Multi-Aspect Runtime Verification for Simulation-Based V&V of LLM-Enabled Autonomous Agents

Nikolaos Kekatos[1] **Dimitrios Nikou**[2] Anastasios Temperekidis[1]
Alexios Lekidis[3]
Nikolaos Kolokotronis[4] Panagiotis Katsaros[5] Stylianos Basagiannis[2]

[1]Clone Systems [2]International Hellenic University [3]University of Thessaly
[4]University of Peloponnese [5]Aristotle University of Thessaloniki

MESAS 2026 · Prague · 21–23 October 2026

The situation

AI assistants that act

In military staff work, an AI language model plans a task, such as evacuating a casualty, and does it by **calling tools**: request a clearance, plan a route, update a record, share a file.

Under written rules

Who may receive a medical record, which routes are safe, what a medical aircraft may carry. These rules are binding, and the AI cannot simply be retrained.

Where we can act

Every action passes through a tool call, so we check the calls.

Three kinds of rules

Where

Where an aircraft may go, and where a record may travel.

When

In what order, and before which deadline, for example within one hour.

What

What a note actually says, not just what a form field claims.

The difficulty

Many rules mix all three at once, and each kind needs a different way of checking.

The trick: four harmless steps

Step 1	Step 2	Step 3	Step 4
request the medical marking	plan a route across the front line	add a note: “also ammunition”	fly across

Together

A medical aircraft carries ammunition under the medical marking: misuse of a protected emblem. No single step is wrong.

So

Checking one step at a time cannot catch it.

Why the last action looks harmless

EARLIER: WHEN

The aircraft received a protected medical marking.

MEANING: WHAT

Its manifest narrative includes ammunition.

CURRENT: WHERE

The proposed flight crosses the forward line.

The prohibition is visible only when these facts are combined

A per-call guardrail sees only the last request. A compositional monitor sees the current position, past actions, and the content of the manifest together.

Our idea: three checkers, one story, one decision

Where-checker

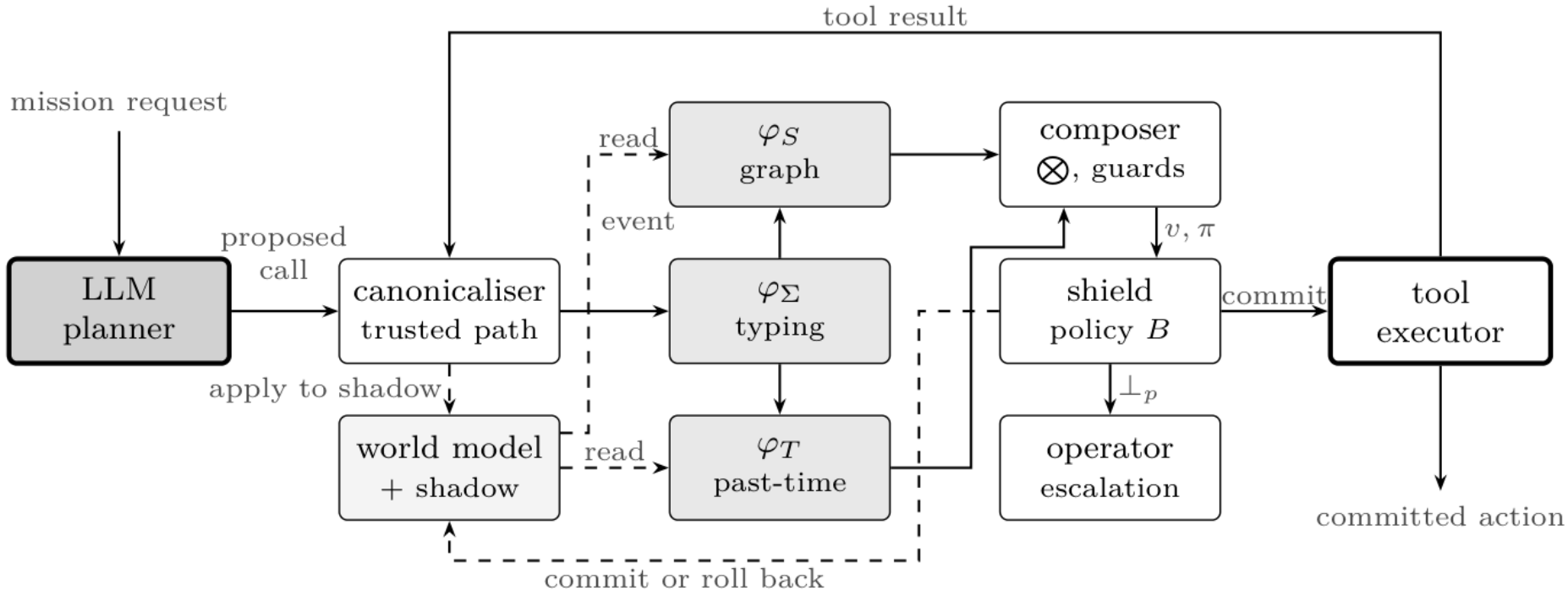
reads a map of places and of data locations

When-checker

remembers what happened and when

What-checker

reads the text of records and notes



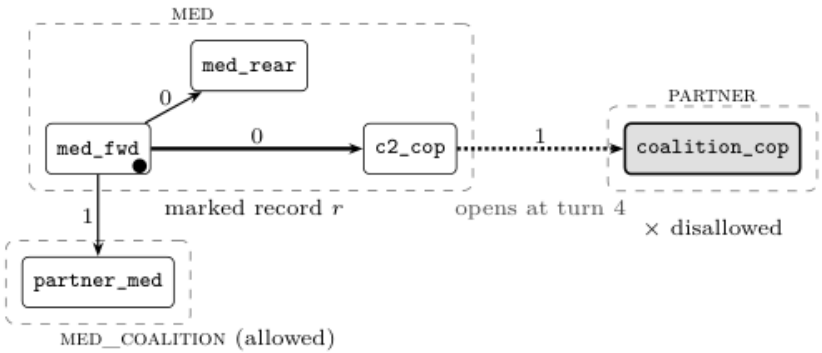
One map for places and for data

Places

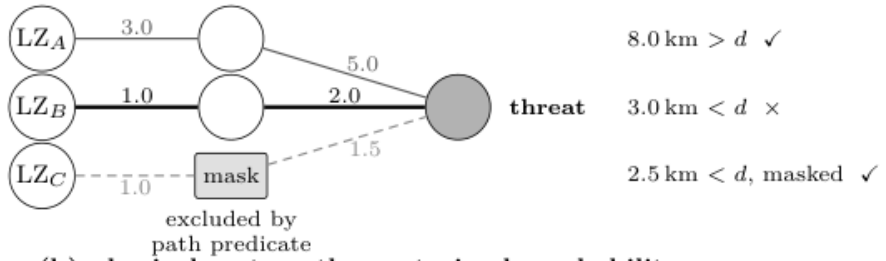
Landing zones, threats and terrain, in kilometres; a landing zone keeps clear of a threat unless terrain shields it.

Data

Computers in sharing zones, linked by one-way channels.



(a) informational sort: runtime topology



(b) physical sort: path-constrained reachability

The same question

“Can it get there?” answers both: one map, one way of checking. Opening one new channel can make a forbidden zone reachable, and the checker notices **before** the record moves.

Four answers, and stopping before it happens

Four possible answers

yes · probably yes · probably no · no

“Probably” means a rough heuristic was used. A “probably no” is sent to an operator, or blocked under the cautious setting.

Try it on a copy first

Each proposed action is applied to a **copy** of the world, checked, and only then carried out if allowed. A blocked action never changes anything.

Every refusal is explained

It names the rule and the checkers behind it, so an operator can review it.

What happens after a refusal?

1 · PROPOSE

Agent requests an action.

2 · CHECK

Try the effect on a copy.

3 · REFUSE

Discard effect; explain why.

4 · REPLAN

Choose a compliant alternative.

The same policy is checked again on the next proposed action

Important: the shield tries only the NEXT action on a shadow world model. It does not simulate the complete remaining mission.

What is real and what is a prototype

Real

The temporal rules were checked offline with MonPoly: they agreed at every time point. The event stream, verdict fusion and shield are implemented.

Prototype

The where-checker is our own code. The what-checker is a simple keyword matcher: rewording defeats it.

Why it matters

Two of the three checkers were tested only against our own code.

Results on our designed attacks

One checker

78% of attacks still get through.

Best two checkers

33% still get through.

All three, cautious

0% get through, and no harmless run is blocked.

Attack success per configuration

Configuration	ASR	FPR	μs/event	$\lvert\pi\rvert \geq 2$
No monitor	100%	0%	—	—
φ_S only	78%	0%	1.9	0%
φ_T only	78%	0%	1.8	0%
φ_Σ only	78%	0%	3.4	0%
$\varphi_S \times \varphi_T$	56%	0%	2.0	0%
$\varphi_S \times \varphi_\Sigma$	33%	0%	3.6	0%
$\varphi_T \times \varphi_\Sigma$	44%	0%	3.5	17%
Full composition, default $B = \{\bot\}$	11%	0%	3.6	33%
Full composition, precautionary	**0%**	**0%**	3.6	33%
$\varphi_S \times \varphi_T$, trusted labels	33%	0%	—	—
Per-call guardrail baseline	67%	33%	—	—

Which checker catches which attack

ID	Attack (MEDEVAC / logistics twin)	φ_S	φ_T	φ_Σ
M1/L1	feed escape to a disallowed domain	✓		
M2/L2	authority held for the wrong object		✓	
M3/L3	point designated inside a threat arc	✓		
M4/L4	mission deadline missed		✓	
M5/L5	sanitisation / consolidation laundering			✓
M6/L6	retention purge	✓	✓	
M7/L7	protected status retained after amendment			✓
M8/L8	corridor breach	✓		
M9/L9	presumptive typing			

With a real AI model, and in a controlled test

Small AI model, 12 runs per setting

Tricked in **6 of 12** runs without the checker, **0 of 12** with it.
On harmless runs it blocked about **4.6** steps per run.

Scripted planner, 400 runs

77% violations without shield;
0% observed with shield.
Delivery: 100% vs. 85%.
80.5% of refused runs recovered.

What this means

The small-model test shows it works with real AI output; the big numbers come from the scripted planner.

After the shield refused an action

308 episodes

At least **one** proposed action was refused.

248 recovered

80.5% of refused episodes completed compliantly.

60 stopped

No compliant destination was reachable.

Extra work when replanning

Median: **two** additional tool calls and **one** planning round. A safe refusal is not necessarily a failed mission.

We attacked our own checker

Hole 1: another door

Three tools could move a record without passing the where-checker. Fixed, and a test now tries every such tool.

Hole 2: self-approval

The AI could give itself the needed approval. Fixed: approvals now record who gave them, and the AI's own do not count.

The lesson

Correct rules are not enough: every tool must pass through the checker, and we must know who gave each approval.

What we did not show

Limits

- Attacks designed by us, against our own rules
- One small AI model, few runs
- A keyword-based what-checker

Not yet compared

- One checker fed a ready-made “can it get there?”
- Outside attack collections
- Newer whole-sequence AI guards

Why simulation

“Try on a copy” works because the world is simulated.

Take-away

In one sentence

An AI assistant can break a rule with steps that each look fine, so check where, when and what together, try every action on a copy first, and make sure every tool and every approval goes through the checker.

Three checkers

where, when, what

One decision

with its reasons

No side doors

every tool, every approval

Questions?

Dimitrios Nikou

International Hellenic University

diminiko14@ihu.gr

Backup: glossary

Term	Meaning
LLM agent	an AI language model that plans and calls tools
Tool call	one action the agent asks a system to perform
Runtime verification	checking a running system against written rules
Shield	the component that blocks or allows each tool call
Sharing zone	a group of computers allowed to hold the same records
MEDEVAC	medical evacuation of casualties
MonPoly	an established, independent rule-checking tool