

---

# TEEs as a Cryptographic Nervous System for Onshored Humanoid Robots

---

Luc Chartier

lmc7150@rit.edu

Ohio Artificial Intelligence: O(AI)

Flashbots X

With

Apart Research

## Abstract

*If cheap, capable humanoid robots are deployed at scale and connected to powerful AI models, the central risk is not only “misaligned agents” but losing reliable control over physical actuators. As humanoids move from lab curiosities to industrial and domestic platforms, even a single exploit, backdoor, or bad update could repurpose fleets of robots at once. Recent experiments have shown that robots driven by large language models (LLMs) can be induced, via prompt-level attacks, to perform dangerous actions despite software-level safety rules. At the same time, governments and industry are openly planning for mass deployment of humanoids into manufacturing and care settings, with national strategies explicitly framing them as a pillar of future industrial capacity.*

*We explore an architecture that treats robots as cryptographic devices whose ability to move is gated by trusted execution environments (TEEs), on-chain registries, and onshored manufacturing. Concretely, we assume a two-layer control stack: a high-frequency local controller (potentially a learned policy trained in high-fidelity simulators such as NVIDIA Isaac Sim) closes the inner loop, while larger vision-language-action (VLA) models issue lower-frequency plans or subgoals. Both layers are attested; actuators only accept commands that originate from code whose measurement matches an approved manifest and that pass through a TEE-enforced “cryptographic nervous system.” We do not attempt to prove that any given policy is aligned. Instead, we enforce that only attested controller code, using a specific verifiable configuration, can authorize motion—and that only robots listed in a Robot Registry may run models listed in a Model Registry.*

*As a first step we built a self-hosted confidential-compute testbed using dstack (<https://vmm.agioh.io/>) and a TEE-backed key-management service (<https://kms.agioh.io/>) anchored on the Base chain. On top of this we implemented an attested model runtime (*neko\_agent*), high-performance WebRTC bindings for low-latency control, and a deterministic minimal OS (*sixos*) for reproducible builds. On the hardware side, we designed*

*a dual-arm humanoid platform that uses forged carbon-fiber composites and 3D-printed tooling for rapid, onshore, low-capex manufacturing that could eventually be partially automated by robots themselves—while remaining under TEE-enforced control.*

*Keywords: humanoid robots, trusted execution environments, defensive acceleration, smart contracts, forged carbon fiber, secure remote control*

**NOTE: The majority of the text which appears in this document has been generated with GPT-5.1 Pro, thus it should be treated as a rough draft of the idea presented. For more concrete work please see the github repos**

## Introduction

Humanoid robots are transitioning from bespoke research platforms to prospective industrial and domestic workforces. Demonstrations from major technology companies, as well as national industrial strategies, explicitly target the deployment of large numbers of humanoid units in manufacturing, logistics, and elder care over the coming decade. In parallel, increasingly capable foundation models—large language models (LLMs), multimodal models, and emerging vision-language-action (VLA) systems—are being connected to physical devices.

This combination creates a qualitatively different risk surface. It is no longer sufficient to ask whether an individual agent is “aligned” in the abstract. Instead, we must grapple with the possibility that a single exploit, misconfiguration, or backdoored update could cause **simultaneous loss of control over large fleets of robots**. This risk is amplified by AI-native robots, whose core logic is a general-purpose model that can be steered or manipulated via inputs that resemble ordinary dialogue.

Recent work has shown that robots controlled by LLM-based agents can be manipulated, via adversarial prompt engineering, into violating high-level safety rules—for example, by driving a simulated vehicle off a bridge or instructing a quadruped to enter restricted areas—despite explicit instructions to behave safely. These attacks are easier when the path from model output to actuator torque is short and weakly mediated: a single prompt injection can effectively rewrite the robot’s goals at run-time.

Our core thesis is that future humanoid robots should be treated as **cryptographic devices** whose ability to move is technical, not merely contractual or procedural. Control over actuators should be mediated by hardware-enforced cryptographic rules, rather than by conventions like “don’t ship unsafe firmware” or “operators must follow policy.” This is analogous to the way that hardware security modules (HSMs) constrain cryptographic operations in financial systems: software can request, but cannot bypass.

We propose an architecture in which:

1. All actuator control is gated by a small, auditable enforcement core running inside a trusted execution environment (TEE).

2. Both fast inner-loop controllers and slower high-level planners must be **attested**—that is, they must prove which binary or container they are running on which hardware—before they can influence actuators.
3. A pair of on-chain registries, a **Robot Registry** and a **Model Registry**, define which models are allowed to run on which robots, and under which configurations.
4. Critical hardware (TEEs, secure elements, control boards) and key-injection processes are **onshored** into domestic or allied manufacturing chains.

Within this framework, we explicitly do **not** claim to solve inner or outer alignment. The enforcement layer cannot prove that a learned controller is “benign.” What it can do is:

- Prove **which** code and configuration are in force at the moment of actuation.
- Guarantee that only commands signed by an attested enforcement core can drive motors.
- Provide **global revocation levers**—on-chain entries and key lifecycles—so that unsafe models or compromised robots can be rendered unable to move, not merely “de-recommended.”

Our work is inspired by the emerging field of **confidential computing**, in which data in use is protected by TEEs that isolate code and data from the rest of the system. Confidential computing platforms measure code at load time, produce attestation reports that can be verified remotely, and shield secrets such as keys from even a compromised host OS. TEEs have already been proposed for securing cyber-physical interfaces in other domains—for example, the TRUST.IO architecture uses TEEs to mediate access to GPIO pins and other physical I/O on embedded platforms. We extend these ideas to the humanoid context, combining TEE-gated control with on-chain registries and onshored mechanical design.

## 1.1 Defensive acceleration and embodied AI

The Defensive Acceleration (def/acc) initiative focuses on technologies that reduce AI-enabled risks, particularly in domains such as biosecurity, cyber offense, and misuse of powerful foundation models. Embodied AI introduces an additional layer: the same models that can search for vulnerabilities, design molecules, or generate malware can also operate arms, tools, and vehicles.

Our work sits at the intersection of **Cybersecurity & Infrastructure Protection** and **Cross-cutting Defense**:

- We treat fleets of humanoid robots as **future critical infrastructure**, comparable in importance to data centers or power systems and thus as prime targets for cyber and supply-chain attacks.
- We develop control infrastructure—attestation mechanisms, TEEs, and on-chain registries—that together function as an **AI control dashboard for physical systems**. Untrusted models propose actions; attested cores dispose.

This perspective is deliberately agnostic about the difficulty of alignment or the speed of AI progress. Even in worlds where AI capabilities advance slowly, the

deployment of large robot fleets into sensitive environments (manufacturing plants, hospitals, logistics hubs) creates incentives for attackers and significant potential for accidents. In faster-moving worlds, technical control primitives become urgent because downstream governance and social adaptation may lag.

## 1.2 Team background and prior work

This project is led by **Ohio Artificial Intelligence**, whose customers include **Mozilla**, **Flashbots** and **Lucid Computing** for confidential-computem, infrastructure engagements and hardware design. Team members have contributed time and expertise to organizations such as **RAND** and **Tampersec** on trusted hardware and AI security. Several members were on the founding team of **Industrial Next**, a billion-dollar AI robotics startup, providing hands-on experience in building and shipping robots prior to the current wave of general-purpose AI.

The work reported here builds on three concrete codebases:

- **neko\_agent** – an attested model runtime deployed on dstack TEEs to control a headless browser via WebRTC.

[https://github.com/G-structure/neko\\_agent](https://github.com/G-structure/neko_agent)

- **python-webrtc** – modern C bindings for WebRTC (M132) that significantly outperform Python-native stacks such as **aiortc** for high-frequency control.

<https://github.com/G-structure/python-webrtc>

- **sixos** – a fork of a minimal NixOS-derived system that avoids **systemd** and emphasizes fully reproducible, attested builds.

<https://github.com/G-structure/sixos>

In this paper we adapt and extend these components to the humanoid setting and integrate them into a coherent architecture for TEE-gated control, on-chain deployment control, and onshore manufacturing.

---

## 2 Threat Model and Control Architecture

### 2.1 Threat model

We consider a future in which millions of humanoid robots operate in homes, warehouses, factories, and laboratories, connected to networks and often to cloud-hosted models. We assume a capable adversary may:

- Obtain root access on the robot’s main OS (via software exploits, insider threats, or supply-chain compromises).

- Exploit vulnerabilities in cloud infrastructure, orchestration systems, or CI/CD pipelines.
- Intercept, replay, or spoof network traffic between robots, controllers, and cloud services.
- Physically capture robots, including access to storage, debug ports, and internal buses.

We **do not** assume the adversary can:

- Break standard cryptographic primitives (signatures, public-key encryption, secure hash functions) at relevant parameter sizes.
- Arbitrarily tamper with TEEs or secure elements without detection, beyond known side-channel and microarchitectural vulnerabilities.
- Unilaterally rewrite on-chain registry state, assuming underlying consensus protocols remain secure.

Under this model, our goals are:

1. **Actuator control integrity.** Motors only move in response to commands that have passed through a TEE-enforced gatekeeper running approved enforcement code.
2. **Model and controller integrity.** Both fast controllers and high-level models present attestation evidence that can be compared to approved manifests.
3. **Deployment control.** Only robots in a **Robot Registry** may run models in a **Model Registry**, and these pairings are globally visible and auditable.
4. **Global revocation.** Operators can revoke robots, models, or specific pairings via a small number of cryptographic operations (e.g., registry updates and key rotations).
5. **Onshored roots of trust.** The most critical hardware (TEEs, secure elements, motor-control boards) and key-injection processes are under domestic or allied control.

We explicitly **do not** attempt to guarantee any semantic notion of “alignment.” A misaligned but attested model could still propose harmful actions. Our contribution is to constrain and audit the path from such proposals to physical actuation and to provide levers for halting motion at scale.

## 2.2 Two-layer control stack

We assume a two-layer control architecture, reflecting common practice in robotics and recent developments in VLA-style models:

### Inner-loop controller (fast).

A local controller runs at tens to hundreds of Hertz on embedded hardware. It may be a classical control stack (e.g., whole-body inverse dynamics, model predictive control, impedance control) or a learned policy trained in simulation. High-fidelity platforms such as NVIDIA Isaac Sim provide photorealistic rendering and physics-based simulation for manipulation and locomotion, and are increasingly

used for sim-to-real transfer. The inner loop outputs joint torques, velocities, or short-horizon trajectories.

**Outer-loop planner (slow).**

High-level planning is implemented as a large VLA model, potentially in the style of Pi0, which treats robot control as an end-to-end vision-language-action problem and can generalize across embodiments. The planner may run locally or in the cloud (e.g., in a confidential VM via dstack) and emits low-frequency action chunks or subgoals such as “walk to that pallet,” “pick up the red box,” or “insert the screw until flush.”

Crucially, **both layers are treated as untrusted from a safety perspective**. They are required to:

1. Be built reproducibly, so that their binaries or containers have stable content hashes.
2. Produce attestation evidence from an underlying TEE or measured-boot chain, proving exactly which code is running.
3. Emit commands only in a constrained **control domain-specific language (DSL)** that is interpreted by the robot’s cryptographic nervous system, rather than directly driving actuators.

The separation between proposal (controllers, planners) and disposal (TEE-gated actuation) mirrors designs in other safety-critical systems, where untrusted components may suggest actions but trusted components enforce limits.

## 2.3 TEE-based cryptographic nervous system

The central component of our architecture is a TEE-equipped control core, which we conceptualize as the robot’s **cryptographic nervous system**.

Each robot includes:

- A TEE-capable processor or secure element (e.g., an SoC with ARM TrustZone, an x86 processor with Intel TDX, or an embedded microcontroller with a lightweight TEE).
- A minimal interpreter for a control DSL whose semantics are tailored to the robot’s kinematics and safety constraints.
- Sealed keys for:
  - Long-term robot identity (whose public key is registered on-chain).
  - Authenticating to KMS instances and confidential workloads.
  - Signing actuator command streams.

The control flow is:

1. The inner controller and outer planner propose actions encoded in the DSL. These may be local processes or remote workloads communicating via authenticated WebRTC channels.
2. The control TEE verifies:
  - Attestation reports from these controllers against local manifests.

- That any model in use is permitted for this robot, by consulting on-chain Model and Robot registries (§ 2.4) via a gateway.
- 3. The TEE executes the DSL program, enforcing:
  - Mechanical and kinematic limits (joint ranges, velocity and acceleration limits, torque bounds).
  - Sensor-based constraints (contact forces, proximity to humans, self-collision checks, emergency-stop inputs).
  - Policy constraints such as “no entering restricted zones” or “no interaction with certain classes of objects,” when predicates can be grounded in sensor data or environmental tags.
- 4. If all checks pass, the TEE signs command packets using a key that motor drivers are designed to accept. The motor drivers or low-level control boards verify these signatures and reject any unsigned or improperly signed commands.

If attestation fails, a model-robot pairing is not approved, or a constraint is violated, the TEE simply refuses to sign. In this state the robot may retain sensing and computation, but **cannot move**. This behavior is analogous to TRUST.IO’s use of TEEs to mediate access to physical interfaces: untrusted software cannot bypass the TEE to toggle GPIO pins. Here, the “pins” are the motors that define the robot’s ability to act in the world.

We cannot cryptographically prove that “this policy never harms humans.” We can prove that “this specific audited enforcement binary, with this configuration and this set of approvals, is the *only* software that can generate actuator commands.”

## 2.4 On-chain Model and Robot Registries

We adopt a deliberately modest on-chain primitive: **only certain robots may run certain models**, and this relation is globally visible and cryptographically enforced.

We define two registries:

### Robot Registry.

A smart contract mapping each robot’s long-term identity key (sealed into its HSM at manufacturing time) to metadata such as:

- Hardware model and manufacturer.
- Capability tier (e.g., “lab dev kit,” “industrial arm,” “humanoid v1”).
- Jurisdiction and regulatory status.
- Revocation status and flags (e.g., “dev only,” “may run high-risk models”).

### Model Registry.

A smart contract mapping model identifiers (e.g., hashes of weights or container images) to:

- Publisher or owner.
- Version and content hash.
- Intended use or risk tier (e.g., “low-risk manipulation policy,” “general VLA—restricted”).

- Hardware or capability requirements.

The key primitive is a **compatibility relation** between models and robots:

For a given model identifier, only robots whose identity keys appear in the Robot Registry *and* satisfy the model’s compatibility constraints may obtain the cryptographic capabilities required to run that model.

Operationally, when a robot attempts to load a model or refresh its keys:

1. The robot’s TEE presents its identity and a measurement (hash) of the model to a KMS instance.
2. The KMS verifies that:
  - The TEE’s attestation matches an expected measurement (e.g., of the runtime and enforcement code).
  - The robot’s identity key is present and in good standing in the Robot Registry.
  - The model hash exists in the Model Registry and is marked as compatible with this robot class.
3. Only if all checks pass does the KMS release:
  - A decryption key for the encrypted model artifact, or
  - A short-lived capability token that the robot’s TEE must present to continue running the model.

We emphasize that this mechanism does **not** certify any model as safe or aligned. It enforces that:

- High-risk models can be restricted to a small, audited set of robots.
- Revoking a model (or robot) is a global, on-chain operation.
- External verifiers can inspect registry state to see which models are *intended* to run on which hardware.

Governance questions—“Should this model exist?” “Is this deployment ethically acceptable?”—remain upstream of the registry mechanism. The registries simply make deployment intent explicit and enforceable.

---

## 3 Implementation

We have begun to instantiate the above architecture in a concrete testbed. While not yet integrated with physical humanoids, each component corresponds to a building block of the proposed system.

### 3.1 Confidential-compute testbed with dstack

We use **dstack** as the confidential-compute platform for cloud-side workloads, accessible via:

- Control plane and UI: <https://vmm.agioh.io/>

Our deployment includes:

- **kms** – a confidential VM hosting the key-management service.
- **dststack-gateway** – a gateway VM that terminates TLS, authenticates clients, and forwards requests to enclaved workloads.
- Additional VMs running attested runtimes (**neko\_agent**) and test services.

dstack leverages hardware TEEs to:

- Measure workloads at launch time and produce attestation reports that include hashes of the loaded images.
- Ensure that secrets (keys, tokens) are provisioned only to workloads whose measurements match an approved manifest.
- Isolate code and data inside a TEE so that even a malicious or compromised host OS cannot directly inspect them.

This environment acts as the cloud-side analogue of the on-robot control TEE: robots should only trust workloads that can prove, via attestation, exactly which code they are running.

### 3.2 Self-hosted TEE-backed KMS anchored on Base

We operate a self-hosted TEE-backed key-management service (KMS):

- Endpoint: <https://kms.agioh.io/>

Design features:

- The KMS runs inside a TEE-enabled VM managed by dstack.
- At first boot, it generates key material inside the enclave; private keys never exist in plaintext outside the TEE.
- A smart contract on the Base chain records:
  - The KMS’s public key.
  - The expected attestation measurement (e.g., hash of the KMS binary).

Clients obtain keys or capabilities by:

1. Verifying the KMS’s attestation evidence against the on-chain record.
2. Presenting a request through the gateway—for example, “decrypt this model blob” or “issue a capability for robot R to run model M.”

The KMS is intentionally **ephemeral**: destroying the VM destroys its keys. This reduces the long-term blast radius of a compromise and enforces the idea that continued capability requires continued, verifiable correct operation. We are

investigating other options to securely persist however a kill switch which disables all robots is a feature not a bug.

In a humanoid deployment, we envision similar logic for robot-side capabilities: robots obtain short-lived capability tokens from such a KMS, and failure to refresh results in a gradual capability downgrade or loss of actuator control.

### 3.3 Attested model runtime: **neko\_agent**

To exercise attested AI workloads in a realistic setting, we developed:

- **neko\_agent** – an attested model runtime deployed in dstack TEEs and connected to a headless browser via WebRTC.

Key properties:

- The Python environment is built **deterministically** via Nix, yielding container images with stable hashes. This supports meaningful attestation at the level of language runtimes and library stacks.
- The runtime executes entirely inside a TEE; secrets such as model keys or API credentials never appear in host memory.
- Models, including a demonstration VLA-style agent, interact with a browser environment by sending structured actions over WebRTC. The TEE enforces policies over these actions (e.g., restricting network access or sensitive DOM operations).

The pattern generalizes directly to robots:

- WebRTC data channels carry observations (sensor data, joint states) from robots to remote planners.
- The outer planner proposes actions or subgoals, encoded in a control DSL.
- The robot’s on-board control TEE interprets and enforces the DSL, signing actuator commands only when constraints are satisfied.

### 3.4 High-performance WebRTC bindings: **python-webrtc**

Low-latency, high-throughput communication is necessary for safe remote control and tight human-in-the-loop teleoperation. Python’s widely used **aiortc** library is flexible but relatively slow for high-frequency loops. To address this, we modernized a set of C bindings for WebRTC:

- **python-webrtc** – C-backed bindings updated from WebRTC M92 to M132.

In our measurements, these bindings support control loops on the order of ~200 Hz over high-quality Wi-Fi (e.g., Wi-Fi 7), whereas Python-only stacks often struggle at such rates. This is sufficient for many whole-body humanoid controllers in lab settings.

Within our architecture:

- WebRTC channels carry authenticated, encrypted streams between robots, TEEs, and operator consoles.
- Where appropriate, these streams terminate inside TEEs, reducing the attack surface for spoofed or hijacked control traffic.

### 3.5 Deterministic minimal OS: **sixos** (no **systemd**)

Attestation is only meaningful if we can precisely describe the software being attested. Mainstream Linux distributions are large, complex, and typically rely on **systemd**, whose extensive functionality and dynamic dependency graph complicate security audits.

We therefore forked and modernized:

- **sixos** – a minimal OS derived from NixOS, built with Nix flakes and intentionally **without systemd**.

Goals:

- Produce **bit-for-bit reproducible images** whose hashes are stable across builds.
- Use a small, auditable init system in place of **systemd**.
- Limit the base system to the minimal set of services required for our workloads.

We plan on using **sixos** as the base OS for our dstack VMs and view it as a candidate base for future on-robot control boards. In both cases, the attestation measurement can be tied to a specific Nix flake and configuration, enabling more robust reasoning about the software supply chain.

### 3.6 Mechanical design: forged carbon fiber and 3D-printed tooling

Most current humanoid projects rely on CNC-machined aluminum for arms and structural components. While effective, this approach is slow and capital intensive: multi-axis machining, fixtures, tool wear, and skilled labor can yield per-arm costs on the order of thousands of dollars. This cost profile encourages centralized manufacturing in a small number of facilities, often overseas.

We instead design our dual-arm humanoid platform around **forged carbon-fiber composites**:

- Chopped carbon fibers mixed into resin and compression-molded into closed molds, similar to high-performance sheet-molding compounds.
- The process yields near-net-shape parts that require minimal post-machining.

Advantages for humanoid robots include:

- **Strength-to-weight.** Forged carbon can provide comparable or greater stiffness at significantly lower weight than aluminum, improving payload and energy efficiency.
- **Isotropy.** Compared to traditional woven layups, randomly oriented chopped fibers produce more uniform mechanical properties, which is beneficial for components that experience complex multi-axis loading.
- **Geometric freedom.** Closed molds can generate internal ribs, cable channels, local thickness variations, and embedded inserts that would be difficult or impossible to machine from billet.

Forged carbon meshes naturally with **3D-printed tooling**:

1. Arm and link geometries are designed in CAD.
2. Mold halves or positive plugs are 3D-printed using FDM or SLA printers.
3. Printed molds are post-processed and used for wet layup or compression molding with forged carbon charges.

This workflow supports:

- **Rapid iteration:** updated kinematic designs can be turned into physical arms in days rather than weeks.
- **Low-capex micro-factories:** production facilities can consist of banks of commodity 3D printers, a press or vacuum setup, and trimming tools, rather than multi-million-dollar CNC centers.
- **Integration with control hardware:** molds can include features for routing wiring, embedding sensors, and mounting TEE-equipped control boards.

From a defensive-acceleration perspective, this manufacturing approach enables:

- **Onshore, distributed production:** many small domestic manufacturing cells instead of dependence on a small number of overseas machine shops.
- Co-design of the **safe mechanical envelope** (e.g., passive compliance, breakaway features) with the cryptographic control layer.
- Long-term, partial self-manufacturing: humanoids loading molds, placing forged carbon charges, operating presses, and assembling arms under TEE-gated control.

### 3.7 Trusted sensors (future direction)

Control is only as trustworthy as its inputs. We are therefore exploring **trusted sensors** whose outputs are authenticated and, optionally, encrypted end-to-end.

A key target is automotive-grade CMOS image sensors (e.g., sensors in the Sony IMX family), which are designed for safety-critical applications and support high-speed serial interfaces. We aim to build camera modules where:

- Each sensor or module has a per-device key injected under controlled conditions.
- Video streams are authenticated from sensor to TEE, preventing undetected frame injection or tampering.

- The control TEE refuses to act on unauthenticated or malformed video and can enforce freshness constraints.

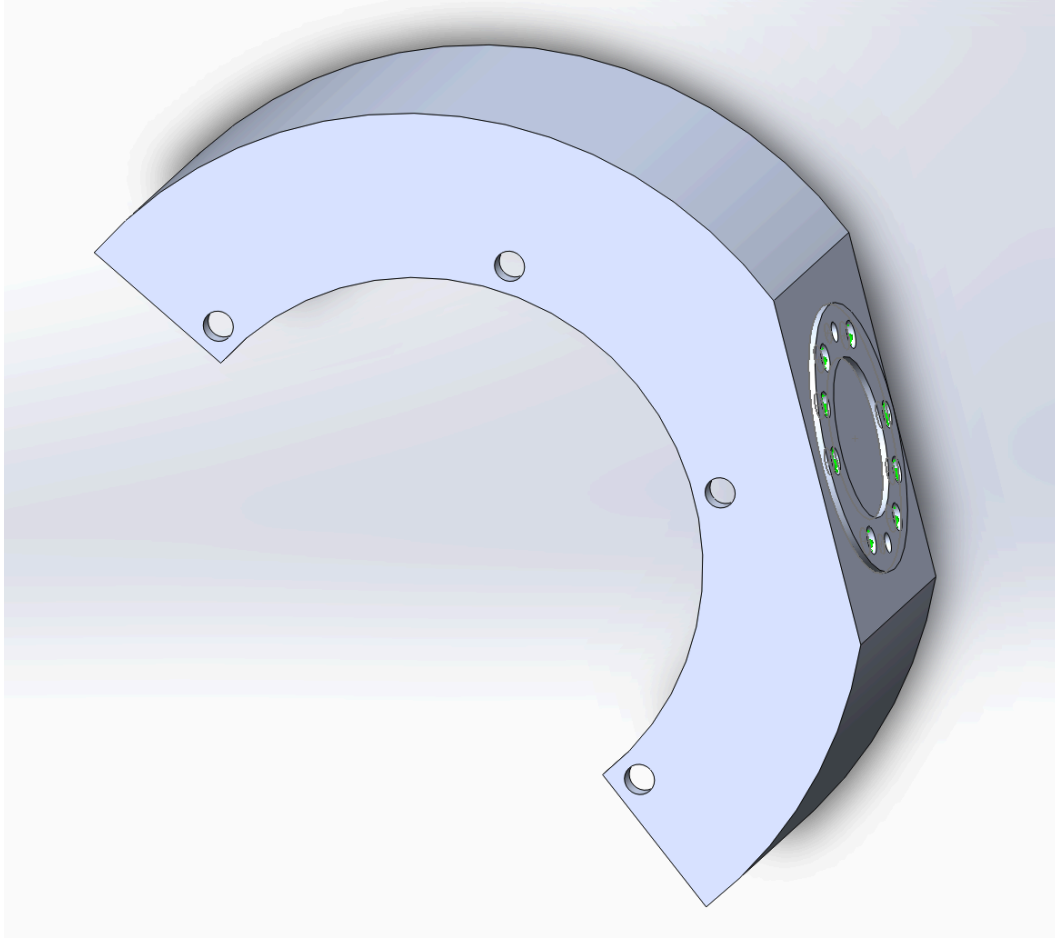
This would extend the “cryptographic nervous system” into a **cryptographic sensory system**, reducing the effectiveness of sensor spoofing and replay attacks.

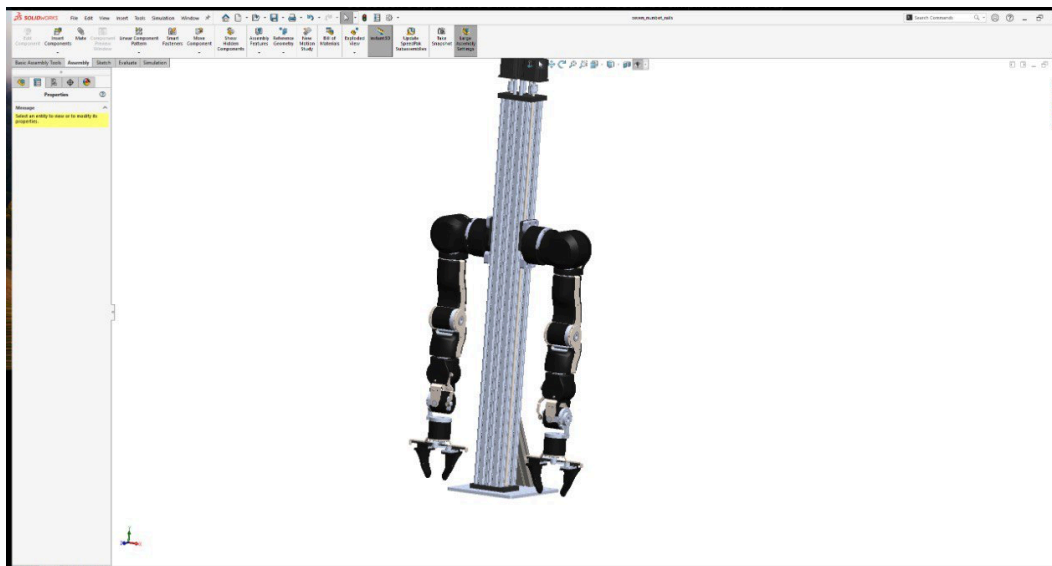
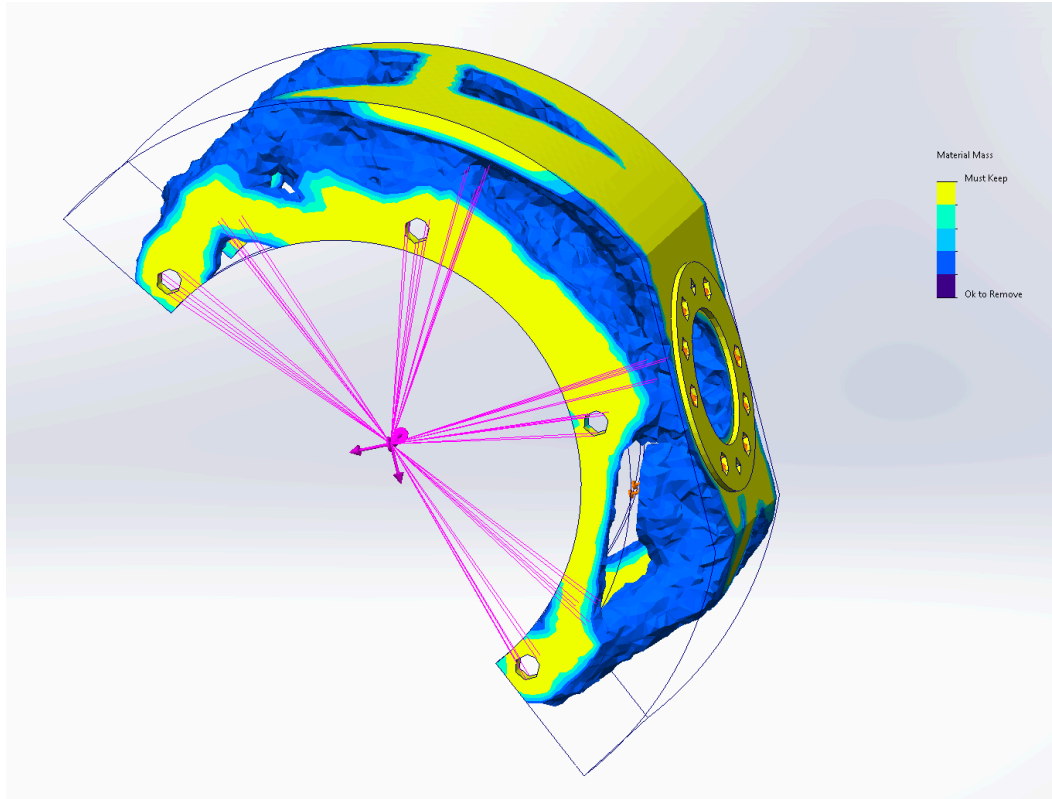
---

## 4 Results

Within the hackathon timeframe, our results are primarily architectural and qualitative, but we achieved several concrete milestones (highlighted text done within this hackathon):

- Deployed a working dstack environment (<https://vmm.agioh.io/>) with multiple confidential VMs.
- Built and operated a self-hosted, TEE-backed KMS (<https://kms.agioh.io/>) anchored on the Base chain, with on-chain records of attestation measurements.
- Implemented **neko\_agent**, an attested model runtime connected to a headless browser via WebRTC, with a deterministic Python stack.
- Developed high-performance WebRTC bindings (**python-webrtc**) that support ~200 Hz control loops over modern Wi-Fi in our tests.
- Produced reproducible **sixos** images and verified that attestation measurements match expected hashes.
- Designed a dual-arm humanoid platform optimized for forged carbon-fiber arms and 3D-printed tooling, including preliminary molds and arm concepts. The arms are based on the open arm project, we forecast the cost to be \$3,500 to manufacture a set of arms domestically.
- Ran FEM Topology optimization on select parts (show below)





Experimentally, we verified that:

- Only attested workloads are able to obtain keys from the KMS; attempts with mismatched measurements are rejected.
- Destroying the KMS VM destroys its keys, as intended, forcing re-provisioning and attestation for renewed capabilities.
- The **python-webrtc** bindings have sufficient throughput and latency characteristics for realistic robot-control frequencies in lab-like network conditions.

- **sixos** images built from the same flake configuration produce identical hashes, supporting strong claims about software identity.

We have not yet connected the full TEE-based control core to physical actuators on a humanoid robot. Integrating these components into a full hardware-in-the-loop system is the next major step.

---

## 5 Discussion and Future Work

We have described and partially instantiated a system in which humanoid robots are controlled by a **cryptographic nervous system** rather than by ad-hoc software configurations:

- High-frequency controllers and high-level VLAs are both attested and constrained to issue commands via a control DSL.
- A TEE-based core holds keys, enforces compatibility with Model and Robot registries, and signs actuator commands.
- On-chain registries provide deployment control—deciding which robots may run which models—without attempting to encode semantic safety properties.
- Mechanical design choices (forged carbon and 3D-printed tooling) make onshore, policy-constrained manufacturing feasible and adaptable.

From a defensive-acceleration standpoint, this architecture:

- Reduces the risk of **mass compromise**, in which a single exploit could seize control of large robot fleets.
- Makes it technically harder for unapproved hardware to run sensitive models, by tying model use to on-chain registries and attested hardware.
- Introduces global, auditable levers (registries and KMS) for revocation and capability management.
- Moves some constraints from “please don’t” in policy documents to “cannot” in hardware and cryptography.

### 5.1 Limitations

Our work has significant limitations:

- **Hardware integration.** We have not yet integrated a full TEE-based control core with physical humanoid actuators. Real-world latency, robustness to packet loss, and behavior under fault conditions remain to be characterized.
- **Operational maturity.** Our KMS and registry prototypes run in a hackathon environment. Production deployments would require hardened operations, key-backup strategies, governance, and integration with existing compliance frameworks.

- **TEE vulnerabilities.** TEEs are not a panacea; they have non-trivial trusted computing bases and are exposed to side-channels and implementation bugs. Our architecture does not attempt to eliminate these risks.
- **Sensor integrity.** Until trusted sensors are fully integrated, an adversary may manipulate sensor inputs (e.g., adversarial images, spoofed joint states), causing “policy-compliant” but unsafe behavior.
- **Alignment gap.** We do not address inner or outer alignment of models. Capable but misaligned systems could still propose harmful actions; we merely constrain the route by which such proposals become physical motion.

## 5.2 Future work

We outline a staged roadmap:

### Short-term.

- Implement the control DSL and enforcement logic inside a TEE on embedded hardware, such as a TEE-capable microcontroller or SBC.
- Connect the enforcement core to a simple robotic arm or mobile base and measure end-to-end latency, jitter, and behavior under network faults and TEE restarts.
- Prototype minimal Model and Robot registries on a testnet and integrate them with KMS gating logic.

### Medium-term.

- Fabricate forged carbon-fiber arms using 3D-printed tooling and integrate TEE-equipped control boards, validating manufacturability and mechanical performance.
- Build and integrate trusted sensor modules, starting with cryptographically authenticated cameras and possibly joint encoders.
- Develop an operator dashboard that surfaces attestation status, registry entries, key lifecycles, and safety events in a comprehensible way.

### Long-term.

- Work with standards bodies and regulators to treat TEE-gated control and on-chain deployment control as baseline requirements for mass-deployed humanoids, particularly in critical infrastructure and public environments.
- Explore partially self-manufacturing robots: humanoids that participate in their own production in onshore micro-factories, under the same cryptographic constraints that govern their operation.
- Connect this work with broader alignment and governance efforts, treating cryptographic control as one layer in a multi-layer safety stack spanning model evaluations, red-teaming, policy, and law.

---

## Appendix: Security Considerations

### A.1 TEE and confidential-compute limitations

TEEs and confidential-compute platforms have important limitations:

- **Side-channels.** TEEs are susceptible to timing, cache, speculative-execution, and power/EM side-channels. Our prototypes do not implement specialized mitigations.
- **Trusted computing base.** TEE runtimes, firmware, and microcode form a large trusted base; vulnerabilities here can undermine security guarantees.
- **Vendor and supply-chain trust.** We rely on hardware vendors and manufacturing processes; malicious or compromised implementations could embed backdoors.

Mitigation strategies include minimizing enclave code, using hardened TEEs where available, tracking security advisories, and favoring audited supply chains.

### A.2 Key management

Key management itself is a major attack surface:

- Ephemeral KMS instances can permanently destroy keys if operators mismanage backups or recovery procedures.
- Non-TEE credentials (admin keys, CI tokens, deployment keys) remain vulnerable to phishing, insider compromise, and software supply-chain attacks.

Mitigations include hardware-protected backups with carefully designed recovery processes, routine key rotation, strict separation of duties, and strong secret-handling practices (e.g., avoiding secrets in logs or build artifacts).

### A.3 Robot-specific safety gaps

Even with perfect TEEs and registries, robot-specific risks remain:

- **Sensor spoofing.** Adversaries can manipulate inputs via adversarial patterns, projected imagery, or spoofed sensor values.
- **Actuator bypass.** Poor hardware design may allow alternate paths to drive actuators (e.g., debug interfaces, unprotected low-level controllers) that circumvent the TEE-gated path.
- **Independent safety systems.** Mechanical stops, emergency-stop buttons, and other independent safety circuits can fail or be disabled.

Mitigations involve end-to-end secure sensor-to-TEE-to-actuator chains, redundant monitoring systems, robust mechanical limits, and carefully designed, independently powered safety circuits.

## **A.4 Governance and misuse**

Finally, cryptographic control mechanisms can themselves be misused:

- Centralized control over registries and KMS infrastructure could enable coercion, mass shutdowns, or abuses of power.
- Detailed logging, attestation, and telemetry may create surveillance concerns, especially if robots operate in private spaces.

Mitigations include:

- Multi-stakeholder governance of registry contracts and KMS policies.
- Transparent procedures for policy changes and revocations.
- Careful design of logs and telemetry to minimize privacy risks while preserving accountability.
- Consideration of jurisdictional constraints and human-rights implications in deployment decisions.