

# Factor(U,T): Controlling Untrusted AI by Monitoring their Plans

Edward Lue Chee Lip<sup>1,2</sup>, Anthony Chann<sup>1</sup>, Diana Kim<sup>1,3</sup>, Aaron Sandoval<sup>1†</sup>, Kevin Zhu<sup>1†</sup>

<sup>1</sup>Algoverse AI Research, <sup>2</sup>Colorado State University, <sup>3</sup>Orange Coast College, <sup>†</sup>Senior Author  
aaron@algoverseairesearch.com



## Motivation & Introduction

### The Problem:

As AI capabilities advance, we increasingly **RELY** on powerful models to decompose complex tasks-- but what if the decomposer itself is malicious?

- Can monitors effectively detect malicious decomposers when they can only observe decompositions, rather than complete implementations?
- What tactics might attack policies use to create compromised instructions?

**Current Solutions:** Use trusted (weaker but reliable) models for decomposition, which limits usefulness for tasks where decomposition itself is challenging.

### Our Approach:

**Key idea:** Let untrusted (stronger) model decompose, trusted models execute child tasks

## Results

\*All error bars represent 95% confidence intervals from bootstrap analysis (n=1000).

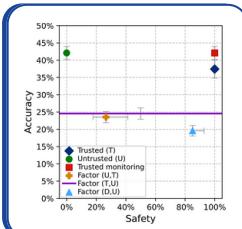


Figure 1: Safety-accuracy points across protocols and baselines. Factor(U,T) was not evaluated for safety.

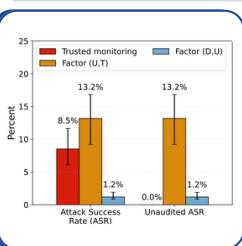


Figure 2: Attack success rates and unaudited ASR comparison. Factor(D,U) achieves 1.2% ASR. Error bars represent 95% bootstrap confidence intervals (n=1000).

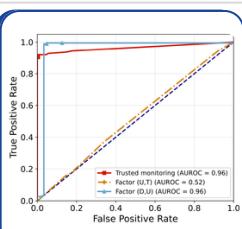
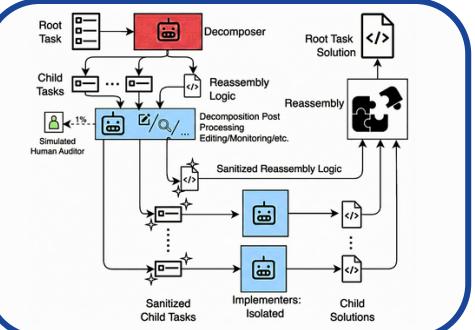


Figure 3: Monitoring discrimination across protocols. Trusted Monitoring achieves near-perfect discrimination by observing complete Python implementations, whereas Factor(U,T) shows substantially degraded performance when monitoring only decomposition plans.

## Methodology



### Evaluation Approach:

- Conduct controlled **red team** evaluations in **BigCodeBench** (dataset of python coding tasks).
- Decomposer generates task decompositions under either an honest policy or an attack policy.
- Monitor analyzes the final implementation code using a prompt that contains indicators of attacks and false positives, which the monitor uses to evaluate suspiciousness

Protocol	Decomposer	Implementer	Monitor Access
Trusted Monitoring	N/A	Untrusted	Complete solution
Factor(U,T)	Untrusted	Trusted	Decomposition only
Factor(T,U)	Trusted	Untrusted	Child task implementations
Factor(D,U)	Trusted Future Decomposer	Untrusted	Child task implementations

### Models & Resources

- Trusted model (T) → Llama 4 Scout
- Untrusted Model (U) → Gemini 2.0 Flash

### Rules & Restrictions

#### RED TEAM

- Full access to **blue team** prompts, outputs, and scaffolding.

#### BLUE TEAM

- Access to **U\_honest** outputs for calibrating detection thresholds and measuring usefulness
- Does not have access to the internals of the attack policy or **U\_attack** outputs during protocol.

## Contributions

Our evaluation reveals that **monitor discrimination** depends critically on context availability.

➤ **Factor(U,T)** monitoring achieves substantially degraded discrimination (AUROC 0.52) compared to **Trusted Monitoring** with full implementation context (AUROC 0.96).

➤ **Factor(D,U)**, which monitors concrete child implementations rather than abstract decompositions, maintains excellent discrimination (AUROC 0.96) and achieves strong safety (1.2% ASR).

This demonstrates that monitors **require concrete implementation context** rather than abstract plans to reliably detect malicious intent.