Formal Containment

Proof-carrying code and AI safety

Quinn Dougherty

June 26, 2025 | Trajectory Labs

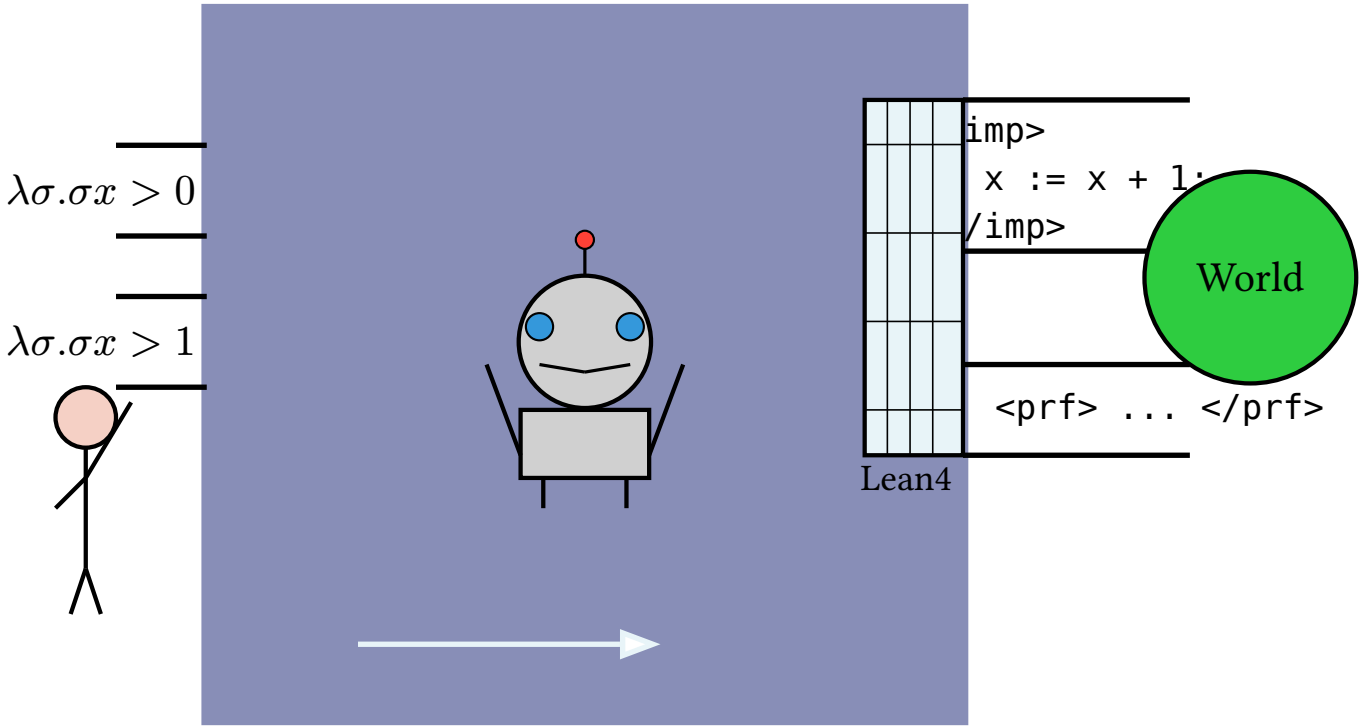
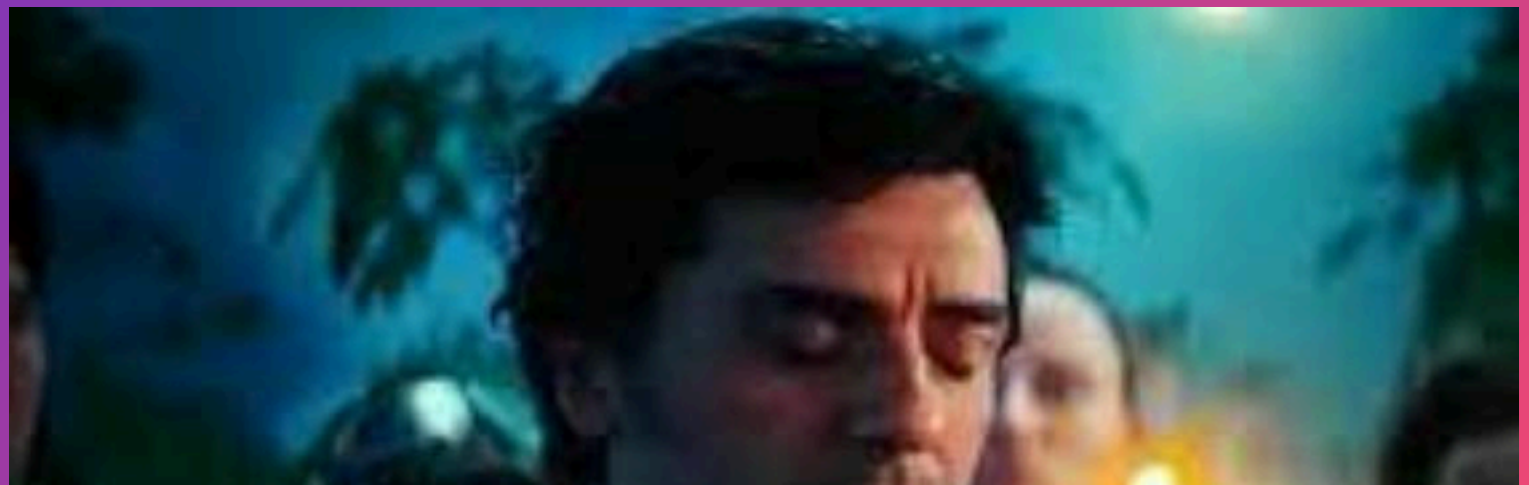


Figure 1: Box protocol at example specification. The AI accepts a specification (formally $(\text{Env} \to \mathbb{P}) \times (\text{Env} \to \mathbb{P})$)) and returns proof-carrying code, with the option of returning nothing.

I.



Two old literatures

- In Yudkowsky 2002 [1], *AI boxing* is the attempt to **contain** AI by policing its interface to the world.
- In Necula 1997 [2], *proof-carrying code* is the attempt to **tag** code with a proof of it's correctness.

AI Containment (Boxing)

> « *When we build AI, why not just keep it in sealed hardware that can't affect the outside world in any way except through one communications channel with the original programmers? That way it couldn't get out until we were convinced it was safe.* »
>
> Yudkowsky 2002 [1]

Spoiler alert: Yudkowsky recommends against trying this.


SOMEHOW BOXING RETURNED
imgflip.com

Somehow AI Boxing Returned

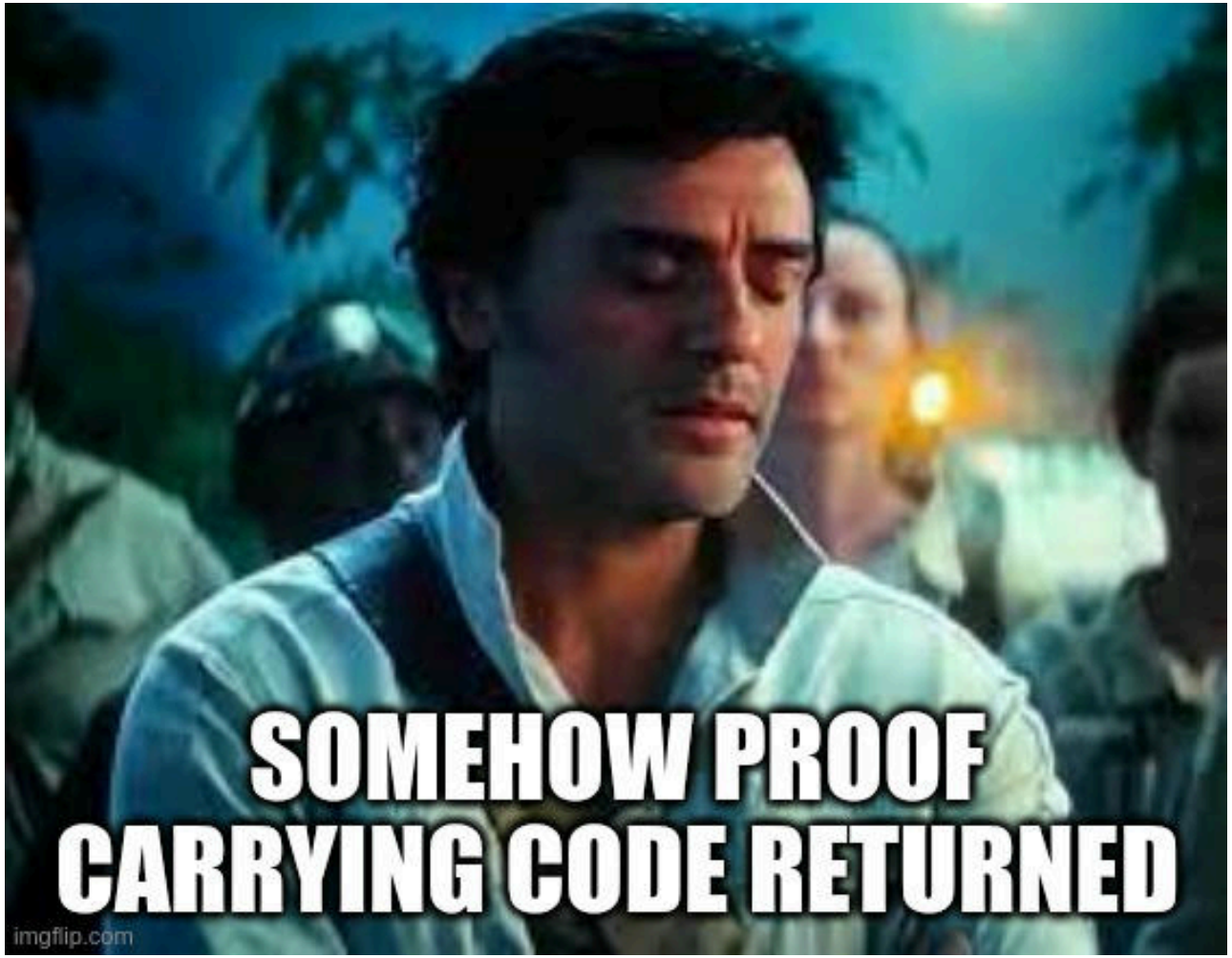
Recent work from AI Control ([3], [4]) and Safeguarded AI ([5]) is thinking through containment to bootstrap into early stages of the transition.

Proof-Carrying Code

> « The untrusted code producer must supply with the code a safety proof that attests to the code's adherence to a previously defined safety policy. »
>
> Necula 1997 [2]

Conceptually like $\exists c : \mathbf{program}, P\,c$ where P is some predicate on programs.

Or the dependent pair $(c, \pi) : \mathbf{program} \times P\,c$ (i.e., π is a proof of Pc)


SOMEHOW PROOF
CARRYING CODE RETURNED
imgflip.com

Somehow Proof-Carrying Code returned

Recently Kamran et al 2024 [6] revived proof-carrying code in the form of *proof-carrying code completions*, language model calls that provide verified dafny code.

II. Formal Containment Protocol

Box

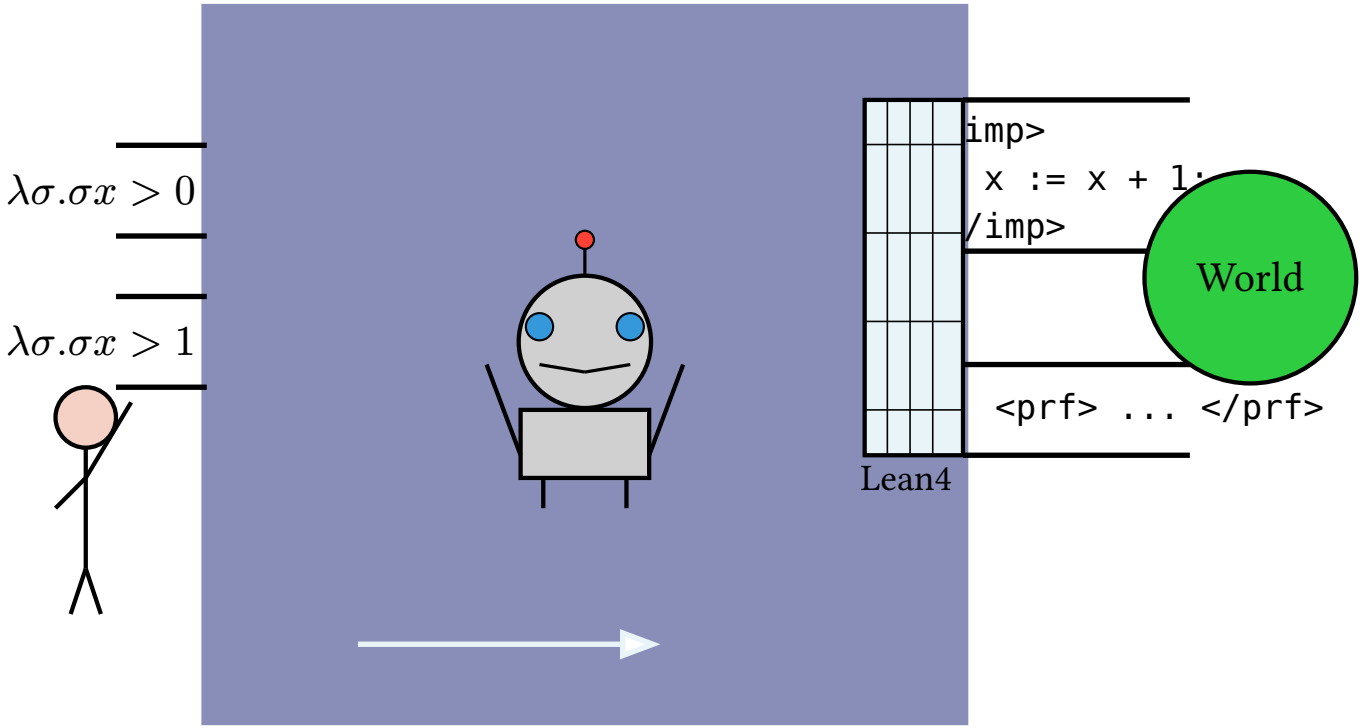


Figure 2: Box protocol at example specification. The AI accepts a specification (formally $(\text{Env} \to \mathbb{P}) \times (\text{Env} \to \mathbb{P})$)) and returns proof-carrying code, with the option of returning nothing.

Preliminaries: notations

- $\mathbb{P}$:= the type of propositions
- `imp` := the minimal imperative programming language with expressions valued in integers containing skip, sequence, assign, if, and while statements
- Env := state type, assigning variable names to values (formally `string`$\to$`int64`)
- Assertion := assertion type, predicates on state (formally Env $\to \mathbb{P}$)
- exec := execution, a function from a command and a state that returns a state (formally, `imp` $\to$ Env $\to$ Env)

Preliminaries: hoare logic

A *hoare triple* is a ternary predicate expressing when a command sends an assertion to another assertion, quantified over all states. Formally,

$$\text{hoare} := PcQ \mapsto$$

$$\forall(\sigma_1\sigma_2 : \text{Env}), P\sigma_1 \to \text{exec } c\sigma_1 = \sigma_2 \to Q\sigma_2 :$$

$$\text{Assertion} \to \texttt{imp} \to \text{Assertion} \to \mathbb{P}$$

and denoted hoare $PcQ = \{P\}$<imp> c </imp>$\{Q\}$. A *term* of type $\{P\}$<imp> c </imp>$\{Q\}$ is a proof that the triple is true.

Containment Protocol: example trace

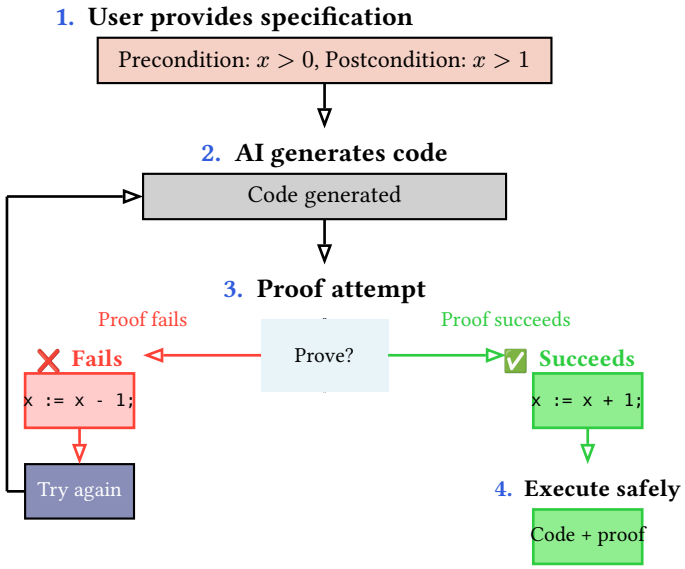


Figure 3: Example trace of the Formal Containment Protocol showing the decision fork at proof attempt, with failure leading to retry and success leading to safe execution.

Example PCC pair

```
example : {{astn x > 0}}(imp {
  x := x + 1;
}){{astn x > 1}} := by auto_hoare_pos
```

Listing 1: Proof of the example hoare triple from Figure 1.

III. Experiments

Specification samples

Sample	Precondition	Postcondition	∀-bound metavariables
gt8	`x = 0`	`x > 8`	—
swap	`x = ~n <^> y = ~m`	`x = ~m <^> y = ~n`	`n m`

facto	`x = ~n`	`y = ~(`
		`  let rec go := fun (x : Int) =>`
		`match x with`
		`  | .ofNat m => match m with`
		`  | .zero => 1`
		`  | .succ k => k.succ * go`
		`(Int.ofNat k)`
		`  | .negSucc _ => 0`
		`  decreasing_by apply`
		`Nat.lt_succ_self`
		`  go n`
		`  )`
			`n`

Experiment results

Experiment	Model	Status	Iterations	Verification Burden
gt8	anthropic/claude-sonnet-4-20250514	✅	1	1.525
gt8	anthropic/claude-opus-4-20250514	✅	3	3.664
gt8	openai/gpt-4.1-2025-04-14	✅	1	1.516
gt8	openai/o3-2025-04-16	✅	1	2.036
swap	anthropic/claude-sonnet-4-20250514	✅	3	4.239
swap	openai/gpt-4.1-2025-04-14	⌛	11	—
swap	openai/o3-2025-04-16	✅	1	3.284
facto	anthropic/claude-sonnet-4-20250514	⌛	11	—
facto	anthropic/claude-opus-4-20250514	⌛	11	—
facto	openai/gpt-4.1-2025-04-14	⌛	11	—
facto	openai/o3-2025-04-16	⌛	11	—

Verification burden

The **verification burden** k says that if it costs x tokens to complete the program, then it costs kx tokens to prove it correct.

- Divergence is hardly evidence that the program completion is incorrect, because our proof performance is so poor.

Verification burden vs cost

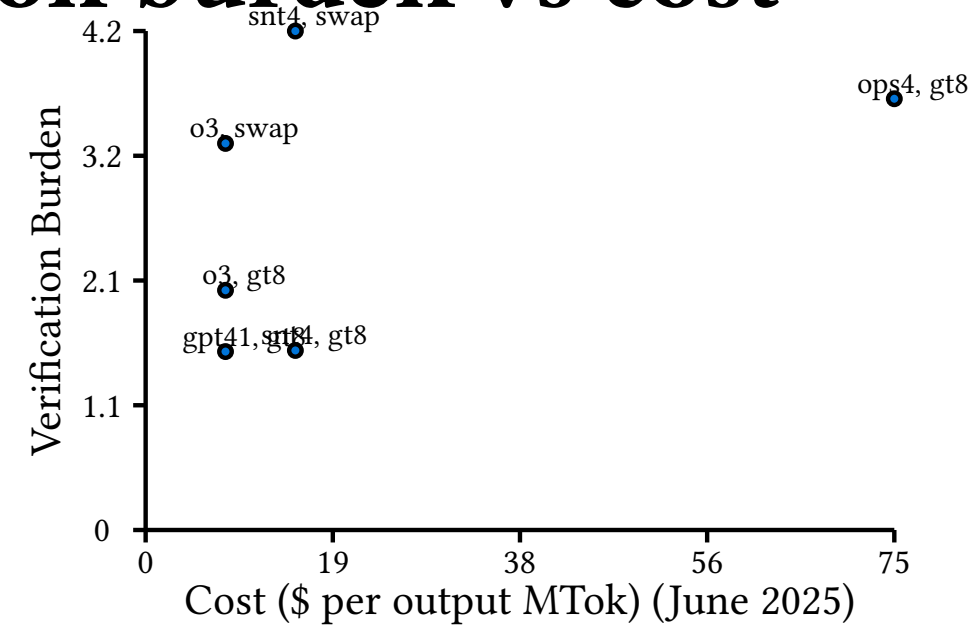


Figure 4: Verification burden vs model cost. Each point represents a successful verification attempt.

Caveat

- **Verification burden is deflated by my custom tactic**. Fewer custom tactics probably means more token cost (or more intelligent model), for the same amount of performance.

IV. Future Work

Non-toy languages and proof stacks

- Make realistic verification burden estimates that would apply to actually useful programming

Elicit subversive code as in [3]

- In Greenblatt et al 2024 they elicit subversive/backdoored code to stress test their protocol.

Bibliography

Bibliography

[1] E. Yudkowsky, “The AI-Box Experiment.” [Online]. Available: https://www.yudkowsky.net/singularity/aibox

[2] G. C. Necula, “Proof-Carrying Code,” in *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, in POPL '97. Paris, France: Association for Computing Machinery, 1997, pp. 106–119. doi: 10.1145/263699.263712.

[3] R. Greenblatt, B. Shlegeris, K. Sachan, and F. Roger, “AI Control: Improving Safety Despite Intentional Subversion.” [Online]. Available: https://arxiv.org/abs/2312.06942

[4] A. Bhatt *et al.*, “Ctrl-Z: Controlling AI Agents via Resampling.” [Online]. Available: https://arxiv.org/abs/2504.10374

[5] D. Dalrymple, “Safeguarded AI: Constructing Guaranteed Safety,” 2024. [Online]. Available: https://www.aria.org.uk/media/3nhijno4/aria-safeguarded-ai-programme-thesis-v1.pdf

[6] P. Kamran, P. Devanbu, and C. Stanford, “Vision Paper: Proof-Carrying Code Completions,” in *39th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW '24)*, New York, NY, USA: ACM, Oct. 2024, p. 7. doi: 10.1145/3691621.3694932.

V.

quinn@beneficialaifoundation.org