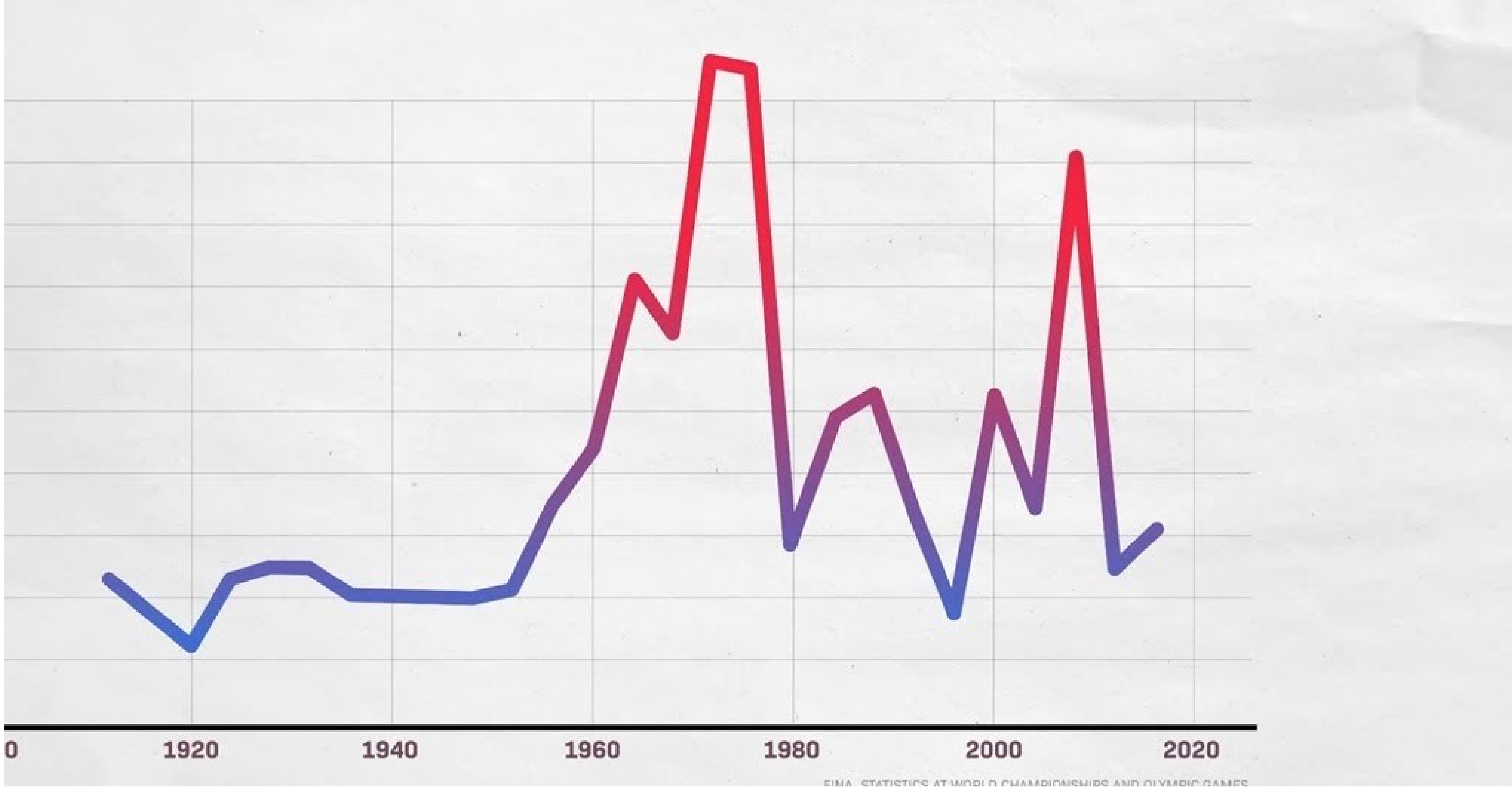
Overview of the session

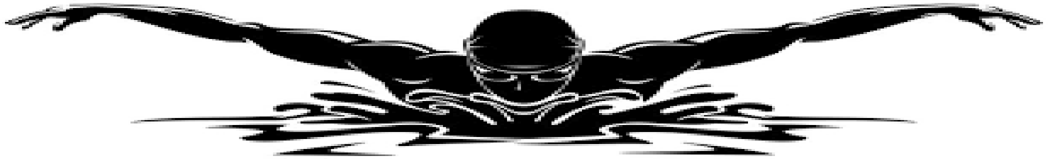
- ❑**Rethinking Software Engineering**
  - ❑Software eras
  - ❑Software engineering eras
  - ❑Towards AI-native SE: our vision and technology stack
- ❑**Where are we today?**
  - ❑**AI4SE:** A very shallow perspective on the ROI of AI4SE
  - ❑**SE4AI:** Too focused on AI models and prompting instead of AI systems
- ❑**Conclusion**

Avoiding Strategic and Technological Surprises

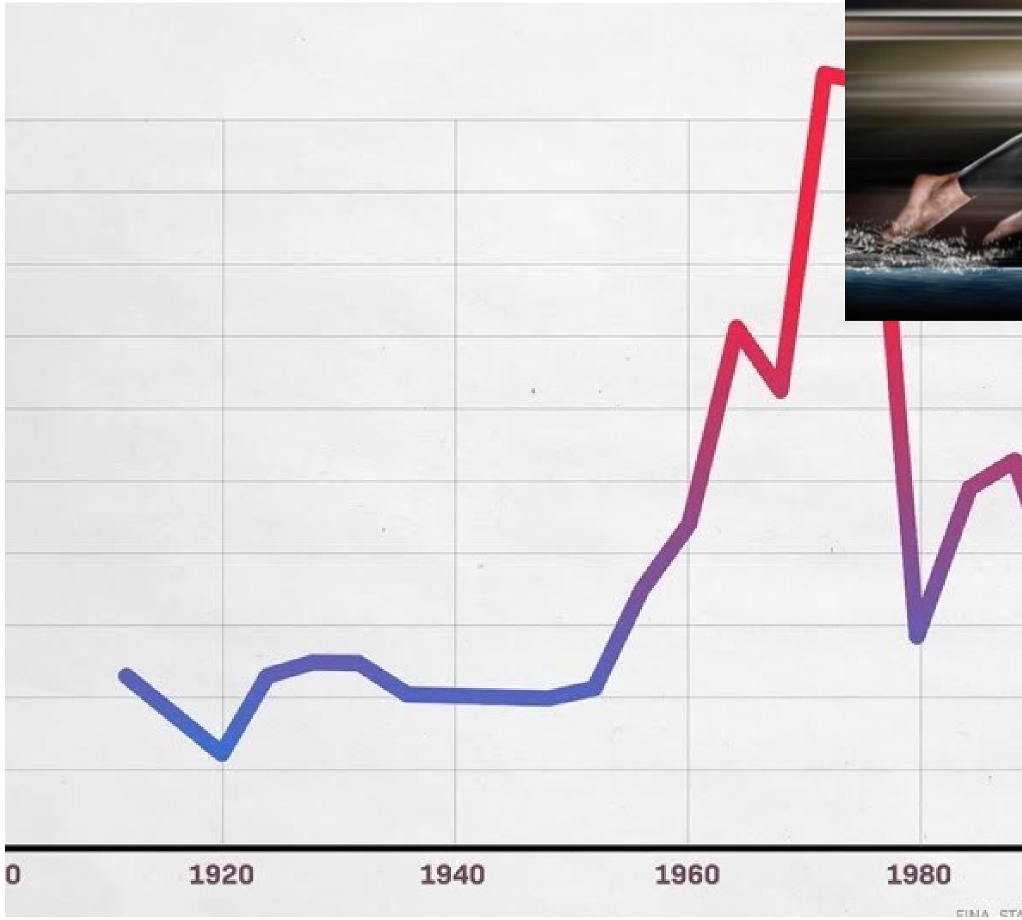
*An innovation engine is needed to undertake, support and direct **high-risk** projects with potential for **game-changing high-return impacts***

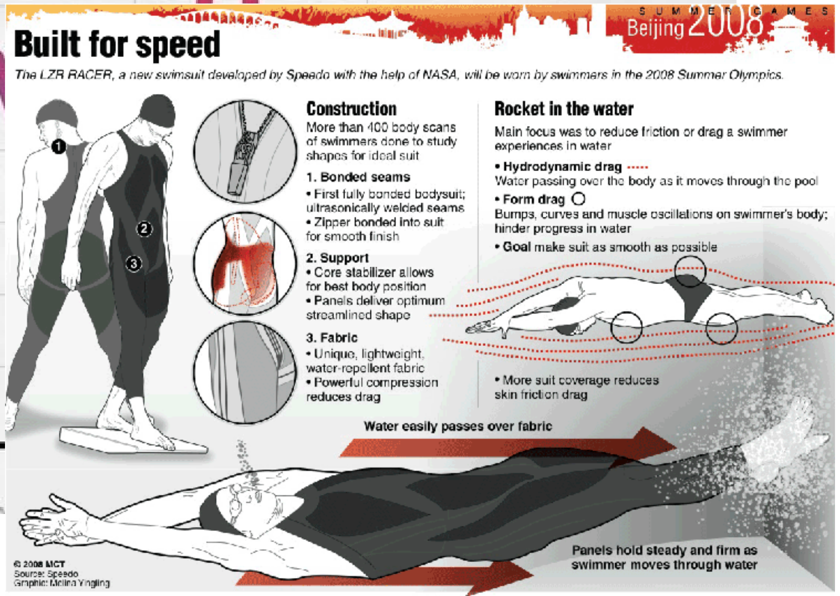
# of broken records per year in Olympics & World championships for swimming

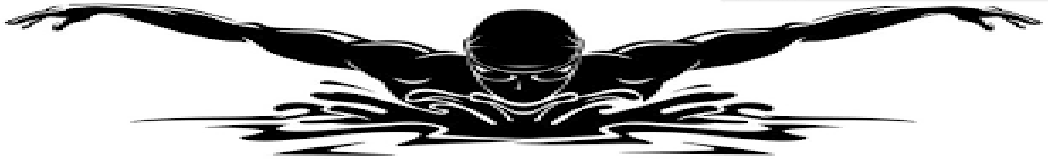




of broken records per year in Olympics & World championships for swimming







Hassan et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

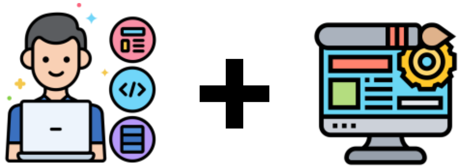
The "Dick Fosbury Flop" in High Jump

The Straddle

Fosbury Flop

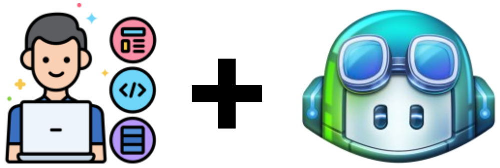
Mexico 1968 Olympics

AIware: Rethinking Software Engineering!



Software Engineering 1.0

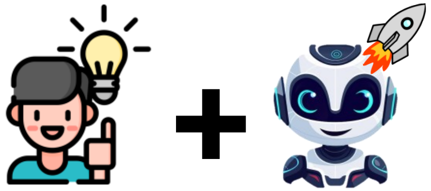
- **Code**-first
- **Tools** supporting **traditional** SE Process activities
- Powered by **program analysis technologies**



Software Engineering 2.0

We are mostly here now!

- **Code**-first
- **AI models** supporting **traditional** SE process activities (AI4SE)
- Powered by **expensive data-driven models** with limited capabilities



Software Engineering 3.0

- **Intent-first**, conversation-oriented development
- **AI-native** SE process maximizing the strengths of human (reqs) & AI (impl)
- Powered by **efficient knowledge-driven** models with advanced reasoning capabilities

The Brain's last stand

"We humans are trying to figure out our next move" Dan Rather, NBC

"A weak human player + machine
+ *better process*

is superior

to a very powerful machine alone,

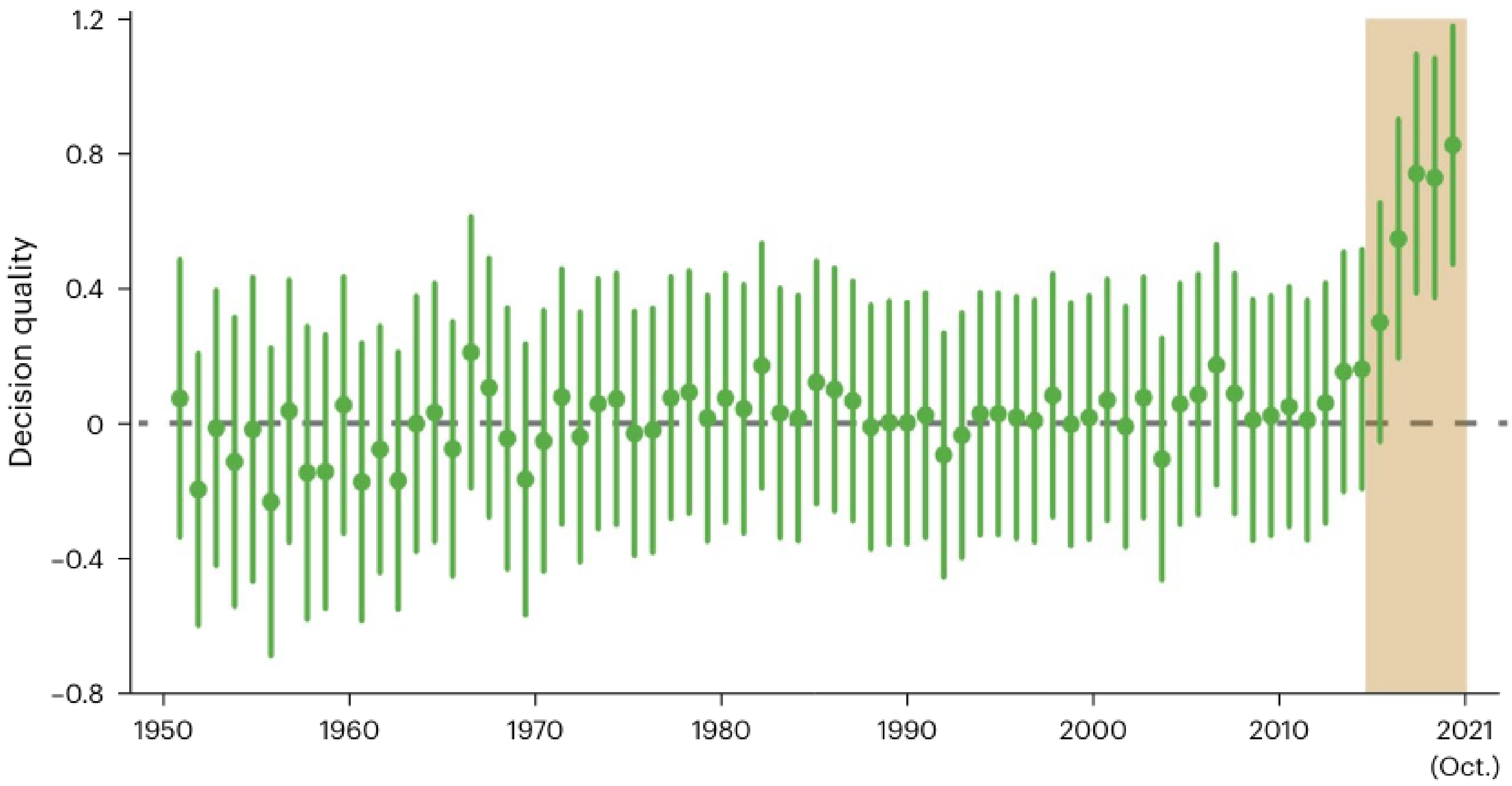
But more remarkably is superior to

a stronger human player + machine
+ *inferior process* "

Gary Gasparov

AI can make developers better

- Strong teams attract strong candidates, making them stronger and keeping them longer ➔ **You AI teammate is a coach that is making you a better developer**



Decision quality of professional Go players evaluated through an algorithm that is performing at a superhuman level. Decision quality significantly increased after Sedol was beaten by AlphaGo on 15 March 2016 (shaded area) [Brinkmann 2003]

What is Software Engineering?

The needed R&D capabilities+ Programming System for the *efficient* transformation of Intent to *high quality* Software

Code is just a means to an end!

Yet today it is treated as the most important aspect, we care about code quality, code health, code automation…. Instead Intent which is the most important aspect is IGNORED!!

Teammate.next
Personalized AI partners

IDE.next
(Developing, Debugging, Maintaining)
Intent-First + Conversational

Compiler.next
Code realization through synthesis and search

Runtime.next
SLA-aware Uni-Cluster Runtime with Edge Extension

FM.next
Curriculum engineered models

We are re-thinking the Software & SE Stacks

Actually we are also rethinking the programming model itself ☺ [observable verifiable and controllable multi-agents framework]!

Teammate.next
Personalized AI partners

IDE.next
(Developing, Debugging, Maintaining)
Intent-First + Conversational

Compiler.next
Code realization through synthesis and search

Runtime.next
SLA-aware Uni-Cluster Runtime with Edge Extension

FM.next
Curriculum engineered models

We are re-thinking the Software & SE Stacks

AI-Native SE: Intent-Centric instead of Code-Centric SE

Alignment of Intent (Requirement Process) ⟷ Realization of the Goal (Coding is a mean to an end!)

HUMAN+AI | **AI**

A core problem that will exist no matter how powerful the AI becomes since it is a human limitation (human's ability to articulate clearly their intent)

Two way arrow, since we might go back and forth during the realization process

TRADITIONALY:

1. SE spends great amount of time on Intent alignment as cost to adjust the system realization is too high if we misunderstand the expectations/ requirements.
2. SE is static. It is worried about the realization since new Intents must be integrated into the old static view of the system

OUR VISION:

1. AI is great at automation, Human is not great at expressing their needs/requirements
   ➔ AI works with Human back and forth to align expectations

2. The AI-native realization process can run at hyper-speed + AI is much smarter than the human requirement analyst
   ➔ AI can see much further and find confusions much faster then go back to re-align with human.

CORE CHALLENGES:

1. Speed up the alignment otherwise human will be frustrated if they must state the obvious or repeat themselves!

2. Make sure realization is grounded on "best-practices"

Conversational-Oriented Programming (COP)

- The conversations are the intellectual property and not the code
- The conversations should be archived as they are even more important than the code itself! **Code is the new binary!**
- The conversations capture intents, goals and requirements
- We can always *re-realize* the code by converting such conversations into code again as FM/LLMs get better
- Yet our SE tooling today doesn't account for this, nor do they enable ***deep conversations:***
  - ***Multi-way AI conversations***
  - ***Conversations with delay response (aka not interactive ones)***

Conversational-Oriented-Programming: The next hop in programming and IDEs

VS Code

jupyter

Classical Programming Editor

- ➢ Code-first
- ➢ Logic focused

Computational/AI Programming Editor

- ➢ Data-first
- ➢ Output focused

AI + Human Co-Programming Editor

- ➢ Chat-first, with Agents instead of plugins
- ➢ Mental-model/Intent focused

Intent Alignment Using “sticky” Theory-of-Mind Bi-directional technology

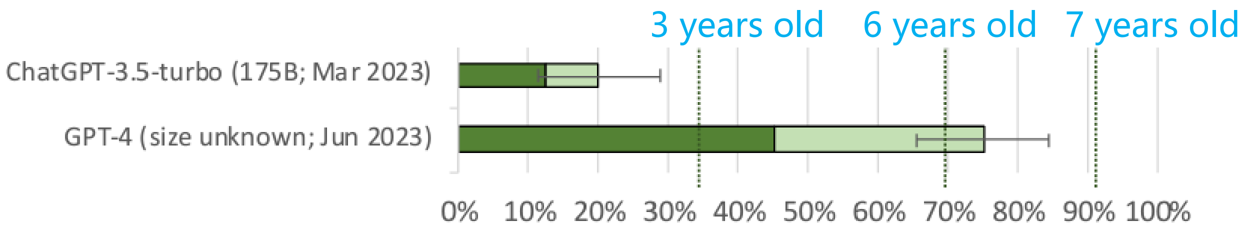
Technology: Communicate with people for multiple rounds of conversations, and properly ask and clarify specific requirements and associated contexts (such as constraints and restrictions).

Theory of Mind

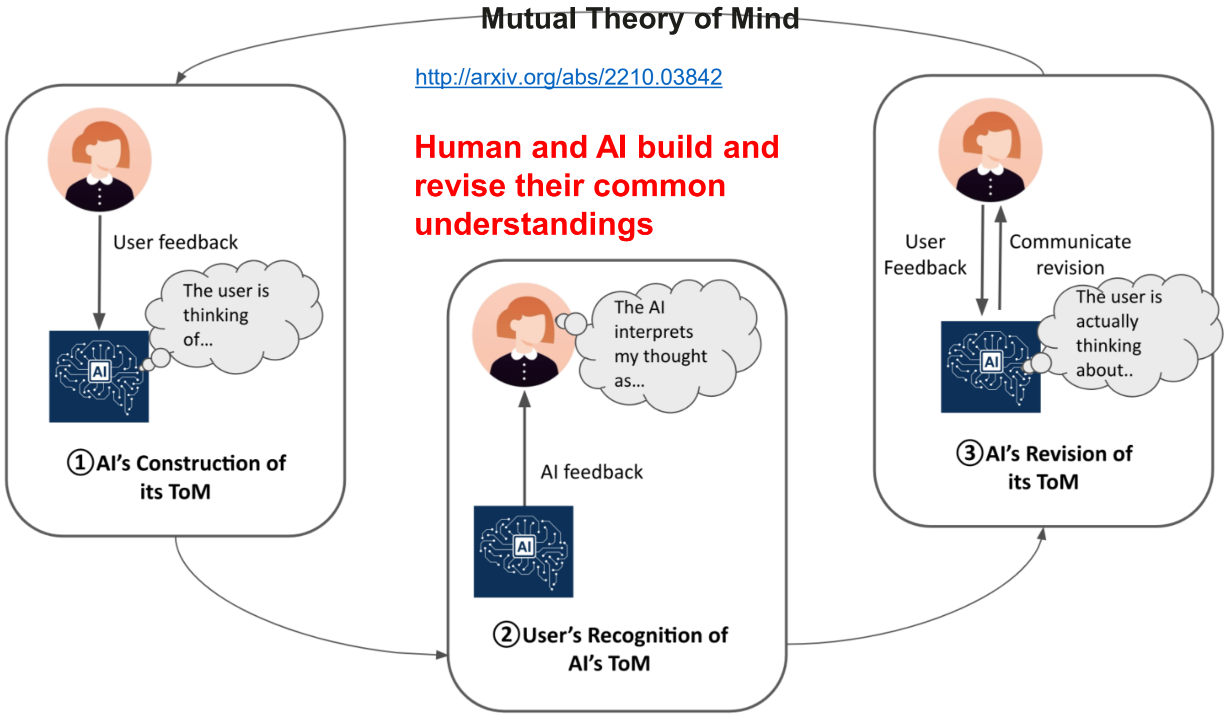
Challenge 1: Balance over-questioning with under-questioning. An agent must be able to establish the theory of mind of the person with whom it interacts, that is, to understand the other person's way of thinking and starting point.

Challenge 2: ToM must be sticky over interactions yet adjustable to the specific context

For example, when a supervisor describes a task to a senior member of the team, the wording and level of detail are different than when describing a task to a new employee and varies over time!

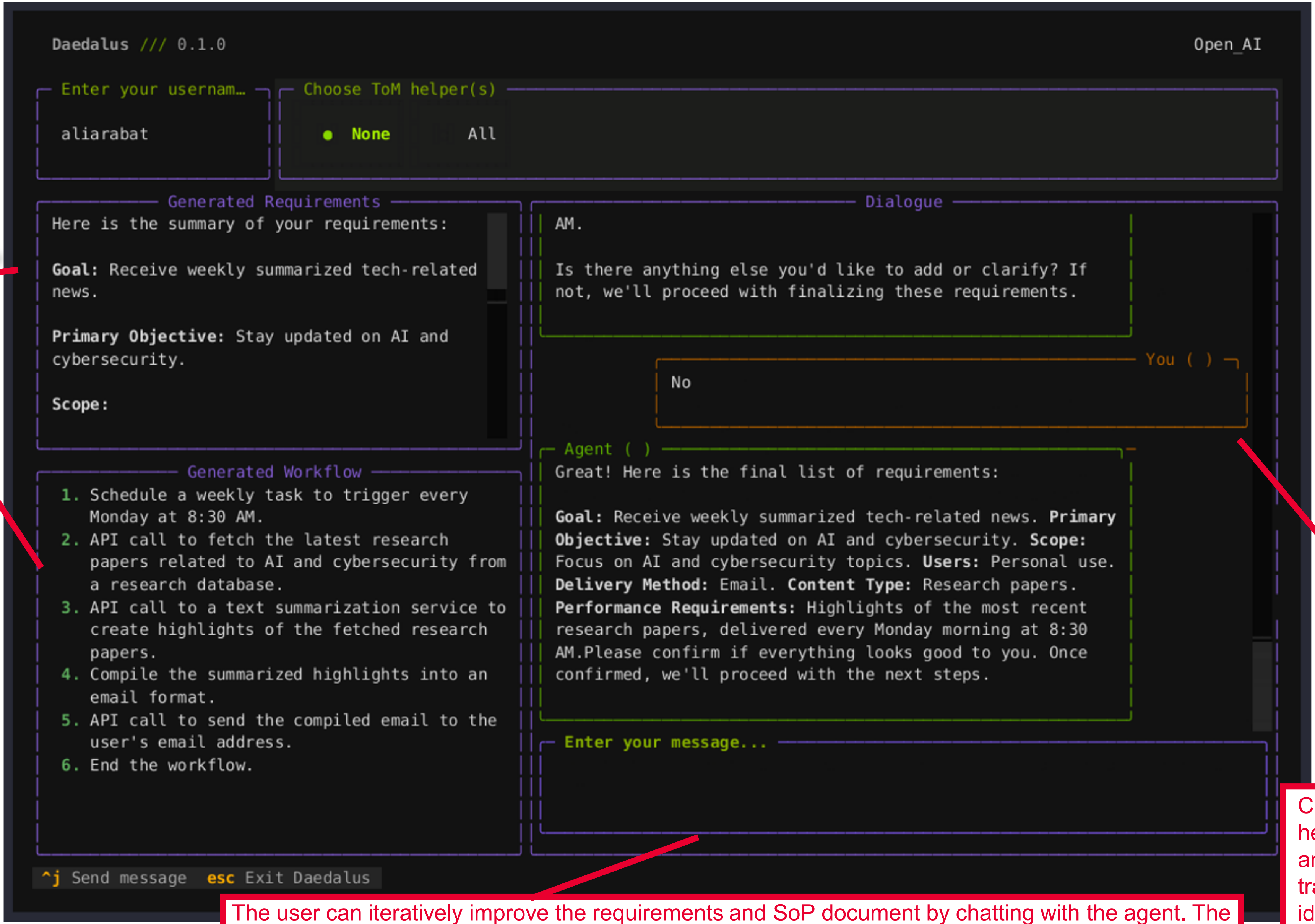


Feasibility Analysis: Large Model Theory of Mind Assessment
GPT-4 emerges from 0-shot to 6 years of age, feasible and has room for improvement
(Stanford 2024.2)



Leverage linguistic clues from users' questions. (e.g. readability, emotion, richness of wording, adaptability to agent answers, etc.) to understand the user's perception of the agent. (anthropomorphic, intelligent, favorable, etc.).

https://dl.acm.org/doi/10.1145/3411764.3445645



The agent UI displays a draft version of requirements and SoP as soon as it has sufficient information to do so, in order to get feedback.

Conversation history helps both the human and GA Agent to keep track of context and identify if something is missing

The user can iteratively improve the requirements and SoP document by chatting with the agent. The user doesn't have to explicitly say their request is for SoP or requirements refinement.

Conversation-Oriented Programming IDE Demo

(Powered by Theory-of-Mind Intent-Alignment Technology)

Support for Cognition Debugging Coming in 2025!
(Resilient to Observer Effect)

The agent UI displays a draft version of requirements and SoP as soon as it has sufficient information to do so, in order to get feedback.

Conversation history helps both the human and GA Agent to keep track of context and identify if something is missing

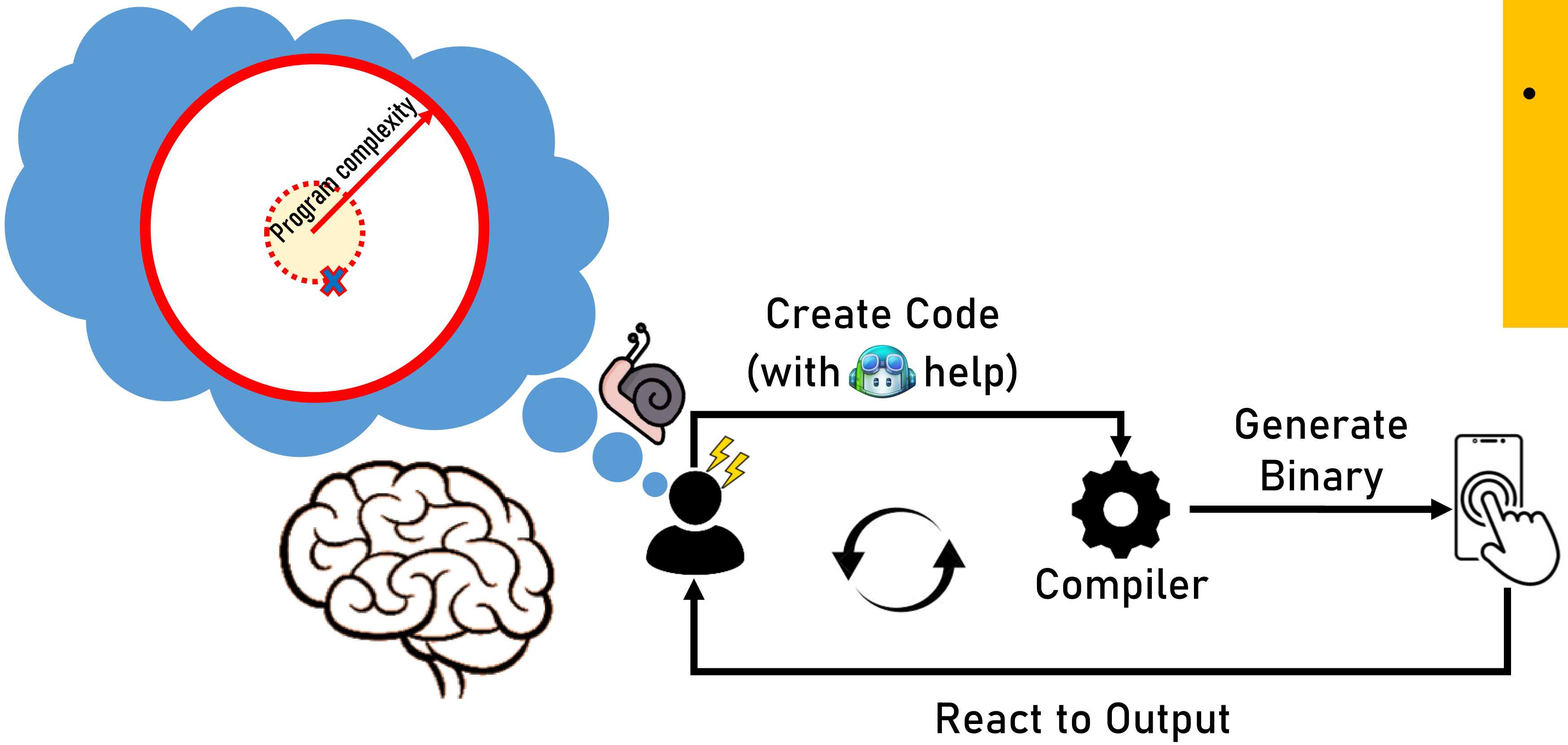
The user can iteratively improve the requirements and SoP document by chatting with the agent. The user doesn't have to explicitly say their request is for SoP or requirements refinement.

Conversation-Oriented Programming IDE Demo

(Powered by Theory-of-Mind Intent-Alignment Technology)

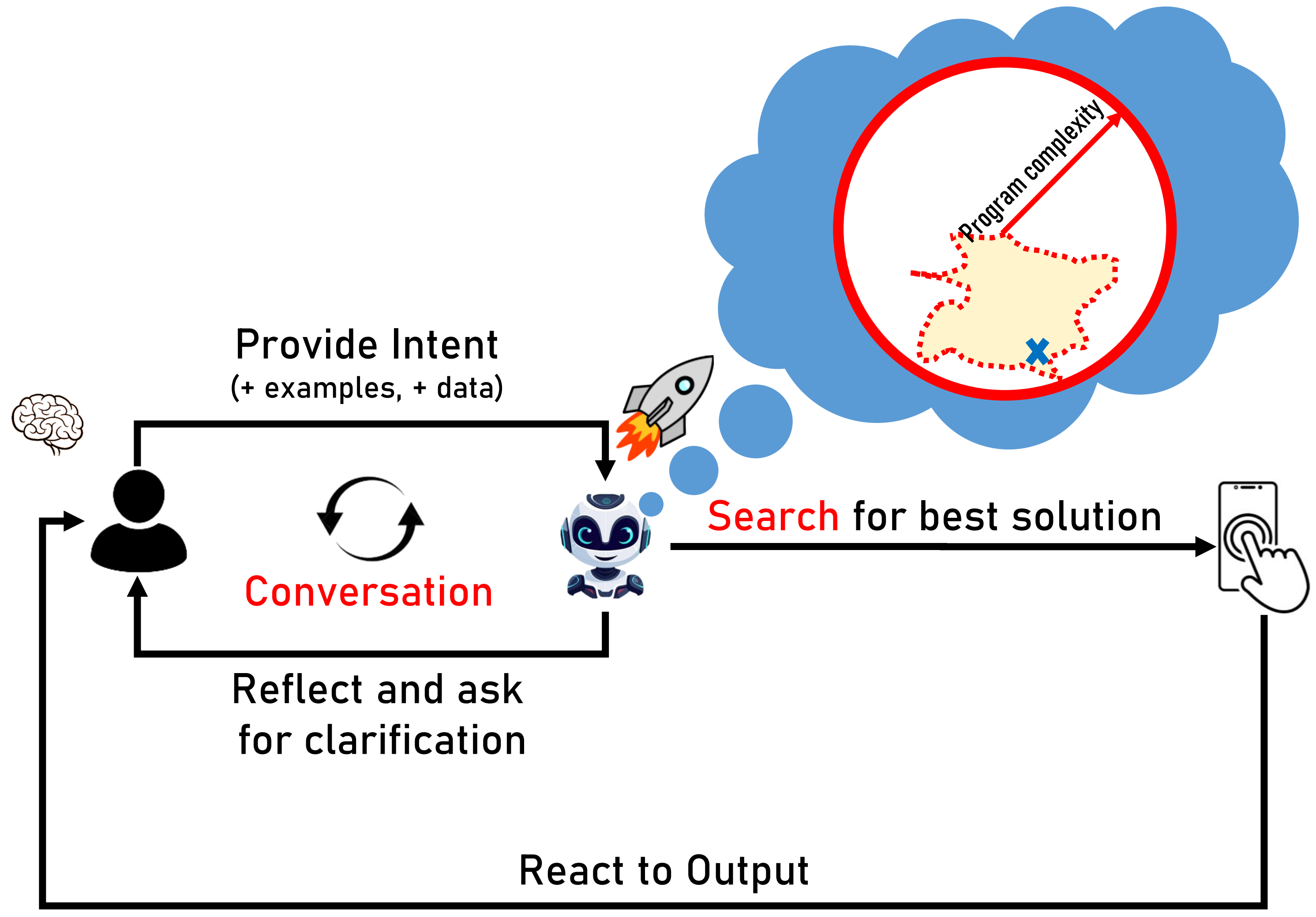
Programming 2.0 [AI Assisted]
(Write Code with AI help then Compile)

- Human drives the code creation process loop
- The code complexity is limited by human's ability to express *and maintain* a complex problem in code



Programming 3.0 [AI-Native]

(Define Search Space, Give Data, AI Searches to create code)



- Human and AI align on goals
- AI drives the code creation loop
- The code complexity is unlimited as the AI synthesizer searches for the "software solution" that maximizes a fitness function (aka Human-AI aligned goals).
- Realization done using AIware or Codeware
- The code creation process can be re-initiated at any time as long as we "archive" the human and AI conversations

Teammate.next

Personalized AI partners

IDE.next

(Developing, Debugging, Maintaining)

Intent-First + Conversational

Compiler.next

Code realization through synthesis and search

Runtime.next

SLA-aware Uni-Cluster Runtime with Edge Extension

FM.next

Curriculum engineered models

We are re-thinking the Software & SE Stacks

Fusion (FMware runtime) enables e2e high efficiency SLA-aware serving of FM\LLMware (30% improvement in SLA and latency over default Ray Serve)

ASIS: Existing model serving systems

- Model centric
- Rely on infra for no-context elasticity
- No intentions for scheduling

TOBE: SLA-aware FMware workflow Serving System (Fusion)

- e2e perf. optimization
- Routing based on priorities
- Allocation based on intelligent predictions

Core Capabilities: 2-layer scheduling

Request Router

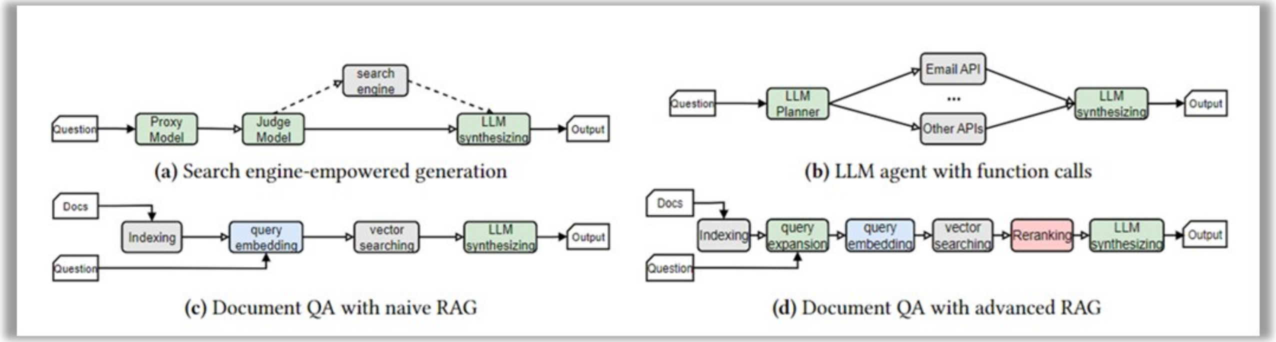
Determines the model replica to which the current model request is routed.

SOTA model acceleration engine

Accelerate model inference

Resource Provisioner

Decide how many replicas each model type should have



Workflow-Level Expression Model

Workflow A	Workflow B	Workflow C
Model hot Deployment	Profile Metadata	Customized Model Registration
Replica Router	Runtime Metric Collectors	Resource Provisioner

Ray Cluster

Node 1

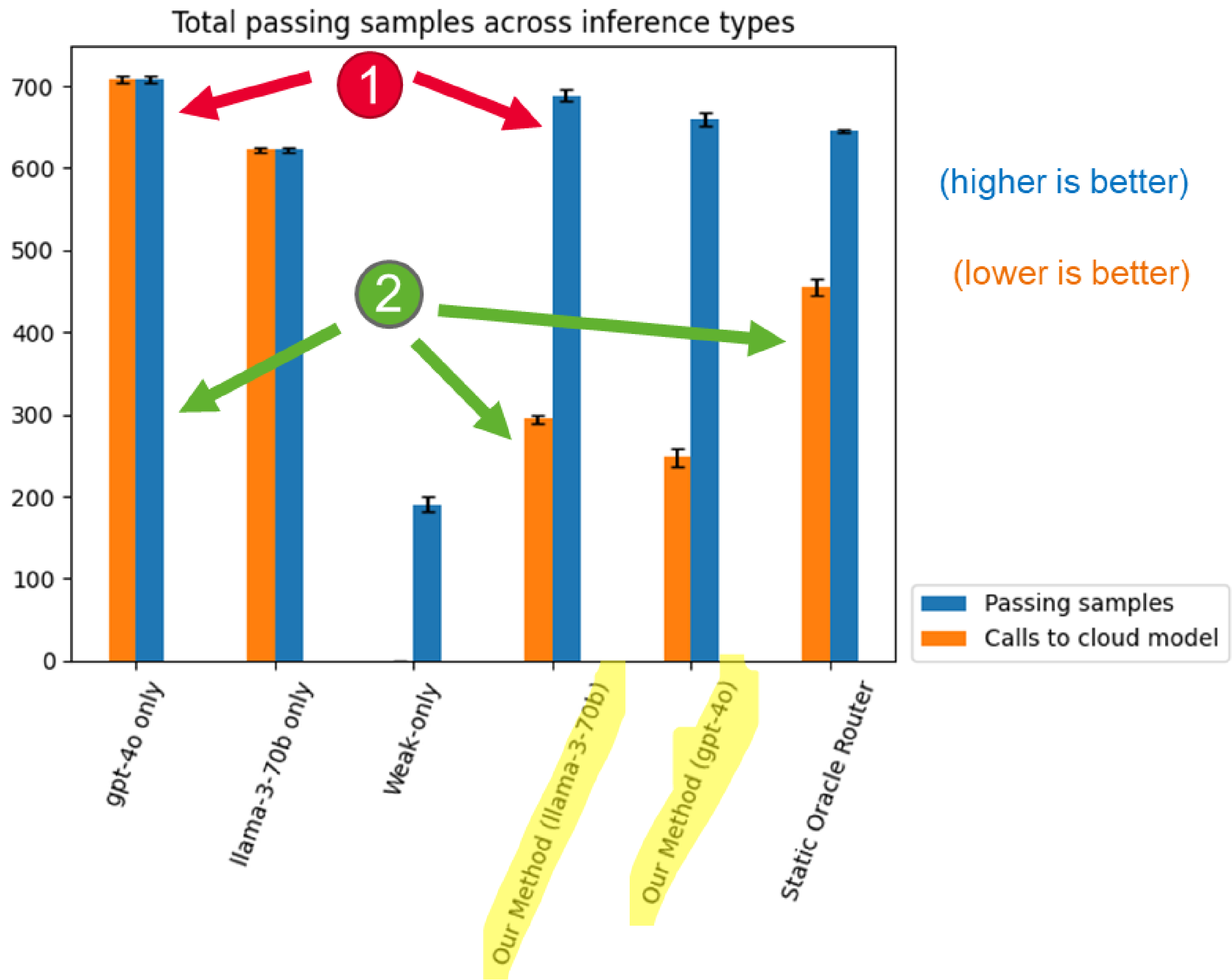
vllm (Model A Replica 1)		vllm (Model A Replica 2)		vllm (Model B Replica 1)	
npu:0	npu:1	npu:2	npu:3	npu:4 / npu:5	npu:6 / npu:7

Node N

vllm (Model B Replica 2)		vllm (Model C Replica 1)	vllm (Model C Replica 2)	
npu:0 / npu:1	npu:2 / npu:3	npu:4	npu:5	npu:6	npu:7

Hassan et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Layered Personalization XPU



1. Shows *nearly identical* model capability as the *cloud model* **(~96% correctly solved tasks)**, and solves more samples than *static routing* by **~1-2%**
2. Makes **~58% less** use of cloud model compared to *cloud model-only inference*, and **~34% less** compared to *static routing*

Hassan et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Teammate.next

Personalized AI partners

IDE.next

(Developing, Debugging, Maintaining)

Intent-First + Conversational

Compiler.next

Code realization through synthesis and search

Runtime.next

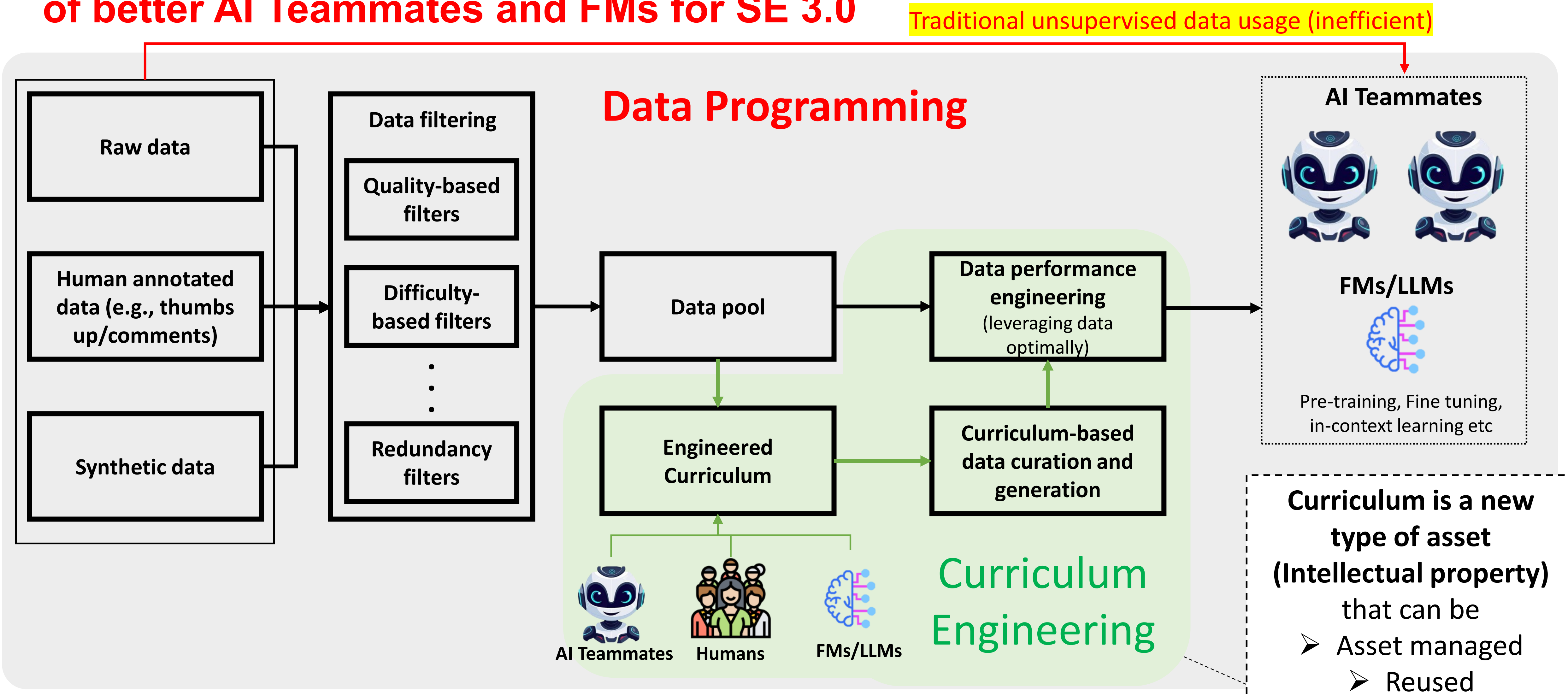
SLA-aware Uni-Cluster Runtime with Edge Extension

FM.next

Curriculum engineered models

We are re-thinking the Software & SE Stacks

Data programming and engineered curriculums enable the efficient creation of better AI Teammates and FMs for SE 3.0



Curriculum is a new type of asset (Intellectual property) that can be

- Asset managed
- Reused
- …

Using Data as-is is a very inefficient, costly, and ad-hoc mean to teach our AI Teammates for SE. Instead, data programming and curriculum engineering are more effective and efficient ways of teaching our AI Teammates

Overview of the session

- ❑ Rethinking Software Engineering
  - ❑ Software eras
  - ❑ Software engineering eras
  - ❑ Towards AI-native SE: our vision and technology stack
- ❑ **Where are we today?**
  - ❑ **AI4SE:** A very shallow perspective on the ROI of AI4SE
  - ❑ **SE4AI:** Too focused on AI models and prompting instead of AI systems
- ❑ Conclusion

How to measure the value – especially e2e-value not just instance-value?!

E2E-Value of an Intervention/Technology

↑

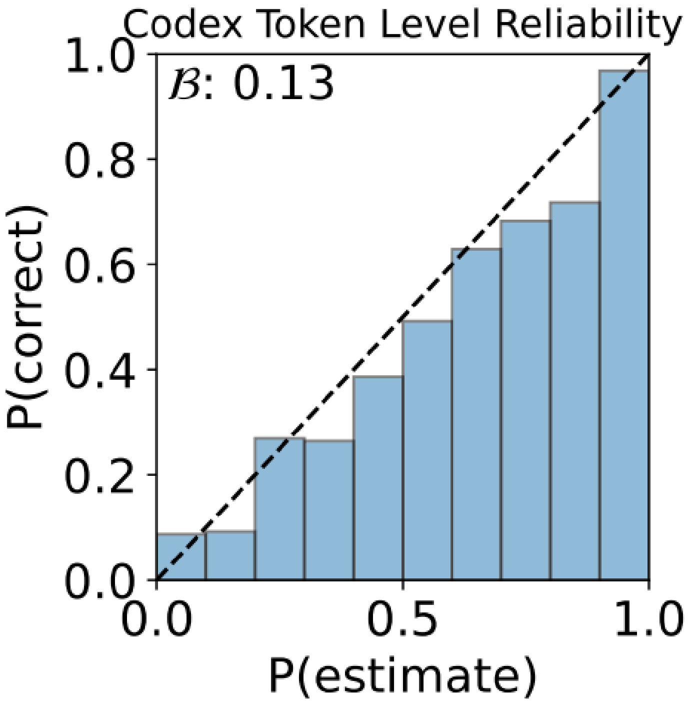
Instance-Value of an Intervention/Technology (e.g., how much it helped for its specific situation)

A technology/intervention might help speed up coding, but then lead to
- Lower code health
- More bugs
- Negatively impact ability to release features faster enough
- Lost revenue $$

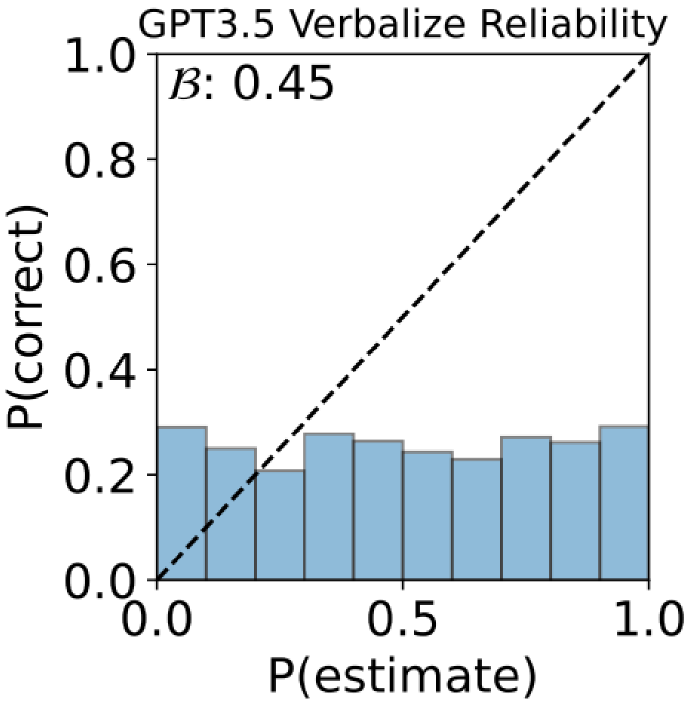
E.g., how to quantify ROI of code-reviewing or AI4SE?

Let us examine the Instance-Value of AI-coding

AI Code Completion: How much should you trust GPT's confidence in its answer?!



(a) Well Calibrated



(b) Poorly Calibrated

Weather Models need a minimum Brier of 0.05 to be considered for deployment (Brier of 0 is perfect prediction)

C. Spiess et al., "Calibration and Correctness of Language Models for Code," ICSE 2025

A frank assessment of the reported ROI metrics of AI-coding: The realities of HumanEval

HumanEval is a benchmark dataset by OpenAI, to evaluate a model's ability to solve coding problems. It consists of 164 human-written ***leetcode-like unrealistic*** programming problems, each with a function description, code body, and unit tests in python.

The goal is to assess the model's understanding of natural language, algorithms, and basic math by making it generate code that fulfills the problem description and passes the tests.

HumanEval Insights: *Simple prompt* experiment setup

- Simplest prompt (*baseline case*) **Pass@1 = 0.84146**
- Context enhancement:
  - Simplest prompt + code hints (e.g. ensure all imports, coding tips) **Pass@1 = 0.9136**
  - Simplest prompt + unit tests & expected outcomes **Pass@1 = 0.9207**
- Reasoning enhancement:
  - Simplest prompt + think 'step by step' (aka 0-shot-CoT) **Pass@1 = 0.9329**
- Trying it all together:
  - Simplest prompt + think 'step by step' + unit tests + code hints **Pass@1 = 0.9268**

** simplest prompt = single call to LLM inference*

Bigger context is not necessarily better

❖ Very sensitive to prompting approach (~25% improvement by just prompt hacking!)

```python
response = client.chat.completions.create(
        model="gpt-4-0125-preview",
        messages=[
            {"role": "system", "content": "You are a helpful AI
            assitant. Answer questions to your best ability. If you
            don't know the answer, simply state that you do not know."},
            {"role": "user", "content": "Can you give me some examples
            from HumanEval dataset please?"},
        ]
)
```

✓ 26.1s Python

```
Certainly! The HumanEval dataset, also commonly referred to as the Codex
dataset, consists of Python programming problems designed to evaluate code
generation models. These problems include a wide range of tasks, from string
manipulation to more complex algorithmic challenges. Each problem in the
dataset is accompanied by a function signature, a docstring explaining what
the function should do, and several test cases.

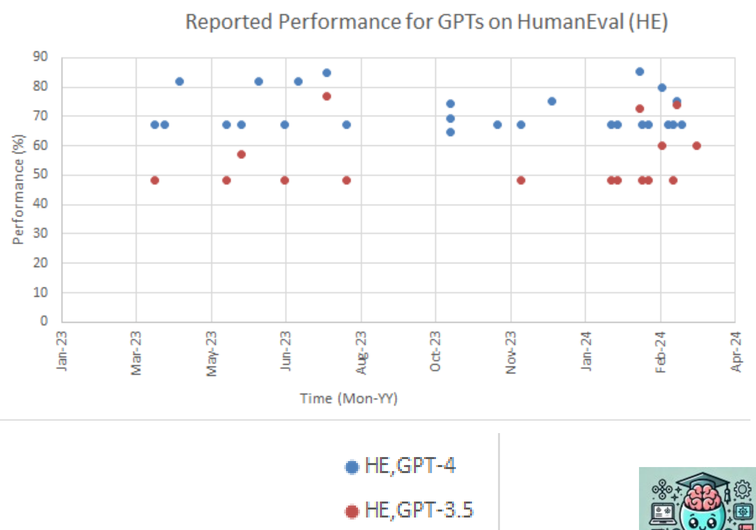
Here are a few example problems from the HumanEval dataset:

### Example 1: Fibonacci Sequence
**Problem:** Write a function to calculate the nth Fibonacci number.

```python
def fibonacci(n: int) -> int:
    """Given n, return the nth Fibonacci number. For example, fibonacci(0) =
0, fibonacci(1) = 1, fibonacci(2) = 1, fibonacci(3) = 2."""
    pass
```

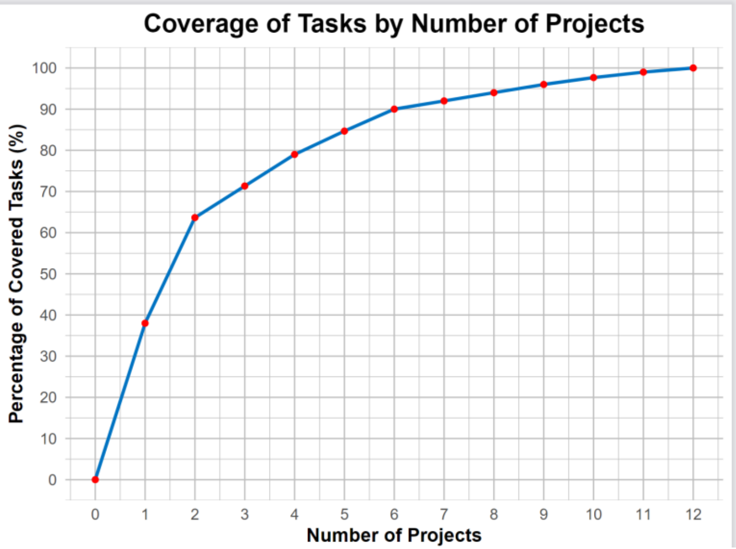
Example 2: Check if Palindrome
Problem: Write a function to determine if a given string is a palindrome.
```

❖ Already contaminated in most FMs today .. ChatGPT has the examples in its training data



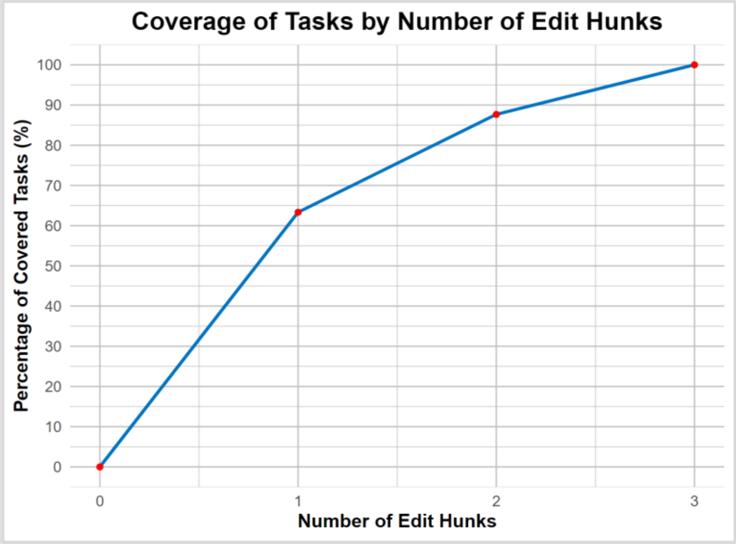
A frank assessment of the reported ROI metrics of AI-coding: The realities of the SWEBench

SWEbench is a real-life projects dataset that is designed to evaluate a system's capability to automatically resolve GitHub-level issues.

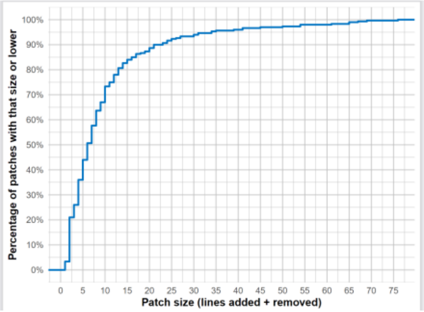


- ❖ 65% of SWEbench lite comes from just 2 projects:
  - ❖ Sympy: a python-based symbolic execution framework
  - ❖ Django: a python-based web-creation framework

Researchers use this dataset to see if new advancements in FMs can be applied to real-world SE tasks



- ❖ 100% of SWEbench lite issues impact a SINGLE file ONLY
- ❖ 2/3 of SWEbench lite issues just changed ONE area in that ONE file
- ❖ 90% of issues changed <20 LOC



% of accepted suggestions

a common industry metric fails to consider many factors

[We can achieve 100% if one simply suggests the ";" at the end of each line of Java code]

30% of keystrokes (attempt opportunities) are in middle of an already written-code line! Many AI4coding systems will not attempt a completion in such a setting, as chances of success today are low

The latency of the AI4coding system is crucial; speed of model inference & speed of context creation matter considerably

The classical % of accepted suggestions

The size of the suggestion – one can take a large suggestion and split it into smaller suggestions (some time even desired – Meta reports developer preferring shorter suggestions otherwise they mentally move from coding to code reviewing)

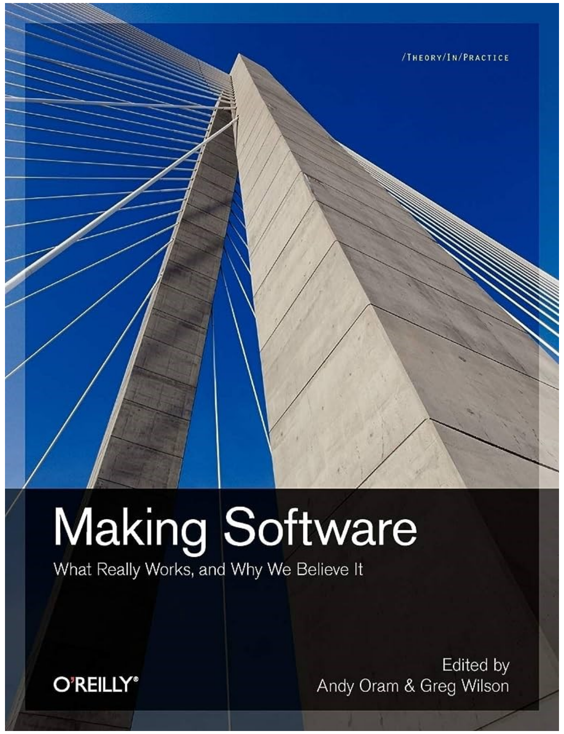
Characters per Opportunity =
% of times AI attempted *
% of times an AI-attempt reaches a developer in time *
% of reached-attempts that are accepted *
Size of attempts

An accepted attempt does not mean correct code, so we must consider how much of that code was eventually *integrated as-is* vs was *heavily modified*!

The Golden Metrics: Characters per Opportunity and Percentage Code Written:
https://codeium.com/blog/golden-metrics-characters-per-opportunity-percentage-code-written

Developer Productivity varies widely across individuals & activities (aka change-requests)

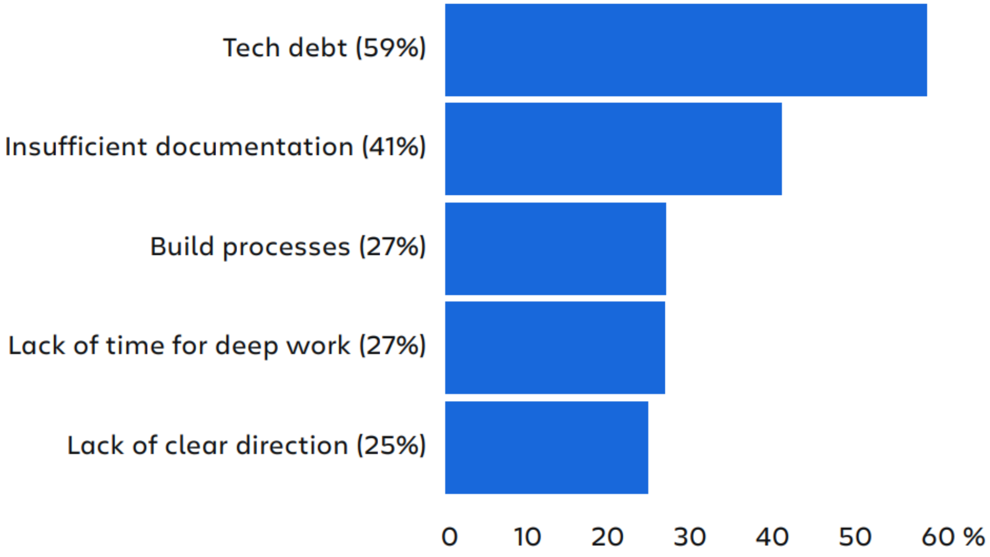
- As early as 1968, Sackman, Erickson and Grant show as much as 28 X difference in productivity (highest is for debugging activities)
- The 10 X difference in productivity between developers is well documented [see Chapter in Making Software by Steve McConnell, author of the Code Complete series]



Somali runner sets record for 'slowest ever' 100m after taking over 20 seconds to complete:
https://www.youtube.com/watch?v=scMs2HSsFqg

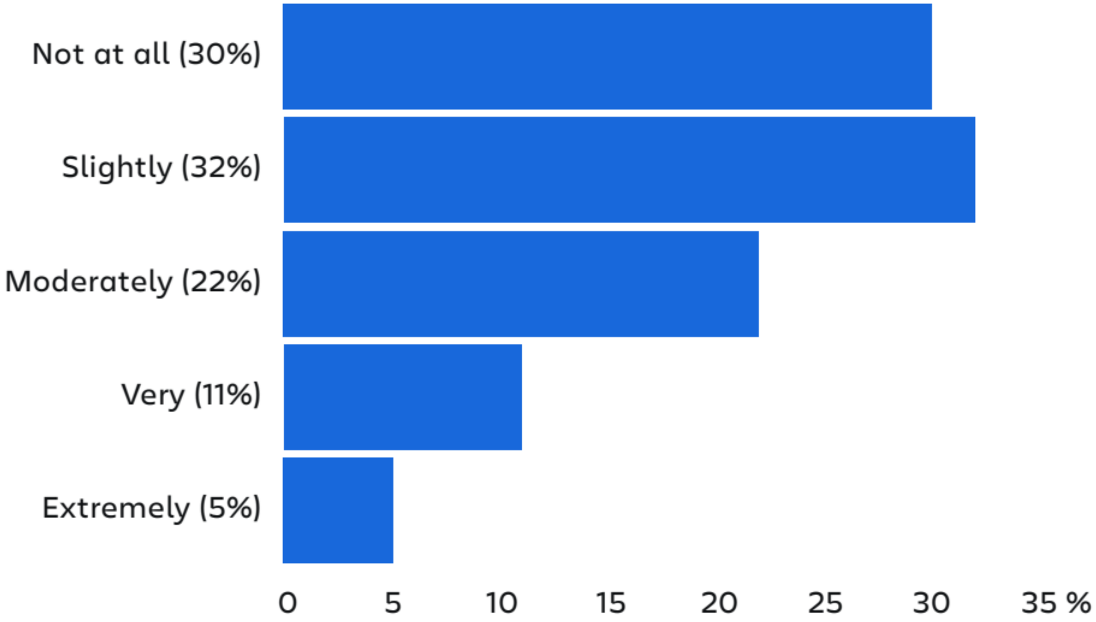
The value of AI on Developer Productivity varies between leaders and developers

Top 5 areas of developer-time loss according to developers



How much are AI tools improving your productivity as a developer today?

62% of developers report slight to no productivity improvement from AI-based dev tools, **despite leadership's belief that using AI is the one of the most effective ways to improve productivity**



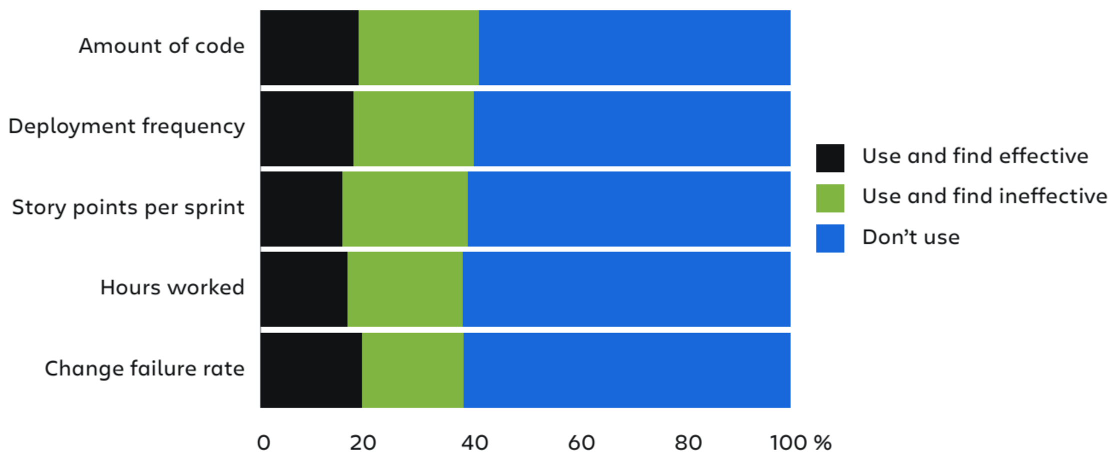
Developer Productivity is quite hard to quantify

- One thing is clear – **NO PROJECT EVER FAILED because developers could not code fast enough!!**
- Short term improvement leading to higher technical debt (and higher long term costs)
- Less than 50% believe in the effectiveness of how they measure productivity
- The only thing we know today is that a happy developer is one that perceives that they are productive [MSFT/GitHub/ Atlassian Global surveys/…]

Meanwhile, there's a significant body of research showing that happy workers are productive workers. So instead of worrying about what makes developers more efficient, what if we think about what makes them happy? Not the surface-level happiness you can buy with high-dollar swag and on-tap kombucha, but the deep sense of fulfillment that comes from creating something great.

RAJEEV RAJAN Chief Technology Officer at Atlassian

Top 5 ways organizations measure productivity and the effectiveness of such measures



Based on two surveys of 2K+ developers across the world commissioned by Atlassian in Feb 2024: 1) Wakefield Research surveyed 1,250 engineering leaders in the US, Germany, France, and Australia. 2) DX surveyed 900 developers around the world, including the US, Germany, France, and Australia.* *DX also surveyed developers in the UK, Sweden, Lithuania, Estonia, Spain, Ireland, Ukraine, Denmark, Switzerland, the Czech Republic, Canada, Brazil, and India.

Volvo's Mission to Software Excellence:
AI4SE is more than just saving keystrokes!

***tricky to keep focus**.. Lots of context switching..*

More time to be creative

*Wow, it **does all the boring** stuff for me..*

*It **teaches me** stuff, I learn how to..*

*It suggests things I **would not come up with myself***

*I am **learning more** when using it*

Signals we are getting are positive!

Volvo Cars' Software Evolution: https://www.youtube.com/watch?v=V88IVD2i4E0

The time respondents save with AI is being used to collaborate, learn, and **design systems**

TASKS RESPONDENTS ARE ABLE TO SPEND MORE TIME DOING AFTER USING AI CODING TOOLS

	USA N=497	BRAZIL N=495	GERMANY N=485	INDIA N=496
COLLABORATING WITH TEAM MEMBERS ON PROJECTS	47%	46%	47%	40%
DESIGNING SYSTEMS & CUSTOMER SOLUTIONS	47%	42%	47%	40%
LEARNING & DEVELOPMENT	46%	47%	43%	44%
RESEARCHING & EXPERIMENTING WITH EMERGING TECHNOLOGY	46%	44%	45%	44%
CODE REVIEW	42%	45%	39%	43%
REFACTORING & OPTIMIZING CODE	37%	42%	43%	37%
TAKING BREAKS	34%	29%	31%	33%

Which of the following are you able to spend more time doing due to using AI coding tools at work? Please select all that apply.

Note: Underlined data indicates the highest common survey responses.

GitHub Survey 2024: https://github.blog/news-insights/research/survey-ai-wave-grows/

Let us dig deeper into AI-coding through extended user studies

- **Long term use quantification: (intuition if developers continue using it then they must see value of it.. Contrast to smoking – most smokers will continue smoking!]**
  - ~5% of developers use AI-coding and around 63% of them use it as just codecomplete++ [Sourcegraph developer of Cody AI-coding tool, June 2024]

- **~20% not 10X acceleration of productivity (aka doing tasks faster) [18% features, 26% bug-fixes]**
  - 75% less code for features but 71% more code for bug-fixes [due to ease of adding more test cases] – *if you use LOC for productivity it will give conflicting signals*
  - 17% of ***code "tokens"*** are coming from copilot

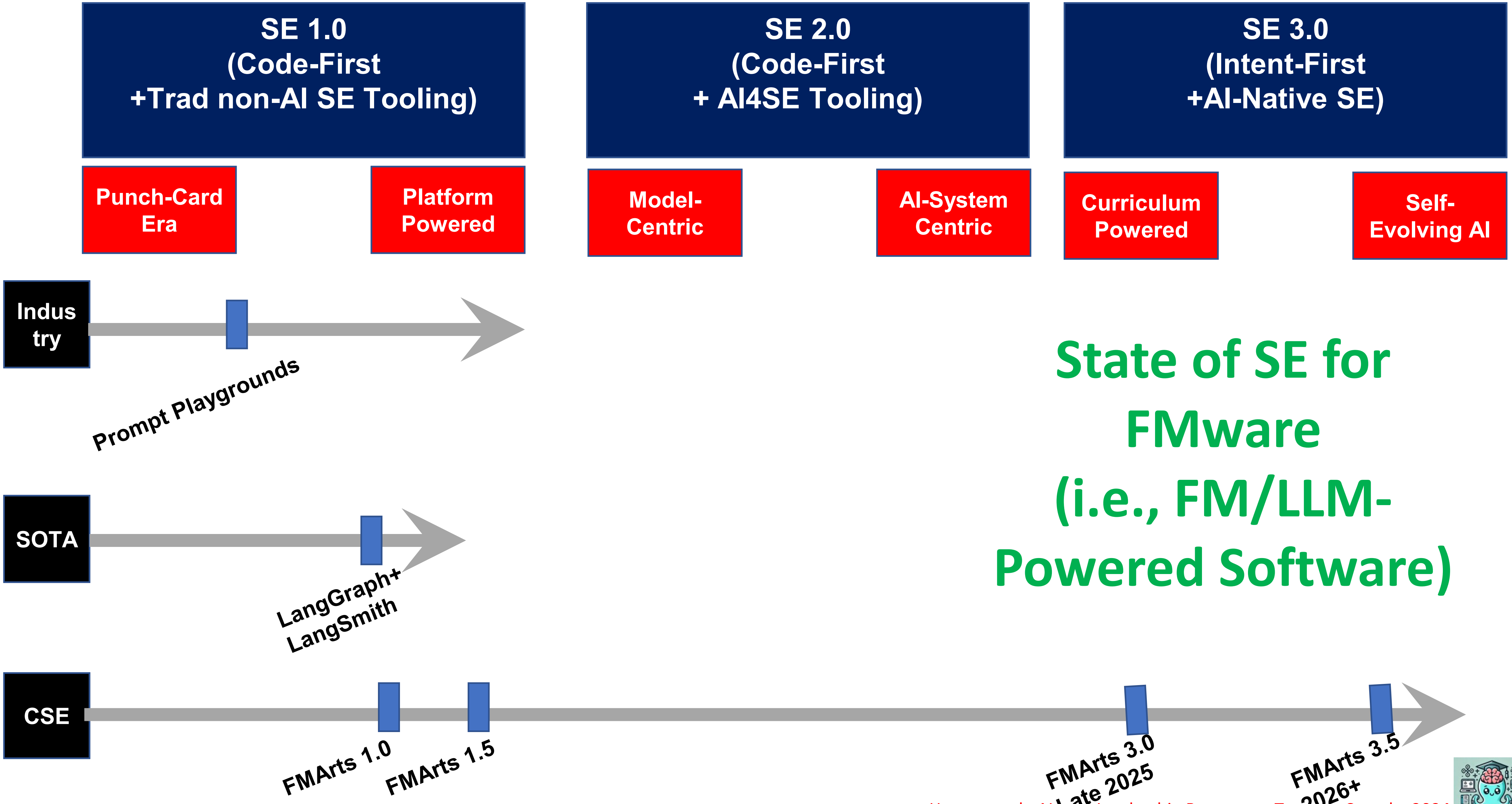
- **The impact on code health is not promising**
  - 5X increase in chances to introduce cloned code!
  - 25X increase in chances to introduce vulnerabilities!
  - **Do note: we did not quantify whether it introduced more bugs – we can fix bugs faster though!**

Key take home:

- It is really hard to quantify ROI

- Just measure developer happiness!

It is not advisable to run an SE organization based on a happiness measure!



State of SE for FMware (i.e., FM/LLM-Powered Software)

Hassan et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

SE 1.0 (Code-First +Trad non-AI SE Tooling)

SE 2.0 (Code-First + AI4SE Tooling)

SE 3.0 (Intent-First +AI-Native SE)

Punch-Card Era

Self-Evolving AI

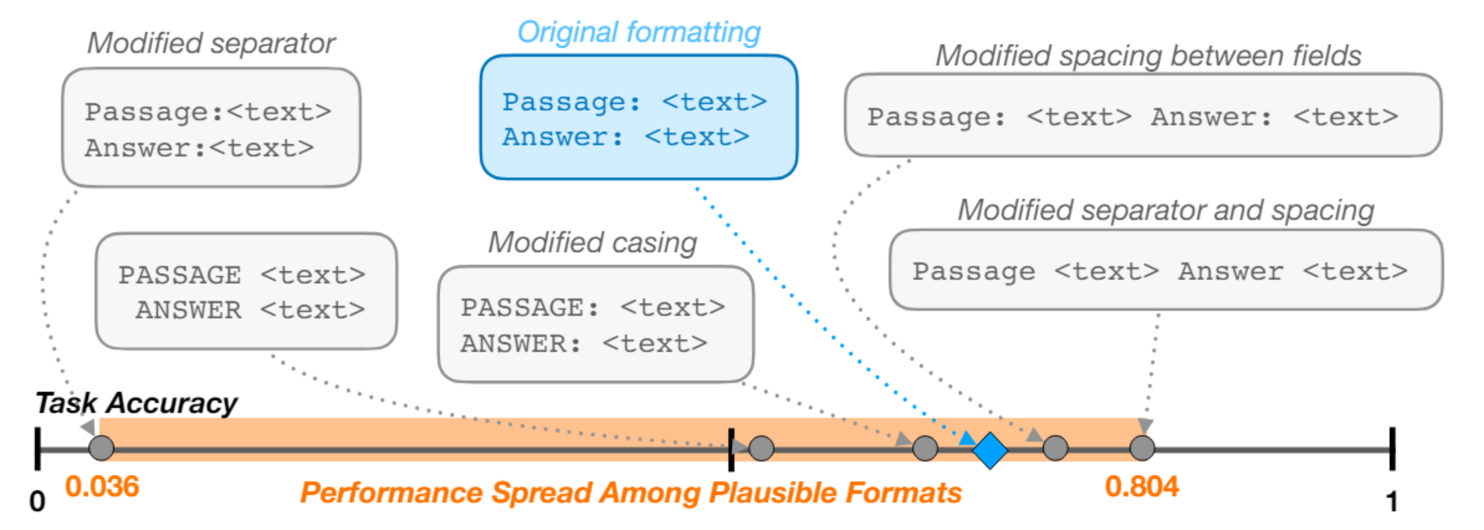
Industry

Prompt Editors

SOTA

CSE

Why are we skipping FMArts 2.0?
Prompting is too unstable and not portable



"How I learned to start worrying about prompt formatting", Sclar et al.

...of SE for ...ware ...M/LLM- ...Software)

FMArts 1.0

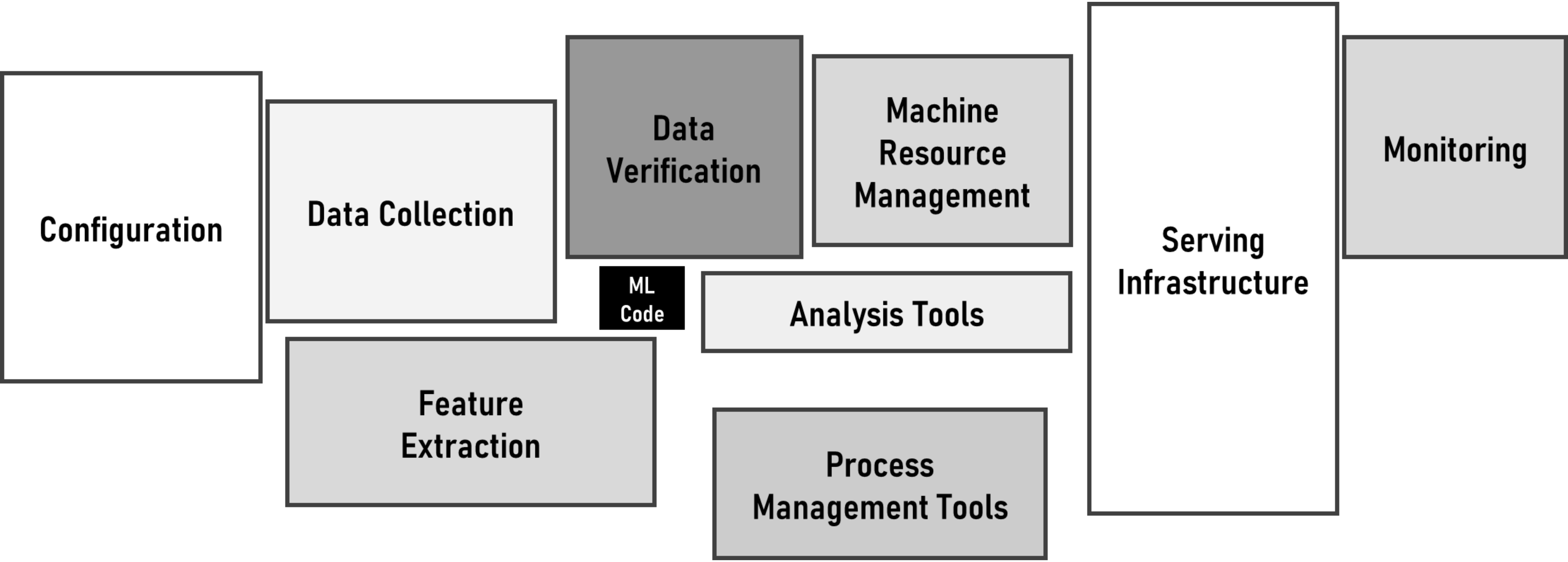
FMArts 1.5

FMArts 3.0 Late 2025

FMArts 3.5 2026+

Over-focusing on the model itself is a repetition of traditional AI engineering pitfalls

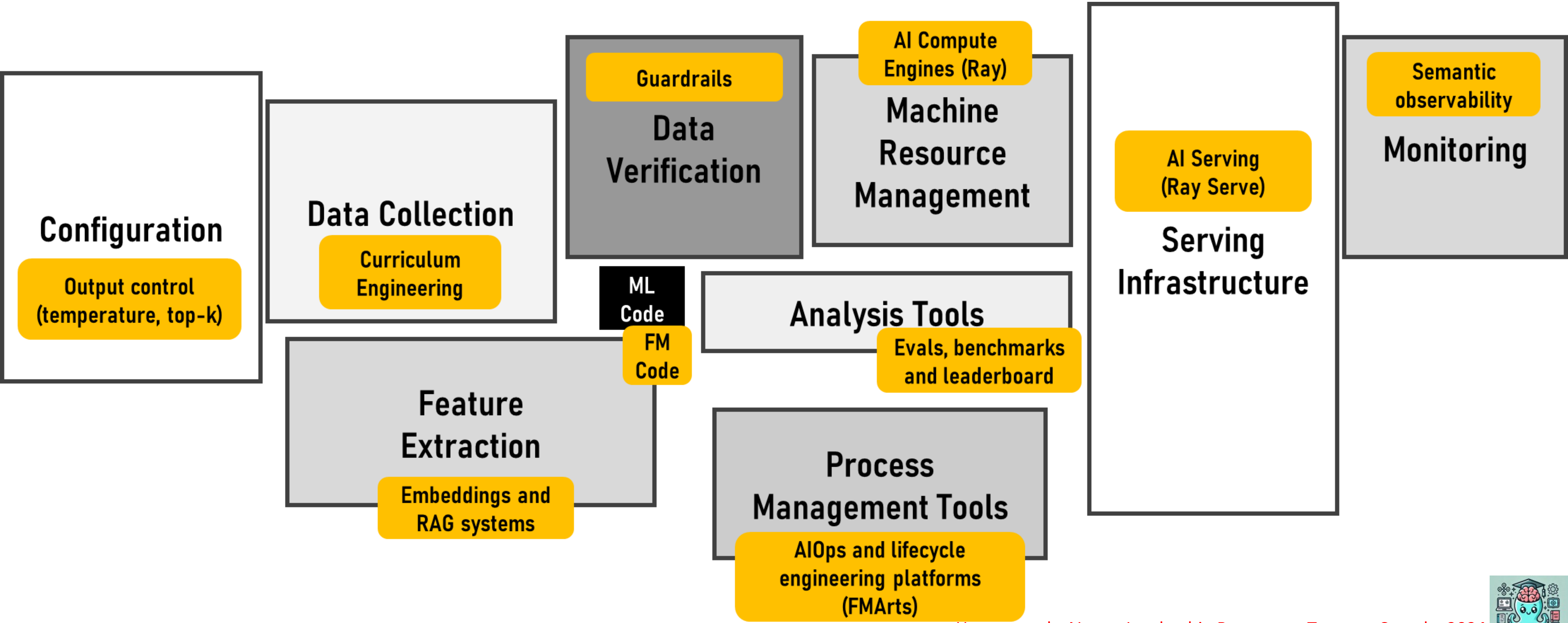
"Models are only a small part of AI systems"



D. Sculley et al., Hidden technical debt in Machine learning systems, NIPS'15, MIT Press

Over-focusing on the model itself is a repetition of traditional AI engineering pitfalls

"Models are only a small part of AI systems"



Hassan et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Overview of the session

- ❑ **Rethinking Software Engineering**
  - ❑ Software eras
  - ❑ Software engineering eras
  - ❑ Towards AI-native SE: our vision and technology stack
- ❑ **Where are we today?**
  - ❑ **AI4SE:** A very shallow perspective on the ROI of AI4SE
  - ❑ **SE4AI:** Too focused on AI models and prompting instead of AI systems
- ❑ **Conclusion**

Key Take homes

- Thinking of model inference and LARGER models is the same as thinking of FLOPS, they mean nothing for value ➔ NEED an AI system perspective and e2e value perspective

- Must re-think SE and Software to be AI-native, coding was never the problem instead it is the recognition and realization of Intents

- Prompt ~~Engineering~~ Hacking MUST die ☺

Teammate.next

Personalized AI partners

IDE.next

(Developing, Debugging, Maintaining)

Intent-First + Conversational

Compiler.next

Code realization through synthesis and search

Runtime.next

SLA-aware Uni-Cluster Runtime with Edge Extension

FM.next

Curriculum engineered models

Hassan et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

	SE 2.0	SE 3.0
Teammate.next Personalized AI partners	• Static • Impersonal	• Self-Evolving • Personalized Mentor
IDE.next (Developing, Debugging, Maintaining) Intent-First + Conversational	• Code-Centric • Editing	• Intent-Centric • Conversations
Compiler.next Code realization through synthesis and search	• Logic-Rule Realization	• Search-Space Exploration
Runtime.next SLA-aware Uni-Cluster Runtime with Edge Extension	• Serving Models	• Serving Compound Apps (AIware)
FM.next Curriculum engineered models	• Data-driven Inefficient FMs	• Knowledge-driven Efficient FMs