System and method for real-time visual defect detection using multi-agent coordination to improve inspection accuracy

Authors: Tran Van Tuan, Zhang Ai Ping

Office: Fii-CPEG研发服务处

E-mail: trantuan22052k@gmail.com

Phone: +86342184225

Table of contents

1. Function point
2. Describe the past solutions and disadvantages
3. Solutions of this proposal
4. Methods
5. Photos
6. Advantages

1 Function point

(1) Requires only a small number of training samples

(2) Capable of detecting a wide range of defects across diverse product categories

(3) Real-time inspection directly on the production line

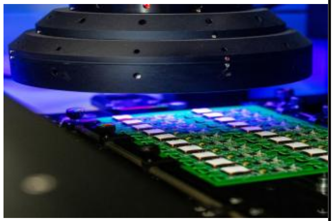
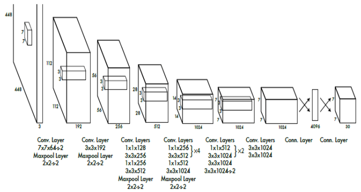
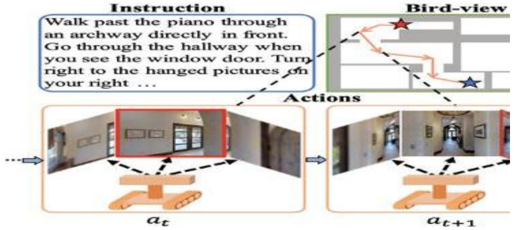
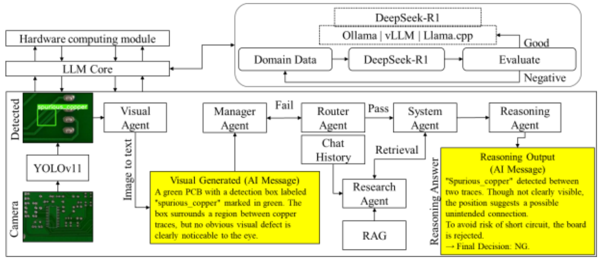
(4) A robust multi-agent system that maintains operates reliably under varying lighting conditions

(5) High accuracy through coordinated inference among agents

(6) Automatically triggers alerts and halts the production line upon defect detection

2 Describe the past solutions and disadvantages

Table 1.1 Comparison between methods

Aspect	Traditional AOI	Vision	Vision Language Model	Our Proposed Method
				
Method	Rule-based	Vision based CNN	Vision based VLLM	Multi-Agent + YOLO
Adaptability	❌ No	⚠ Limit	⚠ Limit	✅ Yes
Explainability	❌ No	❌ No	⚠ Limit	✅ Yes
Pros	• Fast, simple	• Fast, simple	• Fast, simple • Semantic visual description	• Fast (Ollama, vllm, etc, backend) • Adaptability, explainable • No require large amount data training • Reasoning & problem solving
Cons	• Rigid rule • Setup effort	• Need lots of data • No Adaptability	• Low accuracy • Need lots of data • Lack reasoning & problem solving	• Complex system • Multiple components.

Instruction

Walk past the piano through an archway directly in front. Go through the hallway when you see the window door. Turn right to the hanged pictures on your right ...

Visual Generated (AI Message)

A green PCB with a detection box labeled "spurious_copper" marked in green. The box surrounds a region between copper traces, but no obvious visual defect is clearly noticeable to the eye.

Reasoning Output (AI Message)

"Spurious_copper" detected between two traces. Though not clearly visible, the position suggests a possible unintended connection.
To avoid risk of short circuit, the board is rejected.
→ Final Decision: NG.

3 Solutions of this proposal

The solution of this patent is expressed through the following steps

1) step 1, acquiring a batch of image frames, a plurality of consecutive image frames is captured from one or more industrial-grade cameras installed along the production line. Said frames are grouped into a batch of predetermined size n to facilitate batch-based inference, temporal correlation, and parallel processing, represented as $B = \{f_i \mid i = 1,2,3, \ldots, n\}$ $(eq1)$; Where, f_i denotes the frame at index i; n denotes the number of frames in one batch;
2) step 2, performing object detection on each frame, each frame within the batch is individually processed by a pre-trained object detection neural network (e.g., a YOLO-family model). By applying the object detection model, each frame f_i is processed, yielding a set of bounding boxes represented as $BB_i = \left\{bb_{(i,1)}, bb_{(i,2)}, \ldots, bb_{(i,j)}\right\}$ $(eq2)$; Each bounding box $bb_{(i,j)}$ at frame i and index j is represented as $bb_{(i,j)} = (x_1, y_1, x_2, y_2, c, l)$ $(eq3)$,Where j is bounding-box's index at frame index I (f_i); $\{x_1, y_1, x_2, y_2\}$ is coordinates of bounding box; c is the confidence score of YOLO object detection (0 ≤ c ≤ 1); l is the detected object label;

3 Solutions of this proposal

3) Step 3, crop patches based on the bounding boxes with a predefined scaling factor, extracting image patch from bounding boxes, each bounding box $bb_{(i,j)}$ is cropped from the corresponding frame f_i, yielding patches $p_{(i,j)}$stored in P_i, the process can be represented as $p_{(i,j)} = crop\left(f_i, bb_{(i,j)}, s\right)\ (eq4);\ P_i = \left\{p_{(i,1)}, p_{(i,2)}, \dots, p_{(i,j)}\right\}\ (eq5)$; Where s is the scaling factor for adjusting bounding box size; $crop()$ is the cropping function based on bounding box coordinates and scale factor s;
4) Step 4, store the cropped patches into a patch list, each patch $p_{(i,j)}$ is packaged with its bounding box to enable the manager agent to display defect coordinates on the user interface if the router agent confirms the patch description is invalid $P_i = \left\{\left(p_{(i,1)}, bb_{(i,1)}\right), \left(p_{(i,2)}, bb_{(i,2)}\right), \dots, \left(p_{(i,j)}, bb_{(i,j)}\right)\right\}\ (eq6);$ Where, P_i denotes the list of patches; $p_{(i,j)}$ is the patch image at frame i and index j; $bb_{(i,j)}$ is bounding box coordinates;
5) Step 5, for each patch, generate semantic text descriptions and confidence scores using a multi-modal large language model (LLM), the process can be represented as (a) encode each image patch $p_{(i,j)} \in P_i$, the process can be represented as $d_{(i,j)} = VLM\left(p_{(i,j)}\right)(eq7)$, Where $d_{(i,j)}$ is the semantic textual generated from patch $p_{(i,j)}$; (b) Stored the textual description $d_{(i,j)}$ in P_i , represent as $P_i = \left\{\left(p_{(i,1)}, bb_{(i,1)}, d_{(i,1)}\right), \dots, \left(p_{(i,j)}, bb_{(i,j)}, d_{(i,j)}\right)\right\}\ (eq8)$;

3 Solutions of this proposal

6) Step 6, aggregate all patch-image, bounding-box and semantic description into a memory sliding window data structure denote as FrameStates, stored the P_i is in a sliding window FramesState, which holds up to n frames. The process can be represent as $FramesState = \{P_i | i = B - n + 1, \ldots, B\}$ $(eq9)$. If the size of the sliding window exceeds the batch frame length n represent as $|FrameStates| > n$, the oldest element P_{i-n} is removed from FramesState and transferred to ConsistentMemory. This mechanism can be represented as $ConsistentMemory \leftarrow FrameStates[i-n]$ $(eq10)$; Where $ConsistentMemory$ is a long-term, stable storage repository (such as a database); n is represents the maximum number of frames retained in the sliding window (batch size);

7) Step 7, after processing the entire batch, perform multi-agent cooperative reasoning over FramesState. The invention employs a multi-agent architecture comprising distinct software agents, each executing specific inference roles

   a) Manager Agent, evaluates the validity and clarity of each element $P_i \in FramesState$ received from the Visual Agent, this process was aggregate in memory data structure sliding window $FramesState$. If the description is incomplete, ambiguous, or inconsistent with the predefined visual defect detection task, the information is marked as "False" and transferred to the subsequent processing step;

3 Solutions of this proposal

b) Router Agent routes information to other agents based on logical conditions, specifically; (b1) If information is marked as "False" the Router Agent assigns a textual message "Description not clear" along with the corresponding image patch and bounding box coordinates, and returns this data to the Manager Agent for notification through the user interface; (b2) If information is marked as "Pass", indicating sufficient clarity and reliability, the Router Agent directly forwards the data to the System Agent for further aggregation;

c) Research Agent retrieves supplementary contextual knowledge from knowledge extraction models or a retrieval-augmented generation (RAG) system. The retrieved data may include domain-specific knowledge, technical documentation, or specialized guidelines tailored to particular product types.

d) System Agent aggregates information received from the Visual and Research Agents to construct FramesStates, facilitating the final inference step.

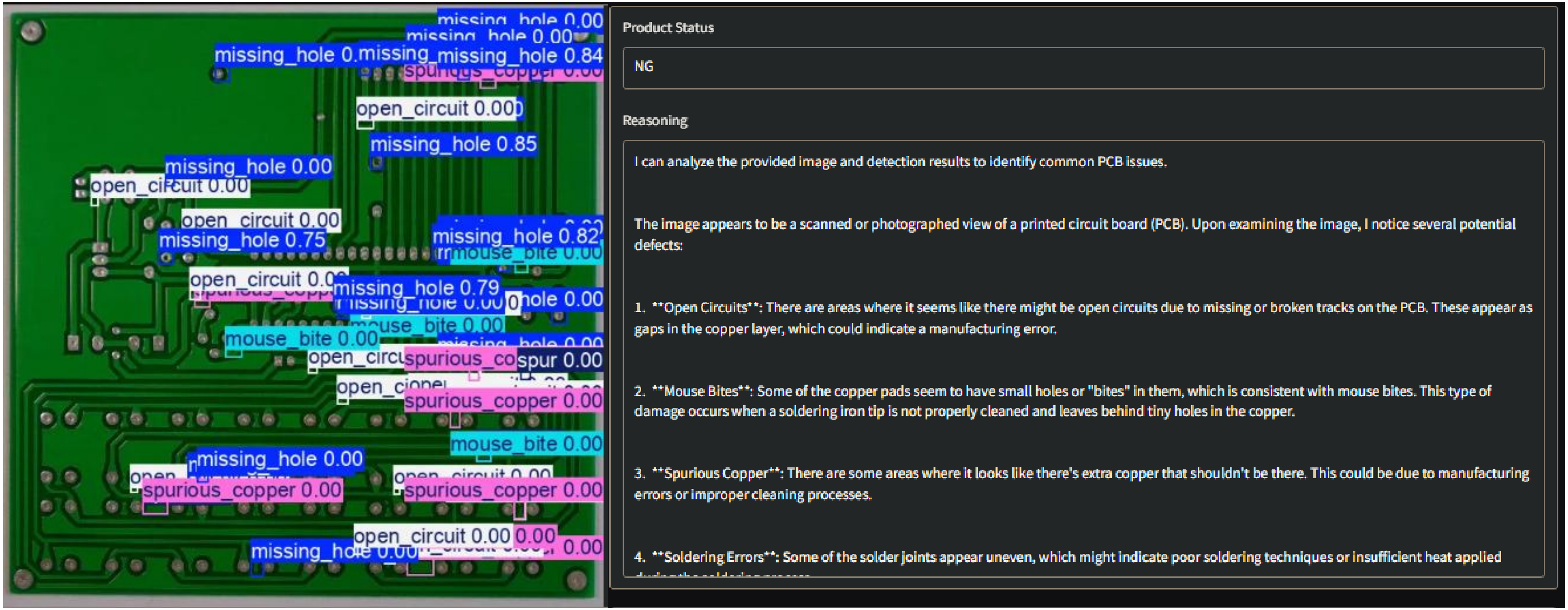
e) Reasoning Agent responsible for performing semantic inference based on aggregated data. This agent makes the final classification decision—NG (Not Good) or OK (Good)—accompanied by explanatory AI reasoning outputs that articulate the analytical logic, confidence levels, and recommended actions.

3 Solutions of this proposal

8) Step 8, make final classification and system-level response actions. Upon completion of multi-agent cooperative inference, the Reasoning Agent delivers the final classification decision for each input batch as either NG (Not Good) or OK (Good),

   a) if classified as NG (Not Good), the system displays a detailed defect alert on the operator interface, a red indicator light and auditory alarm (buzzer) are triggered to immediately notify production staff of the faulty item, the production line may optionally pause or tag the item for removal downstream;

   b) if classified as OK (Good), the interface clearly displays an "OK" confirmation message; the system automatically proceeds to process the next frame batch without interruption; this feedback mechanism ensures real-time defect isolation, supports immediate operator awareness, and maintains uninterrupted workflow continuity for non-defective products;



Product Status

NG

Reasoning

I can analyze the provided image and detection results to identify common PCB issues.

The image appears to be a scanned or photographed view of a printed circuit board (PCB). Upon examining the image, I notice several potential defects:

1. **Open Circuits**: There are areas where it seems like there might be open circuits due to missing or broken tracks on the PCB. These appear as gaps in the copper layer, which could indicate a manufacturing error.

2. **Mouse Bites**: Some of the copper pads seem to have small holes or "bites" in them, which is consistent with mouse bites. This type of damage occurs when a soldering iron tip is not properly cleaned and leaves behind tiny holes in the copper.

3. **Spurious Copper**: There are some areas where it looks like there's extra copper that shouldn't be there. This could be due to manufacturing errors or improper cleaning processes.

4. **Soldering Errors**: Some of the solder joints appear uneven, which might indicate poor soldering techniques or insufficient heat applied during the soldering process.

FIG. 10 illustration of automated PCB defect detection with visual description

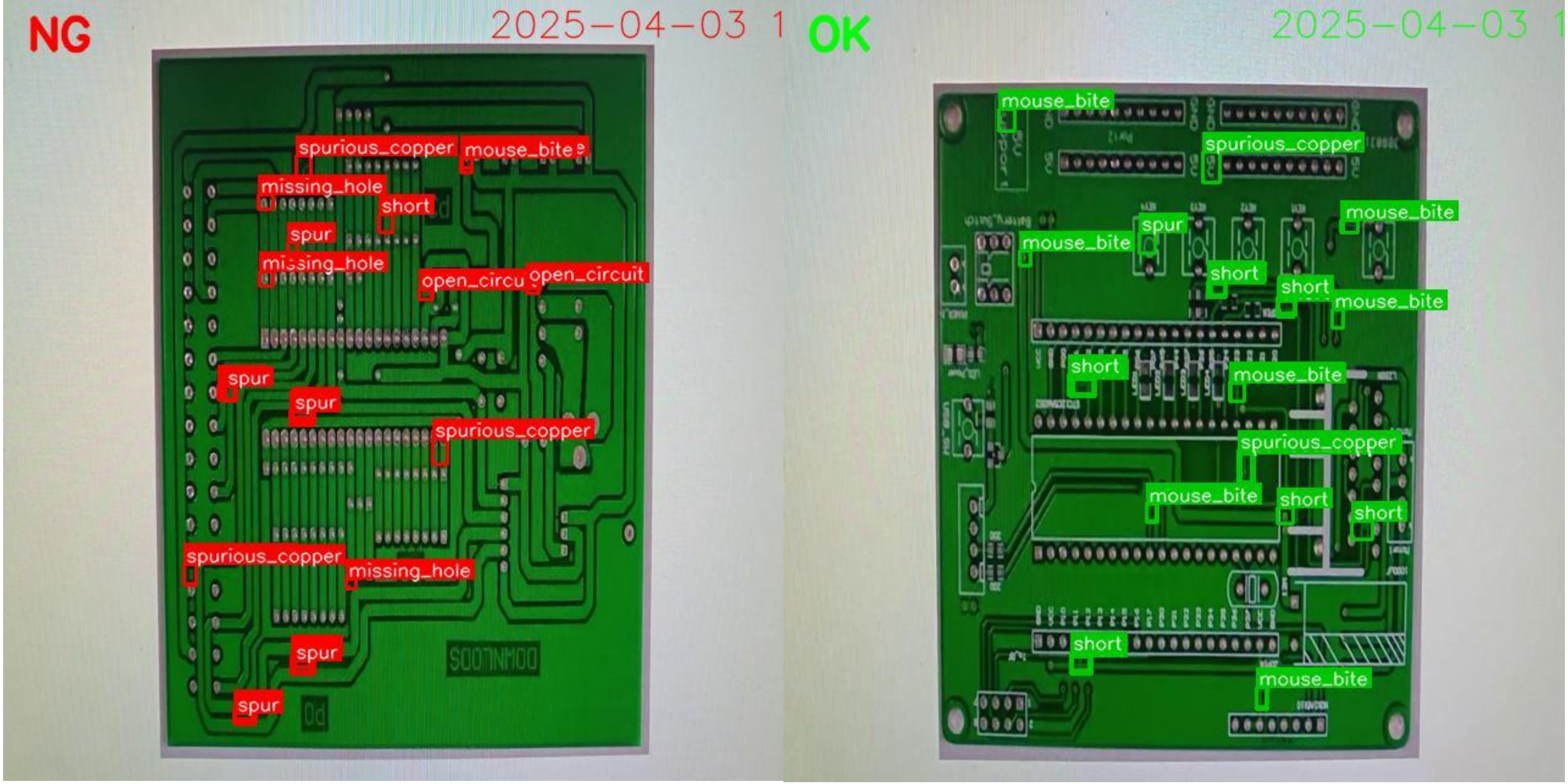


FIG. 11 illustrates the defect detection on different products, the results show NG/OK samples after the reasoning process with multi-agent system

4 Methods

4.1 System overview

The system is designed with a modular architecture

(1) An image acquisition module receives a batch of n frames from videos or industrial cameras

(2) An object detection module defect detection

(3) An image patch cropper module responsible for cropping regions defined by bounding boxes using a specified scaling factors

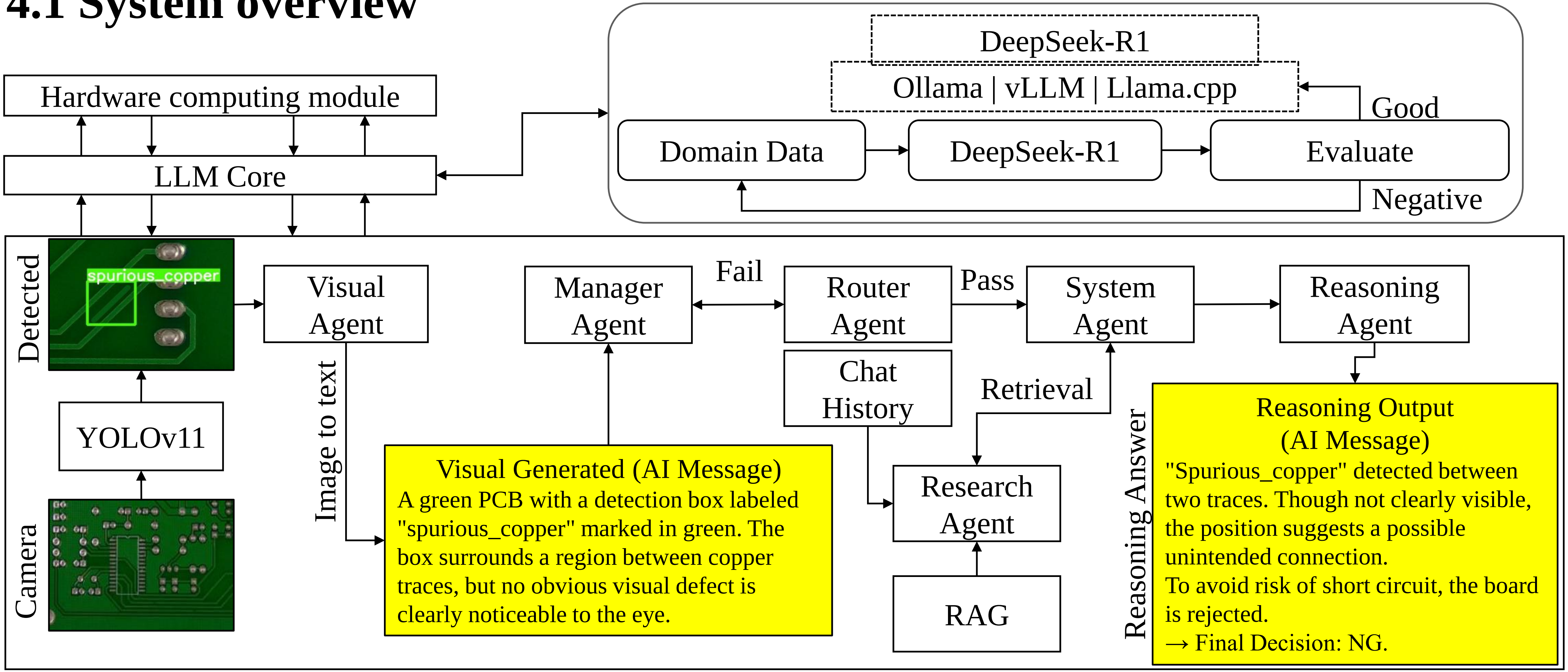
(4) A vision-language module responsible for generating natural language descriptions for each image patch

(5) A memory data structure, stores descriptions in a list format, utilizing a sliding-window mechanism of size n to discard processed descriptions

(6) A multi-agent system with distinct roles but working collaboratively to analyze and synthesize information comprises (a) visual agent; (b) manager agent; (c) router agent; (d) research agent; (e) reasoning agent

4 Methods

4.1 System overview



Visual Generated (AI Message)

A green PCB with a detection box labeled "spurious_copper" marked in green. The box surrounds a region between copper traces, but no obvious visual defect is clearly noticeable to the eye.

Reasoning Output (AI Message)

"Spurious_copper" detected between two traces. Though not clearly visible, the position suggests a possible unintended connection.

To avoid risk of short circuit, the board is rejected.

→ Final Decision: NG.

FIG. 2 illustrates the overall structure of the defect detection and classification system based on multi-agent reasoning

4 Methods

4.2 Object Defect Detection Pipeline

The process comprises the following steps

(1) The system receives input as a batch of n frames, represented as set B

$$B = \{f_i \mid i = 1,2,3, \ldots, n\} \qquad (eq1)$$

Where, f_i denotes the frame at index i; n denotes the number of frames in one batch

(2) Applying the object detection model, each frame f_i is processed, yielding a set of bounding boxes represented as

$$BB_i = \left\{bb_{(i,1)}, bb_{(i,2)}, \ldots, bb_{(i,j)}\right\} \qquad (eq2)$$

Each bounding box $bb_{(i,j)}$ at frame i and index j is represented as

$$bb_{(i,j)} = (x_1, y_1, x_2, y_2, c, l) \qquad (eq3)$$

Where, j is bounding-box's indet at frame index I (f_i); $\{x_1, y_1, x_2, y_2\}$ is coordinates of bounding box; c is the confidence score of YOLO object detection ($0 \leq c \leq 1$); l is the detected object label

4 Methods

4.2 Object Defect Detection Pipeline

(3) Extracting image patch from bounding boxes, each bounding box $bb_{(i,j)}$ is cropped from the corresponding frame f_i, yielding patches $p_{(i,j)}$ stored in P_i

$$p_{(i,j)} = crop\left(f_i, bb_{(i,j)}, s\right) \quad (eq4)$$

$$P_i = \left\{p_{(i,1)}, p_{(i,2)}, \ldots, p_{(i,j)}\right\} \quad (eq5)$$

Where, s is the scaling factor for adjusting bounding box size; $crop()$ is the cropping function based on bounding box coordinates and scale factor s

(4) Each patch $p_{(i,j)}$ is packaged with its bounding box to enable the manager agent to display defect coordinates on the user interface if the router agent confirms the patch description is invalid

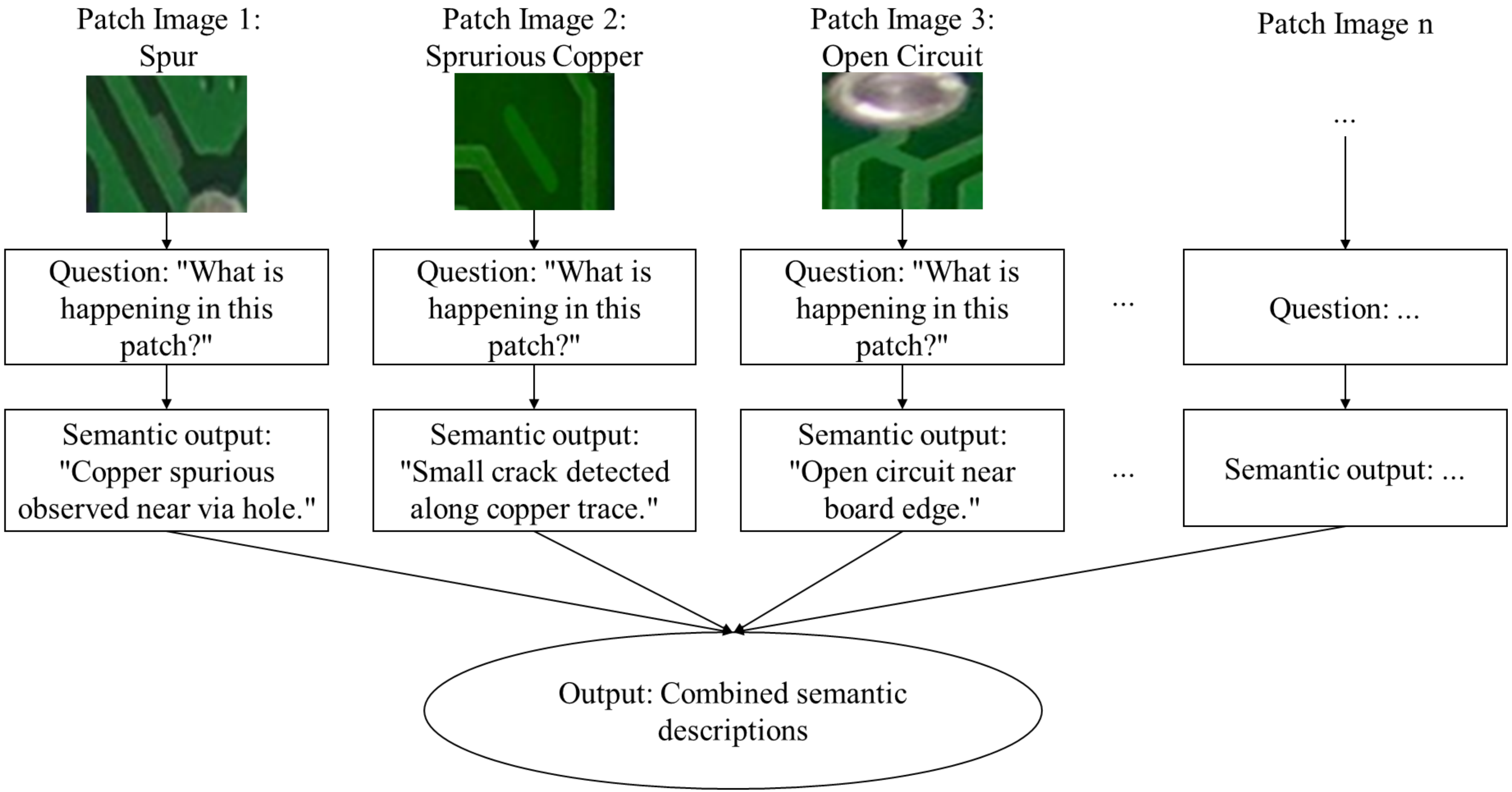
$$P_i = \{\left(p_{(i,1)}, bb_{(i,1)}\right), \left(p_{(i,2)}, bb_{(i,2)}\right), \ldots, \left(p_{(i,j)}, bb_{(i,j)}\right)\} \qquad (eq6)$$

Where, P_i denotes the list of patches; $p_{(i,j)}$is the patch image at frame i and index j; $bb_{(i,j)}$is bounding-box coordinates

(5) Optimizing the detection model, the detection pipeline is implemented in parallel across multiple GPU computing devices to ensure real-time processing speed in production environments

4 Methods

4.3 Vision-Language Semantic Description



Patch Image 1: Spur
Patch Image 2: Sprurious Copper
Patch Image 3: Open Circuit
Patch Image n
...
Question: "What is happening in this patch?"
Question: "What is happening in this patch?"
Question: "What is happening in this patch?"
...
Question: ...
Semantic output: "Copper spurious observed near via hole."
Semantic output: "Small crack detected along copper trace."
Semantic output: "Open circuit near board edge."
...
Semantic output: ...
Output: Combined semantic descriptions

FIG.3 illustrates modeling defect structures as textual descriptions using a vision-language model

4 Methods

4.3 Vision-Language Semantic Description

This process comprises the following comprises

(1) Encode each image patch $p_{(i,j)} \in P_i$, the process is represented as

$$d_{(i,j)} = VLM\left(p_{(i,j)}\right) \qquad (eq7)$$

Where, $d_{(i,j)}$ is the semantic textual generated from patch $p_{(i,j)}$

(2) Stored the textual description $d_{(i,j)}$ in P_i, the process is represent as

$$P_i = \left\{\left(p_{(i,1)}, bb_{(i,1)}, d_{(i,1)}\right), \ldots, \left(p_{(i,j)}, bb_{(i,j)}, d_{(i,j)}\right)\right\} \qquad (eq8)$$

(3) Stored the P_i is in a sliding window FramesState, which holds up to n frames. If full, the oldest element ismoved to ConsistentMemory further retrievel. The process can be represent as

$$FramesState = \{P_i | i = B - n + 1, \ldots, B\} \qquad (eq9)$$

4 Methods

4.3 Vision-Language Semantic Description

(5) When the size of the sliding window exceeds the batch frame length n, the process is represent as $|FrameStates| > n$, the oldest element P_{i-n} is removed from FramesState and transferred to ConsistentMemory. This mechanism is represented by

$$ConsistentMemory \leftarrow FrameStates[i-n] \ (eq10)$$

Where, $ConsistentMemory$ is a long-term, stable storage repository (such as a database); n is represents the maximum number of frames retained in the sliding window (batch size)

(6) Using the VLM for semantic descriptions offers key benefits: it enables integration with reasoning systems, supports AI agents with context-aware inputs, improves explainability through textual outputs, and allows flexible handling of uncertainty via confidence thresholds

4 Methods

4.4. Retrieval-Augmented Generation

To enhance inference by providing context from a vectorized database of QA documents. The Research Agent combines the current query with retrieved semantic data, which is then fed into an LLM to generate context-aware responses

This process can be represent as follows

$$C = Retrieve(FrameStates, QADB) \qquad (eq10)$$

$$RAG(FrameState) = LLM(FrameStates, C) \qquad (eq11)$$

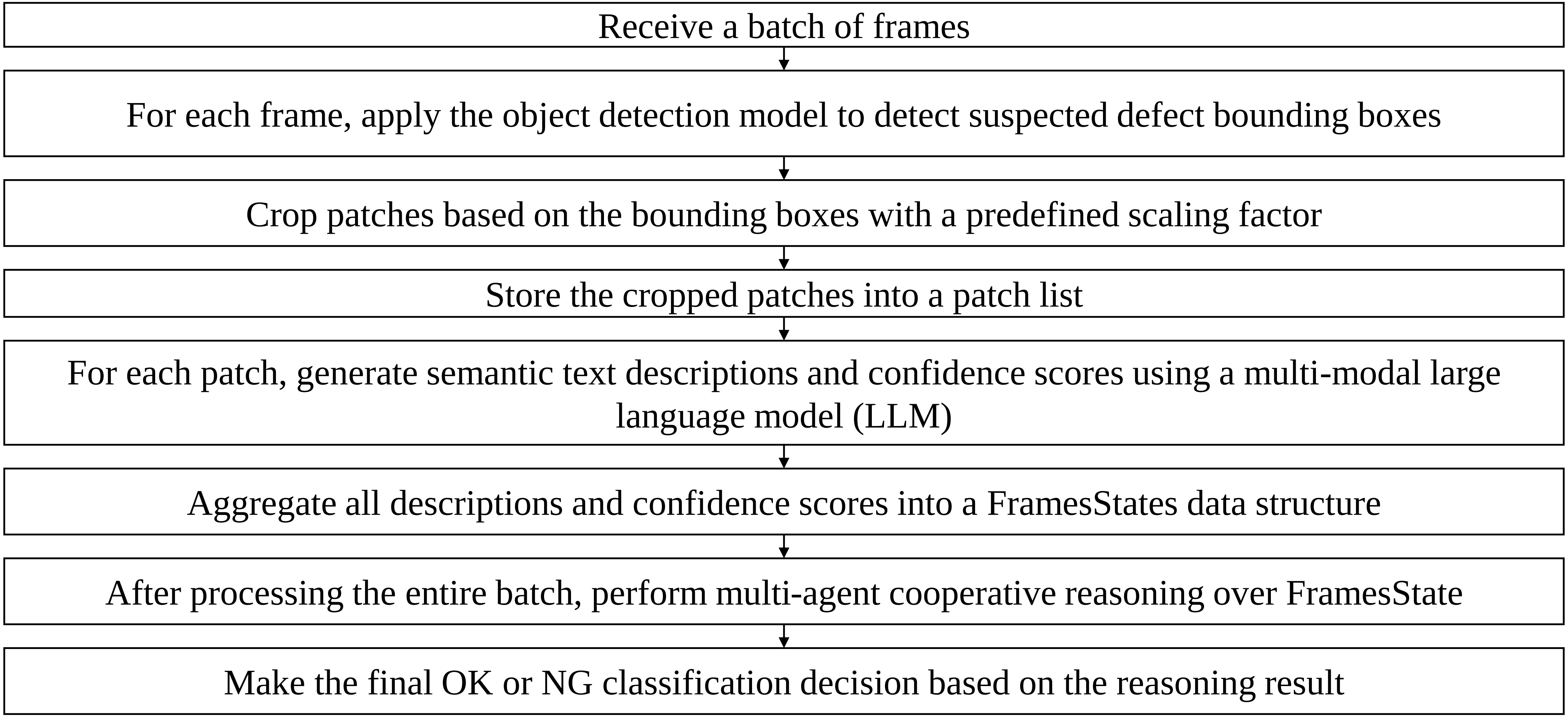
Where

$QADB$ is the vectorized database extracted from human-curated quality assurance documents

$Retrieve(FrameStates, QADB)$ is retrieval function that extracts relevant context from QADB based on the content of FrameStates.
$LLM(FrameStates, C)$: A large language model receiving FrameStates as the query and C as retrieved context, producing an augmented inference response
$RAG(FrameStates)$is the resulting generated by the RAG

4 Methods

4.5. Description of multi-agent inference mechanism



Receive a batch of frames

For each frame, apply the object detection model to detect suspected defect bounding boxes

Crop patches based on the bounding boxes with a predefined scaling factor

Store the cropped patches into a patch list

For each patch, generate semantic text descriptions and confidence scores using a multi-modal large language model (LLM)

Aggregate all descriptions and confidence scores into a FramesStates data structure

After processing the entire batch, perform multi-agent cooperative reasoning over FramesState

Make the final OK or NG classification decision based on the reasoning result

FIG. 6 illustrates the schematic flowchart of defect detection across an image sequence using batch aggregation

4 Methods

4.5. Description of multi-agent inference mechanism

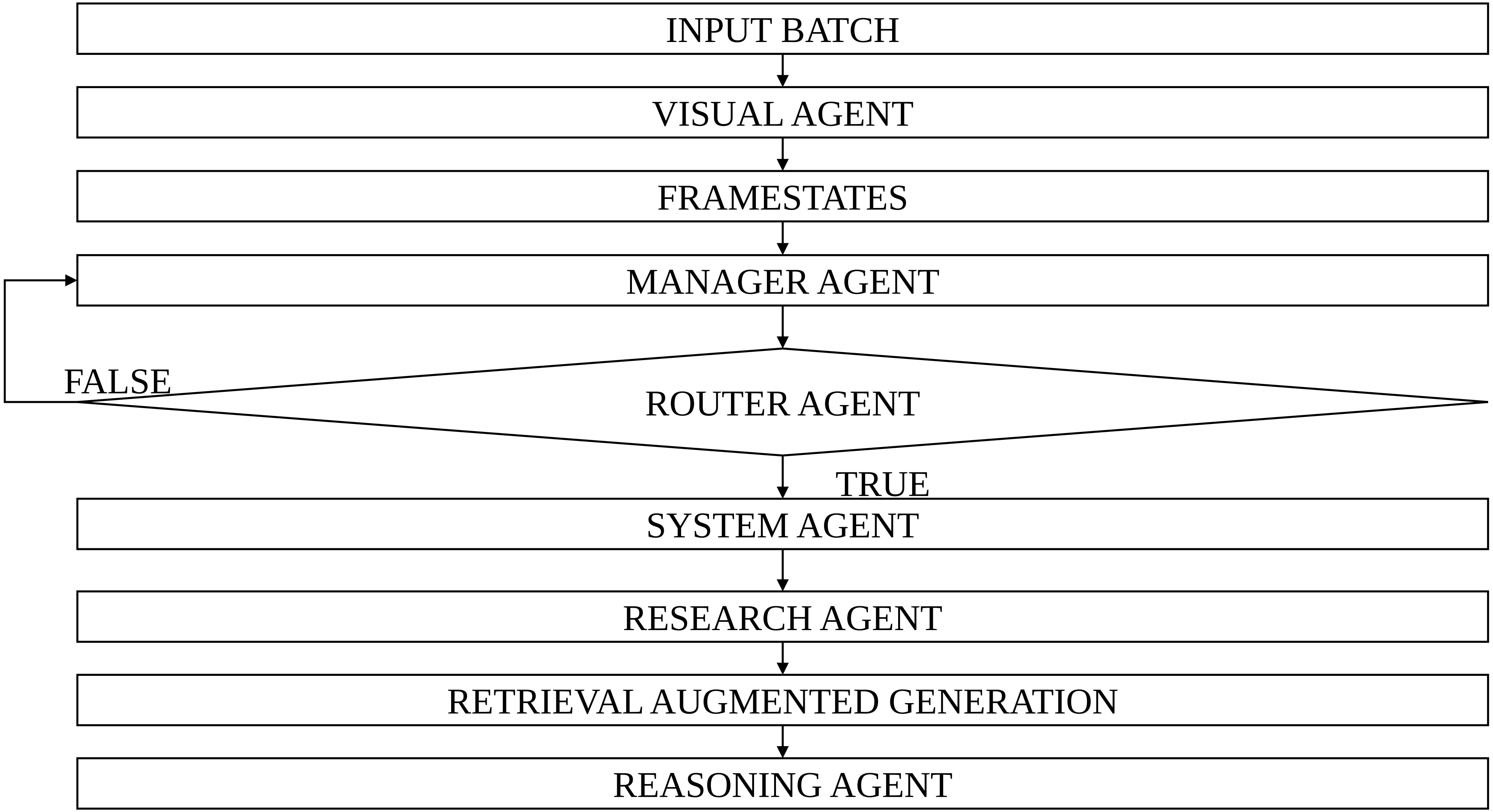
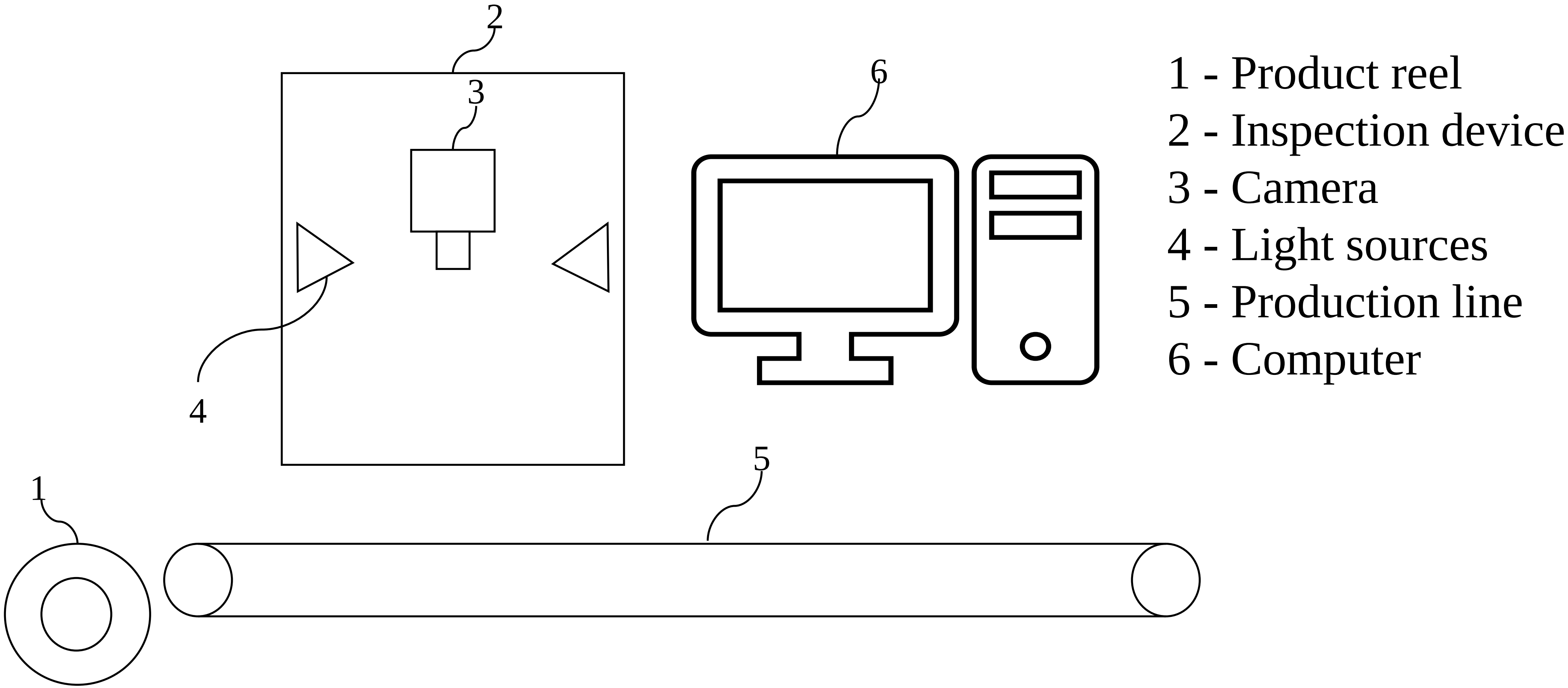


FIG. 7 illustrates the system inference coordination workflow

4 Methods

4.6. Hardware Configuration for Visual Inspection



1 - Product reel
2 - Inspection device
3 - Camera
4 - Light sources
5 - Production line
6 - Computer

FIG. 8 illustrates the hardware module device used for product inspection

4 Methods

4.6. Hardware Configuration for Visual Inspection

The hardware configuration includes the following core components, as illustrated in FIG. 8:

(1) Product reel supplies the input material in a continuous roll format for the production line.

(2) Inspection device, in this embodiment it is a unit comprising an integrated camera and lighting system installed above the inspection zone for capturing surface images of the product.

(3) Camera, in this embodiment it is a high-resolution industrial camera, fixed within the inspection device (2), responsible for acquiring image data.

(4) Light sources, in this embodiment it is a directed LED or synchronized lighting modules mounted alongside the camera to optimize image contrast during acquisition.

(5) Production line, in this embodiment it is a conveyor or transport mechanism that moves the product through the inspection area at a consistent speed and alignment.

(6) Computer, executes the full AI-based visual inspection pipeline, including defect detection, semantic description, temporal aggregation, and multi-agent reasoning. In some embodiments, this computer may be replaced with a high-performance GPU server depending on the computational requirements.

4 Methods

4.7. Finetune object detection model

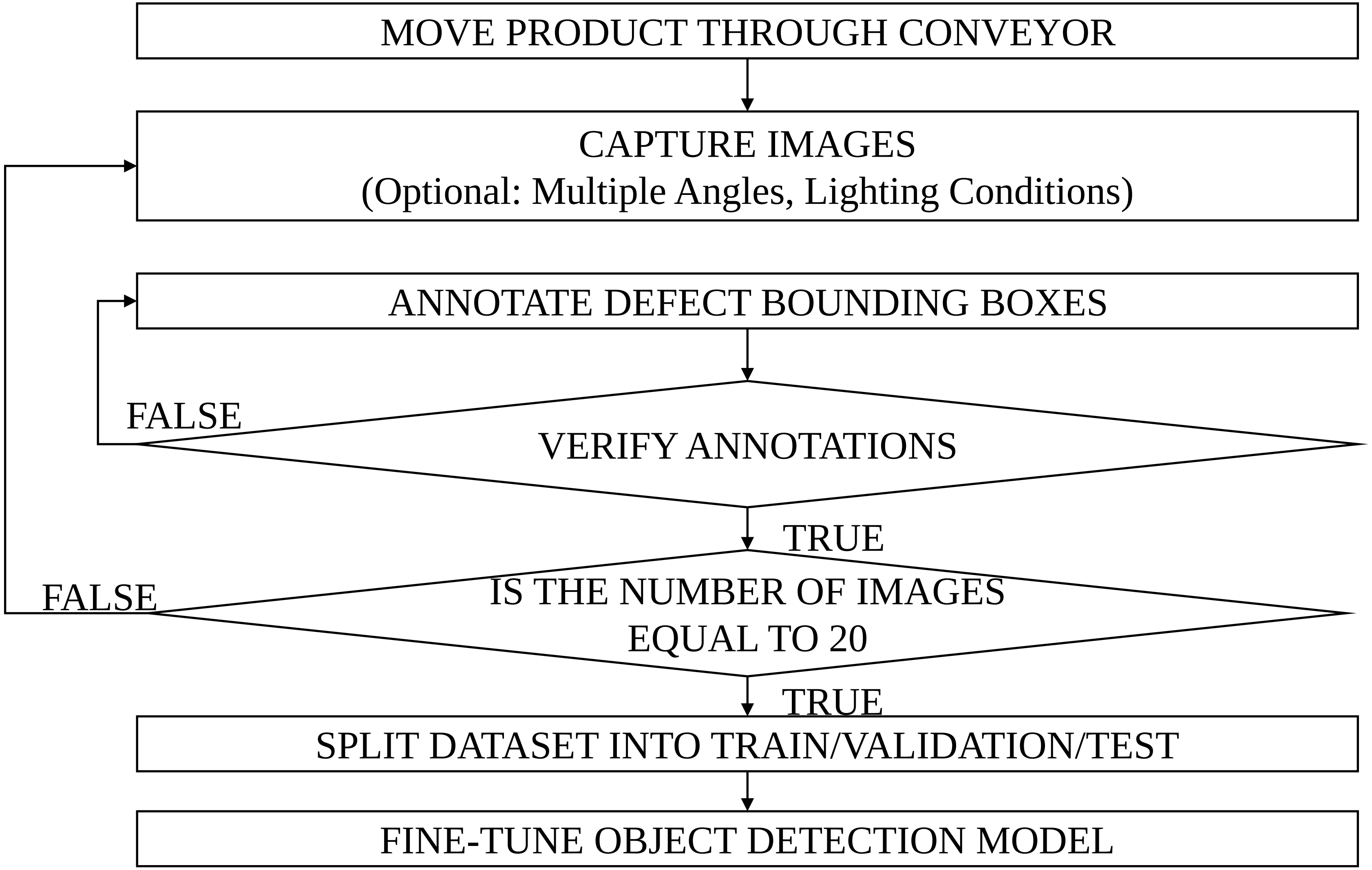

MOVE PRODUCT THROUGH CONVEYOR
CAPTURE IMAGES
(Optional: Multiple Angles, Lighting Conditions)
ANNOTATE DEFECT BOUNDING BOXES
FALSE
VERIFY ANNOTATIONS
TRUE
FALSE
IS THE NUMBER OF IMAGES EQUAL TO 20
TRUE
SPLIT DATASET INTO TRAIN/VALIDATION/TEST
FINE-TUNE OBJECT DETECTION MODEL

FIG. 9 illustrates the data collection process and the fine-tuning procedure for the object detection model

4 Methods

4.7. Finetune object detection model

The object detection module is initially instantiated from a pre-trained network architecture, such as YOLO or SSD. The fine-tuning procedure comprises the following steps

(1) Select a fine-tuning dataset, a compact dataset is chosen that includes a small number of annotated images, where each image contains bounding boxes corresponding to the visual defect categories to be detected.

(2) Perform fine-tuning comprises following steps

  (a) initialize model weights from the pre-trained version.

  (b) use the fine-tuning dataset to update the output layers (and optionally intermediate layers) over a limited number of training epochs with a constrained learning rate.

  (c) monitor performance metrics such as confidence score and Intersection over Union (IoU) on a small validation set to ensure that the model correctly learns the defect patterns.

(3) Deploy the fine-tuned detection module, the fine-tuned object detection model returns preliminary bounding boxes with confidence scores. These results are subsequently passed through the semantic reasoning pipeline and multi-agent inference process for further verification and refinement.

5 Photos

FIG. 1 illustrates YOLO’s defect detection performance in image sequences under varying ambient lighting conditions

FIG. 2 illustrates the overall structure of the defect detection and classification system based on multi-agent reasoning

FIG. 3 illustrates modeling defect structures as textual descriptions using a vision-language model

FIG. 4 illustrates an example of semantic textual descriptions of image patches within a single frame

FIG. 5 illustrates an example of aggregating semantic textual data for multiple patches across multiple frames

FIG. 6 illustrates the schematic flowchart of defect detection across an image sequence using batch aggregation

FIG. 7 illustrates the system inference coordination workflow

FIG. 8 illustrates the hardware module device used for product inspection

FIG. 9 illustrates the data collection process and the fine-tuning procedure for the object detection model

FIG. 10 illustration of automated PCB defect detection with visual description

FIG. 11 illustrates the defect detection on different products, the results show NG/OK samples after the reasoning process with multi-agent system

FIG. 12 illustrates the different layout printed circuits board

FIG. 13 illustrates the batch with n frame

FIG. 14 illustrates hardware resource usage, in this embodiment is demo system with GPU-RTX4090, model core use is llama-3.2-11b-vision-instruct and yolov11-nano

5 Photos

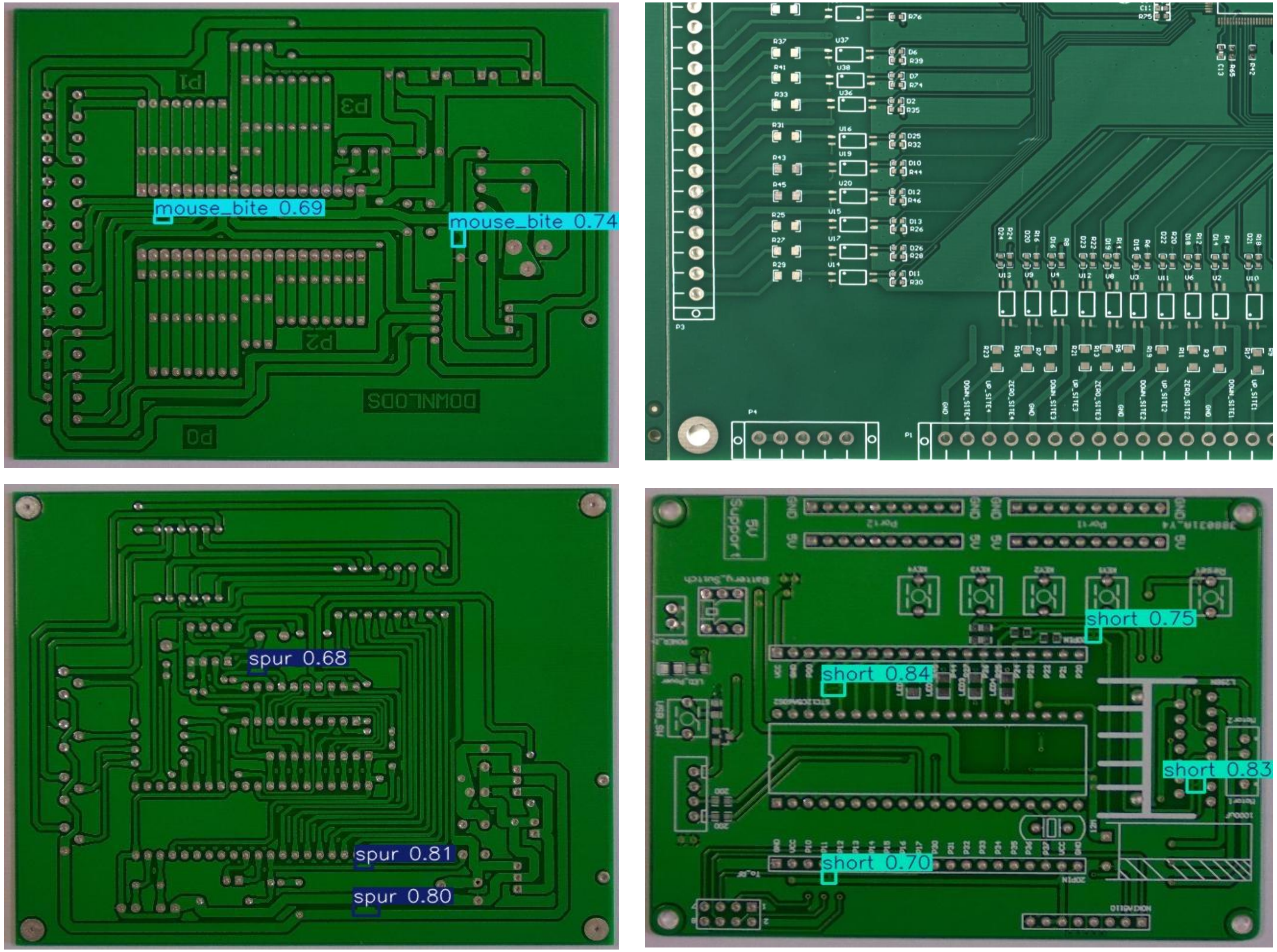


FIG. 12 illustrates the different layout printed circuits board

5 Photos

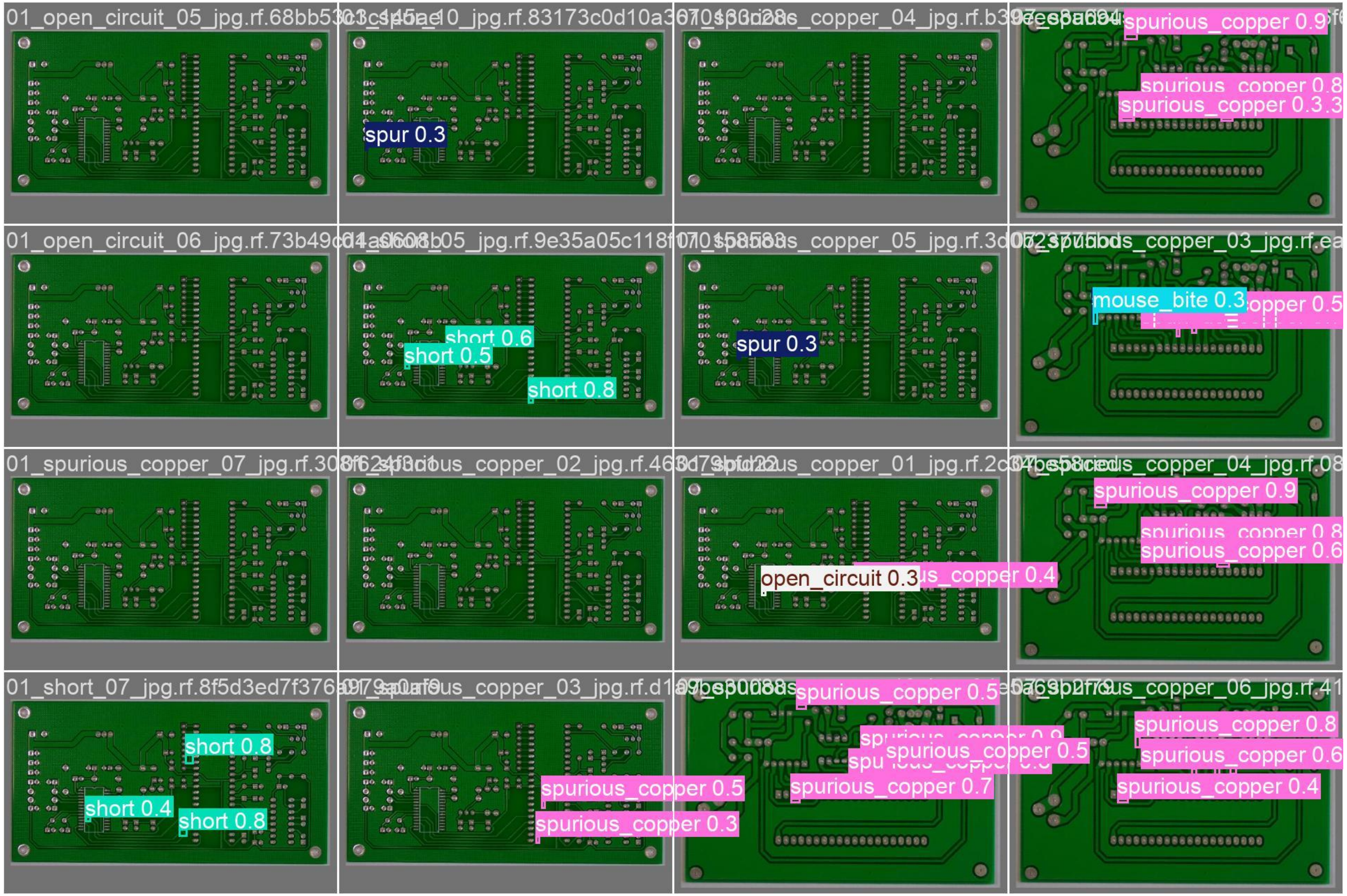
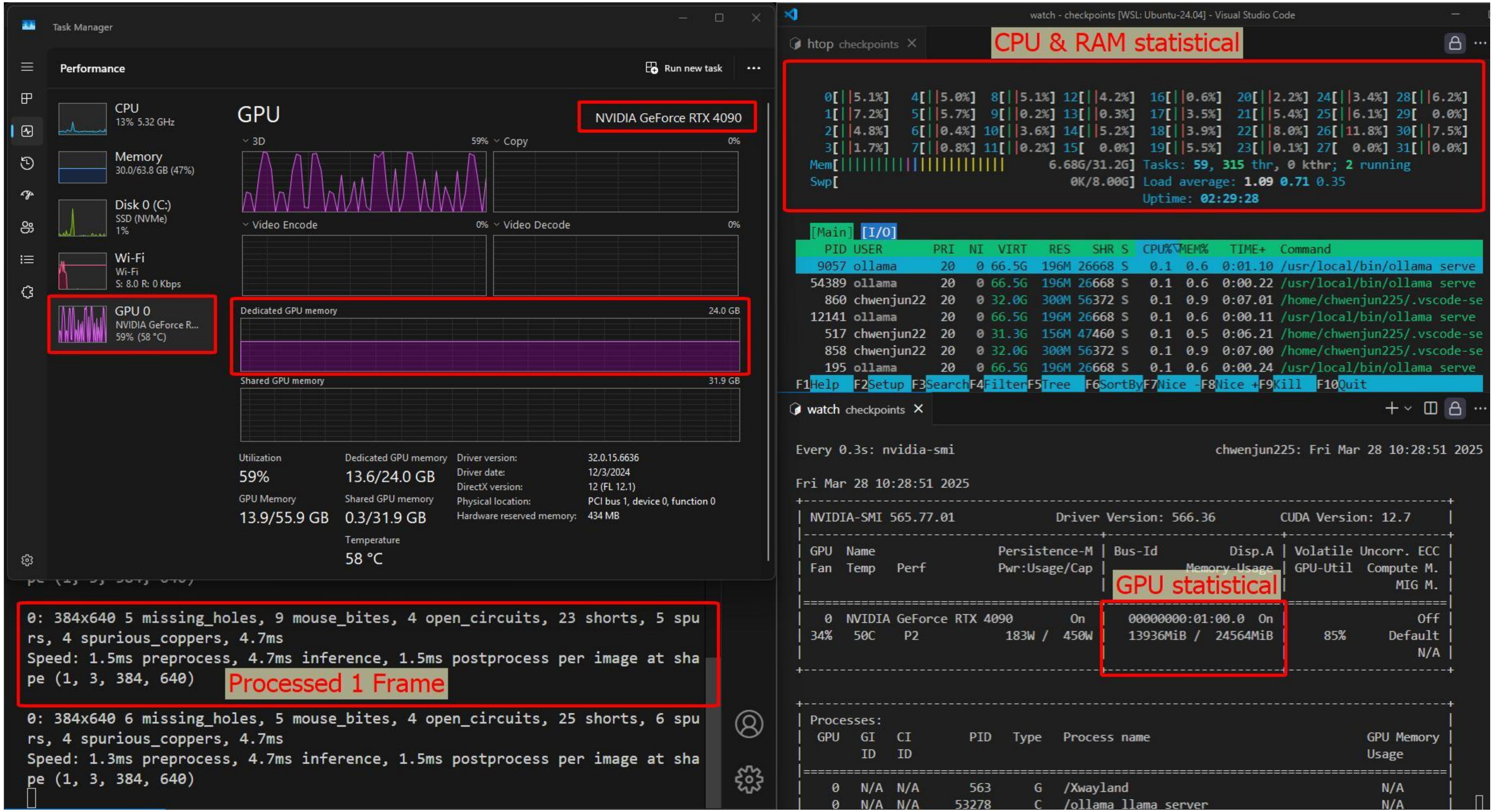


FIG. 13 illustrates the batch with n frame

5 Photos



CPU & RAM statistical

GPU statistical

Processed 1 Frame

Utilization 59%

Dedicated GPU memory 13.6/24.0 GB

GPU Memory 13.9/55.9 GB

Shared GPU memory 0.3/31.9 GB

Temperature 58 °C

Driver version: 32.0.15.6636
Driver date: 12/3/2024
DirectX version: 12 (FL 12.1)
Physical location: PCI bus 1, device 0, function 0
Hardware reserved memory: 434 MB

```
    0[||5.1%]   4[||5.0%]  8[||5.1%] 12[||4.2%]  16[||0.6%]  20[||2.2%] 24[||3.4%] 28[||6.2%]
    1[||7.2%]   5[||5.7%]  9[||0.2%] 13[||0.3%]  17[||3.5%]  21[||5.4%] 25[||6.1%] 29[  0.0%]
    2[||4.8%]   6[||0.4%] 10[||3.6%] 14[||5.2%]  18[||3.9%]  22[||8.0%] 26[|11.8%] 30[||7.5%]
    3[||1.7%]   7[||0.8%] 11[||0.2%] 15[  0.0%]  19[||5.5%]  23[||0.1%] 27[  0.0%] 31[||0.0%]
  Mem[||||||||||||||||||||||       6.68G/31.2G] Tasks: 59, 315 thr, 0 kthr; 2 running
  Swp[                               0K/8.00G] Load average: 1.09 0.71 0.35
                                               Uptime: 02:29:28

  [Main] [I/O]
    PID USER       PRI  NI  VIRT   RES   SHR S  CPU%▽MEM%   TIME+  Command
   9057 ollama      20   0 66.5G  196M 26668 S   0.1  0.6  0:01.10 /usr/local/bin/ollama serve
  54389 ollama      20   0 66.5G  196M 26668 S   0.1  0.6  0:00.22 /usr/local/bin/ollama serve
    860 chwenjun22  20   0 32.0G  300M 56372 S   0.1  0.9  0:07.01 /home/chwenjun225/.vscode-se
  12141 ollama      20   0 66.5G  196M 26668 S   0.1  0.6  0:00.11 /usr/local/bin/ollama serve
    517 chwenjun22  20   0 31.3G  156M 47460 S   0.1  0.5  0:06.21 /home/chwenjun225/.vscode-se
    858 chwenjun22  20   0 32.0G  300M 56372 S   0.1  0.9  0:07.00 /home/chwenjun225/.vscode-se
    195 ollama      20   0 66.5G  196M 26668 S   0.1  0.6  0:00.24 /usr/local/bin/ollama serve
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

```
Every 0.3s: nvidia-smi                                 chwenjun225: Fri Mar 28 10:28:51 2025

Fri Mar 28 10:28:51 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 565.77.01              Driver Version: 566.36         CUDA Version: 12.7     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce RTX 4090        On  |   00000000:01:00.0  On |                  Off |
| 34%   50C    P2            183W /  450W |   13936MiB /  24564MiB |     85%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|    0   N/A  N/A       563      G   /Xwayland                                     N/A    |
|    0   N/A  N/A     53278      C   /ollama llama server                          N/A    |
```

```
pe (1, 3, 384, 640)

0: 384x640 5 missing_holes, 9 mouse_bites, 4 open_circuits, 23 shorts, 5 spu
rs, 4 spurious_coppers, 4.7ms
Speed: 1.5ms preprocess, 4.7ms inference, 1.5ms postprocess per image at sha
pe (1, 3, 384, 640)

0: 384x640 6 missing_holes, 5 mouse_bites, 4 open_circuits, 25 shorts, 6 spu
rs, 4 spurious_coppers, 4.7ms
Speed: 1.3ms preprocess, 4.7ms inference, 1.5ms postprocess per image at sha
pe (1, 3, 384, 640)
```

FIG. 14 illustrates hardware resource usage, in this embodiment is demo system with GPU-RTX4090, model core use is llama-3.2-11b-vision-instruct and yolov11-nano

5 Photos

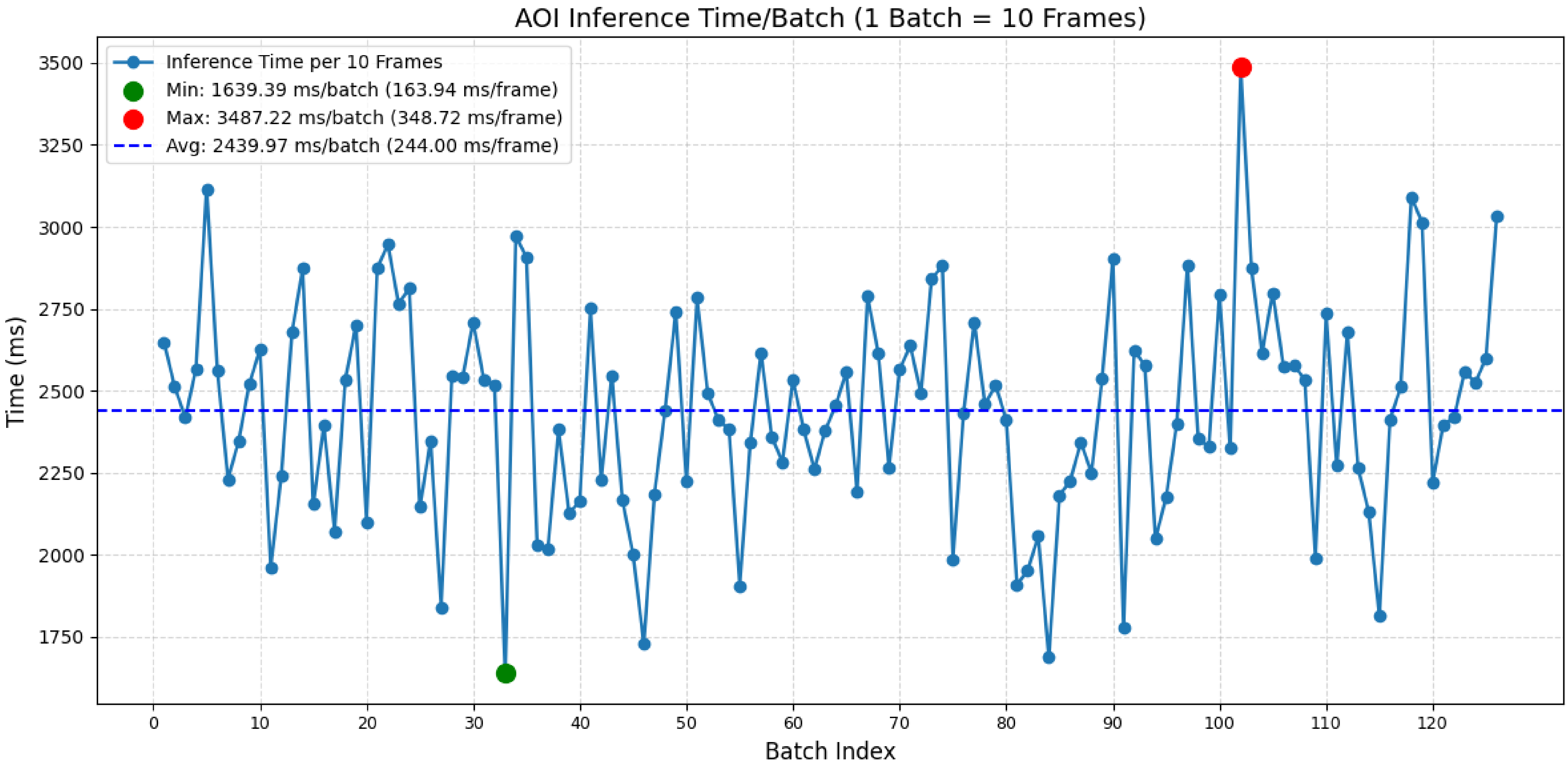


FIG. 15 illustrates inference time per batch, with GPU-RTX4090, model core use is llama-3.2-11b-vision-instruct and yolov11-nano

5 Photos

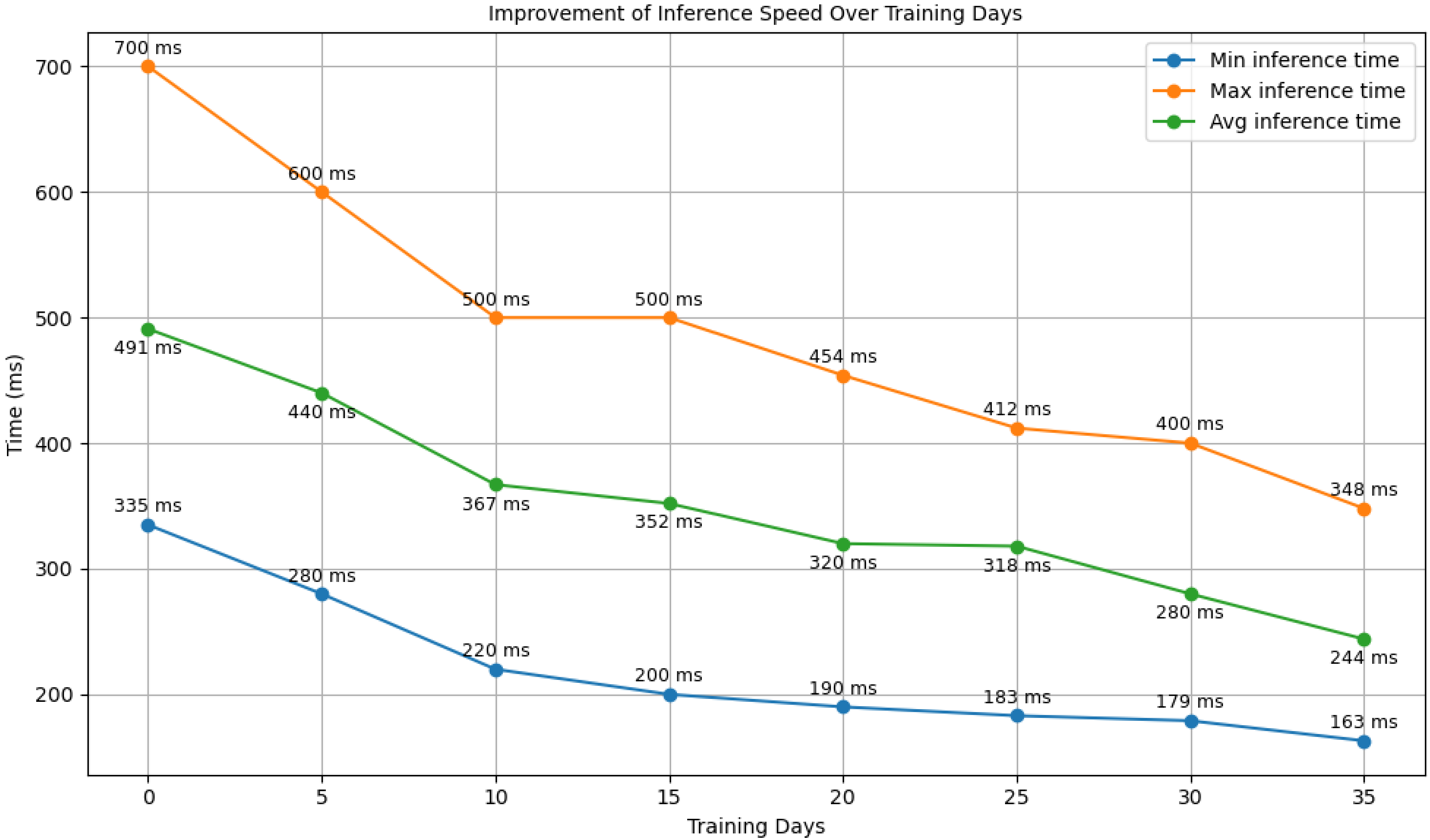


FIG. 16 illustrates the progress of improvement in speed

5 Photos

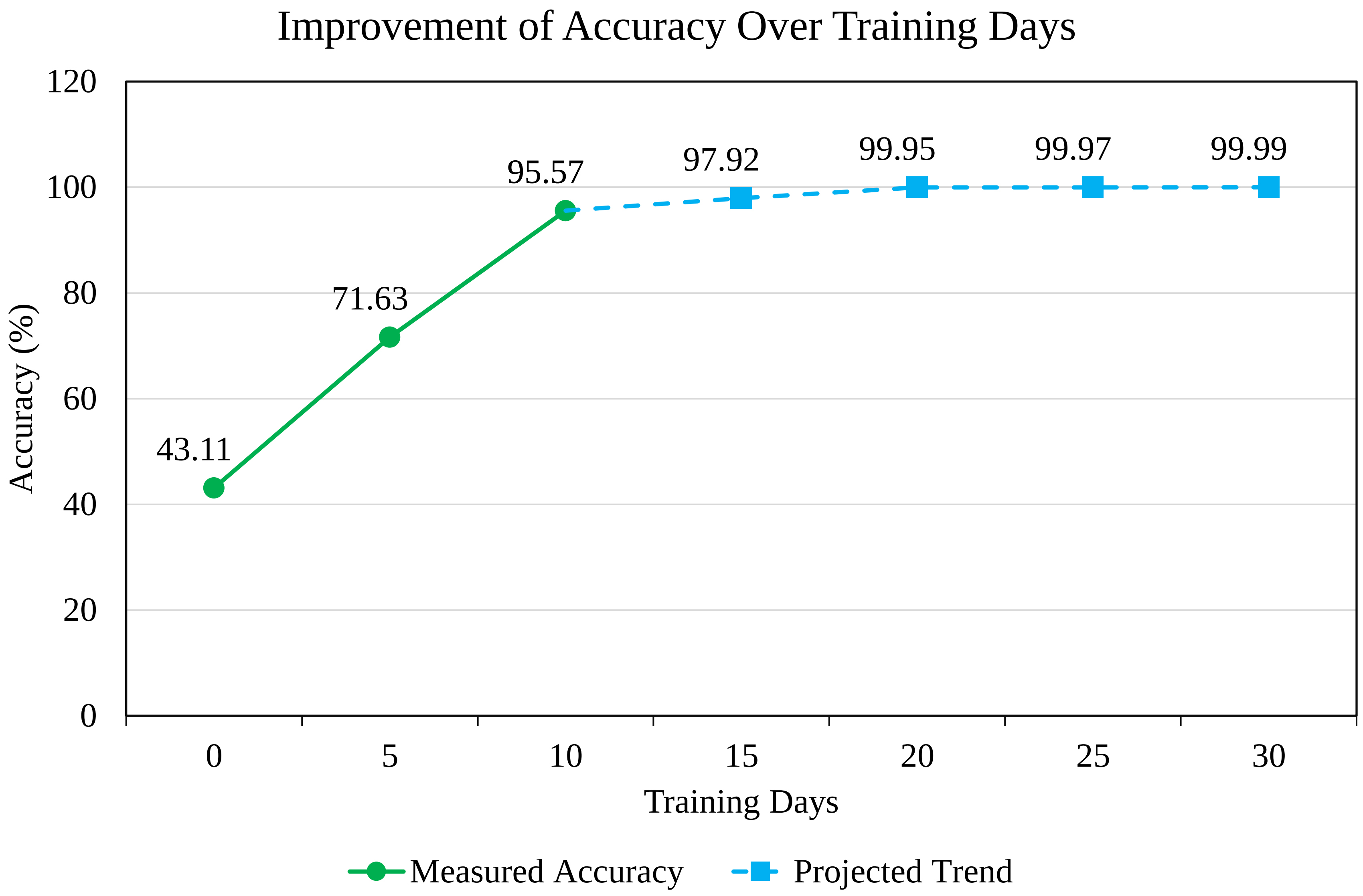


FIG. 17 illustrates the progress of improvement in accuracy

6 Advantages

The proposed invention offers several key advantages over existing visual inspection systems:

(1) High Accuracy Through Multi-Agent Reasoning, by integrating a cooperative multi-agent architecture (including visual, manager, router, research, system, and reasoning agents), the system significantly reduces false positives and false negatives. Each defect is verified semantically and contextually across multiple frames before making a final OK/NG decision.

(2) Explainability and Semantic Transparency, unlike traditional black-box vision models, this system uses a vision-language model to generate natural language descriptions for each defect. These semantic outputs not only assist AI agents in reasoning but also improve explainability for human supervisors.

(3) Temporal Consistency for Robust Detection, defect detection is performed on batches of frames using a sliding-window memory structure. This enables temporal aggregation and filtering, making the system resilient to transient noise, lighting variations, and frame-level inconsistencies often found in video sequences.

6 Advantages

(4) Low Data Requirement and Fast Adaptation, the system only requires a small fine-tuning dataset to adapt the object detection model to new defect types. This dramatically reduces data labeling costs and speeds up deployment in new environments or product lines.

(5) Real-Time Inspection on Production Line, leveraging parallel processing on GPU hardware and lightweight models such as YOLOv11-nano and LLaMA-3.2-11b, the system achieves real-time performance suitable for high-speed manufacturing.

(6) Cross-Product Flexibility, designed with a modular architecture and compatible with various camera setups and product layouts, the system is adaptable to a wide range of industrial applications (e.g., PCBs, semiconductors, mechanical parts).

(7) Automated Decision and Alert System, upon detection of NG-classified defects, the system can trigger automatic alerts and even halt the production line, preventing defective goods from proceeding downstream.

Thank you