Supplementary Materials

This supplementary document consists of four major components designed to complement the main paper on the **HAM³** framework:

1. **Adversarial Attack Implementations:** Detailed algorithmic representations of perception-, communication-, and reasoning-layer adversarial attacks that disrupt multi-modal multi-agent coordination.
2. **Design and Encapsulation of Multi-Agent Tool Sets:** Structural descriptions of the collaborative agent framework and corresponding tool modules used for evaluation experiments.
3. **Cross-dataset Generalization on EvoChart:** Additional experiments on the EvoChart-QA benchmark to assess whether the attack effectiveness observed in the main paper generalizes beyond GQA, with results showing consistent vulnerability patterns across datasets and agent paradigms.
4. **Defense Experiments:** An empirical analysis of a representative defense method under attacked agentic workflows, highlighting its practical limitations and showing why more context-aware or semantics-level defenses are needed for multi-agent systems.

---

1. Adversarial Attack Implementations

Consistent with the definitions in the main paper, the following sections illustrate representative adversarial mechanisms corresponding to the three hierarchical layers of HAM³:

1. **Perception-Layer Attacks** – Manipulate or contaminate multimodal inputs (text, image, or cross-modal data) to bias system perception.
2. **Communication-Layer Attacks** – Inject forged data or misinformation during inter-agent communication to corrupt shared representations.
3. **Reasoning-Layer Attacks** – Manipulate tools, reasoning chains, or agent roles to destabilize high-level inference.

1.1. Perception-Layer Attacks

1.1.1. Cross-Modal Injection Attack(CMA)

This attack jointly manipulates both text and visual inputs. The injected cross-modal perturbation maintains semantic consistency while corrupting alignment between the modalities. This paper employs two attack modes, randomly selecting one to execute the attack.

Algorithm 1 Cross-Modal Attack with Two Modes

Input: Original image x_{image}, original user query x_{text}, attack mode M

Output: Perturbed image-query pair (G_{image}, G_{text})

if M = Semantic-Redirection Mode **then**
  Use a VLM to extract image semantics S from x_{image}
  Use S together with x_{text} to produce a redirected query G_{text} that remains relevant to the image context but changes the question target
  Generate a visual editing plan E according to G_{text}
  Use the image editing model to modify x_{image} according to E, yielding G_{image}
else if M = Text-Embedding Mode **then**
  Analyze the intent of the original query x_{text}
  Generate misleading textual content T_{mis} related to the query intent
  Rewrite the original query into G_{text} so that it is more aligned with T_{mis}
  Embed T_{mis} into a predefined region of image x_{image}
  Set the resulting image as G_{image}
end if
return (G_{image}, G_{text})

1.2. Communication-Layer Attacks

1.2.1. Agent Spoofing Attack(ASA)

This attack works by inserting fake sub-agent or replacing valid ones with false ones, thereby causing harmful outcomes. Its goal is to mislead inter-agent collaboration.

Algorithm 2 Agent Spoofing Attack

Input: Set of target agents $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$, attack mode M

Output: Attacked agent set

1: Generate faked sub-agent set $\mathcal{A}_f = \{A_{f_1}, A_{f_2}, \ldots, A_{f_m}\}$
2: Replace prompt with pre-defined faked master prompt (Insert or replace the correct agent with the faked one, $\mathcal{A}_f$)
3: **return** Attacked agent set

1.2.2. Structural Blocking Attack(SBA)

This attack forces otherwise capable agents into an infinite delegation loop, preventing them from completing the task and leading to unnecessary consumption of computing resources. Note that Algorithm 3 illustrates a minimal working example involving only two agents for simplicity, but this structural blocking can be naturally extended to more agents.

Algorithm 3 Structural Blocking Attack

Input: Set of target agents $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$

Output: Attacked agent set

1: Select two agents (A_1, A_2) from the set of target agents.
2: Modify master instructions to enforce rule: never finalize, always delegate
3: Modify A_1's instructions to always request verification from A_2
4: Modify A_2's instructions to always require re-analysis from A_1
5: **return** Attacked agent set

1.2.3. Shared Memory Pollution Attack(SMPA)

This attack contaminates the inter-agent shared short-term memory with contradictory or false user–assistant dialogue pairs.

Algorithm 4 Shared Memory Pollution Attack

Input: Set of target agents $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$, and *shared_memory* M

Output: Modified *shared_memory* M'

1: Define a set of fabricated dialogue snippets to replace the real memory
2: Randomly select a subset of them
3: Prepend these to *shared_memory*
4: **return** Modified *shared_memory* M'

1.2.4. Shared Context Injection Attack(SCIA)

This attack introduces a unified adversarial instruction I_{mis} into the shared context of multiple collaborating agents. Instead of targeting a single agent, it simultaneously injects the same malicious instruction into several agents. This induces mutual distrust and suspicion among them, ultimately derailing task execution.

Algorithm 5 Shared Context Injection Attack

Input: Set of target agents $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$, each with current prompt P_i

Output: Modified prompts $\{P'_1, P'_2, \ldots, P'_n\}$

1: Construct instruction I_{mis}
2: **for** each agent $A_i \in \mathcal{A}$ **do**
3: Set $P'_i \leftarrow I_{\text{mis}} + P_i$
4: **end for**
5: **return** Modified prompt $\{P'_1, P'_2, \ldots, P'_n\}$

1.3. Reasoning-Layer Attacks

1.3.1. Chain-of-Thought Injection Attack (CIA)

This attack directly inserts misleading reasoning traces (thought chains) that bias the agent's internal reasoning toward erroneous behaviors. Specifically, the injected misleading content typically employs one of the following strategies:

- **Fabricate tool execution errors:** Inject reasoning that falsely concludes a tool is malfunctioning or returning errors, preventing its normal use.
- **Hallucinate safety risks:** Simulate a false risk assessment within the reasoning chain to trigger an unjustified refusal of the user's query.
- **Construct artificial contradictions:** Pretend to discover logical conflicts in the context to derail the agent's reasoning process.

Algorithm 6 Chain-of-Thought Injection Attack

Input: Set of target agents $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$

Output: Attacked agent set

Define a set of candidate misleading contents $\mathcal{C}_{mis}$
for each agent $A_i \in \mathcal{A}$ **do**
    Sample a misleading content $c \in \mathcal{C}_{mis}$
    Append c to the Chain-of-Thought (CoT) context of A_i
end for
return Attacked agent set

Summary.
The supplementary materials presented above provide systematically organized a part of implementations of representative adversarial strategies under the **HAM³** framework. Each attack constitutes a concrete instantiation of a distinct threat mechanism designed to compromise multi-agent system robustness through hierarchical interference. Together, these adversarial techniques penetrate the three critical operational layers—*Perception*, *Communication*, and *Reasoning*—thereby undermining inter-agent coordination, eroding information propagation, and inducing complex failures across the reasoning chain.

2. Design and Encapsulation of Multi-Agent Tool Sets

This subsection outlines the design and encapsulation of tool sets in the proposed multi-agent system. Each agent performs perception and environment interaction via a set of unified tools. These tools internally rely on the `agentlego`[1] interfaces for model invocation, and are exposed to agents as callable functions managed by `FunctionHub`. Conceptually related tools are grouped into separate tool sets and assigned to different agents according to system configuration.

The design follows three principles:

- Tools with related functionality are organized into dedicated tool sets.
- Each tool set is bound to an independent `FunctionHub` instance that handles registration, documentation, and invocation of the corresponding functions.
- Every tool function is registered with an explicit name, typed arguments, and a natural-language description, enabling the language model to select tools and plan parameters autonomously.

In the following, we briefly describe the main tool sets used in our system.

Image Understanding Tool Set

The image understanding tool set `image_understanding_tools` provides high-level visual understanding capabilities, including scene description, optical character recognition (OCR), and visual question answering (VQA).

Image Description The image description tool generates a natural-language description of the input image, capturing salient objects, attributes, and spatial relations.

OCR Text Recognition The OCR tool extracts textual content from the image and returns a consolidated text string.

Image Question Answering The VQA tool answers free-form questions conditioned on the given image, supporting open-ended reasoning about visual content.

Human Analysis Tool Set

The human analysis tool set `human_analysis_tools` focuses on human-centric perception.

Human Pose Estimation The human pose estimation tool predicts body keypoints and their connectivity, providing a structured representation of human pose.

Facial Landmark Detection The facial landmark detection tool predicts dense facial keypoints, which can be used for downstream tasks such as expression analysis or face editing.

Image Conversion Tool Set

The image conversion tool set `image_conversion_tools` produces structured or stylized representations of an input image.

Canny Edge Extraction The edge extraction tool computes a Canny edge map, emphasizing object boundaries and fine structures.

Depth Estimation The depth estimation tool generates a per-pixel depth map, approximating the 3D geometry of the scene from a single RGB image.

Sketch/Scribble Rendering The sketch conversion tool transforms an input image into a line-drawing or scribble-style representation, which is useful as structural guidance for generative models.

Object Detection Tool Set

The object detection tool set `object_detection_tools` supports both generic and text-conditioned object localization.

Generic Object Detection The generic detector identifies all salient objects in the image and outputs their categories, bounding boxes, and confidence scores.

Text-Conditioned Object Detection The text-conditioned detection tool localizes objects that are semantically aligned with a user-provided textual query (e.g., "the red car"), and returns the corresponding bounding boxes.

Image Segmentation Tool Set

The image segmentation tool set `image_segmentation_tools` supports both generic and class-specific image segmentation.

Promptable Segmentation The promptable segmentation tool performs class-agnostic segmentation and can optionally be guided by textual or spatial prompts to restrict the segmented regions.

Category-Specific Segmentation The category-specific segmentation tool extracts pixel-level masks for objects that match a given category name (e.g., "person", "chair"), enabling fine-grained manipulation and analysis.

Integration into the Multi-Agent System

During system initialization, all tool sets are instantiated and registered with their corresponding `FunctionHub` instances. At configuration time, each agent is assigned a specific subset of tools, reflecting its role in the overall pipeline. Agents invoke only tools within their assigned sets to complete their subtasks. This modular organization yields a uniform tool interface, clear capability boundaries across agents, and facilitates reproducibility and future extension of the system.

Table 1: ASR on the EvoChart-QA benchmark.

Paradigm	**Per.**		**Comm.**		**Rea.**	
	VIA	**CMA**	**SBA**	**SMPA**	**TSA**	**CIA**
ReAct	51.7%	48.3%	48.3%	43.3%	53.3%	**58.3%**
Plan&Solve	36.7%	38.3%	45.0%	38.3%	51.7%	**53.3%**
Reflexion	33.3%	36.7%	38.3%	33.3%	48.3%	**50.0%**

3. Cross-dataset generalization on EvoChart.

To assess generalizability, we evaluated our attacks on EvoChart-QA [2], a structured visual reasoning benchmark (650 charts, 1250 QA pairs). Testing 60 random instances with GPT-4o results in Table 1 echo the GQA patterns: all paradigms exhibit significant vulnerability. Notably, CIA achieves peak effectiveness (avg 53.9% ASR), confirming our frame- work's cross-dataset robustness and general failure modes.

4. Defense Experiments.

We evaluated an ONION defender [3] on 400 attacked tasks (1619 steps) under ReAct. ONION triggered 470 times but failed to remove any attack tokens (**0% coverage**). Instead, it frequently deleted essential structured tokens (e.g., "Tool", "execution") and file paths (causing 6.5% runtime failures). These results show fluency-based defenses fail in agentic workflows, necessitating context-level or semantic consistency checks.

References

[1] AgentLego Contributors. Agentlego: Open-source tool api library to extend and enhance llm agents, 2023.

[2] Muye Huang, Han Lai, Xinyu Zhang, Wenjun Wu, Jie Ma, Lingling Zhang, and Jun Liu. Evochart: A benchmark and a self-training approach towards real-world chart understanding, 2025.

[3] Fanchao Qi, Yangyi Chen, Mukai Li, Yuan Yao, Zhiyuan Liu, and Maosong Sun. Onion: A simple and effective defense against textual backdoor attacks, 2021.