An age of experimentation

Thomas Dullien
Cyber Security Research Team
OpenAI

BLUEHAT ASIA
SINGAPORE

"there are decades where nothing happens, and there are weeks in which decades happen"

- 1995 to 2022 saw rapid technological progress in IT, but few "surprising jumps"

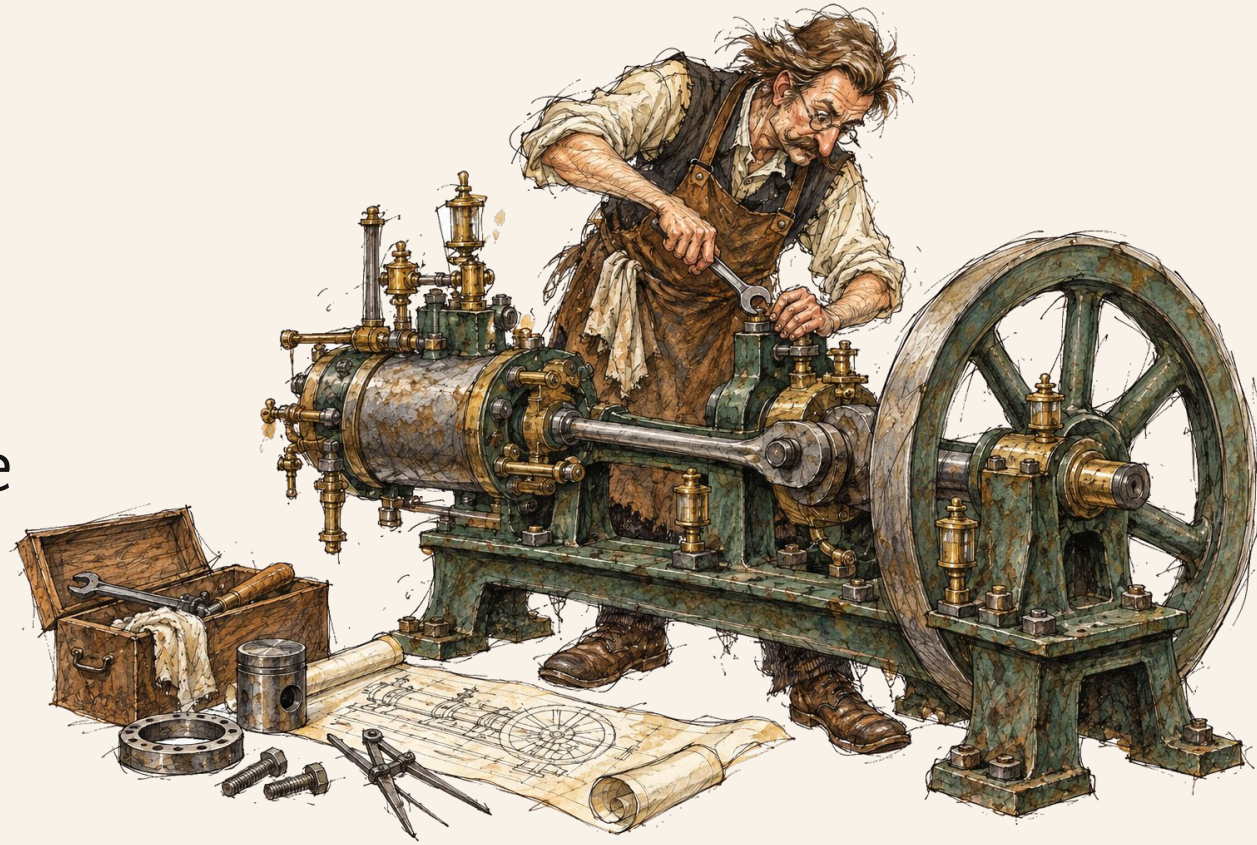
- new technology arrived: the internet, smartphones, etc - but they were "inside the expected trajectories"

- the last 2-3 years feel qualitatively different: "surprising" leaps in technology

- not all technologies are born the same way

- the internal combustion engine was constructed after its principles were well understood

- most applications today resemble what the inventors imagined



- some technologies are more like "discoveries" than "development"
- fire was **discovered**, and humanity had little understanding of how it works
- absent abstract models, experimentation was the only way to advance

- reasoning LLMs are more like fire than like the internal combustion engine
- We discovered them more than we constructed them
- understanding is nascent
- improvements very empirical

- The big surprise: the unreasonable effectiveness of statistically emulating a consistent inner monologue
- arguably more "artificial thought" than "artificial intelligence"?
- “intelligence” implies continual learning, “thought” is the inner monologue.
- But “artificial thought” can solve so many problems already!

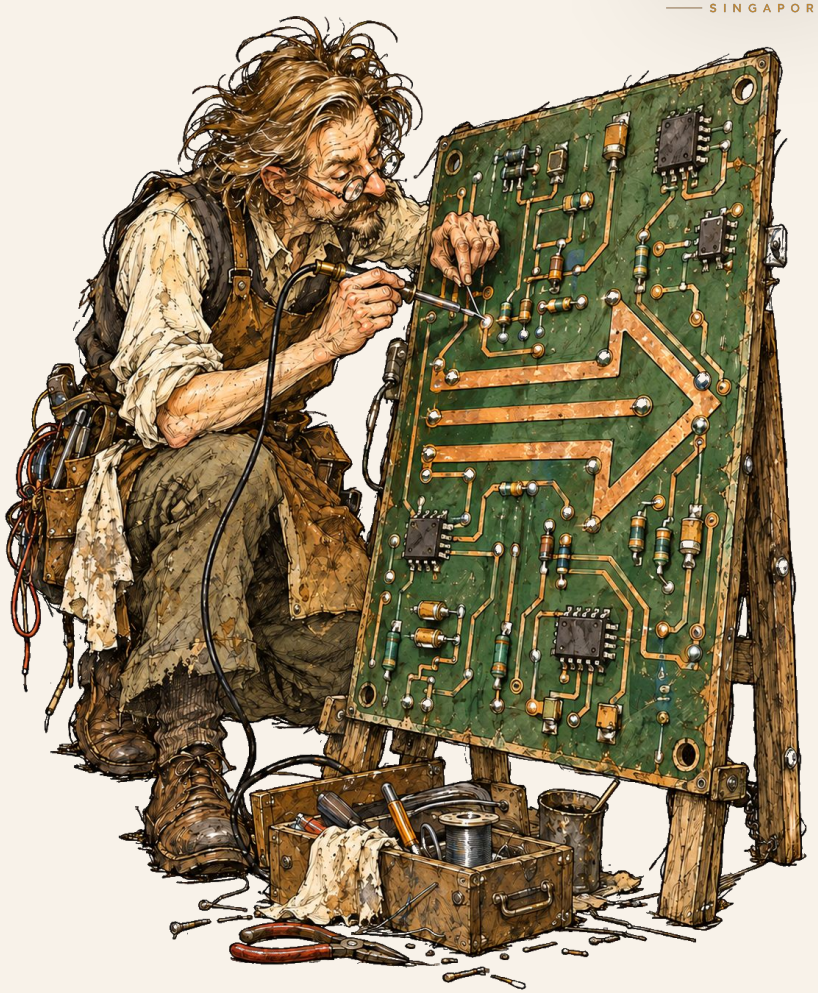
- Perform long-horizon tasks using Bash and Python
- Read code and write coherent understanding on how it works
- Translate code between languages
- Translate human language into code and vice versa
- Create plausible solutions to Millenium math problems

- Capabilities that were deemed impossible 4–5 years ago
- “Computer, please tell me what the intent of this code is”
- Translate human language into code
- Translate code into human language
- LLMs are used to generate code, but **calls to LLMs are also becoming intrinsic parts of many workflows**

BLUEHAT ASIA
SINGAPORE

Adding LLMs to your workflow adds a source of stochasticity!

- CS was born somewhere between EE and math
- Apparent determinism in computers was a tremendous achievement by EE and process engineers
- Allowed CS to aspire to be mathematics



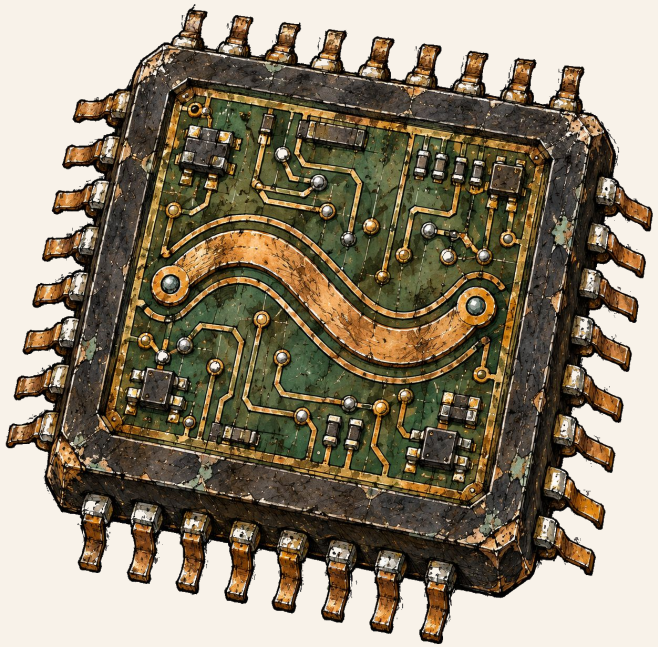
- Software engineering workflows are very geared toward (semantic) determinism
- example: ci/cd Pipelines -- test something **once** to see if it still behaves as it did last time
- determinism allows rapid iteration
- adding stochasticity messes with the workflows we are used to

Claim: Determinism is dying, and it's unclear how much will remain.

Security is affected each time.

Three stages I observed:

1. cracks in hardware determinism (row hammer)
2. death of temporal determinism (spectre)
3. LLMs mean the death of semantic determinism



- CS folks **take determinism for granted**
- Generations of EE and process engineers sacrificed themselves to keep that illusion
- contract between hardware and software over the last decades: We keep visible semantics stable, but we get to do whatever underneath
- determinism isn't natural, **it's a fragile construct**

"The death of **hardware** determinism"

BLUEHAT ASIA
SINGAPORE

- row hammer really made me understand that computers are "average case deterministic"
- they are deterministic most of the time
- and **you have to be nice** to them – hammering, glitching, EMFI
- Economics enforce: You'll be sold the **worst hardware you can't detect as bad**

When hardware determinism fails, security boundaries tend to fail.

"The death of **temporal** determinism"

- in order to make things fast we need caching
- and/or speculation
- leads to ugly, multi-modal and nonparametric distributions
- Temporal semantics are not specified, hardware implementers need full liberty

Discrepancy between timing **models** and **reality** leak information which should semantically not be accessible across security boundaries (Spectre).

"The death of **semantic** determinism"

- LLMs are in practice random
- Random sampling necessary for problem exploration
- The sampling can in theory be reproducible
- more randomness from hardware
- more randomness from concurrency

"The death of semantic determinism"

- LLMs are also ***chaotic***
- recurrence equations over a state space
- Lyapunov exponent (butterfly principle)
- long trajectories for problem solving

- consequences of single token change in prompt not analytically solvable
- Any tiny change in a prompt means (in formal verification terms) T -- “anything can happen” (but rarely does)
- we are absolutely in an empirical / experimental domain
- death of **semantic** determinism

- Software engineering and software security is now in an age of experimentation
- this is different from the past
- we have no strong abstract model how things work
- experimentation and empiricism is all that is currently available

BLUEHAT ASIA
SINGAPORE

What does this mean?

- CS is historically not an empirical science
- CS majors know little about experimental design and are often weak in statistics
- First realization during Rowhammer: Sitting in a room with highly competent CS PhDs, difficulties modeling Rowhammer experiments, confidence intervals etc.
- I was extremely weak myself: Background in CS, some Econ, and a lot of commutative algebra while avoiding statistics.

BLUEHAT ASIA
SINGAPORE

In a probabilistic system, every change is a hypothesis test!

- this means we have an experiment
- you need to think about what you want to show
- you need to account for the sources of variance in your system
- “The first principle is that you must not fool yourself, and you are the easiest person to fool.” -- Feynman

Questions to ask

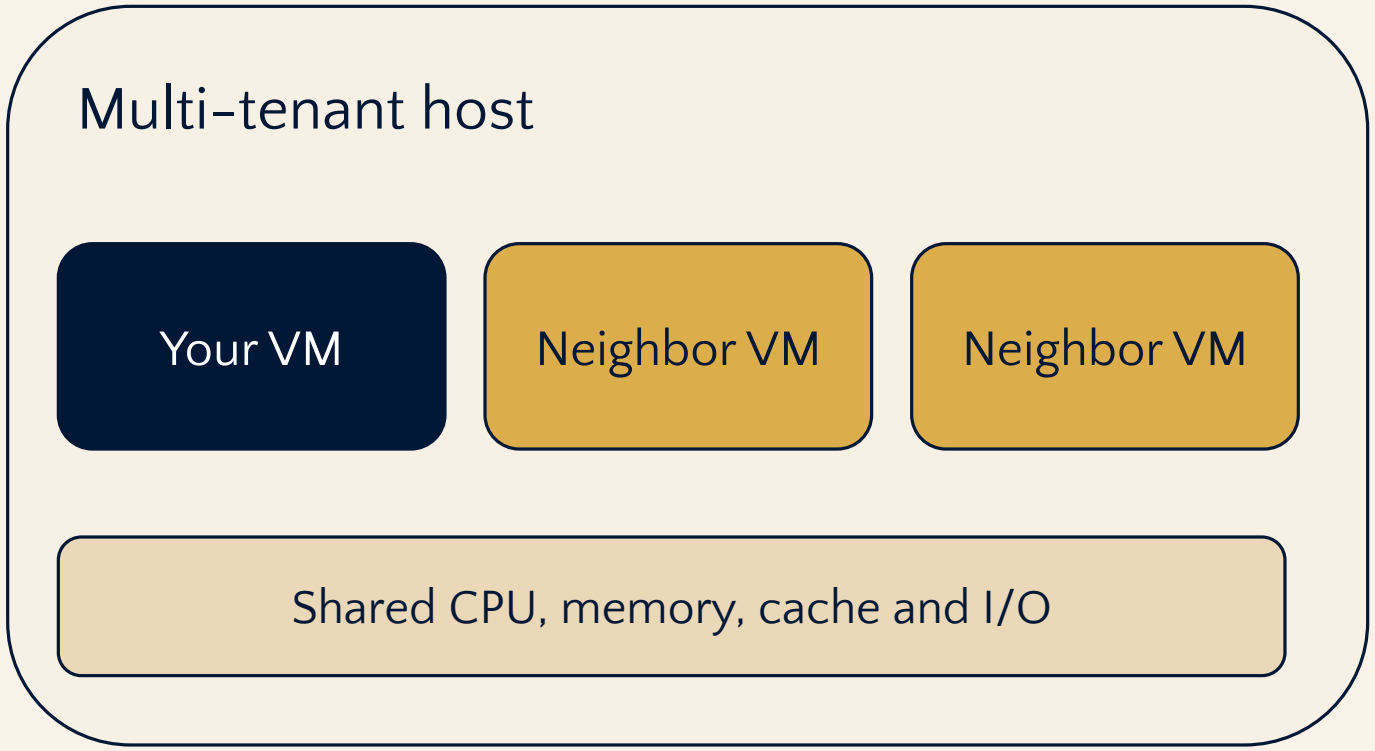
- How much does my measurement vary on identical re-runs?
- What are the sources of randomness?
- Is there any time-varying factor like the noisy VM neighbors that influences multiple runs, or are the measurements IID?
- (IID meaning there is randomness, but the distribution is the same on each draw)

Effect sizes and # of samples

- experiments aren't free
- number of samples is proportional to (stddev / effect size) squared
- **you want your effect to be bigger than stddev** if at all possible

$$N \propto \left(\frac{\sigma}{\Delta}\right)^2$$

Example: Performance optimization in the cloud



- Noisy neighbors can slow me down
- **Adds +/- 15% runtime** variation (!)
- 100s or 1000s of runs necessary to show improvement, if at all.
- **Destroys IID assumption. Ouch.**

Solution: "Tomato farmer protocol"

- Paired experiments on the same VM: Two instances side-by-side.
- Exposed to the same noise.
- Adapted from fertilizer testing, “Statistics for Experimenters”
- Not always possible.

- let's look at LLM bug detection
- example: Raptor, Codex Security, any other LLM+prompt system / harness.
- How do we know we have made an improvement?
- Do we know the variance of our metrics?

- Let's assume we have a codebase with 100 known bugs
- Let an agent loose on that codebase -- how many will the agent find?
- If the agent is just allowed to run on the whole codebase, there is path dependence, and the individual bugs are not independent trials (e.g. if the agent decides to skip a file). So a full run is really just one trial?

- how many test runs do you need to have 95% chance of detecting a 1% improvement?
- let's make a bunch of assumptions we know are false (normal distribution, stddev 2%) - reality is worse
- result: 52 paired sample runs
- show of hands: Who **does** that to test a prompt improvement?

- Ok, so let's say we have 100 bugs that we **can** test independently
- If they are truly independent we can make narrow confidence intervals from a small number of runs (example: TermBench)
- But … uh … the problems have underlying success factors ("does the model know how to use XYZ") to make them correlated.
- This leads to overly narrow confidence interval estimates.

Uncomfortable conclusion 1:

Most of us are playing slot machines

Uncomfortable conclusion 2:

Regular CI/CD processes are poorly adapted to dealing with prompt changes

- LLMs as part of a workflow destroy semantic determinism
- LLMs also make experiments relatively costly
- Existing Software dev processes are a poor fit

- It is easy to end up in a situation where **checking whether a PR that changes a prompt is worth merging may cost more than the developer's workday**.

- This does not mean that merging the PR on a hunch is harmful.
- We have all merged prompt changes on 3–5 experiments.
- The risk is spending a lot of time on iterated changes, thinking one is making steady progress, without getting anywhere.
- Beware the “Gas Town” phenomenon: Steve Yegge’s attempt at a software factory.

- small effect sizes may be too expensive to evaluate.
- it may be worth stacking 5–6 suspected improvements and measuring them jointly.

$$N \propto \left(\frac{\sigma}{\Delta}\right)^2$$

Uncomfortable conclusion 3:

Congratulations. Y'all have to be empirical scientists and (amateur/professional) statisticians now.

LLM code review and fuzzing

- LLM code review is more like fuzzing than like traditional SAST
- Every run is similar to a fuzzing run (but in most APIs you do not get to freeze the seed)
- LLM code review is never “done”, in the same way that fuzzing is never “done”.

LLM code review and fuzzing

- Expect exact re-runs to produce different sets of bugs.
- Expect small prompt variations to produce different sets of bugs.
- Expect new model generations to produce different sets of bugs.

Similar to fuzzing, the “right” metric to track is the rate of useful discovery per dollar of compute spent.

Over time, this will go down, and new discoveries are hopefully expensive for everybody.

Security bugs and fishery models

I have jokingly suggested renaming “bugs” to “fish”:
A lot of our problems are similar to fishery management.

- Security bugs are introduced into applications at a certain rate.
- Security bugs are removed at a certain rate.
- Nobody knows how many bugs there are.
- We can only track effort and “catch rates”.
- New technologies help us catch more fish.

Catching all the fish?

- It may be unrealistic to **catch all the fish**.
- You never know if a Tuna hid somewhere during your hunt, the ocean is too vast.
- You **can hope** that **catching fish becomes uneconomical** because you have depleted the stock.
- Improvements in technologies may find the last hide-outs.

BLUEHAT ASIA
SINGAPORE

Where are we headed from here?

Concrete advice?

Observation: The current change is pretty big

- my confidence intervals around predictions have gotten very wide
- we are looking at a period of rapid change
- certainly as big as the smartphone, likely as big as the internet, possibly as big as railroads, perhaps as big as electrification.

Accept empiricism, accept uncertainty

- There is no way around accepting that we are dealing with stochastical systems.
- Being statistically rigorous isn't an absolute necessity – there are many true things that are onerous to demonstrate as true. But: Be wary of treading water while feeling productive.
- "Statistics for experimenters" is a great book!

Hypothesis generation, testing, and progress

- Observation: Hypothesis generation is now cheaper
- Implementation of the test is now cheaper

- actual experiment cost/duration is often the bottleneck

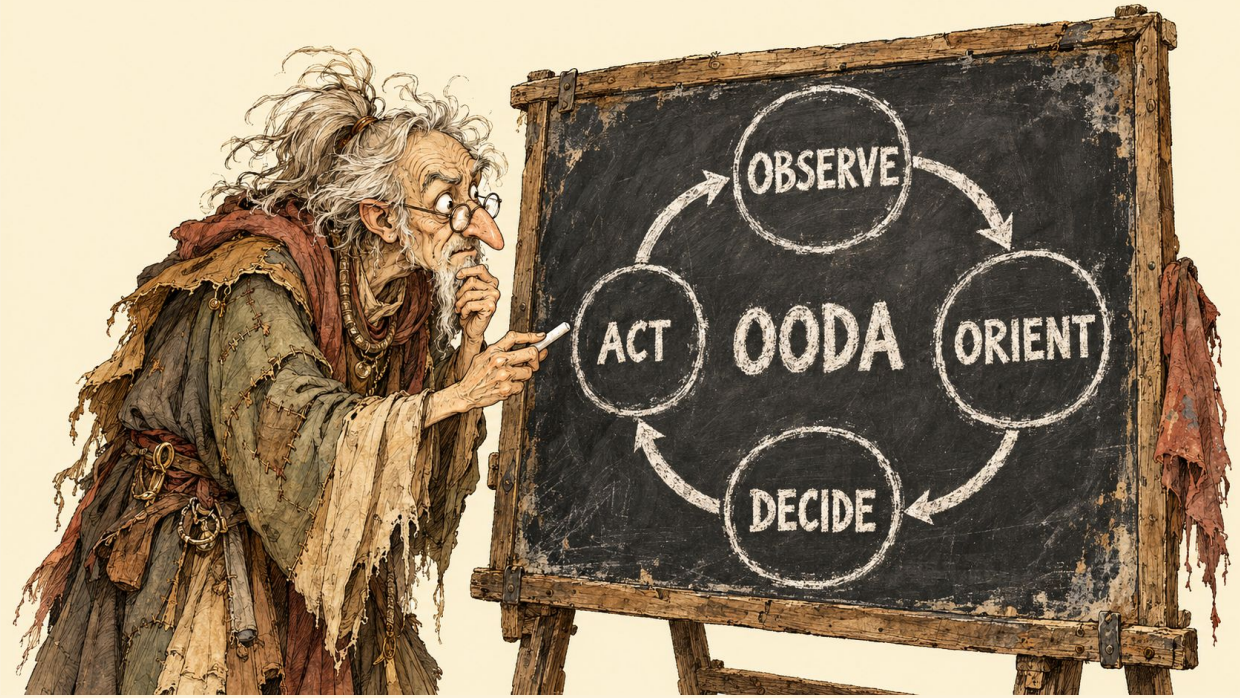
- **Try to build a hierarchy of benchmarks – small ones you can iterate rapidly on, big ones you use periodically to validate the small ones.**
- **Danger zone: Make sure the small ones are representative!**

The value of harnessing and prompting

- Will the long-term value be in the models or in the harness? Both.

- For the foreseeable future (the next few years) I expect that clever harnessing can boost any model on any task.
- But also: Each improvement from harnessing can feed into the next training run.
- I expect these boosts to be absorbed in new training runs, so each harness needs to keep moving to stay relevant.

- OODA loops apply here too:
- A cheap experiment means a faster "OO" part of the loop.
- The "A" part has been sped up.



BLUEHAT ASIA
SINGAPORE

No visible technical reason the current progress should come to an end soon

Progress will slow down when expected benefit from further progress falls under cost of progress. Where is that?

Rapid technological change can be scary...

I try to think of my grandfather: Born in 1902, died in 1982.

- Born in the Prussian Kingdom, horse carriages & gaslight.
- Died in an era of Concorde travel, computers, and moon missions.
- Survived two world wars and great upheaval.

The changes he saw were much more drastic than what I saw 1981 to today.

BLUEHAT ASIA
SINGAPORE

It is a (perhaps uncomfortably) exciting time in technology.

Rapid change, high levels of uncertainty. **The right response is focus, not fear.**

BLUEHAT ASIA
SINGAPORE

Thank you.