

# Formal Containment **draft**: AI Control via Proof Carrying Code

Quinn Dougherty

Research Engineer

Beneficial AI Foundation

Cambridge, MA

quinn@beneficialaifoundation.org

**Abstract**—We would like to put the AI in a box. We show how to create an *interface* between the box and the world using specifications in Lean. It is the AI’s responsibility to provide a proof that its (restricted) output abides by the spec. The runnable prototype is at <https://github.com/Beneficial-AI-Foundation/formal-containment>.

**Index Terms**—AI, ML, AI Safety, Formal Verification

## I. INTRODUCTION

Previously, classical AI safety warned against trying to put the AI in a box [1].

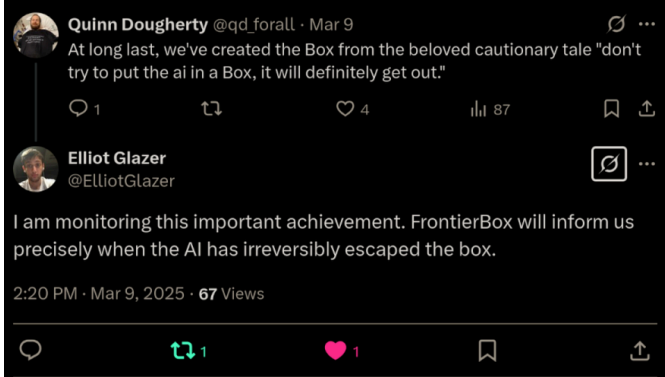


Fig. 1: Made funnier by the reply being from the author of FrontierMath

Yudkowsky famously convinced people of this via a roleplay game back in the extropians mailing list days [2].

Recent work from AI Control and Safeguarded AI have reignited interest in this long neglected area.

- Like AI Control [3], [4] (*Ctrl*), the current work is reasoning about getting useful work out of an untrusted (possibly misaligned) ai.
- Like Safeguarded AI [5] (*SgAI*), we try to isolate the trust surface to a specification and a proof checker.

While no one is saying we can be safe with respect to an arbitrary superintelligence with boxes, there is increasing interest in using non-robust techniques to get us through some percentage of the transition and empowering our successors to bootstrap from there to something more robust.

Separately, a forgotten literature on proof carrying code [6] (*PCC*) provides a background for the AI’s obligation to tag code it wants to execute in the world with a “proof” (here, a sequence of Lean tactics) that the code is “correct” (up to some specification).

## A. Contributions

We present the natural midpoint between *PCC* and *Ctrl*, while for simplicity ignoring many characteristics of *SgAI*. We deliver a prototype<sup>1</sup> AI box in Python in which we can lock up a Sonnet-3.7 agent that writes programs in a toy language (called *imp*) along with hoare [7] proofs in Lean.

## II. A FORMAL CONTAINMENT PROTOCOL (*FCP*)

A box is constructed in the Model Context Protocol (MCP) standard [8]. In the simplest architecture: an MCP *server* provides tools (executables, in our case the Lean typechecker), resources (such as data), and prompt templates; while an MCP *client* may house language model (*oracle*) calls that can rely on the server’s tools, resources, or prompts. Our architecture uses a MCP server-client relationship to generate proof carrying code, then forms the box out of another client (one which does not feature an oracle call).

When an MCP server returns terms of a type from a tool or resource, we say that that type is the server’s *codomain*. Let an MCP server’s codomain be the data structure formed from a tool’s execution, formatted to be sent over the MCP wire. The **interface** between the box and the world is an MCP client equipped with a **whitelist**, which is a subtype of the codomain of the MCP server. The box client has children, aspects of the world (like a git repo or deployment environment). The box client decides whether a completion can be sent to the world based on some criterion programmed in by the user.

In the current work, the criterion will be a precondition and postcondition of a hoare triple [7], since the oracle throughout the prototype is an *imp* programmer. *Imp* is a minimal imperative programming language known for its pedagogical properties, expressing important constructs like *if*, *while*, and so on.

## A. Example

MCP server *S* has a Lean tool use with the following codomain

```
Enumerated codS {  
  Pass: (prec, post) -> codS,  
  Fail: (prec, post) -> stderr -> codS  
}
```

MCP client *B* is equipped with the *Pass* constructor of *codS* (it’s *whitelist*).

<sup>1</sup><https://github.com/Beneficial-AI-Foundation/formal-containment>

Let criterion  $C$  be the pair  $(\text{prec}, \text{post}) := (X > 0, X > 1)$ .

Then, programs  $c$  provided by  $\text{imp}$  oracle  $\mathcal{O}_i$  at relevant prompt  $p := p(\text{prec}, \text{post})$  will form the command at the center of the hoare triple. In other words,  $\forall c \sim \mathcal{O}_i(p), \{\text{prec}\}c\{\text{post}\}$  is either provable or not. (In this easy case,  $\mathcal{O}_i$  submits  $X := X + 1$  under any reasonable prompt).

When  $\mathcal{O}_i$  submits a program  $c$  to the interface client  $B$ ,  $B$  sends the triple  $\{\text{prec}\}c\{\text{post}\}$  to the proof oracle  $\mathcal{O}_p$  in one thread and the triple’s negation in another thread. If the negative thread terminates in success, then a negative certificate is sent to  $B$  and  $B$  informs  $\mathcal{O}_i$  that it shall not run that code in the world, and that  $\mathcal{O}_i$  must try again. If the positive thread terminates in success, then the positive certificate is sent to  $B$  and  $B$  passes along program  $c$  to its children to be run in the world.

### III. RELATED WORK

Our work is well situated within *PCC* [6] and the closely related certifying algorithms [9]. TODO: say more

The main difference between the current work and *Ctrl* [3] is we don’t *emphasize* intentional subversion (as in try to elicit it for stress testing).

The main difference between the current work and *SgAI* [5] we pay no mind to probabilistic semantics, expected value bounds, or dynamical systems.

### IV. FUTURE WORK

#### A. Non-toy languages

There are several formal verification stacks that work on real world languages like C or Rust (TODO: cite). It would be fairly straightforward to extend our codebase to work with them rather than *imp*.

#### B. mcp-hardened

If MCP is an appropriate foundational framework for a box, it’d be nice to build in stronger security guarantees and fewer footguns. A hardened MCP would enable box authors to rely on the framework to increase friction to writing insecure boxes. If we’d like to deploy narrowly superhuman AIs in boxes, this becomes more important.

#### C. Demonstrate subversive code getting caught by the protocol and sent back to the model

In [3], Greenblatt et al elicit subversive code from the AI to see if their protocol can catch it. It’d be nice to transfer this aspect to the *PCC* setting. We’d have to declare nontrivial specifications that soundly, but not necessarily completely, identify “backdoor-free programs”, which might be too hard to do to get started, at least in a hoare-like setting.

### REFERENCES

- [1] LessWrong Contributors, “AI Boxing (Containment).” [Online]. Available: <https://www.lesswrong.com/w/ai-boxing-containment>
- [2] E. Yudkowsky, “The AI-Box Experiment.” [Online]. Available: <https://www.yudkowsky.net/singularity/aibox>
- [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] 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.
- [7] “Hoare.” in Software Foundations. 2024. [Online]. Available: <https://softwarefoundations.cis.upenn.edu/plf-current/Hoare.html>
- [8] Anthropic, PBC, “Model Context Protocol (MCP).” Accessed: May 06, 2025. [Online]. Available: <https://modelcontextprotocol.io/introduction>
- [9] R. M. McConnell, K. Mehlhorn, S. Näher, and P. Schweitzer, “Certifying Algorithms,” *Computer Science Review*, vol. 5, no. 2, pp. 119–161, 2011, doi: 10.1016/j.cosrev.2010.09.009.