Automation and Agentic AI

CS453 Automated Software Engineering

Shin Yoo | COINSE@KAIST

What is an LLM?

Let’s go back to 2012

Hindle et al., ICSE 2012

- One of my favourite papers: On Naturalness of Software (https://dl.acm.org/doi/10.5555/2337223.2337322)
- “Programming languages, in theory, are complex, flexible and powerful, but the programs that real people actually write are mostly simple and rather repetitive, and thus they have usefully predictable statistical properties that can be captured in statistical language models and leveraged for software engineering tasks.”
- But what is “naturalness”?

John: Hi, nice to meet you. How are you?

Mary: I’m ____, _____ ___. ___ ___?

a) fine, thank you. And you?

b) okay, I guess. But why?

What about code?

- It is not “natural”, in the sense that we have artificially created the grammar for programming languages.
  - Programming languages do evolve, but how?
    - Intentionally? New grammars, language consortiums, etc…
    - Gradually? Languages do affect each other, a newer and more popular style eventually gets accepted, etc…

Python: for _ __ _____ …

a) i in range

b) (int i = 0;

Java: for _ ___ _ _ _ _ …

a) i in range

b) (int i = 0;

Statistical Language Model

- Given a set of tokens, $\mathcal{T}$, a set of possible utterances, $\mathcal{T}^*$, and a set of actual utterances, $\mathcal{S} \subset \mathcal{T}^*$, a language model is a probability distribution p over utterances $s \in \mathcal{S}$, i.e., $\forall s \in \mathcal{S}[0 < p(s) < 1 \wedge \sum_{s \in \mathcal{S}} p(s) = 1$

- An utterance (or a sentence) is a sequence of tokens (or words). Suppose we have N tokens, $a_1, a_2, \ldots, a_N$ that consist s. What is $p(s)$?

  - $p(s) = p(a_1)p(a_2 \mid a_1)p(a_3 \mid a_1 \,.\, a_2)p(a_4 \mid a_1, a_2, a_3)\ldots p(a_N \mid a_1 \ldots a_{N-1})$

  - But these conditional probabilities are hard to calculate: the only feasible approach would be count each utterance that qualifies, but $\mathcal{S}$ is too big, let alone $\mathcal{T}^*$.

Large Language Model

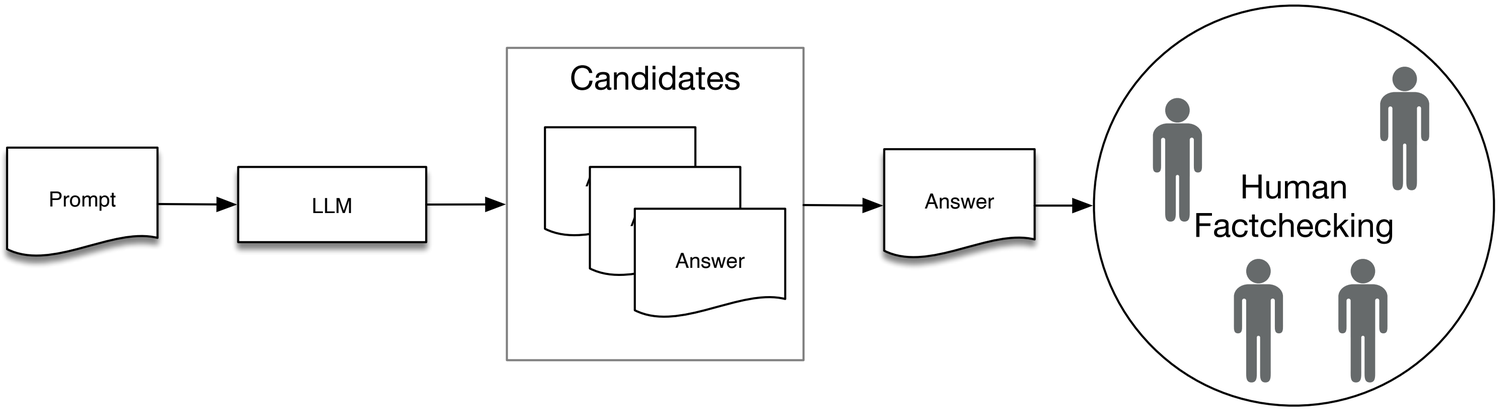
(really, a very large and powerful statistical language model)

- Mainly Transformer-based DNNs that are trained to be an auto-regressive language model, i.e., given a sequence of tokens, it repeatedly tries to predict the next token.
- They **seem to** get the semantics of the code and **work across natural and programming language, which is unprecedented!**
- But they are inherently restricted as a statistical language model.
  - No memory/state, no background/domain knowledge…

Code is a unique artefact because it executes.

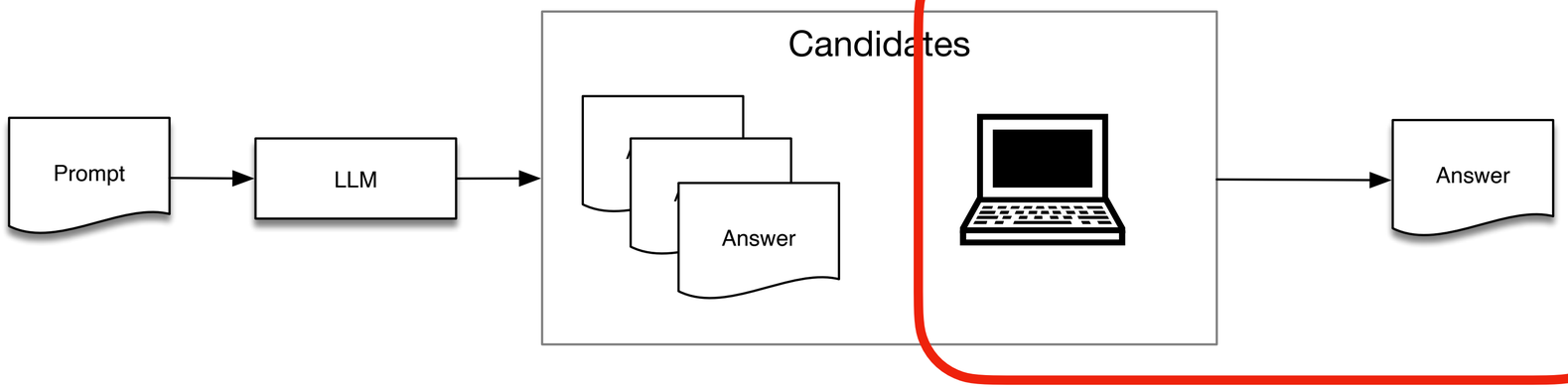
(And we've been doing dynamic analysis for a long time)

NL + LLM Pipeline



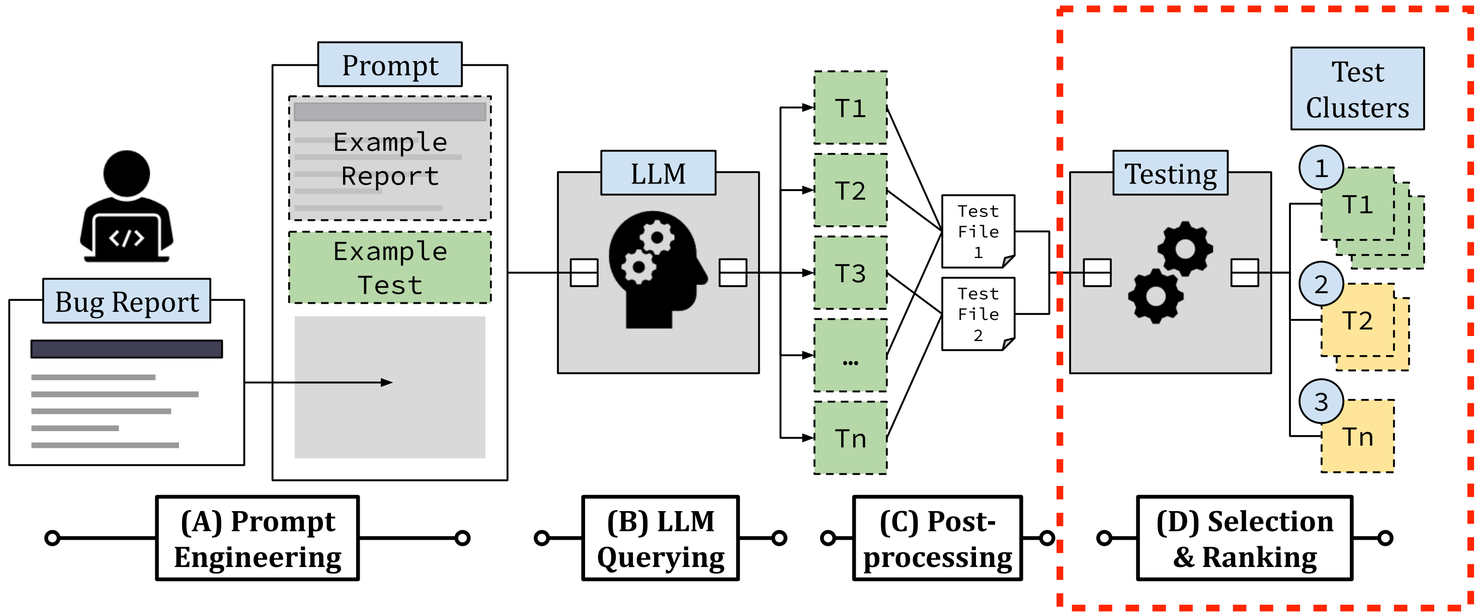
Isn't this testing? :)

PL/NL + LLM Pipeline



Execution-Based Filtering of LLM Output

LLM-based Bug Reproduction (Kang, Yoon & Yoo, ICSE 2023)



- Any test that does not fail in the buggy version are filtered out!
- Failure type and error messages are considered when clustering tests —> larger clusters are more reliable

LLMs will generate incorrect comments. But…

Kang, Milliken & Yoo (https://arxiv.org/pdf/2406.14836)

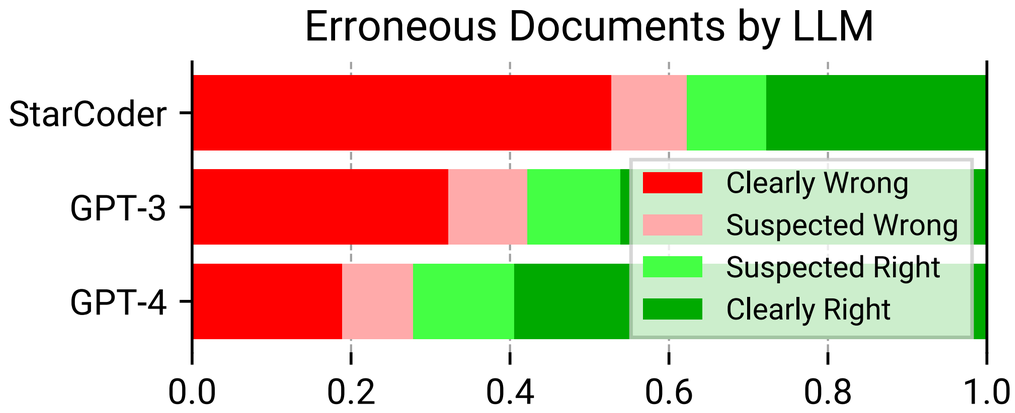


Figure 1: Comment factual accuracy by generating LLM.

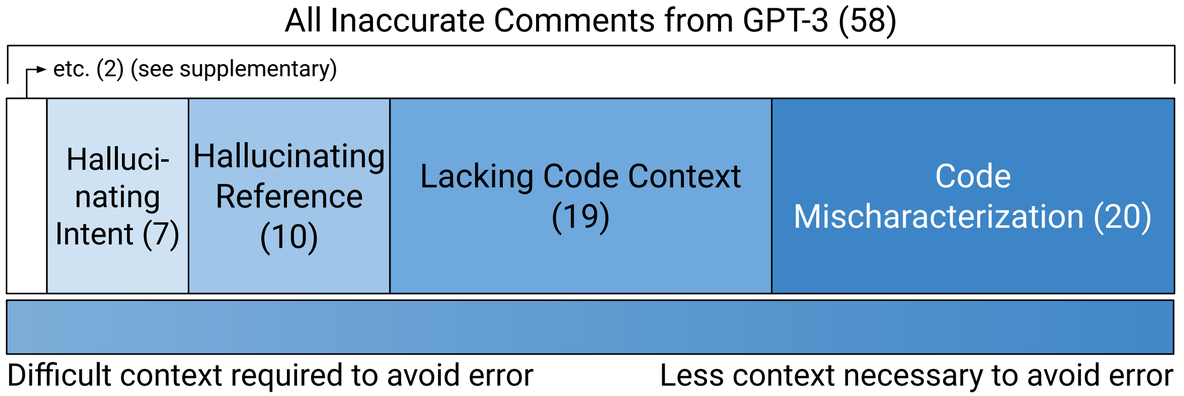
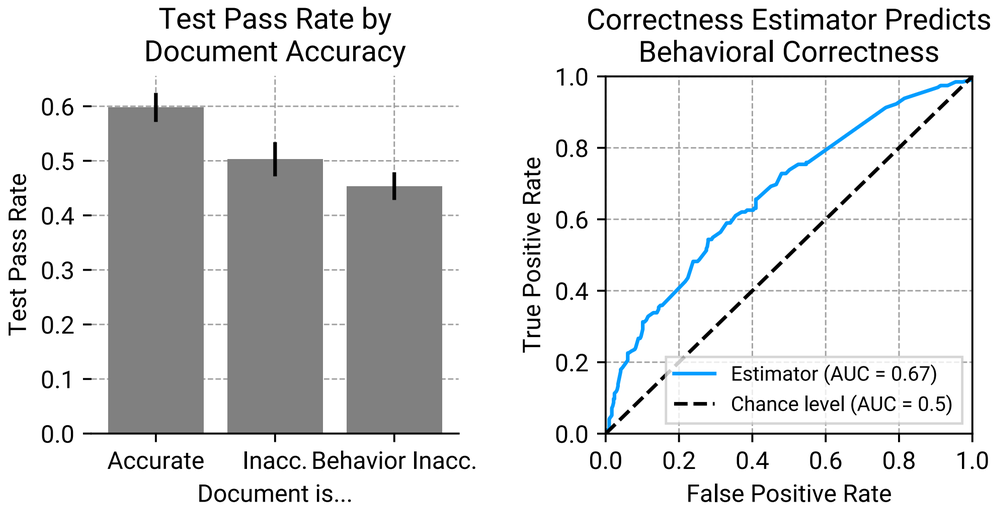


Figure 2: Diagram of error taxonomy for GPT-3 comments.

Evaluating code comment quality is notoriously hard. We tell LLMs to synthesise test cases based on comments —> if generate test cases fail, comments are not good.

Test Pass Rates Correlates with Doc Quality

Kang, Milliken & Yoo (https://arxiv.org/pdf/2406.14836)



(a) Pass rate by accuracy, with 95% confidence intervals.

(b) ROC graph of correctness estimator with actual correctness.

Figure 4: Relationship between comment accuracy and suggested indicators.

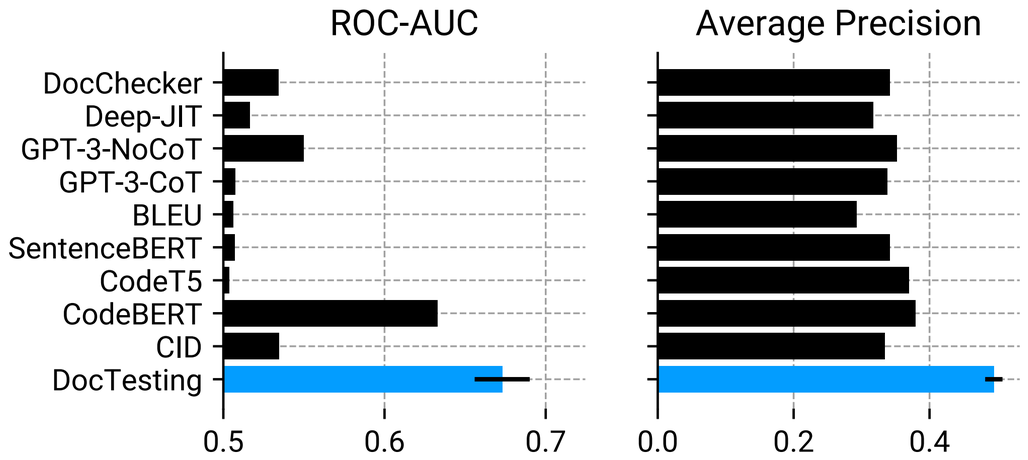


Figure 5: ROC-AUC and AP values compared with baselines. For our approach (blue), we present the mean value from five runs, along with its 95% confidence interval.

If we then tell LLMs to synthesise test cases based on comments
—> the pass rate of generated tests correlates with comment quality.

What is an agent?

agent | ˈeɪdʒ(ə)nt |

noun

1 a person who acts on behalf of another person or group: *in the event of illness, a durable power of attorney enabled her nephew to act as her agent.*
- a person who manages business, financial, or contractual matters for an actor, performer, writer, etc.: *his agent was able to negotiate a long-term contract.*
- a person or company that provides a particular service, typically one that involves organizing transactions between two other parties: *speak to your letting agent about refurbishing the property.*
- a person who works secretly to obtain information for a government or other official body: *a trained intelligence agent.*

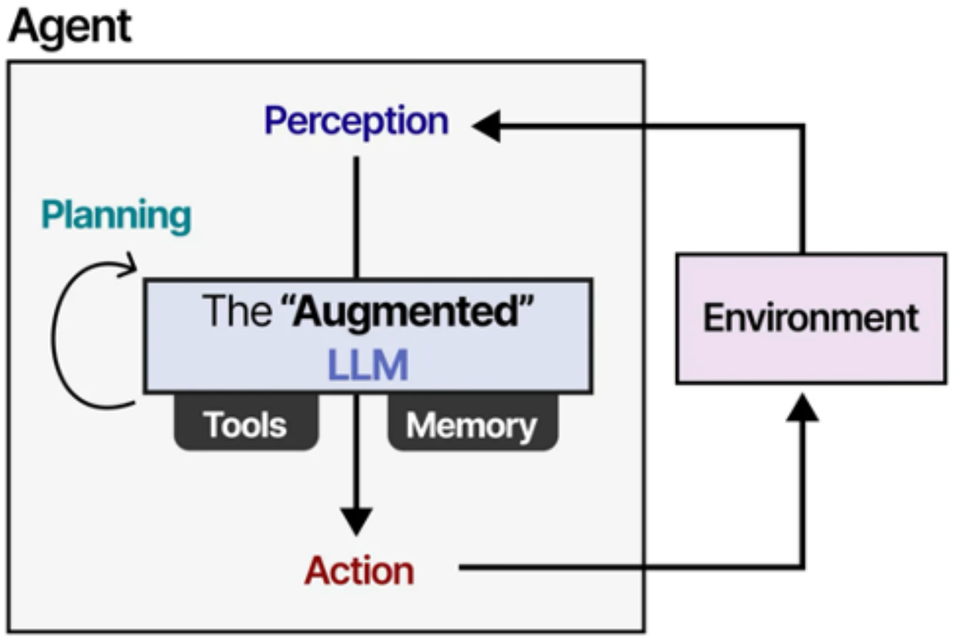
2 a person or thing that takes an active role or produces a specified effect: *these teachers view themselves as agents of social change.*
- a substance that brings about a chemical or physical effect or causes a chemical reaction: *there is an urgent need for new antimicrobial agents to combat infections | the bleaching agent used is hydrogen peroxide.*
- *Grammar* the doer of an action, typically expressed as the subject of an active verb or in a **by** phrase with a passive verb.
- *Computing* an independently operating program, typically one that performs background tasks such as information retrieval or processing on behalf of a user or other program.

AI Agent

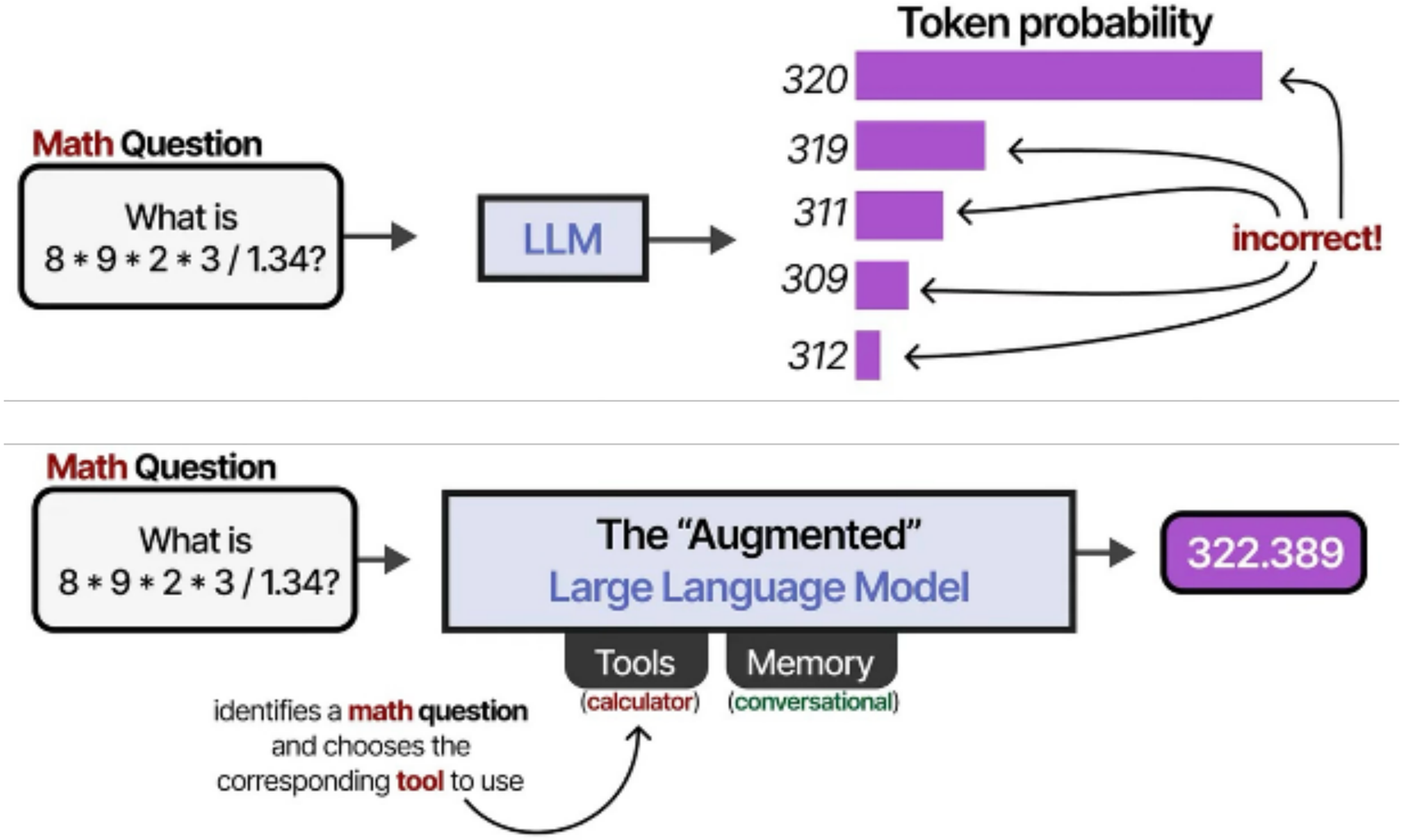
- Autonomy: given a task, it can plan the actions for completing the task and make decisions accordingly, on its own.
- Proficiency: it can find the appropriate tools and use them to complete the given task.
- LLMs can be more than chatbots.
  - In fact, many chatbots are internally complicated agent systems, I suspect. But more about this later.

AI Agent

- Autonomy: given a task, it can plan the actions for completing the task and make decisions accordingly, on its own.
- Proficiency: it can find the appropriate tools and use them to complete the given task.
- LLMs can be more than chatbots.
    - In fact, many chatbots are internally complicated agent systems, I suspect. But more about this later.



from "A Visual Guide to Agents" by Maarten Grootendorst
https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-llm-agents



from "A Visual Guide to Agents" by Maarten Grootendorst
https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-llm-agents

Why build agents?

- “If we wait a bit, maybe LLMs will be even smarter, achieve AGI, and we won’t need complicated agents… no?”
- This is a big question, and not just a rhetorical one 🤓 But here are my personal, biased answers for now.
  - Will AGI be possible based on the Transformer architecture?
  - What are we going to do about the lack of memory?
  - Context is now much longer than before, but it is still finite.
  - Some things are just easier when handled symbolically.

Why work on agents?

- If what we said in the previous slide is true, then we are looking at the beginning of a new form of software system
  - Truly neuro-symbolic: LLMs and traditional programming language making decisions together
  - Intelligent and multi-modal, therefore directly human facing with free-form inputs
  - We’re living in a really exciting time, whether you believe in AGI or not :)

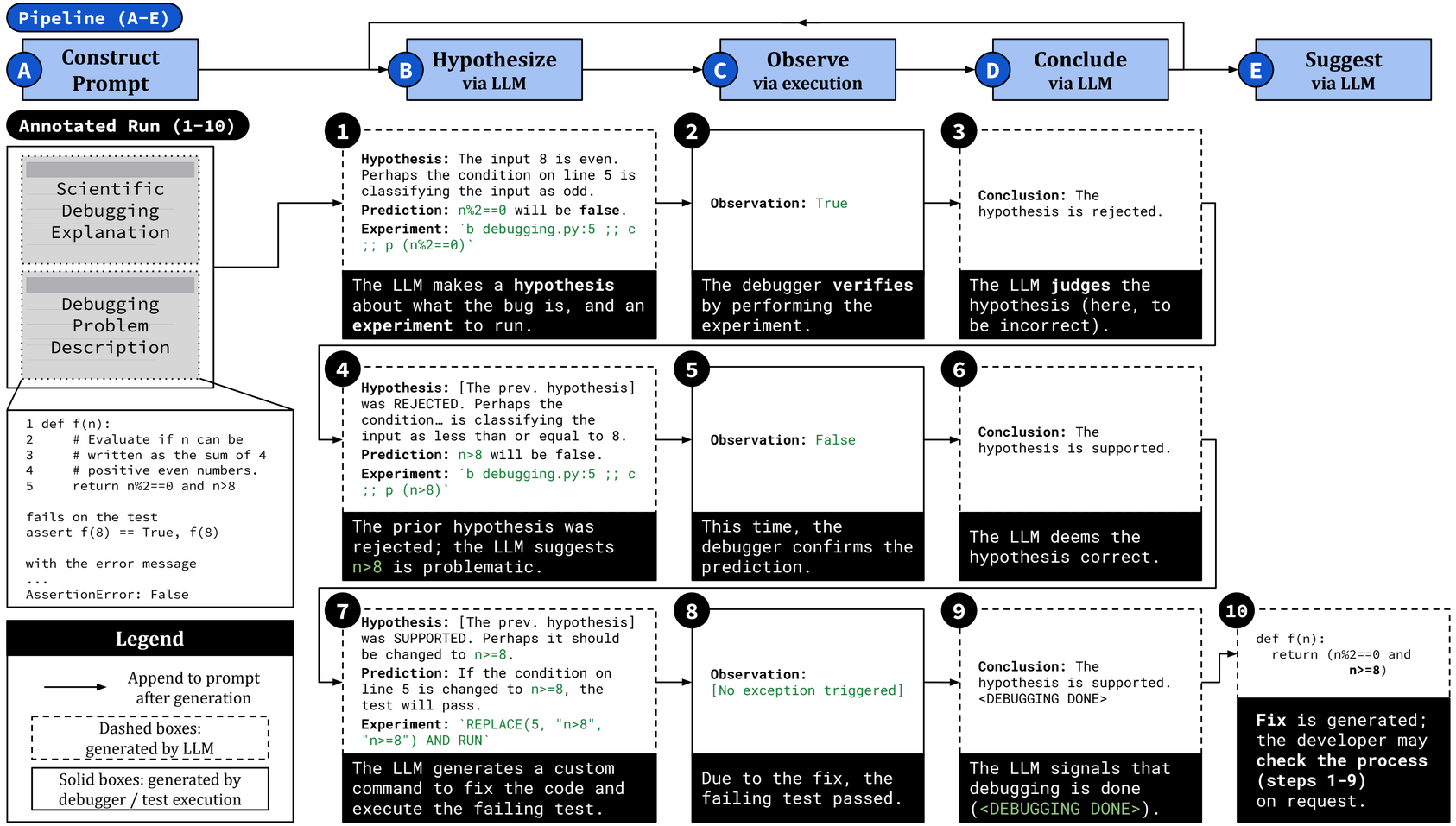
Our First Agentic Tool

(back in 2022-2023, without knowing)

- AutoSD: an APR technique that performs Scientific Debugging
  - Zero-shot: only instructions about SD are given
  - Tool usage: it has access to debuggers to evaluate its hypotheses
- Scientific Debugging (Zeller, 2009)
  - Debugging should systematically follow the process of scientific discovery: hypothesis, design of experiments, observation, and conclude or refine.
- Our primary aim was to make APR **explainable**. We thought the SD process can be an explanation in itself.

AutoSD: Zero-shot Automated Debugging

Kang et al., EMSE 2025



Agentic Design Elements in AutoSD

- Use of debuggers: AutoSD uses `pdb` to repair Python programs (mutated HumanEval). Since Python is interpreted, its debugger is relatively straightforward to use. Similarly, Java is friendly to AutoSD.
  - We later tried to port AutoSD to C/C++; `gdb` was much pickier.
- DSL for Code Edit: a fairly simple set of edit commands, based on line numbers and substring matching
  - Multi-hunk, multi-file patches may require a more sophisticated approach.
- Test Execution: the efficacy of the generated patch is validated using test execution.
  - Again, we benefit from Python's simple test framework.

Second Agent had Function Call Support

AutoFL by Kang & An, FSE 2024

- Our next aim was to make Fault Localization (FL) **explainable**.
  - On one hand, this seemed like an entirely reasonable direction. Perfect Bug Understanding assumption has been discussed as a major weakness of FL for a long time.
  - On the other hand, what is a good explanation for a FL result? The fact that patching this location leads to test pass? A description of how the test failure occurred?

Second Agent had Function Call Support

AutoFL by Kang & An, FSE 2024

- We also wanted to apply LLMs to FL (hammer and nail problem) but did not know how.
  - Mainly because of the context length limit. You cannot ask LLM to pinpoint a code location that has not been included in the context!
  - Wu et al. (https://arxiv.org/abs/2308.15276) assumes we have the faulty method, and tried statement level localization within the method (because a method fits the context length).
  - We tried sampling top-k methods from SBFL then querying the LLM - didn't perform particularly well.
  - How do we go repository-level?

AutoFL: LLM based FL

Kang, An & Yoo, FSE 2024 (https://arxiv.org/abs/2308.05487)

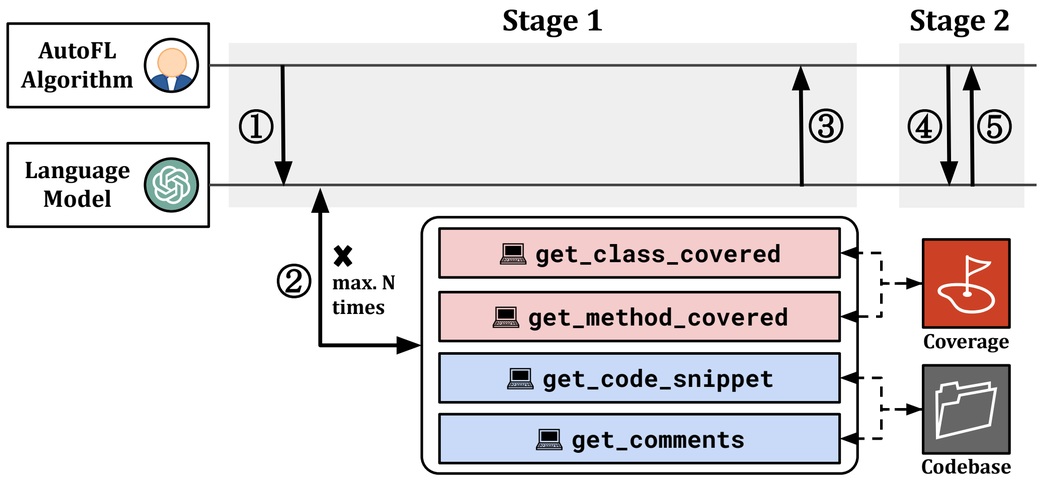


Figure 1: Diagram of AutoFL. Each arrow represents a prompt / response between components, with the circled numbers indicating the order of interactions. Function invocations are made at most N times, where N is a predetermined parameter of AutoFL.

1. Prompt LLM to start by looking at the list of classes covered by the failing test.

2. Up to 10 function calls.

3. After 10 calls, or when it is ready, the LLM generates a user-facing explanation.

4. We prompt the LLM to come up with buggy methods.

5. The model responds.

AutoFL: LLM based FL

Kang, An & Yoo, FSE 2024 (https://arxiv.org/abs/2308.05487)

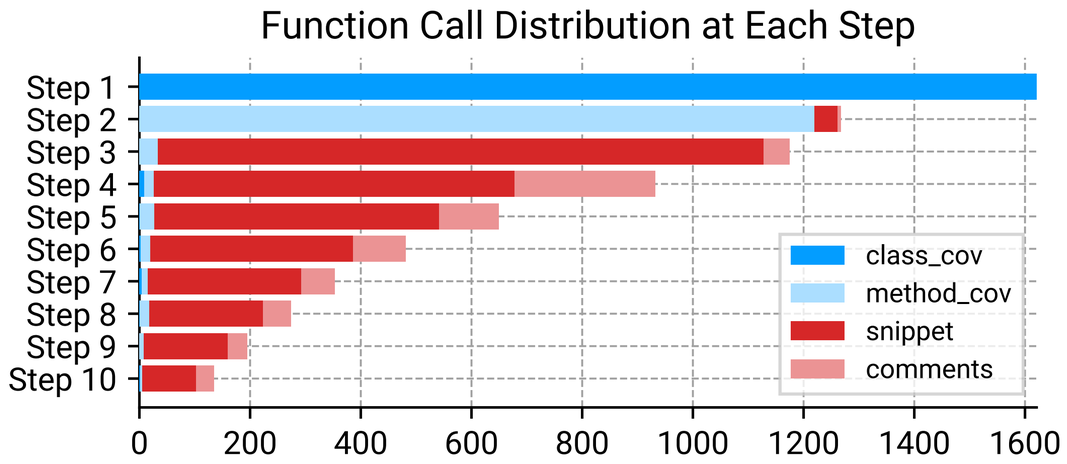


Figure 5: Function call frequency by step over all five runs of AutoFL. The total length at each step decreases as AutoFL can stop calling functions at any step; e.g. about 400 AutoFL processes stopped calling functions after the first step.

- Patterns do seem reasonable.
- The model gradually narrows down the scope, eventually looking at snippets (i.e. methods).
- Occasionally it looks at comments.

AutoFL: LLM based FL

Kang, An & Yoo, FSE 2024 (https://arxiv.org/abs/2308.05487)

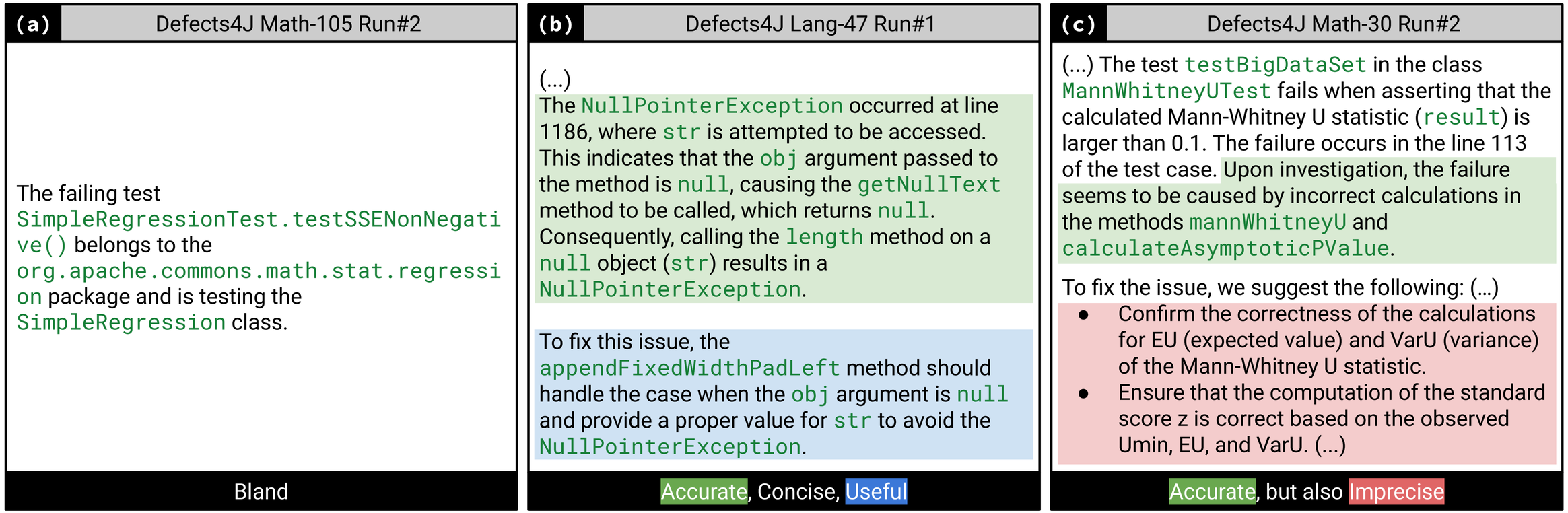
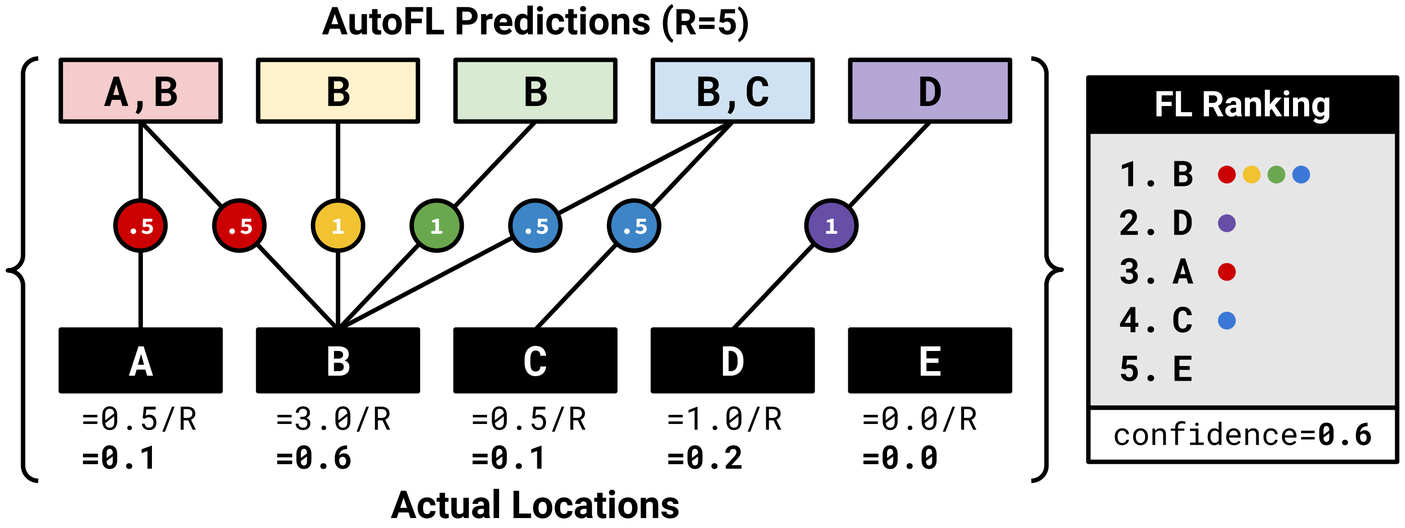


Fig. 7. Example explanations from AutoFL. Explanations (b) and (c) are truncated for clarity.

AutoFL: LLM based FL

Kang, An & Yoo, FSE 2024 (https://arxiv.org/abs/2308.05487)



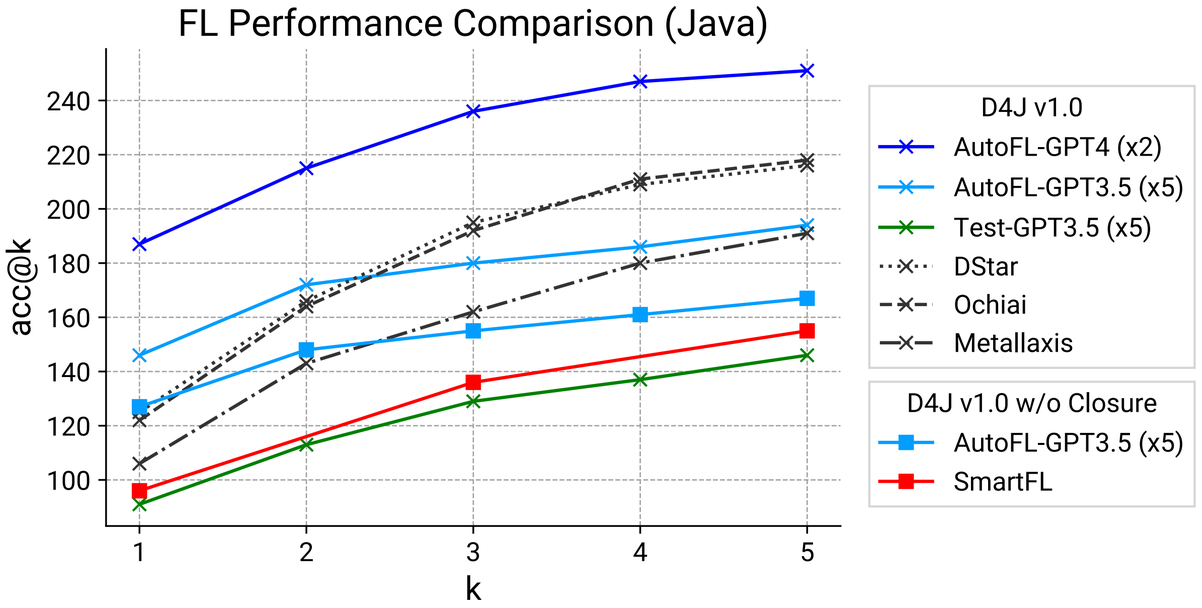
$$\text{score}(m) = \frac{1}{R}\sum_{k=1}^{R}\left(\frac{1}{|r_k|}\cdot[m \in r_k]\right)$$

where r_i is the set of methods reported as culprits in run i

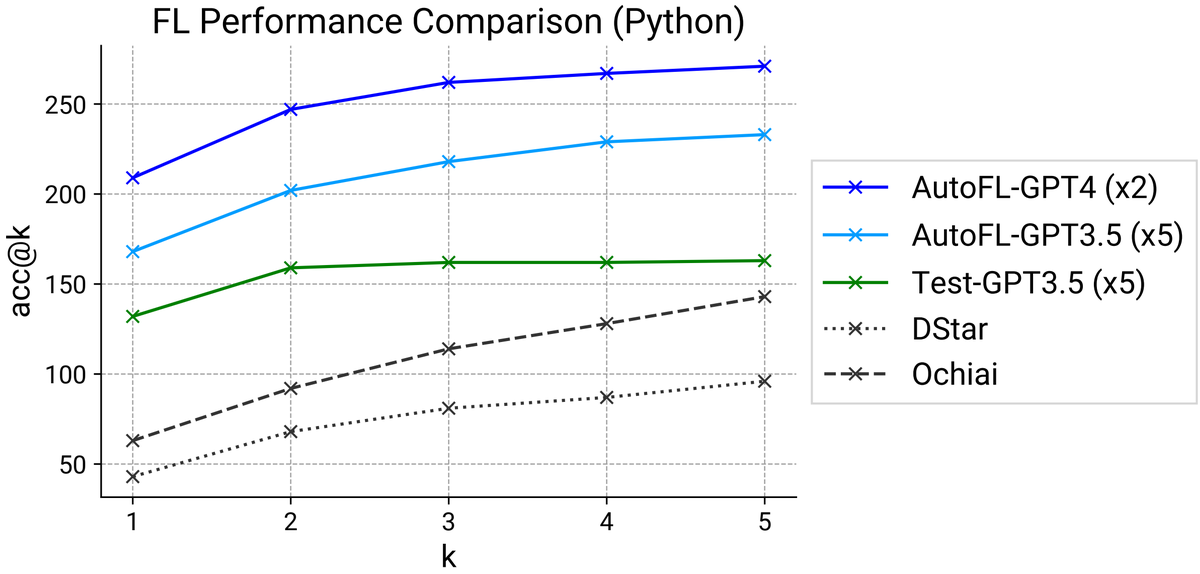
$$\text{confidence} = \max_{m \in M} score(m)$$

AutoFL: LLM based FL

Kang, An & Yoo, FSE 2024 (https://arxiv.org/abs/2308.05487)



(a) FL evaluation on Defects4J



(b) FL evaluation on BugsInPy

Fig. 3. Performance of various FL techniques on the Defects4J and BugsInPy benchmarks.

Agentic Design Elements in AutoFL

- Use of function calls: compared to AutoSD, there are more choices for AutoFL in terms of which function to call.
  - The functions do require some pre-processing, but these are engineering cost that is not beyond usual testing scenarios in CI environments.
- Decision to terminate: AutoFL decides when Stage 1 can be terminated.
  - This is more difficult than the case with AutoSD, which operates within the framework of Scientific Debugging. In general, knowing when it is done is difficult, when given an open task.

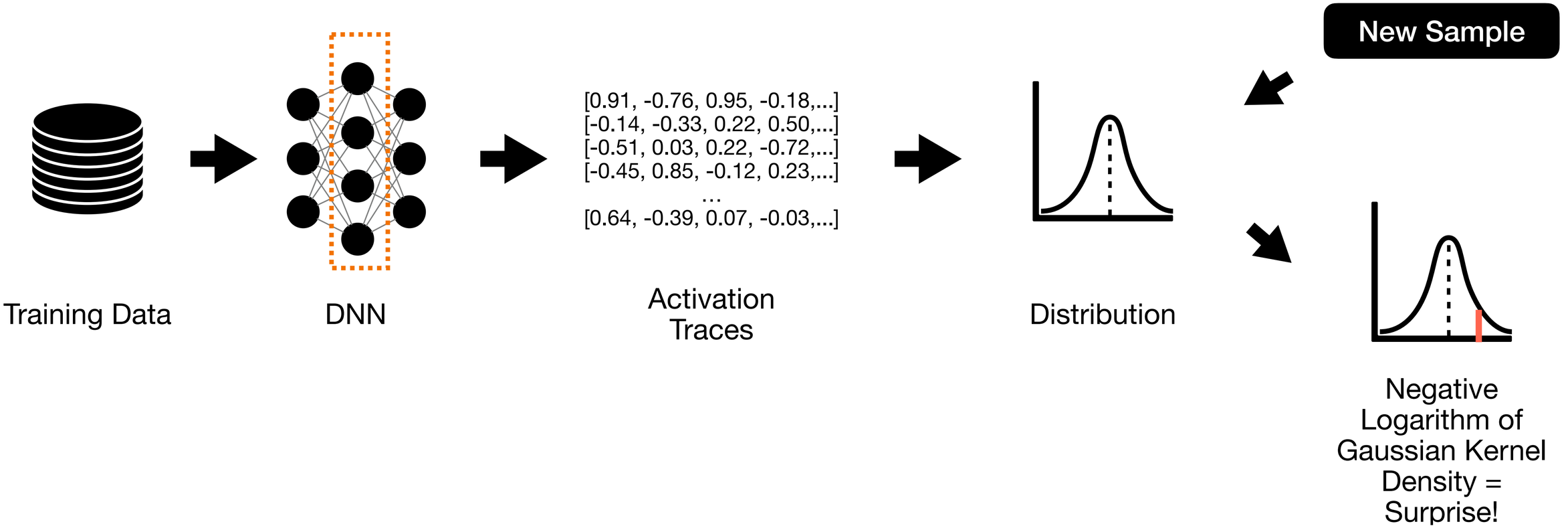
Current SOTA Agents

- Agents we have discussed today are defined by a set of very tight harness (i.e., a frame that defines what actions are allowed and not allowed).
- Widely discussed agents (OpenHand, OpenClaw, Claude Code) tend to have much more freedom, with almost no fixed + structured workflow harness.
- There is a wide design spectrum between two, and we haven’t explored them in full.

Test Adequacy for LLMs

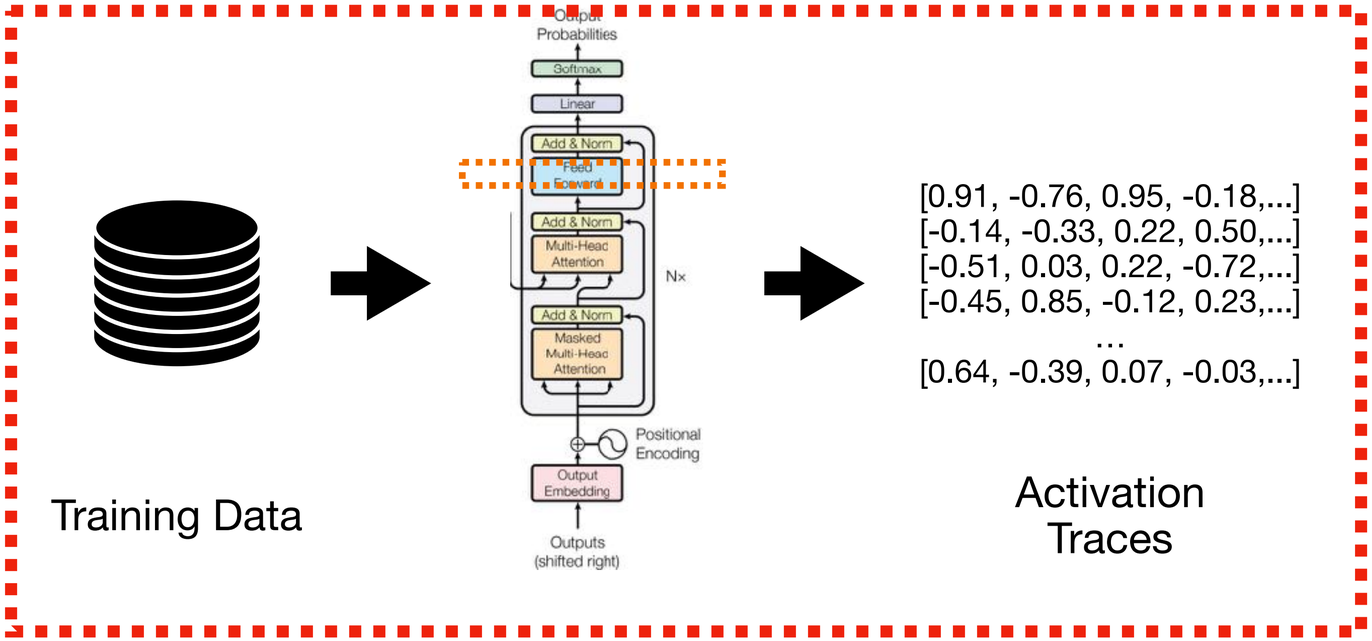
Recap: Surprise Adequacy

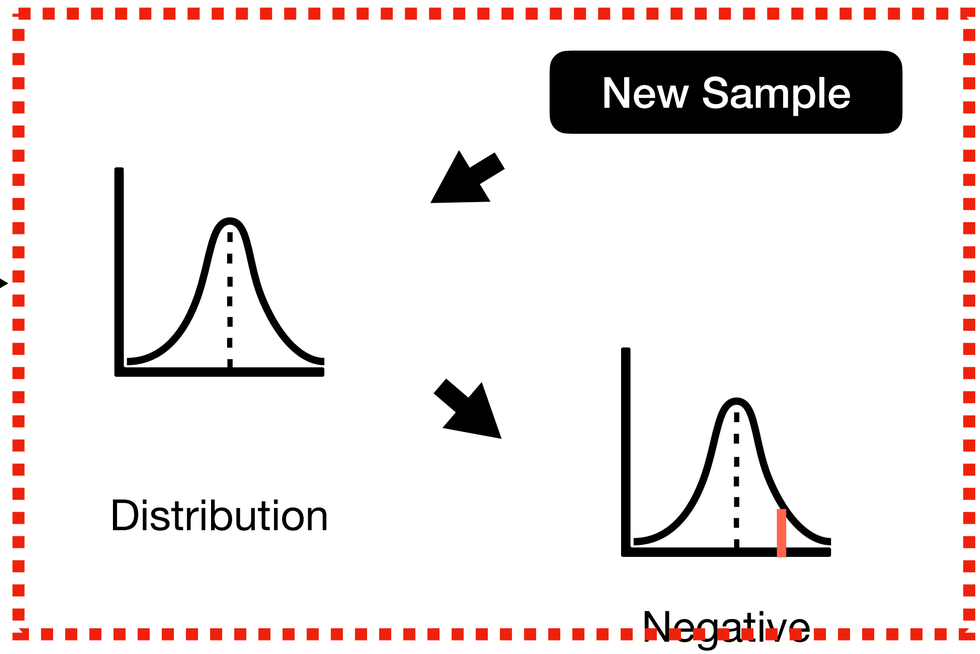
(i.e., out-of-distribution-ness measure as test adequacy)



But what do we do with LLMs/Agents?

Surprise Adequacy does not fit as it is.



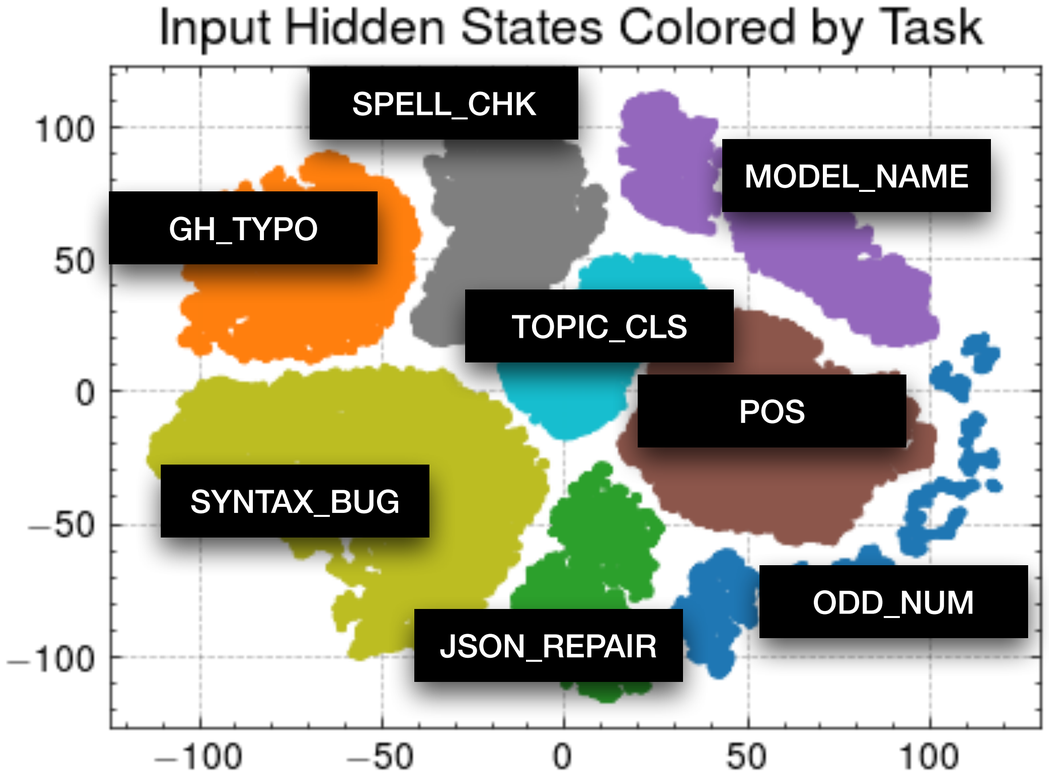


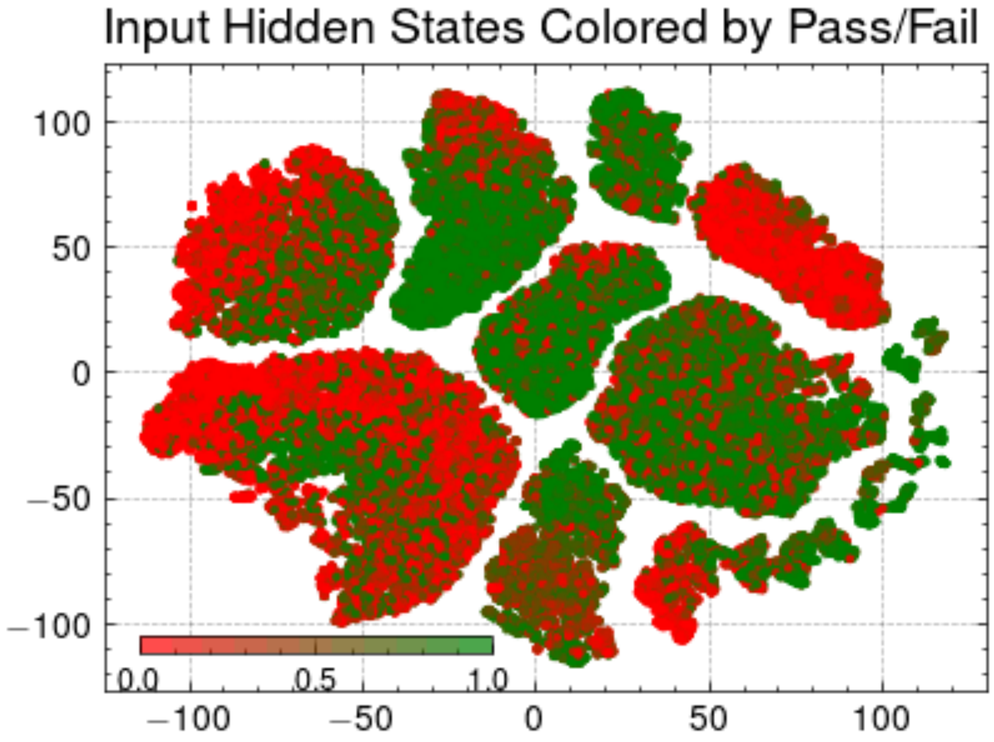
Massive Training Data (Essentially entire Internet + α)

Training and Inference form two heterogeneous distributions

Task-specific Modelling of Input Distribution

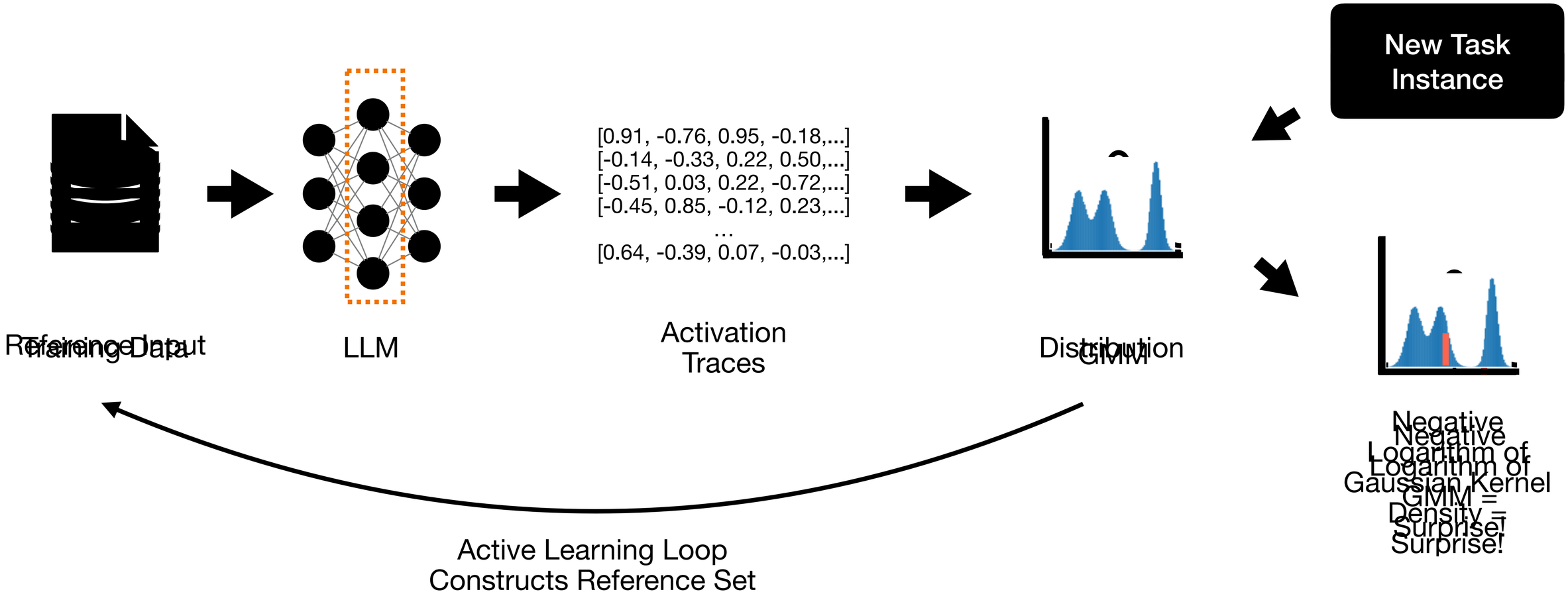
Clotho (FSE 2026 https://arxiv.org/abs/2509.17314)





Surprise Adequacy Strikes Again (SA^2)

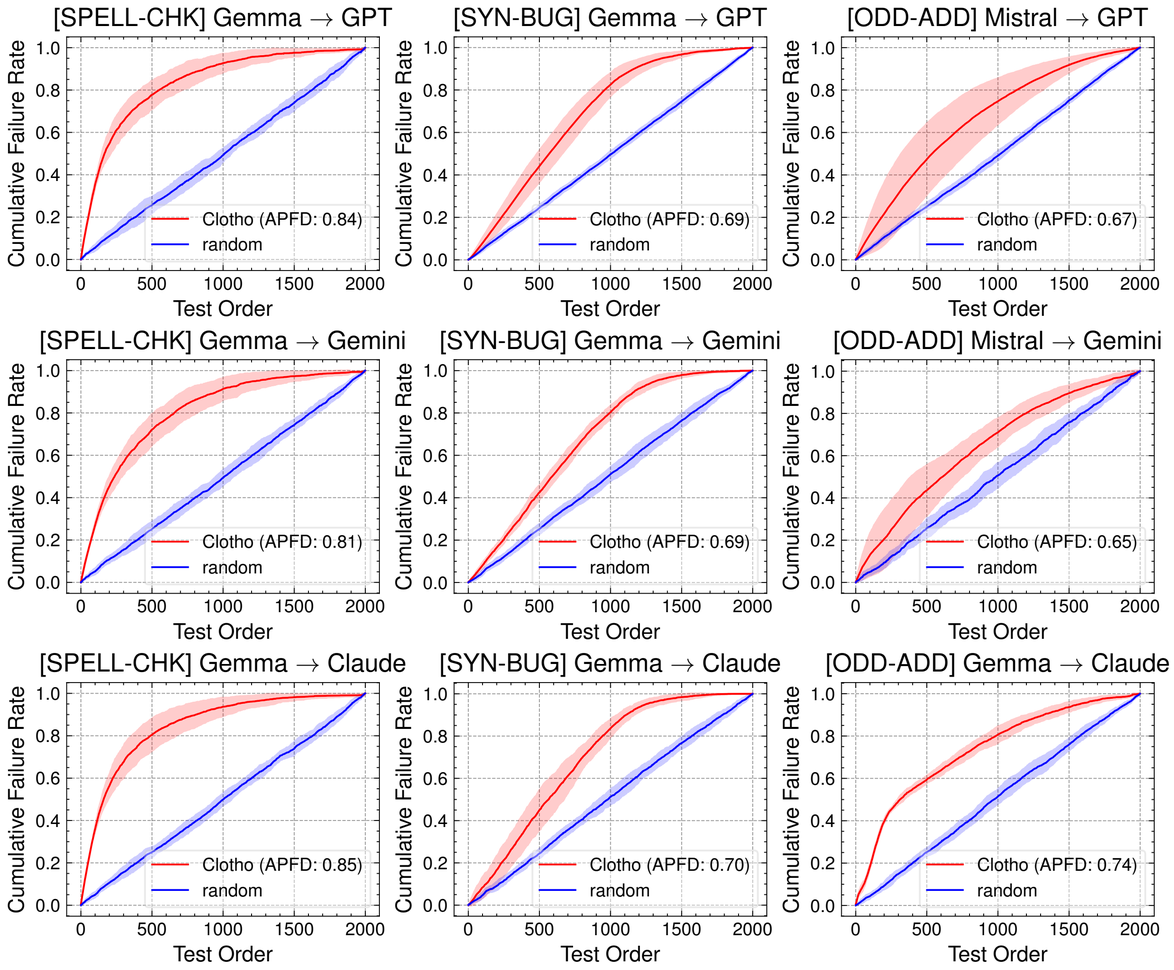
(this time, w/ task-specific reference sets)



Failure modes transfer cross models!

Clotho (FSE 2026 https://arxiv.org/abs/2509.17314))

- SA computed by Clotho shows moderate correlation with pass rates in SLMs.
- However, if you prioritize inputs according to SA computed by Clotho, and run inference on proprietary LLMs, the ordering still achieves high APFD!



Conclusion

- Fundamentally, the usefulness of LLMs depends on the statistical naturalness of language.
- Agent harness can be designed around techniques we have studied in this course. Open-ended agents are more flexible but may require more instructions.
- Agents are, in the end, just another type of software systems. We do not fully understand how to best test them yet.