From Prediction to Proof: Rethinking AI for Systems and Networks

Matthew Caesar
University of Illinois at Urbana-Champaign
caesar@illinois.edu

What is Artificial Intelligence?

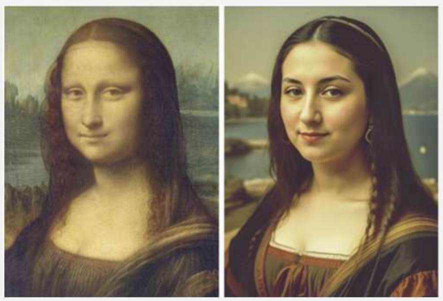
artificial intelligence

/ˌɑːtɪfɪʃl ɪnˈtɛlɪdʒ(ə)ns/ **noun**

1: the capability of computer systems or algorithms to imitate intelligent human behavior

2: a branch of computer science dealing with the simulation of intelligent human behavior by computers

“Human thinking” has achieved success



AI Art

AI Music

KHANMIGO

by Khan Academy

AI Education

AI Search

AI Management

AI Games

“Human thinking” is prone to problems

Semantic incoherence

- Hallucination
- Mode collapse
- Semantic drift
- Interface misalignment
- Context drift

EU AI Act

Proposal for a

Regulation of the European Parliament and of the Council Laying Down Harmonised Rules on Artificial Intelligence (Artificial Intelligence Act) and Amending Certain Union Legislative Acts

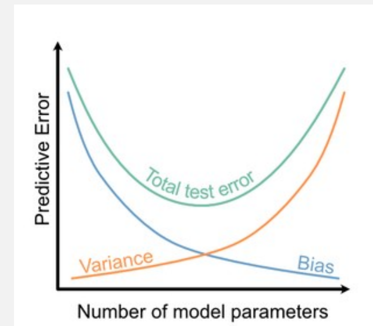
2021/0106 (COD)

Model Opacity

- Non-transparency
- Causal confusion
- Overconfidence
- Catastrophic forgetting
- Prompt sensitivity

Security

- Reward hacking
- Specification gaming
- Role breakage
- Deployment drift
- Data poisoning

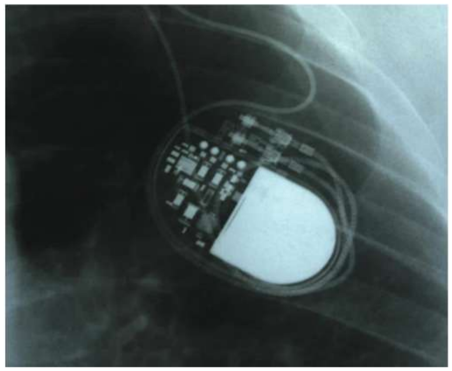


Data Errors

- Bias
- Model inversion
- OOD Errors
- Spurious Correlations
- Over/underfitting

Our field: Systems and Networking

Financial and trading networks



Medical devices

Clouds and online services

Societal infrastructures

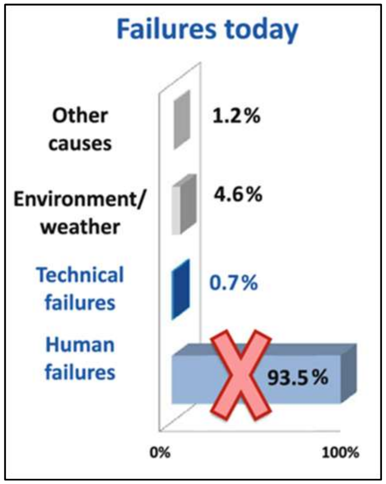
- We work on infrastructures behind modern society
- These infrastructures are complex, huge, dynamic
- Yet critical to get right

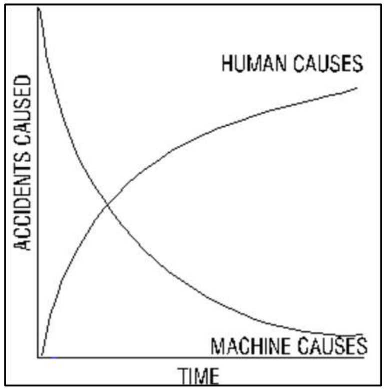
Humans aren’t good at getting things right

- Human error is #1 cause of problems in systems/networks
  - 80% of data center outages, 95% of data breaches, 95% of data loss incidents as a direct and immediate cause
  - Some categories of faults are 100% human-caused (misconfigurations, vulnerabilities, social engineering attacks, software bugs, insider attacks, …)
  - Consistently listed by CISOs as top risk; #1 category on IBM Threat Index

- Daily news is filled with vulnerabilities, misconfigurations, errors
  - Human error is the largest (increasing) contributor to failure

- “Human thinking” isn’t such a great approach to designing and operating systems





Should AI be the goal of the systems community?

- Should we be striving to build systems that "think like humans"?
  - AI may be useful, but it might also not be exactly what we want

- We may want to think more deeply about what we actually want from AI
  - We may be able to come up with something better

An alternative proposal

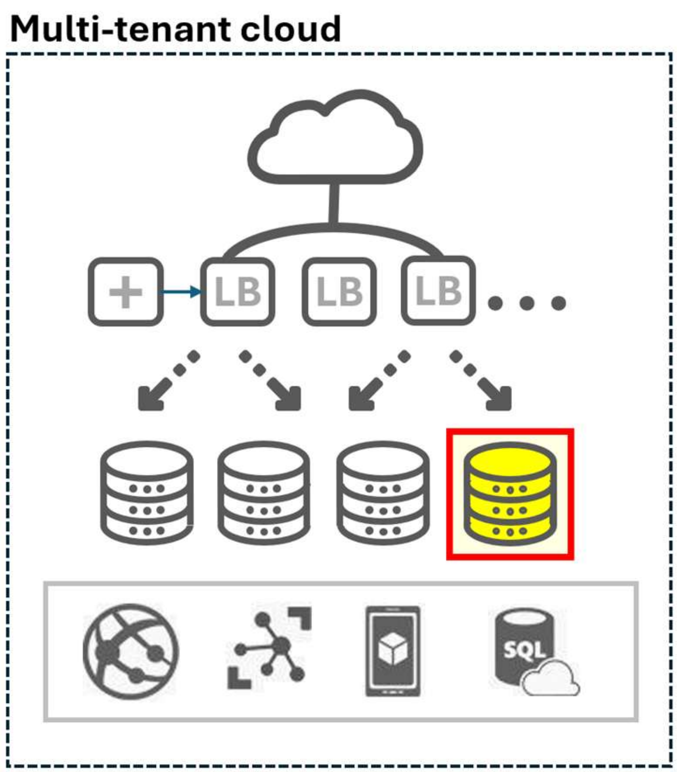
Artificial Reasoning

-- The capability of computer systems to derive optimal, fully-correct, understandable and analyzable solutions, through a formal sequence of logical steps

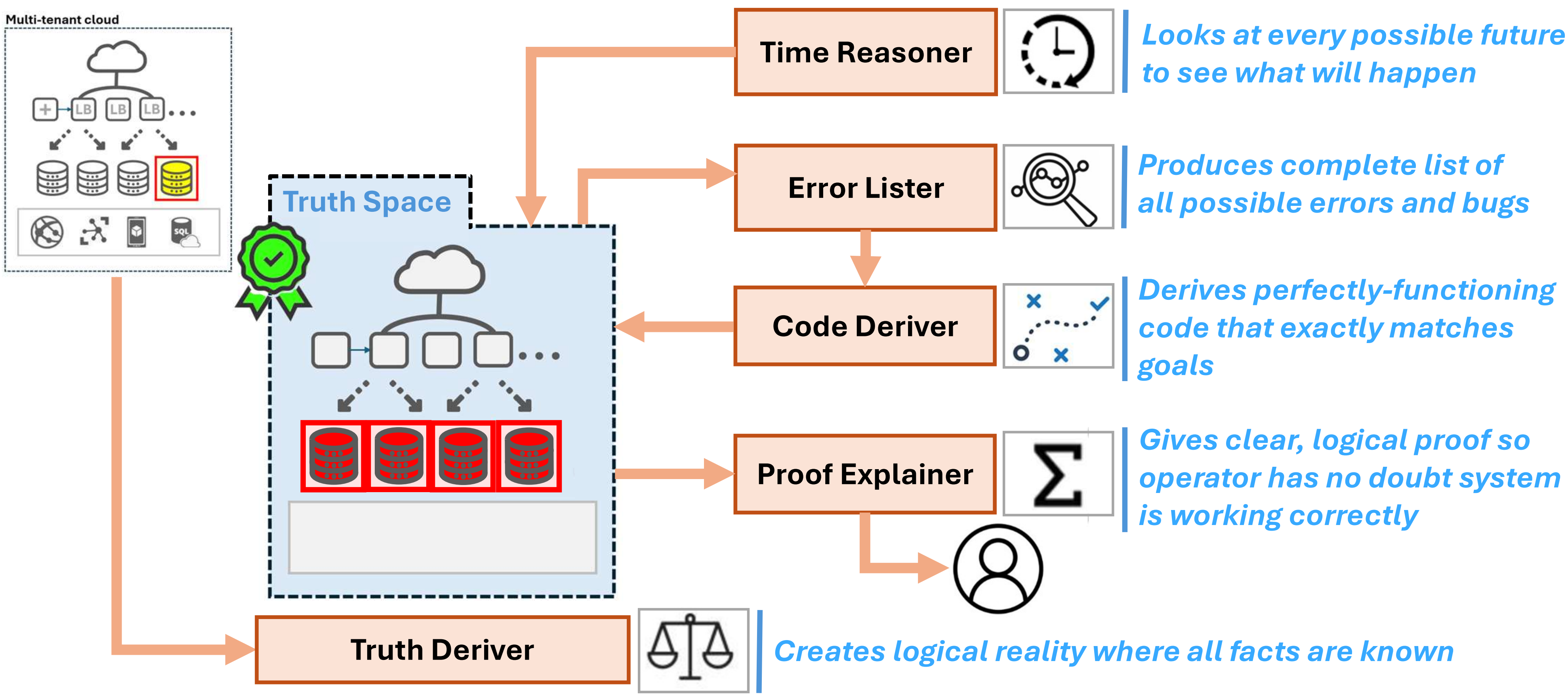
Focus on things we care about: safety, resilience, correctness, efficiency, explainability

A complement, not a replacement for AI techniques

What would an AR for Systems look like?



What would an AR for Systems look like?

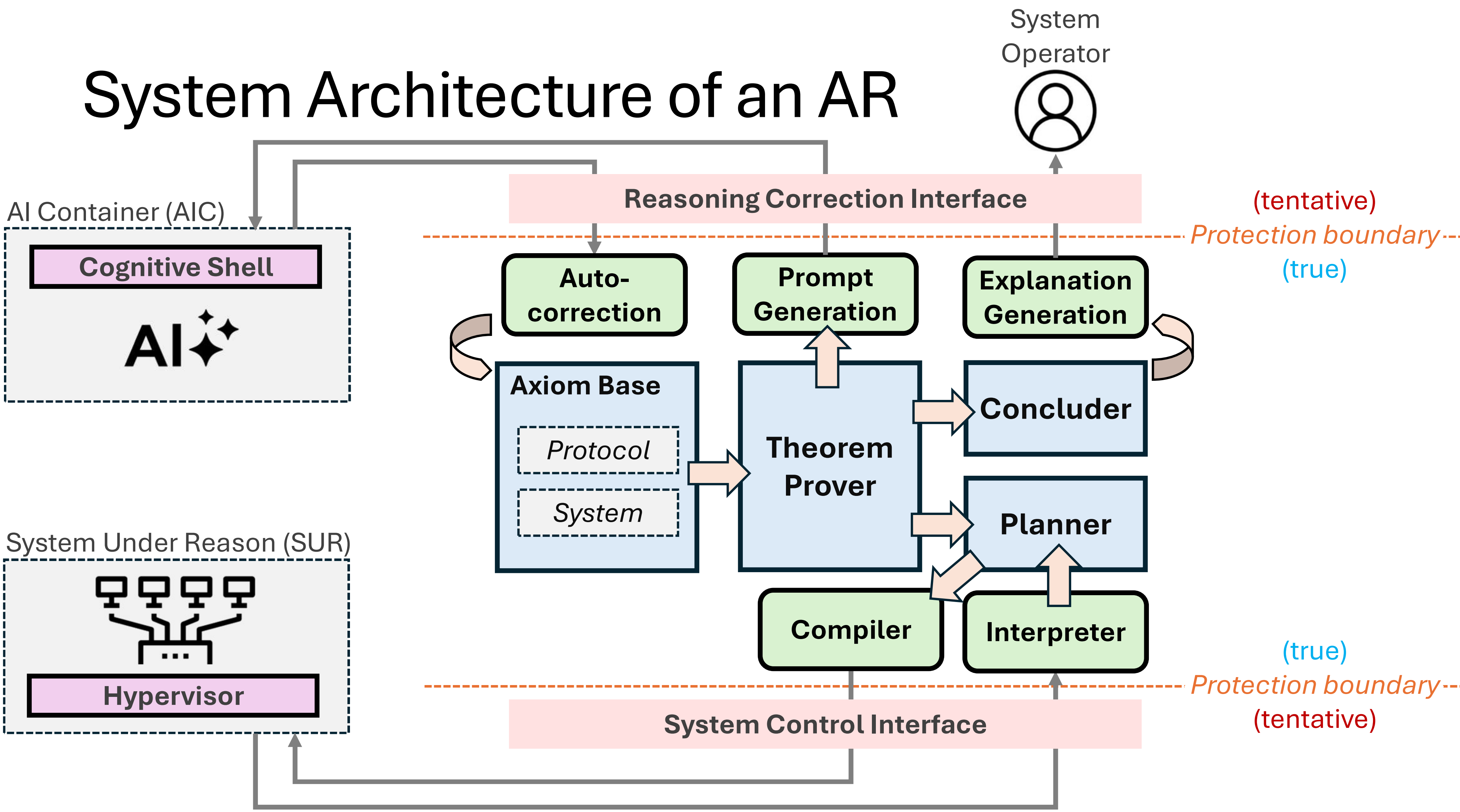


Towards a solution

- Seems difficult, but recent breakthroughs make progress towards this vision
  - *Formal methods* enable perfect and precise modeling, proving, and synthesis of diverse platforms
  - *Virtualization technologies* allow manipulation of time and inputs, log capture, and deterministic analysis and execution of real software
  - *Optimization techniques* enable exact and rigorous derivation of optimal behaviors in complex environments

How can we collectively leverage these components to build generalized Automated Reasoning for networked systems?

System Architecture of an AR



Key Challenges and Solution Approaches

1. How can we formally reason about dynamic, real systems where vagaries of execution behavior really matter?

→ *Solution approach:* System-Guided Formal Modeling
(“guide” formal modeling with running of real implementation code)

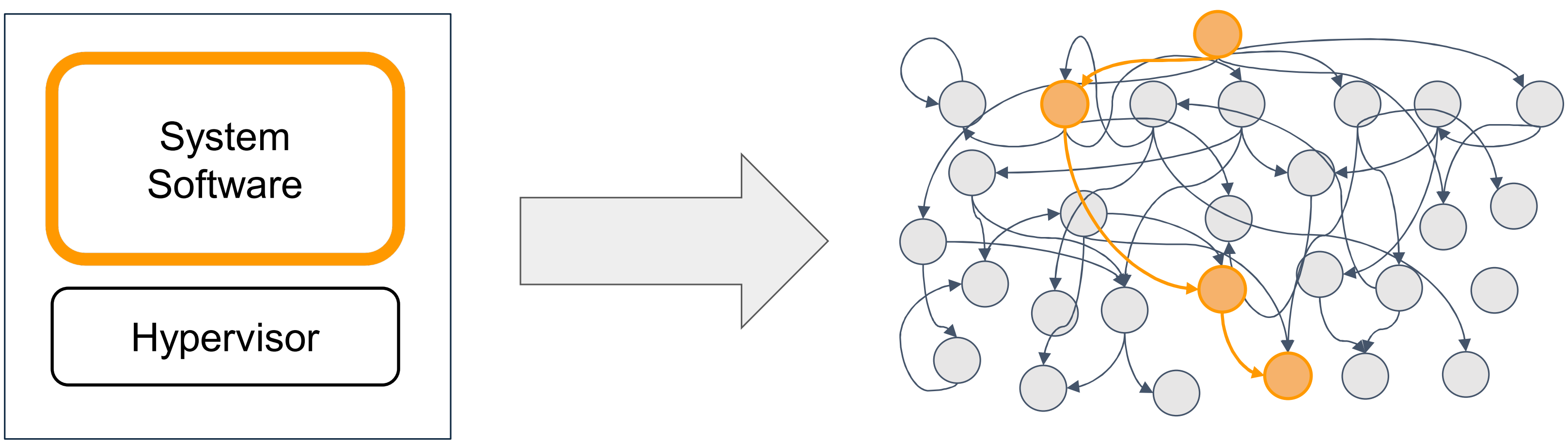
2. How can we safely integrate less-trusted AI inputs into the reasoning process?

→ *Solution approach:* Cognitive Input Autocorrection
(automatically repair inputs from AIs to match correctness specification)

How to create a model for reasoning?

One option: run the system’s software in an emulated environment (e.g., VMs)

Limitations: replication overheads, execution overheads, limited coverage



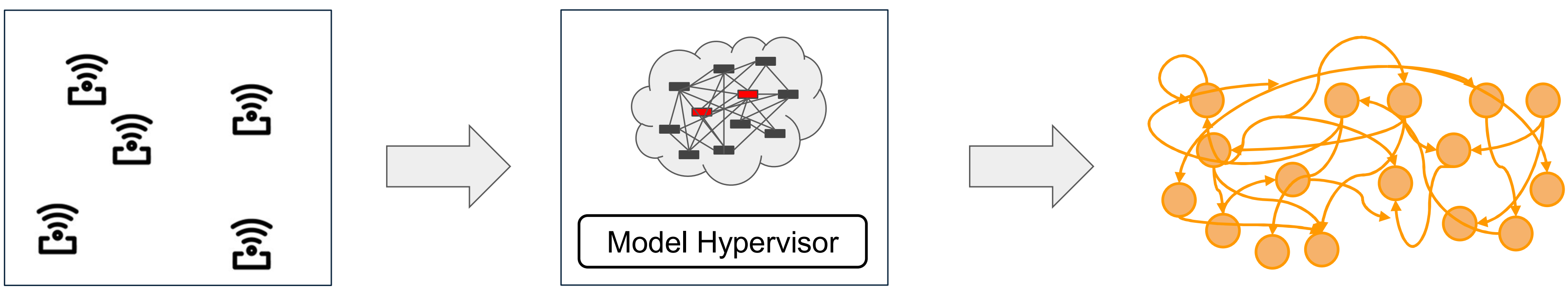
Software-Based Model

State Space Exploration

A more scalable approach to modeling

Idea: create a model with **formal methods**

- Can leverage rigorous techniques to efficiently, exhaustively search
- Lower bandwidth/compute requirements to traverse model states



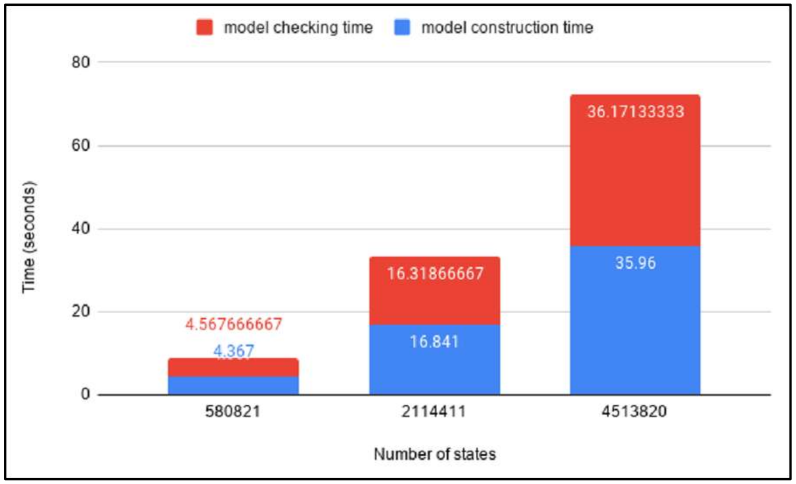
Deployment System Model Workaround Search

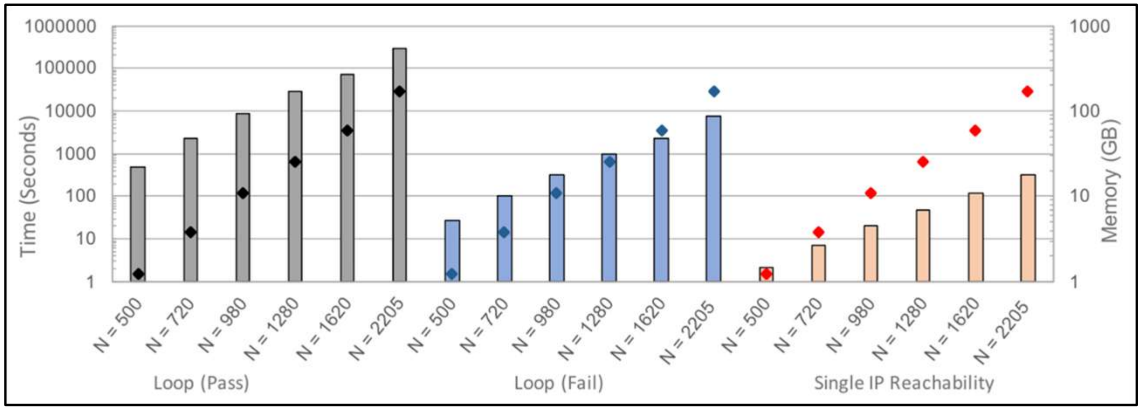
Benefits of formal modeling

- Unveil corner cases that are hard to argue about without models
- Better predictability compared to pure emulation
  - Automatic state transition without synchronization from the deployment
  - The model-based representation is less likely to deviate from the deployment state
- More optimization opportunities compared to working with real hw/sw
  - E.g., partial-order reduction (Only the orders of packets entering the stateful components are relevant) for simulating multiple connections through a stateful network

Challenges in applying formal modeling for reasoning

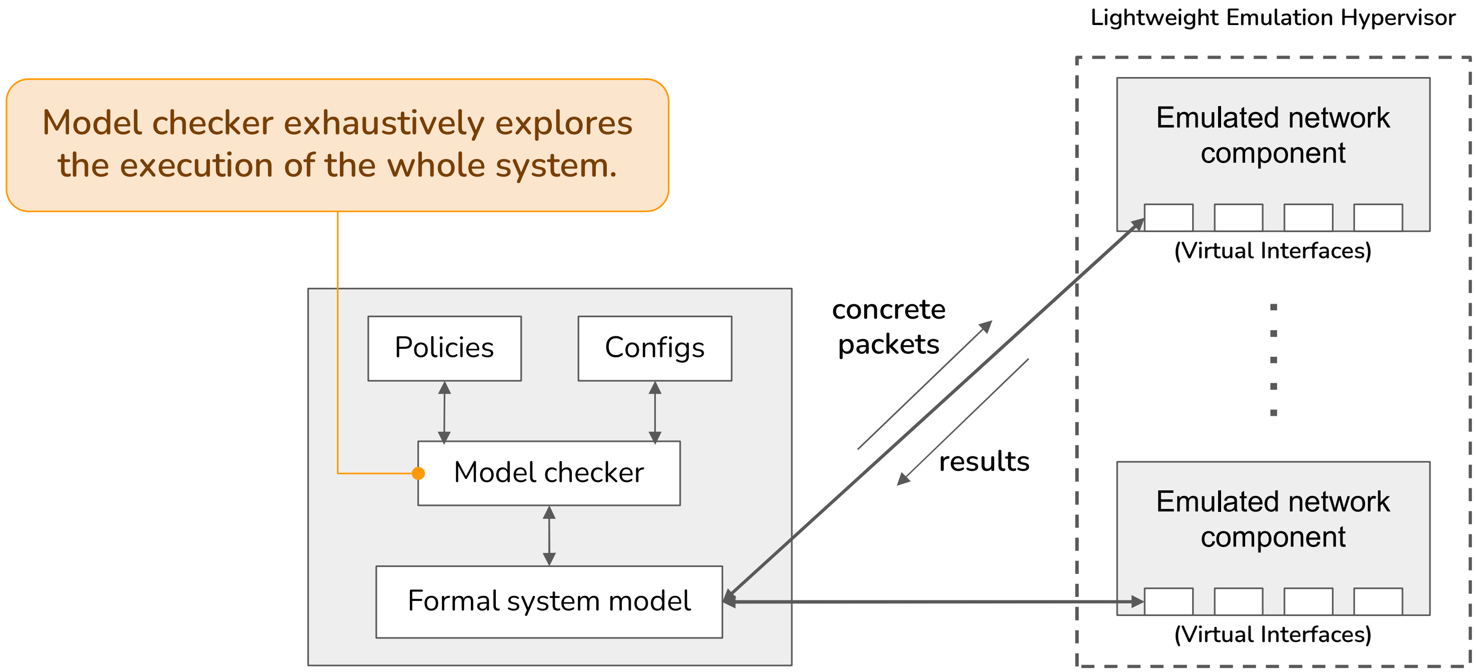
- Models must be truthful and precise
  - Real-world systems face complexities, scale, non-determinism, environmental interactions (e.g., time/event triggers), distributed protocols, energy/bw constraints
- Models with unnecessary details have lower scalability





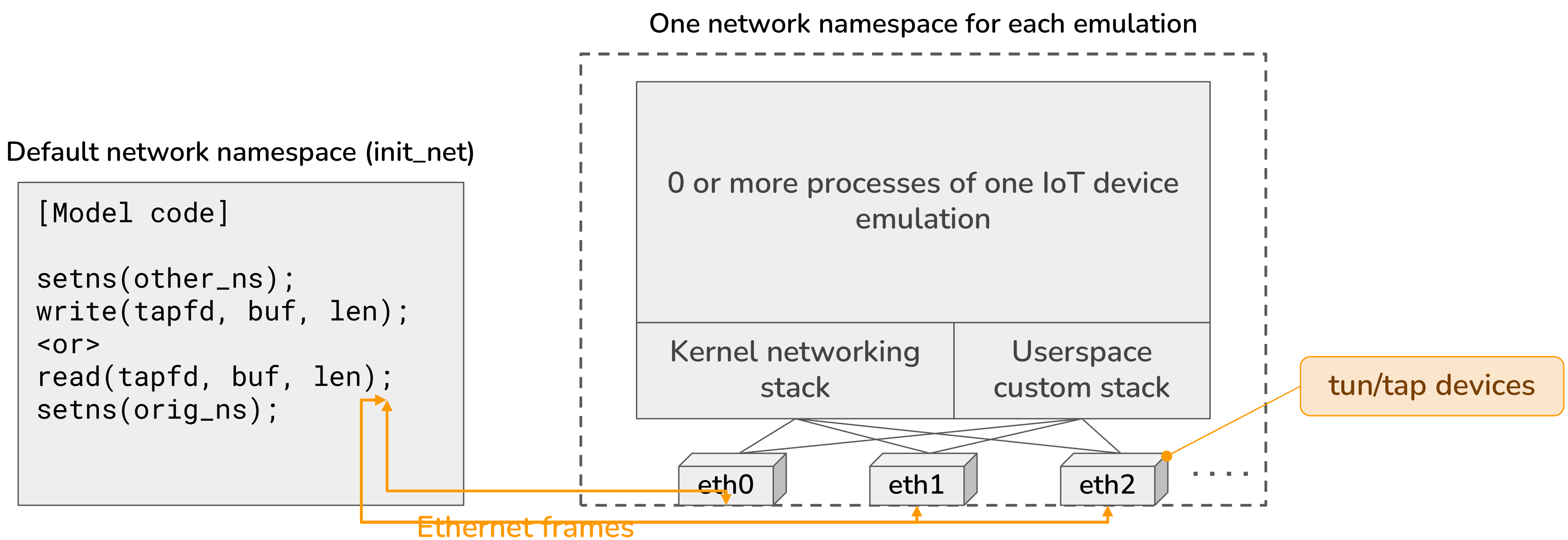
Formal Emulation: Combine models with emulation

- For each emulated component, we communicate with them from formal models by injecting and interpreting concretized packets and events
- State space is trimmed by synchronizing with deployed system



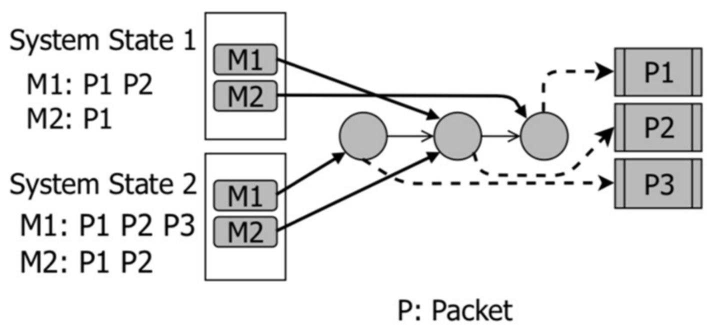
Formal Emulation: Combine models with emulation

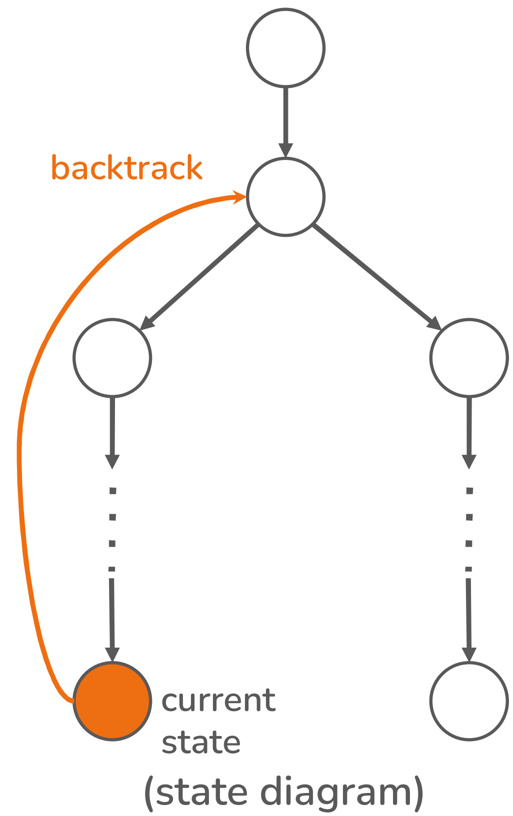
- We employ virtual interfaces and network namespaces for lightweight emulation
- The packets to/from emulation instances are interpreted for model state transition



Formal Emulation: Tracking emulation states

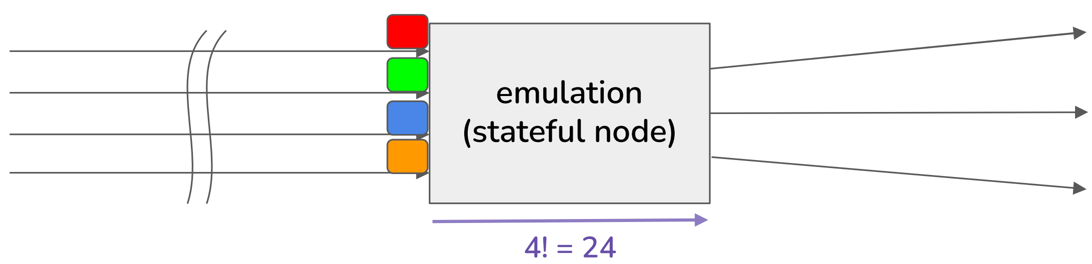
- Emulation state := initial state + history of events
  a. Events: packet arrivals, sensor updates, etc.
- Emulation instances need to be in the right state before injecting packets
  a. Reset to the initial state
  b. Replay the history of events
- Hashing histories of events to reduce memory overhead





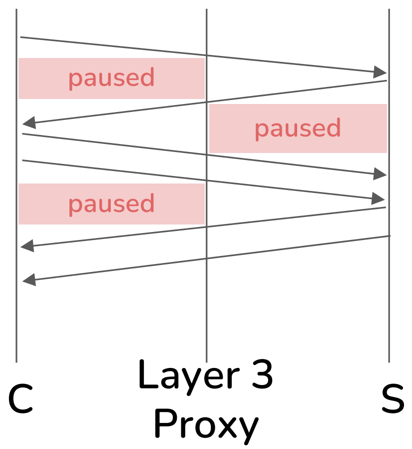
Formal Emulation: Dynamic multi-connection coordination

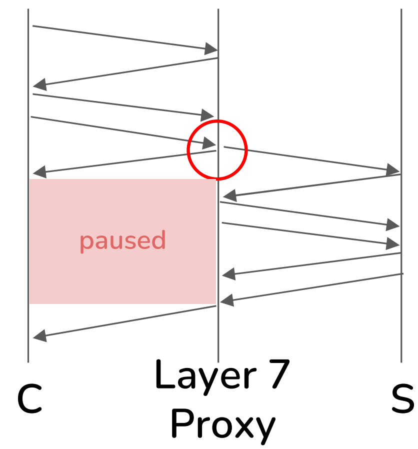
- How to model the non-deterministic nature of multiple connections in a networked component?
  - Our model implemented the non-deterministic choices for the model checker explore.
  - Apply partial-order reduction (POR) to reduce unnecessary search space.
  - POR heuristic:
    - Pick an arbitrary connection until every connection is about to enter an emulation.
    - Explore all orderings of the connections entering the emulations. (And repeat.)



Formal Emulation: Dynamic multi-connection coordination

- How to handle new connections initiated by emulations?
  - Parse received packets, and add new connections to the model state.
- What about L3 vs L7 proxies?
  - How to tell if a packet has gone through a L3 proxy or belongs to a new connection?
  - We treat all proxied packets as new connections. “Pause” connections when necessary.



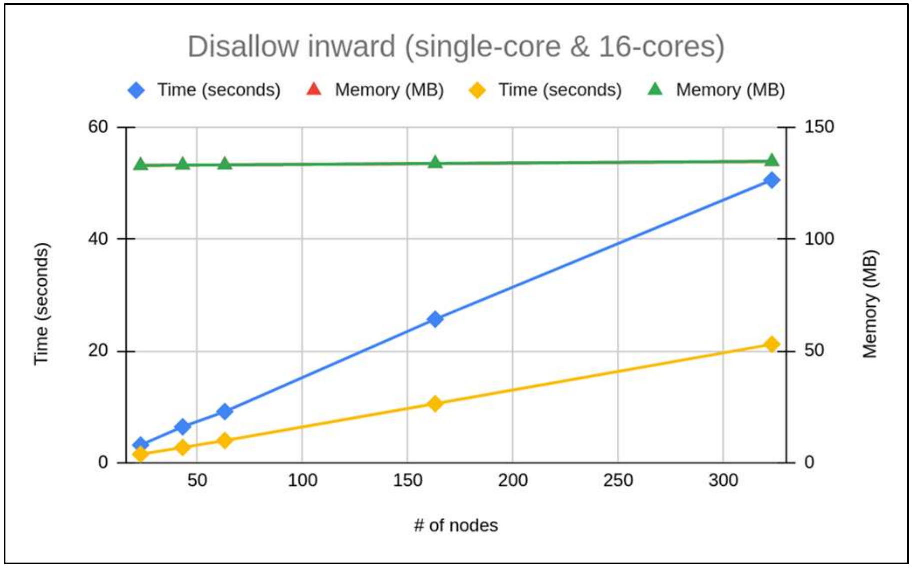


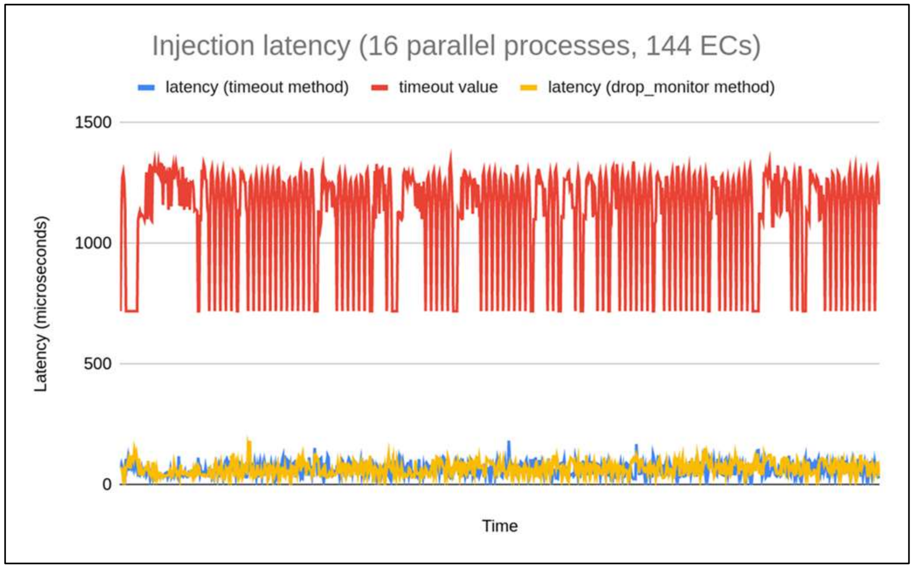
Formal Emulation: Association interpretation

- Challenge
  - Once we inject a packet, how do we know if the packet is dropped or not?
- Method 1 (drop timeout estimation)
  - Adjust the drop timeout based on injection RTT estimate, similar to TCP retransmit timeout.
  - Cumulative estimate of time := $\mu_{latency} + \sigma_{latency} \times max\left(4,\ ceil\left(\frac{2 \times (Number\ of\ jobs)^{1.5}}{Number\ of\ total\ cores}\right)\right)$
- Method 2 (Linux per-packet drop_monitor)
  - Since 5.4, we can request for per-packet drop alerts from kernel.

	Drop timeout	Kernel drop_monitor
Advantages	• Available for all types of emulations • Easy to implement	• No false violation when there is no tail drops
Limitations	• Potential false violations under high load • Longer wait time for dropped packets	• Only appropriate to hypervisors/kernels supporting this method

Performance Results





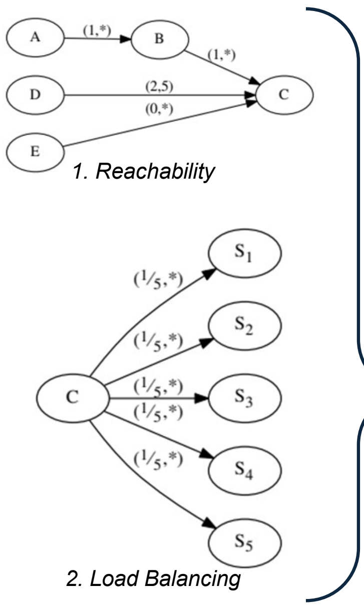
- CPU time grows linearly with network size, memory usage almost constant
- Timeout method generally faster than drop-monitor; drop monitor has additional overhead from registering and checking drops in kernel

Can we still use AI?

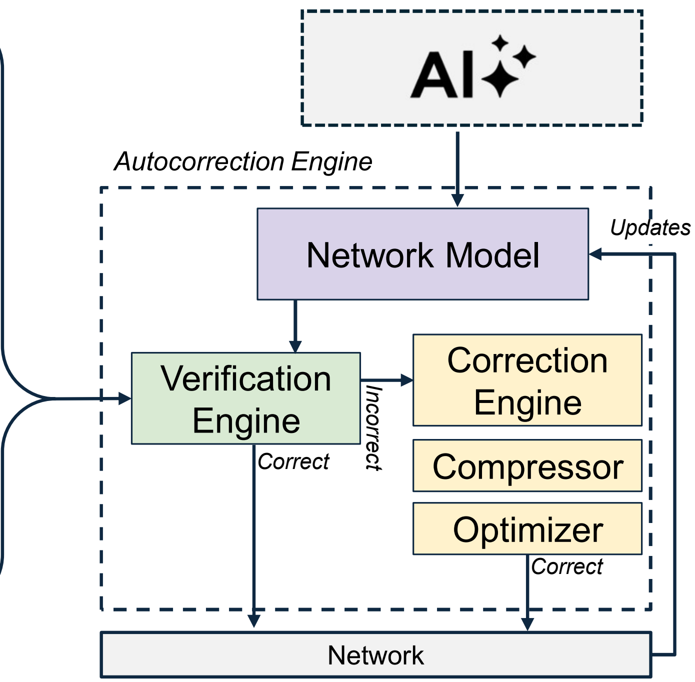
- AI has proven very useful to systems
  - But it can also be wrong
  - But we don’t want to just not use AI

- Idea: can we automatically correct AI inputs?
  - React quickly without need for operator in the loop
  - Transparent integration with existing workflows/APIs
  - Operator can view fixes to get insights on understanding their errors

Cognitive Input Autocorrection



Policy Graphs (intent-based AI guardrails)



System Architecture (repair updates to match provided policy graphs)

- Autocorrection layer synthesizes repairs to inputs in real time
- Fixes/patches derived from formal methods, guaranteeing compliance with provided specifications
- Optimization-based framework to place observation and correction programs

Two ideas for future work in Automated Reasoning

- Formally Verified AI Systems
  - There have been great strides in formal verification of AI techniques, and networked systems separately
  - Can we apply these techniques to build AI-based systems with formal guarantees on correctness, QoS, etc.?
- Using Reason to Design Systems
  - Much of research is getting automated
    - E.g., we rely on AI more and more for algorithm design
    - Has been harder to do for certain things, like architecture, which require robustness
  - Formal logics are good at deriving rigorous designs in other disciplines
    - Can we automatically derive system architectures with AR?

Conclusions

- The future of AI in systems isn’t about mimicking minds, it’s about mastering the goals important to our community
  - Artificial Reasoning may open the door to a new kind of AI
  - Centered around critical properties such as correctness, trust, and understanding
- We presented some abstractions that can help bring this vision closer to reality
  - Leveraged recent advances in formal methods, modeling, virtualization, and optimization to achieve scale, completeness, rigor
  - Early results demonstrate benefits and practicality of approaches