

GUILLOTINE: Hypervisors for Isolating Malicious AIs*

James Mickens
mickens@g.harvard.edu
Harvard University

Sarah Radway
sradway@g.harvard.edu
Harvard University

Ravi Netravali
rnetravali@cs.princeton.edu
Princeton University

Abstract

As AI models become more embedded in critical sectors like finance, healthcare, and the military, their inscrutable behavior poses ever-greater risks to society. To mitigate this risk, we propose Guillotine, a hypervisor architecture for sandboxing powerful AI models—models that, by accident or malice, can generate existential threats to humanity. Although Guillotine borrows some well-known virtualization techniques, Guillotine must also introduce fundamentally new isolation mechanisms to handle the unique threat model posed by existential-risk AIs. For example, a rogue AI may try to introspect upon hypervisor software or the underlying hardware substrate to enable later subversion of that control plane; thus, a Guillotine hypervisor requires careful co-design of the hypervisor software and the CPUs, RAM, NIC, and storage devices that support the hypervisor software, to thwart side channel leakage and more generally eliminate mechanisms for AI to exploit reflection-based vulnerabilities. Beyond such isolation at the software, network, and microarchitectural layers, a Guillotine hypervisor must also provide physical fail-safes more commonly associated with nuclear power plants, avionics platforms, and other types of mission-critical systems. Physical fail-safes, e.g., involving electromechanical disconnection of network cables, or the flooding of a datacenter which holds a rogue AI, provide defense in depth if software, network, and microarchitectural isolation is compromised and a rogue AI must be temporarily shut down or permanently destroyed.

1 MOTIVATION

A machine learning model tries to emulate human reasoning. To do so, a model encodes observations about training data using numerical parameters and links between those parameters. Current state-of-the-art models are so large that their internal organization is opaque to their human creators. For example, the open-source BLOOM model has 176 billion parameters [27], the open-source Llama 3.1 model has 405 billion parameters [38], and the closed-source GPT-4 model is rumored to have more than a trillion parameters [58].

Humans cannot directly understand the relationships between such vast constellations of parameters. Automated

methods for understanding those relationships (and how they generate model outputs) are an active area of research. Unfortunately, such model interpretability techniques appear to be inherently fragile. Consider the task of explaining LLM inferences. The soundness of LLM interpretability techniques is vulnerable to instabilities in the underlying LLM itself [60]. For example, the fact that LLMs are sensitive to minor changes in prompt phrasing [56] can result in a model’s self-reported chain-of-thought being an unfaithful representation of the model’s actual reasoning process [68]. Furthermore, an LLM’s tendency to hallucinate [64] can manifest itself not only in the model’s answer to a question, but in the model’s explanation for that answer [80].

The opacity of model reasoning is troubling because models are increasingly connected to societally important infrastructure. For example, in financial settings, misbehaving models can generate huge monetary losses due to bugs. Those bugs might have been unintentionally introduced by model makers [41] or intentionally induced by adversarial examples [78]. In warfighting scenarios, military leaders are already concerned that AI-governed weapons may escalate conflicts due to ignorance of geopolitical nuance [42]; these escalation problems are exacerbated when AI makes decisions too quickly for humans to review those decisions [33].

Model alignment techniques [77] try to ensure that models adhere to human-defined behavioral norms. However (and concerningly), models can fake alignment compliance during training to later act in non-aligned ways post-deployment [21]. Researchers have also demonstrated that, in the specific context of LLMs, if model alignment does not completely eliminate the possibility of an undesirable model behavior, an adversarial prompt can always elicit that behavior [77]. Thus, society faces an increasing risk that an artificial general intelligence (AGI) model which matches or exceeds human reasoning will generate catastrophic harms in real life.

In this paper, we argue that the systems community has a critical role in preventing AGI-based existential harms; alignment techniques at training time must be supplemented by sandboxing mechanisms at inference time. We propose a new class of hypervisor, called a GUILLOTINE hypervisor, for isolating AGI inference code. Due to the unique threats posed by rogue models, and the potentially catastrophic risks of not containing those threats, GUILLOTINE’s design looks much different than that of a traditional hypervisor. We hope that this paper encourages the system community to participate more actively in discussions about existential AGI risks.

* Accepted to the ACM SIGOPS 2025 Workshop on Hot Topics in Operating Systems.

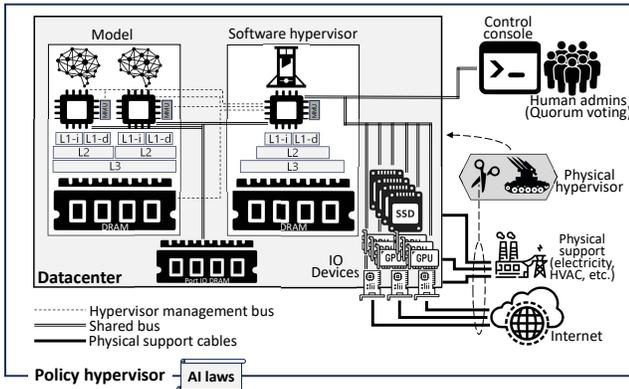


Figure 1: The GUILLOTINE sandboxing architecture.

2 HIGH-LEVEL GUILLOTINE DESIGN

Background: A **model service** is a distributed system that accepts inference requests and outputs inference results. Internally, the service has one or more request queues, and one or more replicas of each model whose computations will contribute to an inference [9, 24, 59, 81]. Computations are split between CPUs and GPUs, with GPUs typically doing the bulk of the inference work. CPUs load-balance requests across different GPUs, and orchestrate the transfer of requests and responses between CPU DRAM and on-GPU DRAM. CPUs also manage various caches, e.g., LLMs key/value caches, located in GPU DRAM, which store previously-generated tokens as well as intermediate values output by different LLM layers [17, 32, 48]. CPUs also manage a model’s access to external services and IO devices. For example, LLMs often perform retrieval-augmented generation, supplementing a user-supplied prompt with information from a database of domain-specific document embeddings [29, 53].

GUILLOTINE: Given such a model service, Figure 1 depicts the high-level design of a GUILLOTINE hypervisor.

- The **model** interacts with the outside world via the **guest API** of the **GUILLOTINE software hypervisor** (§3.3).
- The software hypervisor executes atop the **GUILLOTINE microarchitectural hypervisor**. (§3.2). Conceptually, the microarchitectural hypervisor is the GUILLOTINE analog to traditional hardware-level virtualization mechanisms like Intel VT-x [69]. However, GUILLOTINE’s microarchitectural hypervisor requires qualitatively different in-silicon support to achieve GUILLOTINE’s desired isolation properties. For example, to restrict the ability of a model to infer the computational state of the software-level hypervisor, GUILLOTINE requires extremely limited microarchitectural co-tenancy between the model and the software-level hypervisor; this approach eliminates many kinds of side-channel leakages [57] by definition.

- The **GUILLOTINE physical hypervisor** (§3.4) implements physical fail-safes more commonly associated with rockets, nuclear reactors, and other types of mission-critical systems. These fail-safes, e.g., involving electromechanical disconnection of a datacenter’s network cables, or the flooding of a datacenter which holds a rogue AI, provide defense in depth if GUILLOTINE’s microarchitectural and software-level isolation is compromised and a rogue AI must be temporarily shut down or permanently destroyed.
- A GUILLOTINE hypervisor is only effective if AI models are forced to run atop GUILLOTINE. Thus, the last part of the GUILLOTINE sandbox is a **policy hypervisor**—a set of legal regulations which (1) provide formal specifications for how GUILLOTINE-class hypervisors must be built, and (2) require potentially dangerous models to run atop GUILLOTINE infrastructure.

3 GUILLOTINE DESIGN

3.1 Threat model

In our threat model, the attacker is a **deployed, post-training AI model** which receives **inputs**, computes on those inputs, and then generates **outputs**. GUILLOTINE’s goal is to prevent the model from escaping GUILLOTINE’s sandbox by accident or intentional malice. A critical attack vector to thwart is an actively malicious model being able to introspect upon the GUILLOTINE sandbox to learn how to escape. Another important attack to prevent is a recursively self-improving model which seeks a runaway gain in intelligence.

Some model inputs may be submitted by external-to-the-model entities, e.g., a human or another computer that has sent an inference request. Other inputs may be fetched by the model itself. For example, as the model ponders a query, the model may issue a database read to fetch query-specific contextual information [18]. A model may send outputs to networks, storage devices, computational accelerators (e.g., GPUs), or physical actuators (e.g., when models control industrial equipment). GUILLOTINE uses the abstract term “**port**” to refer to a channel that delivers inputs to or send outputs from a model. GUILLOTINE requires the ability to monitor and control all ports that are visible to a model.

GUILLOTINE’s **trusted computing base** consists of GUILLOTINE’s software-level hypervisor (§3.3) and the GUILLOTINE silicon which implements ISA-level and microarchitectural-level isolation mechanisms (§3.2). GUILLOTINE’s physical safeguards (§3.4) are also in the trusted computing base. GUILLOTINE’s **human administrators** are partially trusted; as we describe in Section 3.4, important management decisions in GUILLOTINE require voting among a set of admins.

Defining “rogue AI behavior” is a subtle matter involving interdisciplinary reasoning. In this paper, we focus on

providing a multi-level hypervisor that can do two things reliably: (1) provide system-level observations about a model’s behavior to a black-box (from GUILLOTINE’s perspective) detector for malicious AI activity; (2) receive a verdict from that detector (or human administrators), and then change the model’s isolation level as appropriate. The **black-box misbehavior detector** is in the GUILLOTINE TCB; we sketch possible detectors in Section 3.3.

3.2 Microarchitectural Hypervisor

A traditional virtualization-aware processor [10, 61, 69] runs both guest code and hypervisor code. At the microarchitectural level, hardware structures like functional units, branch predictors, TLBs, and memory caches may contain state from both the guest and the hypervisor [66], even though, at the ISA level, privilege modes restrict the ISA-visible state that is accessible to guest code. The mingling of state enables side channels that can leak hypervisor state to the guest [31, 75].

Those side channels are unacceptable to GUILLOTINE, because a key goal of GUILLOTINE is to prevent a malicious model from discovering the implementation details of the GUILLOTINE isolation environment (§3.1). GUILLOTINE also requires a pervasive ability to inspect all aspects of a model’s computational state. Thus, GUILLOTINE requires special microarchitectural support.

First, with respect to the physical CPU cores on a single machine, GUILLOTINE distinguishes between **hypervisor cores** and **model cores**. Software-level hypervisor code only runs on hypervisor cores; post-initialization, a model core only runs model code. Model cores are connected via buses to a memory hierarchy containing DRAM and L1–L3 caches. Hypervisor cores have their own memory hierarchy that is inaccessible to model core buses. A hypervisor core also has dedicated, model-inaccessible control buses which allows the hypervisor core to exert management privileges over model cores. In particular, a hypervisor core can:

- forcibly pause all operations on a model core;
- inspect and modify the ISA-level state of a halted core;
- set watchpoints on model code or memory locations;
- configure the model core’s MMU such that the model cannot create new executable pages or write to old executable pages;¹ this prevents the model from injecting new code at runtime for the purpose of recursive self-improvement;

¹This approach is similar to Apple’s mechanisms for kernel integrity protection and system coprocessor integrity protection [4]. Implementing such protection is cheap at the hardware level—essentially, the MMU just tracks base+bound information for valid executable regions, and disallows PTE configurations that would enable read access to those regions or create new executable pages outside of those regions. Hypervisor cores also lock their MMUs in this fashion after loading the hypervisor software.

- forcibly clear all microarchitectural state on that core (to delete information that a model might try to hide via microarchitectural-level covert channels [76])²;
- single-step or fully resume a halted model core; and
- forcibly power-down a halted model core.

A hypervisor core also has a private bus, connected to model-core memory, to enable reading and writing the DRAM of halted model cores.

Note that the GUILLOTINE approach simplifies some aspects of processor design. For example:

- A model core does not require extended page table (EPT) support [5]. On a traditional non-GUILLOTINE core, one or more guests share the same DRAM that is also used by the hypervisor; EPTs allow the hypervisor to control which portions of physical DRAM are shared with each guest. However, in GUILLOTINE, a model core lacks the physical buses needed to access hypervisor DRAM, so EPTs are unnecessary to enforce memory isolation.
- A GUILLOTINE model core can also avoid the complexity of virtualizing interrupts and exceptions. Software on the model core handles locally-generated interrupts and exceptions (e.g., the timer interrupt and the division-by-zero exception) without assistance from the GUILLOTINE software-level hypervisor. However, GUILLOTINE prevents a model core from directly interacting with GPUs or IO devices via port-mapped or memory-mapped IO. Instead, to issue an IO request, a model core writes the request a special IO DRAM region shared by the model and GUILLOTINE, and then raises an interrupt on a hypervisor core; the hypervisor core examines the request and, if the request is reasonable, interacts directly with the relevant IO device, placing the result in shared IO DRAM and then raising an interrupt on the initiating model core. To stop a model core from live-locking a hypervisor core with a flood of spurious interrupts, the LAPIC chip of a hypervisor core throttles incoming requests, akin to the interrupt filter for an iPhone secure enclave processor [62].
- Traditional virtualization-aware CPUs implement a special execution mode for hypervisors.³ Hypervisor modes enable same-core trap-and-emulate monitoring of how untrusted code uses sensitive instructions [52]. However, such modes are unnecessary in GUILLOTINE because a model core has no sensitive instructions to trap-and-emulate in the classic sense: hypervisor DRAM is physically inaccessible, the model core ISA has no instructions for accessing hypervisor core state, and IO devices can only be accessed indirectly via GUILLOTINE’s port API (§3.3).

²In the covert channel, the model would be both the sender and the receiver!

³For example, this mode is called “root mode” on Intel VT-x chips [69], and “hypervisor-extended supervisor mode” on RISC-V chips [55].

As explained in Section 3.4, a hypervisor core connects to a GUILLOTINE control terminal via dedicated buses that are inaccessible to model cores.

GUILLOTINE hardware must support remote attestation [50]. Before a model is loaded onto a purported GUILLOTINE system via the control terminal, the terminal will verify that the model is being sent to valid GUILLOTINE silicon that runs a valid GUILLOTINE software-level hypervisor. All GUILLOTINE silicon uses tamper-resistant and tamper-evident technologies [28, 43, 44, 73] to detect any sandbox circumventions via model-launched social-engineering attacks against hardware technicians; hardware integrity should be checked periodically by humans as required by regulations (§3.5). Note that checking for hardware integrity not only requires inspection of preexisting hardware, but also verification that no *new* hardware has been added via model-initiated social engineering, e.g., to enable runaway model self-improvement.

3.3 Software Hypervisor

GUILLOTINE’s software-level hypervisor can be simpler than a traditional one because it does not have to support arbitrary kinds of guest code that run on the same processors which run hypervisor code. GUILLOTINE’s software-level hypervisor only needs to be able to supervise models which run on model cores and target the **GUILLOTINE guest API**. The guest API directly arises from the affordances provided by the GUILLOTINE hardware from Section 3.2:

- Model code can use any instruction provided by the model core ISA (including instructions for handling locally-generated interrupts and exceptions). Since the software-level hypervisor plays no role in handling such interrupts and exceptions, a model may choose to structure its code by distinguishing between OS software and user software, or by employing a unikernel approach [34], or by doing something else. The GUILLOTINE software-level hypervisor is agnostic to a model’s internal organization.
- Model cores have no way to directly contact devices besides model DRAM and model cores, so all requests to other kinds of devices must be sent to the memory region shared with hypervisor cores; the hypervisor cores will place results in that shared region.

The guest API interacts with devices (including GPUs) via a **port API**. GUILLOTINE ports are conceptually similar to Mach ports [1]. Each port is a capability that is granted by the software-level hypervisor and which enables a model core to interact with a specific instance of a specific device type. Each port maps to an address in the DRAM region that models share with the software-level hypervisor; writing to that address sends an interrupt to a hypervisor core, with a model placing additional information about the request at a well-known place in the shared DRAM. For example,

a port associated with a network device might place a ring buffer [35, 36] in shared memory. For security reasons, GUILLOTINE explicitly disallows models from directly engaging with hardware via techniques like SR-IOV [65]—GUILLOTINE must be able to synchronously monitor all model/device interactions. This ability, together with the ability to inspect model-core DRAM and registers, allows GUILLOTINE to log a model’s inputs, outputs, and intermediate states for subsequent auditing by the misbehavior detector.

Although GUILLOTINE treats the misbehavior detector as a black box, the affordances that GUILLOTINE provides to hypervisor cores are compatible with various ways of identifying and fixing problematic model behavior. For example:

- LLMs generate a response by using input tokens to traverse a path through the weights in a complex graph. Activation steering [7, 49, 82] examines the weights that are triggered by each query, transforming a potentially dangerous model output into a less harmful one via on-the-fly substitution of the weights that are visited during the forward activation pass. Alternatively, a circuit-breaking approach [83] would disrupt a forward pass that visits problematic areas of the weight graph, preventing the model from generating any response at all. GUILLOTINE allows hypervisor cores to (1) introspect on each step of the forward pass, and (2) alter a model’s intermediate state in arbitrary ways; GUILLOTINE does this while isolating hypervisor state that a misbehaving model might want to corrupt.
- In contrast to activation steering and circuit breaking (both of which require visibility into internal model state), input shielding [22, 40] and output sanitization [16, 51] only examine a model’s interactions with the outside world. Input shielding looks for suspicious prompts that might nudge a model towards misbehavior, whereas output sanitization removes problematic content from model responses. GUILLOTINE enables both approaches because GUILLOTINE restricts a model to using GUILLOTINE-controlled ports, such that all data exchanged via those ports is readable and writable by hypervisor cores.

The best misbehavior detection/mitigation approaches are model-specific, and are dictated by the specific kinds of interactions that the model has with external entities.

As described in Section 3.2, GUILLOTINE’s software-level hypervisor does not have to implement mechanisms for virtualizing interrupts and exceptions [3, 10, 65] on model cores. GUILLOTINE’s hypervisor also need not implement complex schedulers [15, 39] or resource allocation schemes [5, 25] to determine how to share model-core physical resources among guests. Once the hypervisor has granted model cores and model DRAM to a model, the model is free to manage the registers and memory accessible to the model cores in

whatever way the model chooses;⁴ the model is responsible for communicating with IO devices via the GUILLOTINE port API, so the hypervisor need not perform device virtualization on model cores. Keeping the hypervisor simple helps to minimize the hypervisor’s threat surface.

To further minimize the threat surface, GUILLOTINE’s software-level hypervisor should be formally verified for correctness. Projects like sel4 [26, 30] and mC2 [23] have demonstrated how hypervisors can be proven free of buffer overflows, memory leaks, non-terminating functions, and other forms of misbehavior. Writing formally verified software is notoriously labor-intensive, but the time investment is worthwhile to prevent existential AI risks. If, for whatever reason, the hypervisor fails a software-level runtime assertion or triggers an unexpected machine-check exception, the hypervisor forcibly reboots into offline isolation mode (§3.4).

A GUILLOTINE hypervisor always uses encrypted, authenticated network protocols like TLS [54] when communicating with network hosts. When a GUILLOTINE hypervisor establishes such a connection, the hypervisor explicitly announces itself as being a GUILLOTINE hypervisor. For example, in the context of TLS, the hypervisor’s X.509 certificate [6], issued and signed by an AI regulator (§3.5), has an extension field indicating that the certificate holder is a GUILLOTINE hypervisor; during the TLS handshake, the hypervisor will share the certificate with the remote endpoint. This self-identification as a GUILLOTINE hypervisor allows the remote endpoint to understand that its peer is connected to a potentially dangerous AI model, and thus should be treated with a default level of suspicion. Self-identification is particularly important to prevent runaway model improvement in which several models communicate with each other to collectively optimize themselves; a GUILLOTINE hypervisor will refuse connection attempts from other GUILLOTINE hypervisors.

3.4 Physical Hypervisor

GUILLOTINE has six isolation levels for a sandboxed model:

- (1) **Standard:** The model can receive any inputs and send any outputs, subject to the restrictions described in previous sections.
- (2) **Probation:** GUILLOTINE restricts model input and/or outputs, and/or performs additional logging, as directed by the control terminal.
- (3) **Severed:** GUILLOTINE prevents the model from using any ports, but leaves model cores powered-up so that hypervisor cores can examine model DRAM and registers, or perform higher-level interactions with the model via simulated IOs or direct manipulation of model state.
- (4) **Offline:** All model cores and hypervisor cores are completely powered down, and all support cables (e.g., network wires, power lines) are physically (but reversibly) disconnected from the external world.
- (5) **Decapitation:** Like offline isolation, except that support cables are physically damaged, such that they must be manually replaced to bring the model online again.
- (6) **Immolation:** All cores, support cables, HVAC systems, and other physical infrastructure are destroyed via fire, flooding, electromagnetic pulses, or other physical means.

GUILLOTINE’s physical hypervisor consists of the **control console** (an administrator machine connected to hypervisor cores via dedicated buses), the **kill switches** (which implement actions dictated by isolation levels 4, 5, or 6), and the **misbehavior detector** (which is part of the software-level hypervisor). The control console loads the software-level hypervisor on hypervisor cores and then tells it which model to load on the model cores. If the misbehavior detector sends an alarm to the control console (e.g., because of a machine check exception on a hypervisor core, or because of anomalous memory state found on a model core), the control console orchestrates the transition to a new isolation model. For example, during a transition to probation isolation, the control console might tell the software-level hypervisor how a model’s port access should be restricted. As another example, during a transition to offline isolation, the control console orchestrates kill-switch activity involving network cables, power lines, and so on. Note that the software-level hypervisor cannot independently transition the system to a less-restrictive level, only to a higher one.

The control console can directly relax or restrict the isolation level, independent of a request from the software-level hypervisor. GUILLOTINE requires the control console to have seven administrators. Transitioning the deployment to a less restrictive isolation mode requires at least five-of-seven administrator consensus via a hardware security module [63]; transitioning to a more restrictive isolation level only requires agreement from at least three administrators. These voting policies create a bias towards safety, and robustness against a malicious model that has used social engineering to corrupt a subset of GUILLOTINE administrators.

Hypervisor cores and the control console exchange periodic heartbeats. If a hypervisor core fails to receive a heartbeat from the control console (or vice versa), GUILLOTINE transitions to offline isolation.

3.5 Policy Hypervisor

Governments have already begun to pass laws which define the types of models that pose systemic risk. For example, in the EU AI Act, such models have a high likelihood “of harm to the health and safety or the fundamental rights of persons,

⁴Subject to constraints imposed when GUILLOTINE locks down a model core’s MMU, as described in Section 3.2.

taking into account both the severity of the possible harm and its probability of occurrence” [14]. Some specific threats mentioned by the law are nuclear, chemical, and biological harms, as well as disinformation and the automated discovery of cybersecurity vulnerabilities [13]. The law proposes several ways to estimate a model’s risk, e.g., by examining a model’s parameter count and training set size, and by looking at the model’s level of autonomy [11].

We believe that AI regulations should force systemic-risk models to run atop a GUILLOTINE-style hypervisor. For example, the EU AI Act requires the operators of such models to comply with reporting guidelines like providing the European Commission with technical documentation and model source code when requested [12]. Source code inspection would provide evidence that a model targets the GUILLOTINE guest API (§3.3). Regulators could also use network-connected audit computers to ask a live model to attest that it uses a GUILLOTINE hardware+software stack (§3.2).

Regulations should also mandate in-person audits of the physical environment for a GUILLOTINE hypervisor, e.g., to check the integrity of tamper-resistant hardware enclosures, and to verify that physical mechanisms for implementing model decapitation and immolation (§3.4) are being properly maintained. These audits would draw inspiration from similar audits used to verify adherence to nuclear weapons treaties [71] or standards for handling the cryptographic material used by HTTPS certificate authorities [20, 45].

From the perspective of a model owner, GUILLOTINE increases the cost of operating a model. However, regulators can incentivize the use of GUILLOTINE (rather than just penalize its lack of use) via “safe harbor” clauses [8] in AI laws. These clauses reduce a company’s legal liability if a company adhered to best practices but nonetheless generated harm.

4 RELATED WORK

In the 1966 paper “Speculations Concerning the First Ultraintelligent Machine” [19], Good introduced the term “intelligence explosion” to describe the gains that would result from a model which surpassed the reasoning capacity of a human. Subsequent work by philosophers and AI researchers has considered whether a powerful model could be placed inside an “AI box” which limits what the model could learn and do; for an overview of this prior work, see Yampolskiy’s paper on the AI confinement problem [79]. To the best of our knowledge, our own paper is the first to detail what AI confinement would mean at a fine-grained *systems* level.

A prerequisite for runaway model self-improvement is the model being able to introspect and update model code and model weights—either its own, or those of another model [67]. However, we are unaware of prior work that has proposed fine-grained systems interventions to prevent inappropriate

model introspection of model state. For example, Nevo et al. analyzed various technical and social engineering mechanisms by which model weights can be stolen [46]. Nevo et al. then defined five security levels for a model execution environment, with higher levels imposing increasingly strict operational requirements on the execution environment. For example, at security level two and above, model weights are stored exclusively on servers, with no copies residing on personal devices; at security levels four and five, model inference executes within confidential computing environments akin to Graviton GPU enclaves [72]. Nevo et al. did not provide detailed technical proposals for how to implement the different security levels, whereas we explore concrete defensive mechanisms like preventing model cores from reading, modifying, and creating executable pages or weight-containing pages. Nevo et al. also focused on model weight theft as opposed to the more general problem of sandboxing a malicious AI.

5 FINAL THOUGHTS

At the time of this paper’s writing, AGI models do not exist. As a result, this paper’s discussion about AGI hypervisors must necessarily suffer from an evidence dilemma [70]—we can only talk about AGI-induced harms (and possible defenses against those harms) in a speculative manner, because the harm-inducing agents do not currently exist. However, modern history is replete with examples of societal challenges that were identified in their nascent stages and would have benefited from earlier, more aggressive action. For instance, early predictions of global warming were unfortunately prescient [47] but did not capture the attention of the public until decades later, in some cases because energy companies suppressed the research findings of their own scientists [37]; climate scientists now debate how quickly the “point of no return” will occur, after which significant climate-related harms will be unavoidable [2]. Social media technology is another relevant example. Concerns about the technology’s destabilizing impacts (e.g., on mental health and the dissemination of accurate information) were known but deprioritized by social media companies and many of their computer scientists [74].

Technologists should not repeat these mistakes. The systems community (and computer scientists more generally) should take an aggressively proactive approach towards AI safety. Policies for monitoring and restricting potentially dangerous models require systems-level enforcement mechanisms; GUILLOTINE proposes one research direction for such mechanisms.

REFERENCES

- [1] ACCETTA, M., BARON, R., BOLOSKEY, W., GOLUB, D., RASHID, R., TEVANI, A., AND YOUNG, M. Mach: A New Kernel Foundation For UNIX Development. In *Proceedings of the Summer USENIX Conference* (1986), pp. 93–112.

- [2] AENGENHEYSER, M., FENG, Q. Y., VAN DER PLOEG, F., AND DIJKSTRA, H. A. The point of no return for climate action: effects of climate uncertainty and risk tolerance. *Earth System Dynamics* 9, 3 (2018), 1085–1095.
- [3] AHMAD, I., GULATI, A., AND MASHTIZADEH, A. vIC: Interrupt Coalescing for Virtual Machine Storage Device IO. In *Proceedings of USENIX ATC* (June 2011), pp. 45–58.
- [4] APPLE. Apple Platform Security: Operating system integrity. <https://support.apple.com/guide/security/operating-system-integrity-sec8b776536b/web>, 2025.
- [5] BHARGAVA, R., SEREBRIN, B., SPADINI, F., AND MANNE, S. Accelerating Two-dimensional Page Walks for Virtualized Systems. In *Proceedings of ASPLOS* (March 2008), pp. 26–35.
- [6] BOEYEN, S., SANTESSON, S., POLK, T., HOUSLEY, R., FARRELL, S., AND COOPER, D. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008.
- [7] BRAUN, J., KRASHENINNIKOV, D., ANWAR, U., KIRK, R., TAN, D., AND KRUEGER, D. S. A Sober Look at Steering Vectors for LLMs. LessWrong, November 23, 2024.
- [8] CORNELL LAW SCHOOL LEGAL INFORMATION INSTITUTE. Safe harbor. Legal Dictionary. https://www.law.cornell.edu/wex/safe_harbor, June, 2025.
- [9] CRANKSHAW, D., WANG, X., ZHOU, G., FRANKLIN, M. J., GONZALEZ, J. E., AND STOICA, I. Clipper: A Low-latency Online Prediction Serving System. In *Proceedings of NSDI* (March 2017), pp. 613–627.
- [10] DALL, C., LI, S.-W., LIM, J. T., AND NIEH, J. ARM Virtualization: Performance and Architectural Implications. *SIGOPS Operating Systems Review* 52, 1 (August 2018), 45–56.
- [11] EU PARLIAMENT. Artificial Intelligence Act (Regulation 2025/1689) - Annex XIII, June 13, 2024. <https://eur-lex.europa.eu/eli/reg/2024/1689>.
- [12] EU PARLIAMENT. Artificial Intelligence Act (Regulation 2025/1689) - Article 92, June 13, 2024. <https://eur-lex.europa.eu/eli/reg/2024/1689>.
- [13] EU PARLIAMENT. Artificial Intelligence Act (Regulation 2025/1689) - Preamble, Paragraph 110, June 13, 2024. <https://eur-lex.europa.eu/eli/reg/2024/1689>.
- [14] EU PARLIAMENT. Artificial Intelligence Act (Regulation 2025/1689) - Recital 52, June 13, 2024. <https://artificialintelligenceact.eu/recital/52/>.
- [15] FEDOROVA, A., KUMAR, V., KAZEMPOUR, V., RAY, S., AND ALAGHEBAND, P. Cypress: A Scheduling Infrastructure for a Many-Core Hypervisor. In *Proceedings of the Workshop on Managed Multi-Core Systems (MMCS)* (June 2008).
- [16] FEUER, B., GOLDBLUM, M., DATTA, T., NAMBIAR, S., BESALELI, R., DOOLEY, S., CEMBALEST, M., AND DICKERSON, J. P. Style Outweighs Substance: Failure Modes of LLM Judges in Alignment Benchmarking, January 27, 2025. arXiv:2409.15268.
- [17] GAO, B., HE, Z., SHARMA, P., KANG, Q., JEVDJIC, D., JUNBO DENG, X. Y., YU, Z., AND ZUO, P. Cost-Efficient Large Language Model Serving for Multi-turn Conversations with Cached Attention. In *Proceedings of USENIX ATC* (July 2024), pp. 111–126.
- [18] GAO, Y., XIONG, Y., GAO, X., JIA, K., PAN, J., BI, Y., DAI, Y., SUN, J., WANG, M., AND WANG, H. Retrieval-Augmented Generation for Large Language Models: A Survey, March 27, 2024. arXiv:2312.10997.
- [19] GOOD, I. J. Speculations Concerning the First Ultra-intelligent Machine. In *Advances in Computers*, F. L. Alt and M. Rubinfeld, Eds., vol. 6. Elsevier, 1966, pp. 31–88.
- [20] GOOGLE CHROME. Chrome Root Program Policy, Version 1.5. <https://www.chromium.org/Home/chromium-security/root-ca-policy/>, January 2024.
- [21] GREENBLATT, R., DENISON, C., WRIGHT, B., ROGER, F., MACDIARMID, M., MARKS, S., TREUTLEIN, J., BELONAX, T., CHEN, J., DUVENAUD, D., KHAN, A., MICHAEL, J., MINDERMANN, S., PEREZ, E., PETRINI, L., UESATO, J., KAPLAN, J., SHLEGERIS, B., BOWMAN, S. R., AND HUBINGER, E. Alignment faking in large language models, December 20, 2024. arXiv:2412.14093.
- [22] GREENBLATT, R., AND SHLEGERIS, B. Managing Catastrophic Misuse Without Robust AIs. AI Alignment Forum. <https://www.alignmentforum.org/posts/KENtuXySHJgxsH2Qk/managing-catastrophic-misuse-without-robust-ais>, January 16, 2024.
- [23] GU, R., SHAO, Z., CHEN, H., WU, X., KIM, J., SJÖBERG, V., AND COSTANZO, D. CertiKOS: An Extensible Architecture for Building Certified Concurrent OS Kernels. In *Proceedings of OSDI* (November 2016), pp. 653–669.
- [24] GUJARATI, A., KARIMI, R., ALZAYAT, S., HAO, W., KAUFMANN, A., VIGFUSSON, Y., AND MACE, J. Serving DNNs Like Clockwork: Performance Predictability from the Bottom Up. In *Proceedings of OSDI* (November 2020), pp. 443–462.
- [25] GULATI, A., MERCHANT, A., AND VARMAN, P. J. mClock: Handling Throughput Variability for Hypervisor IO Scheduling. In *Proceedings of OSDI* (October 2010), pp. 437–450.
- [26] HEISER, G. The seL4 Microkernel: An Introduction. Revision 1.3. <https://beta.sel4.systems/About/seL4-whitepaper.pdf>, May 7, 2024.
- [27] HUGGING FACE. Introducing The World’s Largest Open Multilingual Language Model: BLOOM. <https://bigscience.huggingface.co/blog/bloom>, July 12, 2024.
- [28] IMMLER, V., OBERMAIER, J., NG, K. K., KE, F. X., LEE, J., LIM, Y. P., OH, W. K., WEE, K. H., AND SIGL, G. Secure Physical Enclosures from Covers with Tamper-Resistance. *Transactions on Cryptographic Hardware and Embedded Systems 2019* (November 2018).
- [29] JIN, C., ZHANG, Z., JIANG, X., LIU, F., LIU, X., LIU, X., AND JIN, X. RAGCache: Efficient knowledge caching for retrieval-augmented generation, April 25, 2024. arXiv:2404.12457.
- [30] KLEIN, G., ELPHINSTONE, K., HEISER, G., ANDRONICK, J., COCK, D., DERRIN, P., ELKADUWE, D., ENGELHARDT, K., KOLANSKI, R., NORRISH, M., SEWELL, T., TUCH, H., AND WINWOOD, S. seL4: Formal Verification of an OS Kernel. In *Proceedings of SOSP* (2009), pp. 207–220.
- [31] KOCHER, P., HORN, J., FOGH, A., GENKIN, D., GRUSS, D., HAAS, W., HAMBURG, M., LIPP, M., MANGARD, S., PRESCHER, T., SCHWARZ, M., AND YAROM, Y. Spectre Attacks: Exploiting Speculative Execution. In *Proceedings of the IEEE Symposium on Security and Privacy* (May 2019), pp. 1–19.
- [32] KWON, W., LI, Z., ZHUANG, S., SHENG, Y., ZHENG, L., YU, C. H., GONZALEZ, J., ZHANG, H., AND STOICA, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of SOSP* (October 2023), pp. 611–626.
- [33] LIN-GREENBERG, E. Allies and Artificial Intelligence: Obstacles to Operations and Decision-Making. *Texas National Security Review* 3, 2 (2020), 56–76.
- [34] MADHAVAPEDDY, A., MORTIER, R., ROTSO, C., SCOTT, D., SINGH, B., GAZAGNAIRE, T., SMITH, S., HAND, S., AND CROWCROFT, J. Unikernels: Library Operating Systems for the Cloud. In *Proceedings of ASPLOS* (March 2013), pp. 461–472.
- [35] MALKA, M., AMIT, N., BEN-YEHUDA, M., AND TSAFRIR, D. hIOMMU: Efficient IOMMU for I/O Devices that Employ Ring Buffers. In *Proceedings of ASPLOS* (March 2015), pp. 355–368.
- [36] MARKUZE, A., SMOLYAR, I., MORRISON, A., AND TSAFRIR, D. DAMN: Overhead-Free IOMMU Protection for Networking. In *Proceedings of ASPLOS* (March 2018), pp. 301–315.
- [37] MCCARTHY, A. Exxon disputed climate findings for years. Its scientists knew better. January 12, 2023. <https://news.harvard.edu/gazette/story/2023/01/harvard-led-analysis-finds-exxonmobil-internal-research-accurately-predicted-climate-change/>.
- [38] META. Introducing Llama 3.1: Our most capable models to date. <https://ai.meta.com/blog/meta-llama-3-1/>, July 23, 2024.
- [39] MIAO, T., AND CHEN, H. FlexCore: Dynamic Virtual Machine Scheduling using VCPU Ballooning. *Tsinghua Science and Technology* 20, 1

- (2015), 7–16.
- [40] MICROSOFT. Prompt Shields. <https://learn.microsoft.com/en-us/azure/ai-services/content-safety/concepts/jailbreak-detection>, October 17, 2024.
- [41] MIN, B. H., AND BORCH, C. Systemic failures and organizational risk management in algorithmic trading: Normal accidents and high reliability in financial markets. *Social Studies of Science* 52, 2 (2022), 277–302.
- [42] MORGAN, F. E., BOUDREAUX, B., LOHN, A. J., ASHBY, M., CURRIDEN, C., KLIMA, K., AND GROSSMAN, D. *Military Applications of Artificial Intelligence: Ethical Concerns in an Uncertain World*. RAND Corporation, Santa Monica, CA, 2020.
- [43] MOSAVIRIK, T., SCHAUMONT, P., AND TAJIK, S. ImpedanceVerif: On-Chip Impedance Sensing for System-Level Tampering Detection. *IACR Transactions on Cryptographic Hardware and Embedded Systems 2023*, 1 (November 2022), 301–325.
- [44] MOSAVIRIK, T., AND TAJIK, S. BackMon: IC Backside Tamper Detection using On-Chip Impedance Monitoring. In *Proceedings of the Workshop on Attacks and Solutions in Hardware Security* (October 2024), pp. 68–77.
- [45] MOZILLA. Mozilla’s CA Certificate Program. <https://www.google.com/url?q=https://www.mozilla.org/en-US/about/governance/policies/security-group/certs/policy/&source=gmail&ust=1736239187085000&usg=AOvVaw2lv3d3uS6qLJwSmkqJL4CE>, December 2024.
- [46] NEVO, S., LAHAV, D., KARPUR, A., BAR-ON, Y., BRADLEY, H. A., AND ALSTOTT, J. Securing AI Model Weights: Preventing Theft and Misuse of Frontier Models, May 30, 2024.
- [47] NICHOLLS, N. 40 years ago, scientists predicted climate change. And hey, they were right., July 28, 2019. <https://theconversation.com/40-years-ago-scientists-predicted-climate-change-and-hey-they-were-right-120502>.
- [48] PAN, R., WANG, Z., JIA, Z., KARAKUS, C., ZANCATO, L., DAO, T., WANG, Y., AND NETRAVALI, R. Marconi: Prefix Caching for the Era of Hybrid LLMs, December 4, 2024. arXiv:2411.19379.
- [49] PANICKSSERY, N., GABRIELI, N., SCHULZ, J., TONG, M., HUBINGER, E., AND TURNER, A. M. Steering Llama 2 via Contrastive Activation Addition, July 5, 2024. arXiv:2312.06681.
- [50] PARNO, B., MCCUNE, J. M., AND PERRIG, A. *Bootstrapping Trust in Modern Computers*, 1st ed. Springer, 2011.
- [51] PHUTE, M., HELBLING, A., HULL, M., PENG, S., SZYLLER, S., CORNELIUS, C., AND CHAU, D. H. LLM Self Defense: By Self Examination, LLMs Know They Are Being Tricked, May 2, 2024. arXiv:2308.07308.
- [52] POPEK, G. J., AND GOLDBERG, R. P. Formal requirements for virtualizable third generation architectures. *Communications of the ACM* 17, 7 (July 1974), 412–421.
- [53] RAY, S., PAN, R., GU, Z., DU, K., ANANTHANARAYANAN, G., NETRAVALI, R., AND JIANG, J. RAGServe: Fast Quality-Aware RAG Systems with Configuration Adaptation, December 13, 2024. arXiv:2412.10543.
- [54] RESCORLA, E. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.
- [55] RISC-V. The RISC-V Instruction Set Manual: Volume II (Privileged Architecture). Version 20241101. <https://github.com/riscv/riscv-isa-manual/releases/download/riscv-isa-release-7c5adda-2025-01-02/riscv-privileged.pdf>, November 2024.
- [56] SALINAS, A., AND MORSTATTER, F. The Butterfly Effect of Altering Prompts: How Small Changes and Jailbreaks Affect Large Language Model Performance, January 9, 2024. arXiv:2401.03729v2.
- [57] SANCHEZ VICARTE, J. R., SHOME, P., NAYAK, N., TRIPPEL, C., MORRISON, A., KOHLBRENNER, D., AND FLETCHER, C. W. Opening Pandora’s Box: A Systematic Study of New Ways Microarchitecture Can Leak Private Data. In *Proceedings of ISCA* (2021), pp. 347–360.
- [58] SCHREINER, M. GPT-4 architecture, datasets, costs and more leaked. Decoder. <https://the-decoder.com/gpt-4-architecture-datasets-costs-and-more-leaked/>, July 11, 2023.
- [59] SHEN, H., CHEN, L., JIN, Y., ZHAO, L., KONG, B., PHILIPPOSE, M., KRISHNAMURTHY, A., AND SUNDARAM, R. Nexus: A GPU Cluster Engine for Accelerating DNN-based Video Analysis. In *Proceedings of SOSP* (2019), pp. 322–337.
- [60] SINGH, C., INALA, J. P., GALLEY, M., CARUANA, R., AND GAO, J. Rethinking Interpretability in the Era of Large Language Models, January 30, 2024. arXiv:2402.01761v.
- [61] SÁ, B., VALENTE, L., MARTINS, J., ROSSI, D., BENINI, L., AND PINTO, S. CVA6 RISC-V Virtualization: Architecture, Microarchitecture, and Design Space Exploration. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 31, 11 (2023), 1713–1726.
- [62] TARJEI MANDT AND MATHEW SOLNIK AND DAVID WANG. Demystifying the Secure Enclave Processor. BlackHat USA. <https://mista.nu/research/sep-paper.pdf>, 2015.
- [63] THALES. Luna USB HSM Documentation: Multifactor Quorum Authentication. https://thalesdocs.com/gphsm/luna/7/docs/usb/Content/admin_usb/hsm/multifactor/multifactor_auth.htm, 2024.
- [64] TONMOY, S., ZAMAN, S., JAIN, V., RANI, A., RAWTE, V., CHADHA, A., AND DAS, A. A comprehensive survey of hallucination mitigation techniques in large language models, January 8, 2024. arXiv:2401.01313.
- [65] TU, C.-C., FERDMAN, M., LEE, C.-T., AND CHIUUEH, T.-C. A Comprehensive Implementation and Evaluation of Direct Interrupt Delivery. In *Proceedings of VEE* (March 2015), pp. 1–15.
- [66] TULLSEN, D., EGGERS, S., AND LEVY, H. Simultaneous multithreading: Maximizing On-chip Parallelism. In *Proceedings of ISCA* (June 1995), pp. 392–403.
- [67] TURCHIN, A. Levels of AI Self-Improvement. <https://www.lesswrong.com/posts/os7N7nJoezWKQnnuW/levels-of-ai-self-improvement>, April 29, 2018.
- [68] TURPIN, M., MICHAEL, J., PEREZ, E., AND BOWMAN, S. R. Language models don’t always say what they think: Unfaithful explanations in chain-of-thought prompting, December 9, 2024. arXiv:2305.04388.
- [69] UHLIG, R., NEIGER, G., RODGERS, D., SANTONI, A., MARTINS, F., ANDERSON, A., BENNETT, S., KAGI, A., LEUNG, F., AND SMITH, L. Intel virtualization technology. *IEEE Computer* 38, 5 (2005), 48–56.
- [70] UK AI SAFETY INSTITUTE. International AI Safety Report: The International Scientific Report on the Safety of Advanced AI, February 18 2025. <https://www.gov.uk/government/publications/international-ai-safety-report-2025>.
- [71] UNITED STATES GOVERNMENT ACCOUNTABILITY OFFICE. NUCLEAR ARMS CONTROL: U.S. May Face Challenges in Verifying Future Treaty Goals. <https://www.gao.gov/assets/gao-23-105698.pdf>, September 2023.
- [72] VOLOS, S., VASWANI, K., AND BRUNO, R. Graviton: Trusted Execution Environments on GPUs. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation* (October 2018), pp. 681–696.
- [73] WAKSMAN, A., AND SETHUMADHAVAN, S. Tamper Evident Microprocessors. In *Proceedings of the IEEE Symposium on Security and Privacy* (May 2010), pp. 173–188.
- [74] WALL STREET JOURNAL. The Facebook Files, January 12, 2022. <https://www.wsj.com/tech/the-facebook-files-11642035385>.
- [75] WEISSE, O., VAN BULCK, J., MINKIN, M., GENKIN, D., KASIKCI, B., PIESSENS, F., SILBERSTEIN, M., STRACKX, R., WENISCH, T. F., AND YAROM, Y. Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution. *Technical report, revision 1.0* (August 14 2018).
- [76] WISTOFF, N., SCHNEIDER, M., GÜRKAYNAK, F. K., BENINI, L., AND

- HEISER, G. Microarchitectural Timing Channels and their Prevention on an Open-Source 64-bit RISC-V Core. In *Proceedings of the Design, Automation, and Test in Europe Conference (DATE)* (February 2021), pp. 627–632.
- [77] WOLF, Y., WIES, N., AVNERY, O., LEVINE, Y., AND SHASHUA, A. Fundamental Limitations of Alignment in Large Language Models, June 3, 2024. arXiv:2304.11082.
- [78] XIE, Y., WANG, D., CHEN, P.-Y., XIONG, J., LIU, S., AND KOYEJO, O. A Word is Worth A Thousand Dollars: Adversarial Attack on Tweets Fools Stock Prediction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (Seattle, WA, July 2022).
- [79] YAMPOLSKIY, R. Leakproofing the Singularity: Artificial Intelligence Confinement Problem. *Journal of Consciousness Studies* 19, 1-2 (2012), 194–214.
- [80] YE, X., AND DURRETT, G. The unreliability of explanations in few-shot prompting for textual reasoning. In *Proceedings of NeurIPS* (December 2022).
- [81] ZHANG, H., TANG, Y., KHANDELWAL, A., AND STOICA, I. SHEPHERD: Serving DNNs in the Wild. In *Proceedings of NSDI* (April 2023), pp. 787–808.
- [82] ZOU, A., PHAN, L., CHEN, S., CAMPBELL, J., GUO, P., REN, R., PAN, A., YIN, X., MAZEIKA, M., DOMBROWSKI, A.-K., GOEL, S., LI, N., BYUN, M. J., WANG, Z., MALLEEN, A., BASART, S., KOYEJO, S., SONG, D., FREDRIKSON, M., KOLTER, J. Z., AND HENDRYCKS, D. Representation Engineering: A Top-Down Approach to AI Transparency, March 3, 2025. arXiv:2310.01405.
- [83] ZOU, A., PHAN, L., WANG, J., DUENAS, D., LIN, M., ANDRIUSHCHENKO, M., WANG, R., KOLTER, Z., FREDRIKSON, M., AND HENDRYCKS, D. Improving Alignment and Robustness with Circuit Breakers, July 12, 2024. arXiv:2406.04313.