PATENT APPLICATION

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

Type of report to be declared: Patent Technical Disclosure

Inventor of the patent: Tran Van Tuan, Zhang Ai Ping

Writer and technician of the technical disclosure document: Tran Van Tuan, Zhang Ai Ping

Phone: +86342184225

Email: trantuan22052k@gmail.com

ABSTRACT

The present invention discloses a method and system for automatically detecting and classifying visual surface defects across various types of products using a cooperative multi-agent reasoning mechanism. The method includes applying an object detection model to identify suspected defect regions within input image sequences, cropping these regions into image patches, and employing a vision-language model to generate semantic descriptions along with confidence scores derived from the object detection model. These descriptions are aggregated into a structured list-based data format. AI agents subsequently analyze the aggregated semantic data to determine a final classification of NG (Not Good) or OK, based on temporal recurrence and semantic consistency of detected defects.

FIELD OF THE INVENTION

This invention relates to automated visual inspection systems and, in particular, multi-agent reasoning to increase defect detection accuracy.

BACKGROUND OF THE INVENTION

1 The technical problem to be solved

1.1 High rates of false positives and false negatives in existing visual defect detection systems due to lack of collaborative reasoning

Current inspection processes primarily rely on the output of a single object detection model. Although these models have been trained on large, fully labeled datasets, they frequently misclassify minor defects such as mouse bites, spurious copper, open circuits, and short circuits during printed circuit board (PCB) visual inspections. Due to the extremely small size and intricate nature of these defects, object detection models are prone to high rates of false positives and false negatives, despite extensive training [Radford et al., 2021].

Currently, defect detection systems utilize either standalone deep learning models or individual algorithms for defect detection. However, this approach does not fully exploit the potential of coordinated interactions among multiple specialized agents within a system. Reliance on a single model inherently limits accuracy in defect identification and classification, as no individual model can adequately perform all necessary inspection tasks [Liu, 2024].

If each agent within the system were optimized for a specific task (e.g., identifying suspect regions, providing detailed defect descriptions, verifying defect accuracy), then each detected defect could be cross-verified by another agent, with yet another agent providing an in-depth description of the defect's nature. However, existing methods have not yet optimized inter-agent coordination, resulting in significant limitations in accurate defect detection in complex manufacturing environments [Chen et al., 2024].

1.2 Degraded defect detection performance of yolo in image sequences under varying environmental conditions

Popular object detection models such as YOLO (You Only Look Once) perform effectively on individual frames in real-time. However, YOLO can generate inconsistent candidate bounding boxes across sequential frames due to varying lighting conditions, motion blur, or occlusions [Tung et al., 2018]. When applied independently to each frame, YOLO may overlook persistent defects or misclassify transient noise as defects.

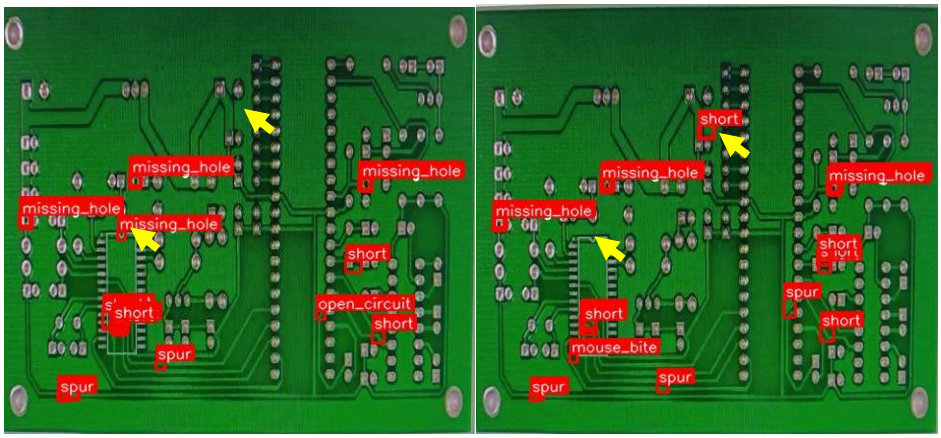

FIG. 1 illustrates YOLO's defect detection performance in image sequences under varying ambient lighting conditions.

Defects may appear only briefly in a small portion of a video sequence or even in a single frame, causing YOLO to easily miss these defects. Although improvements have been made to extend YOLO's capability to detect objects across multiple frames, minor variations in scene or lighting conditions between frames continue to pose significant challenges for accurate detection [Zheng et al., 2024]. Consequently, when applied in real-world operational environments, there is a need for a solution to enhance YOLO's defect detection capabilities in consecutive frame sequences, especially under continuously varying lighting conditions.

1.3 Difficulty distinguishing genuine defects from transient artifacts using only single-frame confidence scores

Relying solely on single-frame confidence scores to determine defects can lead to significant inaccuracies. A high confidence score may not necessarily indicate a genuine defect but could result from temporary phenomena such as lighting reflections, surface stains, or image noise. Conversely, a low confidence score in a single frame may cause the system to overlook obvious defects appearing consistently across adjacent frames. This issue is particularly critical in applications such as security monitoring or quality inspection, where accuracy in defect detection is paramount.

Recent studies have shown that integrating information across multiple sequential frames significantly enhances object detection and tracking accuracy. For instance, Sun et al. (2022) proposed a method leveraging temporal consistency in video object detection, reducing computational costs and improving the performance of single-stage detection models. This approach utilizes position and size prediction networks to filter background regions and avoid unnecessary computations on low-level feature maps for specific frames.

Additionally, vision-language models have been explored to enhance object detection capabilities in real-world environments. Du et al. (2022) proposed a method involving prompt representation learning for open-vocabulary object detection, enabling systems to detect previously undefined objects by continuously learning prompt representations from positive and negative proposals relative to actual labels in images. Furthermore, exploiting temporal consistency has proven crucial in improving object detection performance. Li et al. (2023) presented an online video object segmentation method leveraging temporal consistency to minimize prediction inconsistencies and enhance video object segmentation performance.

These approaches indicate that incorporating temporal information and employing hybrid vision-language models are necessary for enhancing accuracy and reliability in defect detection, especially in continuous operation systems with stringent quality requirements.

2 Related works

2.1 Yolo-based object detection in industrial inspection

The YOLO (You Only Look Once) family, including YOLOv3 [Redmon et al., 2018] and YOLOv5, has become a popular standard for real-time object detection tasks, particularly in industrial applications such as printed circuit board (PCB) inspection. These models achieve high processing efficiency by analyzing each frame independently. However, they exhibit limitations in maintaining temporal consistency and contextual reasoning. For instance, patent US20170147905A1 describes an end-to-end object detection process but does not ensure frame-to-frame stability, leaving the system vulnerable to transient noise or motion blur. Patent CN107527009B similarly employs YOLO for detecting residual objects in surveillance environments but fails to address temporal stability issues. Consequently, many practical implementations rely on post-processing steps such as manual filtering or human verification to mitigate false positives and missed detections.

2.2 Vision-language integration for semantic inspection

Recent advancements in multi-modal large language models, such as CLIP [Radford et al., 2021] and Flamingo [Alayrac et al., 2022], have enabled the generation of natural language descriptions that accurately reflect visual anomalies. These descriptions significantly support inspection engineers in defect classification and prioritization [Zhang et al., 2023]. Patent US12205356B2 proposes semantic description methods for identifying and distinguishing various defect types, particularly for tracking structural cracks in bridges, roads, and buildings. However, current systems primarily apply language models to individual image patches without comprehensive integration into the overall defect detection or verification pipeline. Additionally, confidence scores generated by vision-language models often remain uncalibrated and are not aggregated across multiple detections, limiting effectiveness in automated defect classification.

2.3 Multi-agent reasoning architectures

Agent-oriented frameworks such as LangChain Agents [Chase et al., 2023] and Auto-GPT [Richter et al., 2023] have enabled the coordination of multiple AI components like perception, planning, and verification within unified systems. Initially designed for general language tasks such as chatbots and virtual assistants, these frameworks are increasingly being adapted for industrial applications. Patent US11237933B2 presents a multi-agent plan recognition method utilizing classical symbolic AI, while patent US20200184383A1 introduces multi-agent reinforcement learning for classifying user intents. Nonetheless, current research and applications lack tight integration between object detection agents, defect-description language agents, and reasoning agents within a unified real-time inspection loop.

2.4 Temporal aggregation and multi-frame analysis

Several methods have been proposed to enhance defect detection stability through temporal modeling techniques, including Kalman filters, object tracking algorithms, and frame-result aggregation models [Wojke et al., 2017]. These methods help smooth detection results across frames but typically lack semantic understanding capabilities. Patent US11847775B2 utilizes historical data for vision-based defect detection, while patent US11987236B2 employs temporal aggregation for 3D object localization from single images. However, few contemporary systems effectively combine temporal stability with semantic reasoning to reliably distinguish genuine defects from transient artifacts such as reflections or motion blur.

3 Description of the 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

FIG. 16 illustrates the progress of improvement in speed and accuracy

FIG. 17 illustrates the progress of improvement in accuracy

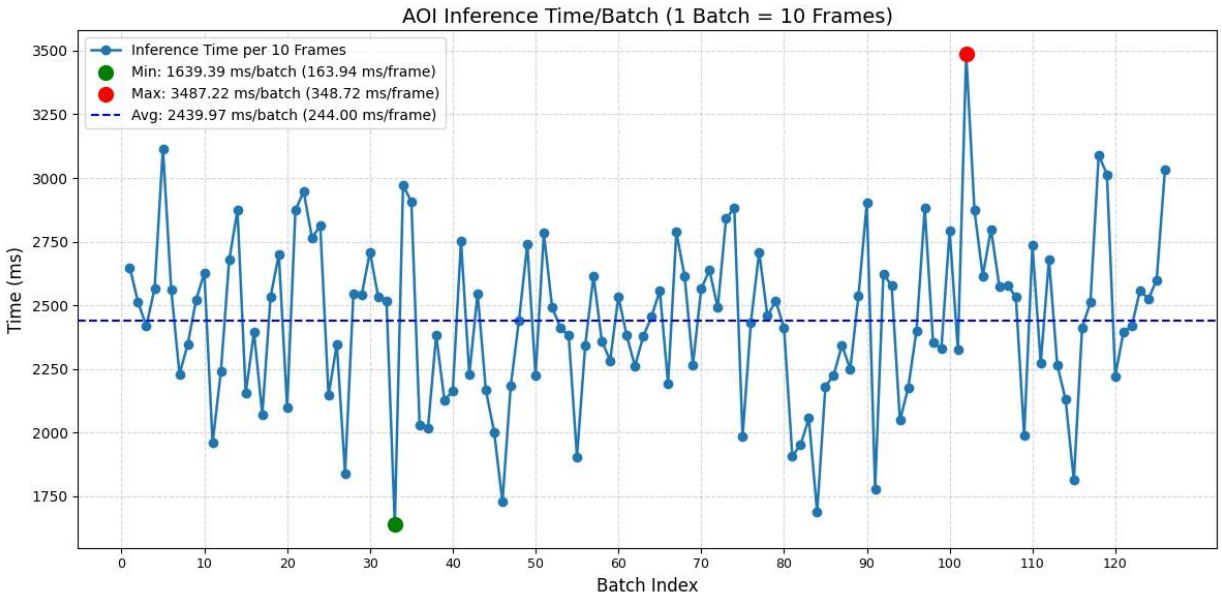

FIG. 15 illustrates inference time per batch, with GPU -RTX4090, model core use is llama-3.2-11b-vision-instruct and yolov11-nano

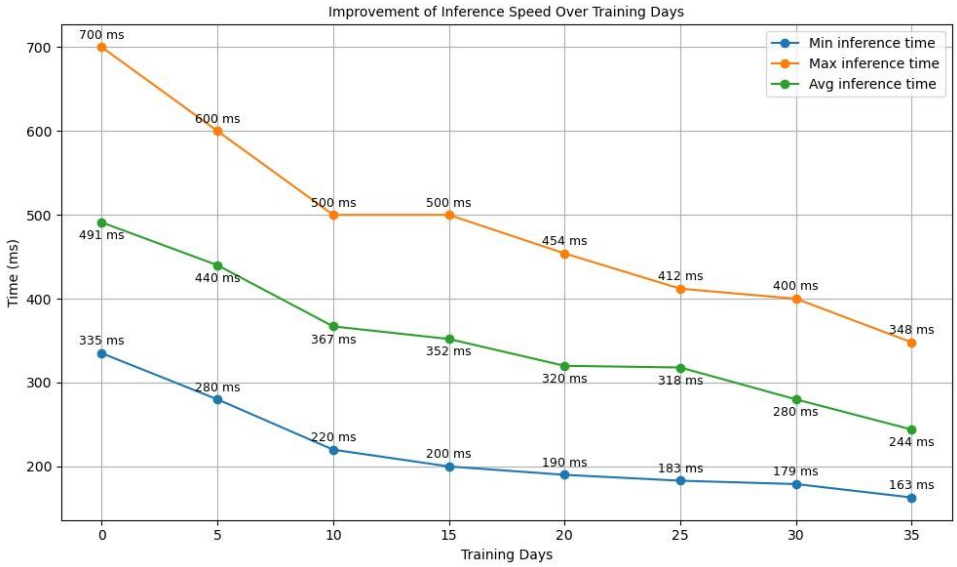

FIG. 16 illustrates the progress of improvement in speed

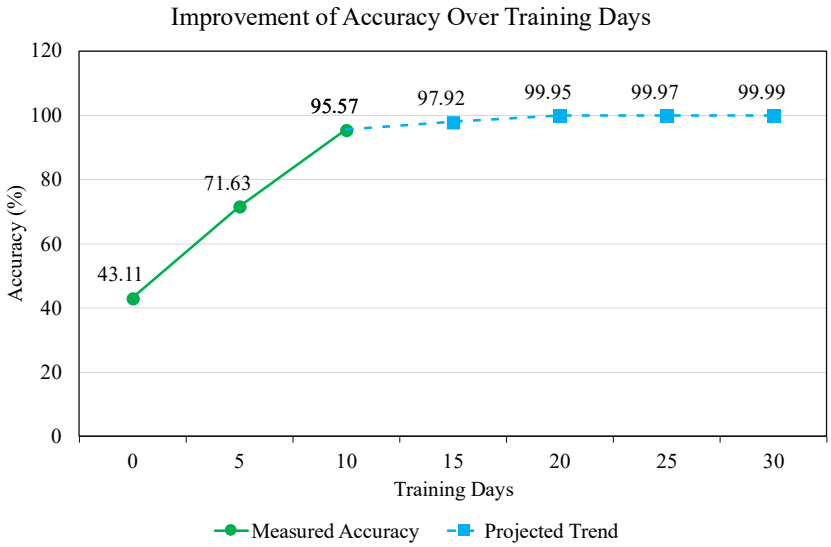

FIG. 17 illustrates the progress of improvement in accuracy

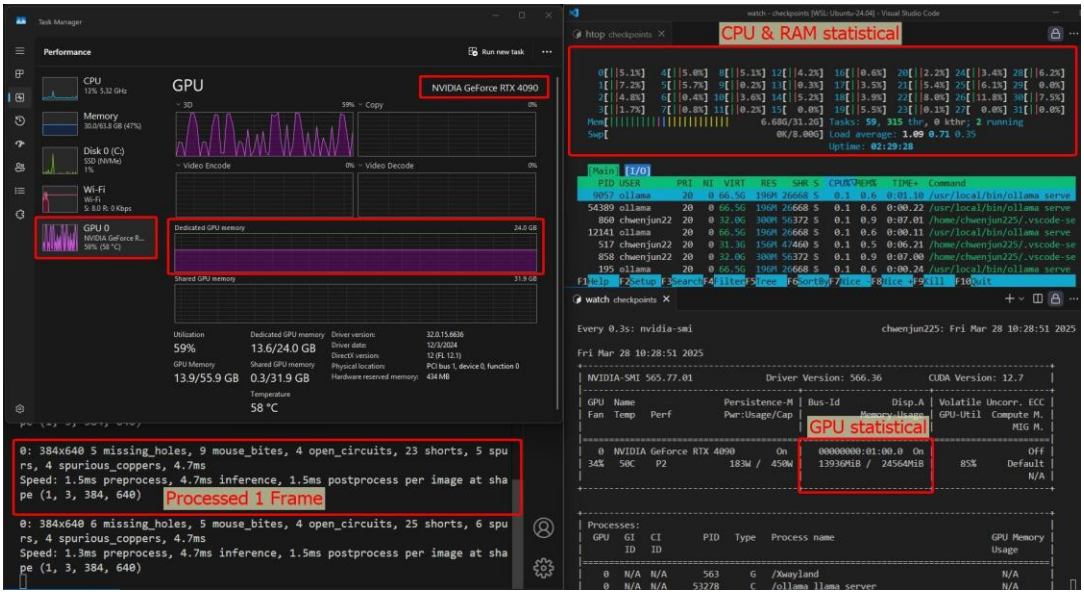

FIG. 14 illustrates hardware resource usage, in this embodiment is demo system with GPU-RTX4090, model core use is llama-3.2-11b-vision-instruct and yolov11-nano

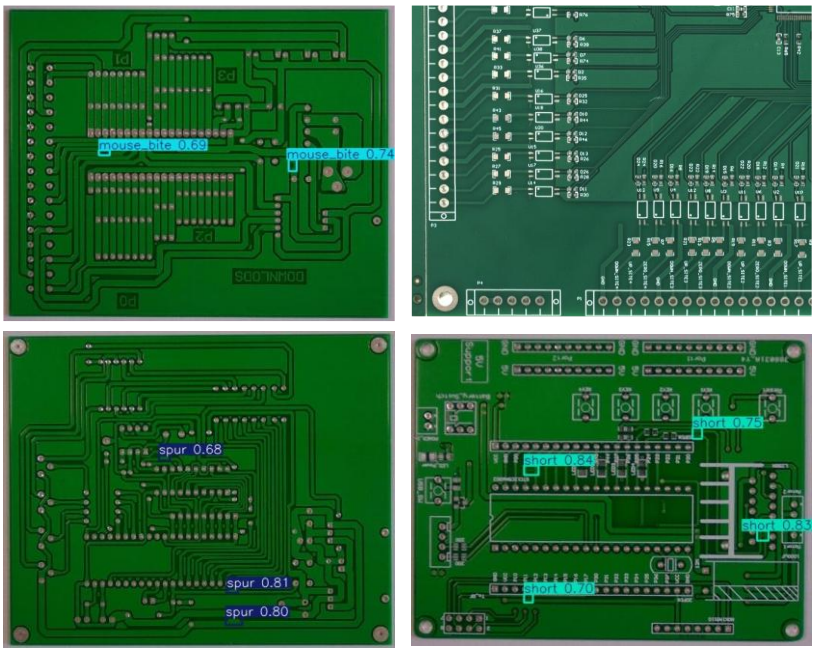

FIG. 12 illustrates the different layout printed circuits board

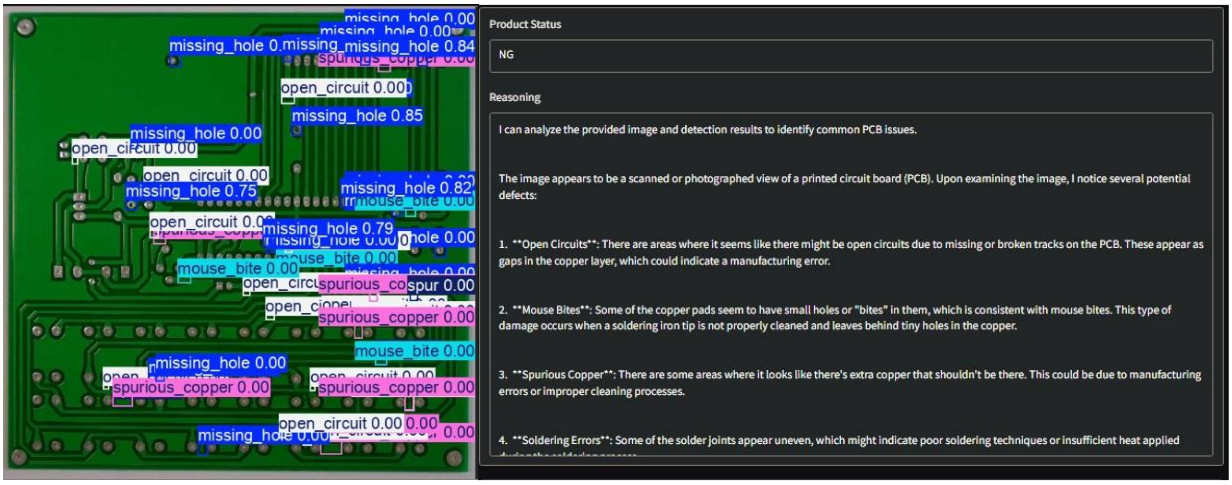

FIG. 10 illustration of automated PCB defect detection with visual description

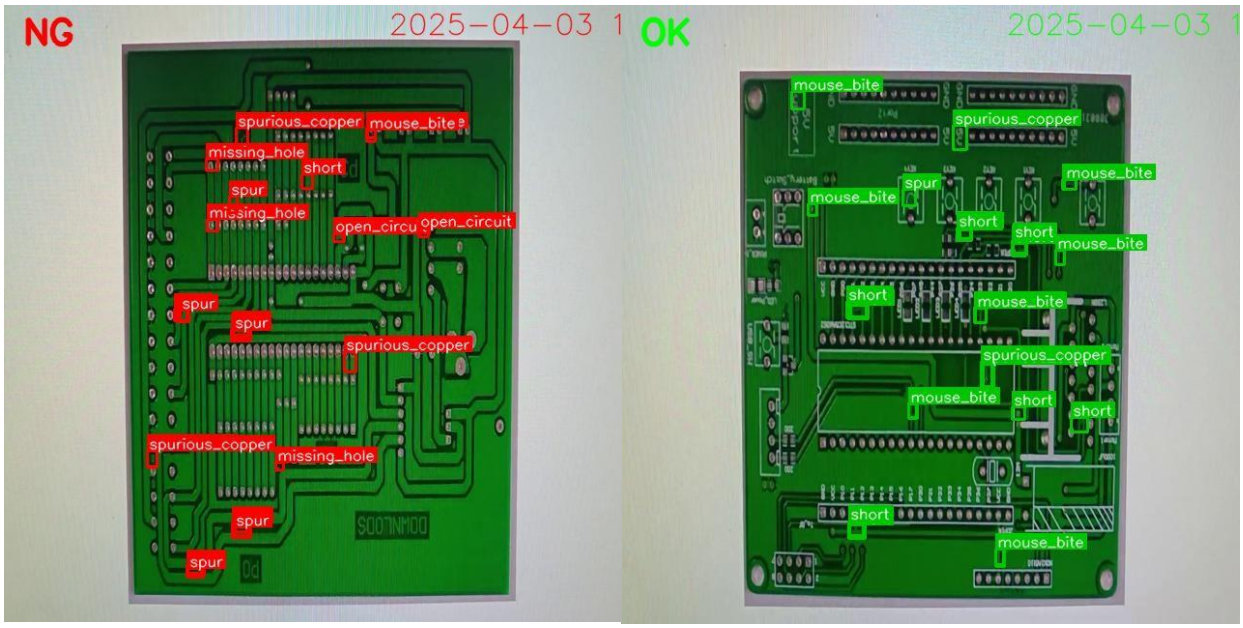

FIG. 11 illustrates the defect detection on different products, the results show NG/OK samples after the reasoning process with multi-agent system

4 Detailed description of the invention

4.1 System overview

The present invention proposes an automatic visual defect inspection system based on computer vision, wherein deep learning models and artificial intelligence (AI) agents are organized into a chain-of-agents architecture to collaboratively perform three primary functions: defect detection, defect description, and defect verification, with input comprising a batch of n frames. The invention aims to significantly reduce false positives and false negatives in industrial product visual inspections by simultaneously utilizing textual descriptions from a vision-language model and contextual text across frames.

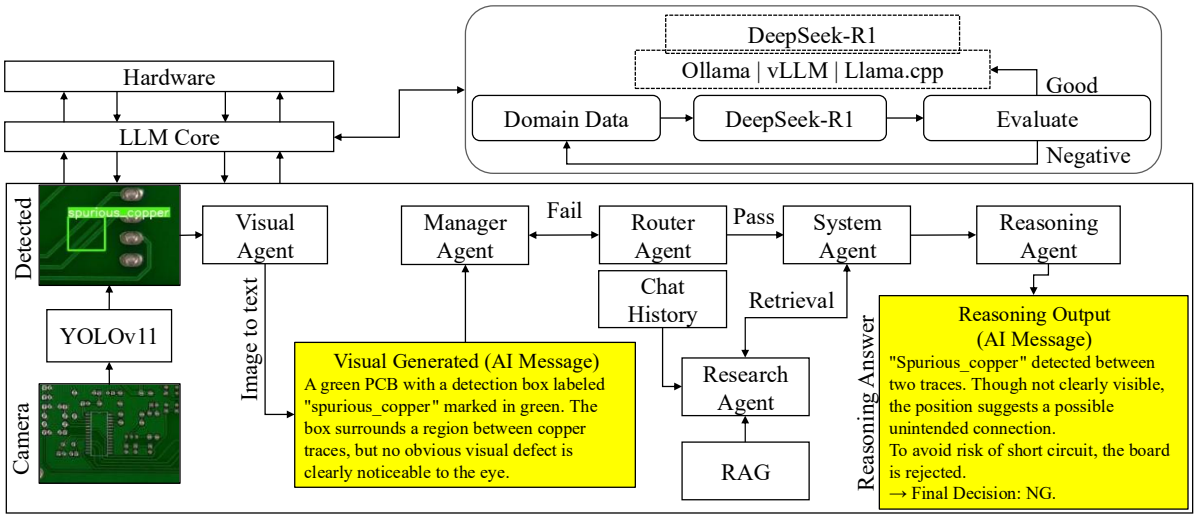

FIG.2 illustrates the overall structure of the defect detection and classification system based on multi-agent reasoning

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 $BB_i = \{bb_{(i,1)},\ bb_{(i,2)},\ \ldots,\ bb_{(i,j)}\}\ (eq2)$; 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 ($0 \le c \le 1$); l is the detected object label;

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 $p_{(i,j)} = crop(f_i,\ bb_{(i,j)},\ s)\ (eq4)$; $P_i = \{p_{(i,1)},\ p_{(i,2)}, \ldots, p_{(i,j)}\}\ (eq5)$; 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 $P_i = \{(p_{(i,1)}, bb_{(i,1)}), (p_{(i,2)}, bb_{(i,2)}), \dots, (p_{(i,j)}, bb_{(i,j)})\}$ $(eq6)$; 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 $d_{(i,j)} = VLM(p_{(i,j)})(eq7)$, 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 $P_i = \{(p_{(i,1)}, bb_{(i,1)}, d_{(i,1)}), \dots, (p_{(i,j)}, bb_{(i,j)}, d_{(i,j)})\}$ $(eq8)$;

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 $FramesState = \{P_i | i = B - n + 1, \dots, B\}$ $(eq9)$. 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;
   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.

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;

This feedback mechanism ensures real-time defect isolation, supports immediate operator awareness, and maintains uninterrupted workflow continuity for non-defective products.

4.2 Object Defect Detection Pipeline

According to an aspect of the invention, the system includes a pipeline for detecting suspected surface defect regions in input frames. This pipeline utilizes a pre-trained deep learning object detection model, typically YOLO (You Only Look Once), to detect these regions within each frame. The process comprises the following steps

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

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

Where

f_i is denotes the frame at index i

n is 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

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

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

(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:

$$p_{(i,j)} = crop(f_i, bb_{(i,j)}, s) \qquad (eq4)$$
$$P_i = \{p_{(i,1)}, p_{(i,2)}, \dots, p_{(i,j)}\} \qquad (eq5)$$

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) Assigning bounding box coordinates to image patches, each image 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

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

Where

P_i is 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.

4.3 Vision-Language Semantic Description

In another aspect of the invention, the system integrates an image-to-text processing module utilizing a multimodal large-language model, also known as a vision-language model (VLM), to generate natural language descriptions for each image patch. This module serves as a semantic translator, converting visual information from image patches into textual descriptions comprehensible to the reasoning agents within the system.

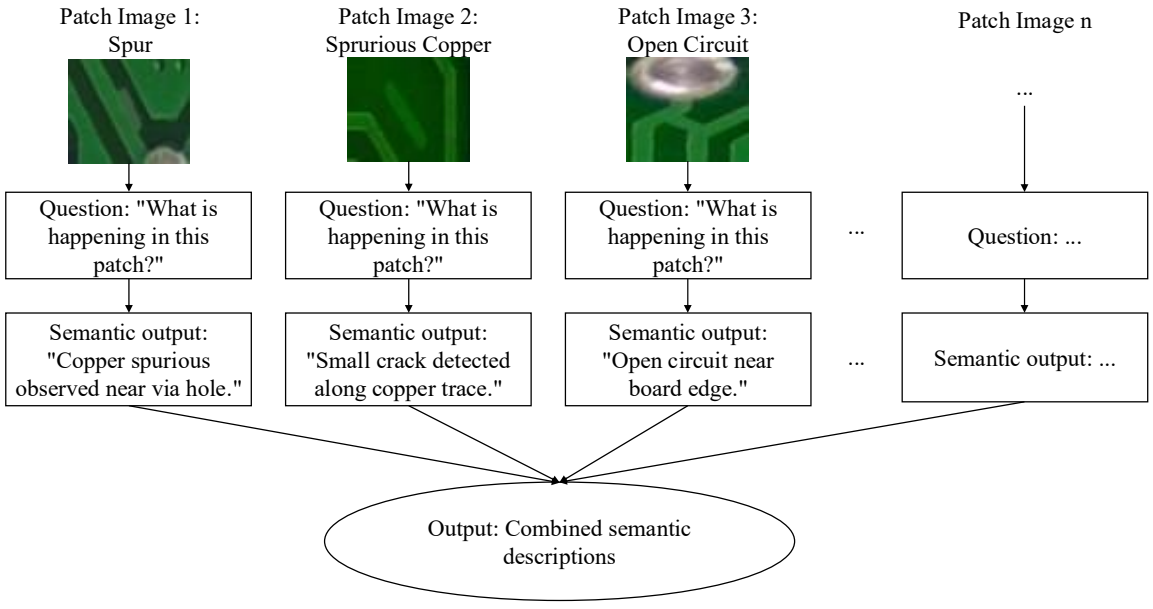

FIG.3 illustrates modeling defect structures as textual descriptions using a vision-language model

This process comprises the following steps

(1) The primary technical goal of the module is to semantically encode visual defects from image data into natural language descriptions. These semantic descriptions enable subsequent reasoning agents to understand the nature of defects, thereby bridging raw image signals and downstream contextual analysis logic. Integration of the vision-language model: In various embodiments, this module may integrate any vision-language model capable of automatically generating textual descriptions from input images. Suitable model providers include Meta, OpenAI, and HuggingFace. In a preferred embodiment aimed at optimal hardware and module compatibility, the invention utilizes the Llama-3.2-11b-vision-instruct model provided by MetaAI, deployed either in a server or edge computing environment, depending on computational requirements and latency constraints.

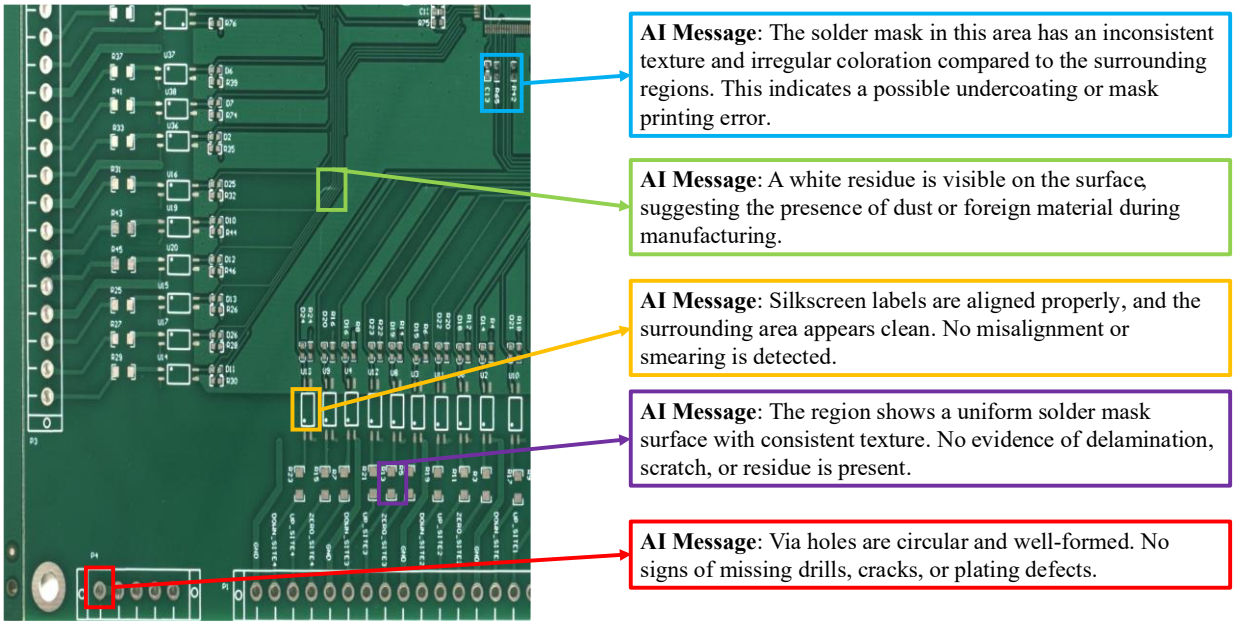

FIG.4 illustrates an example of semantic textual descriptions of image patches within a single frame

(2) For each image patch $p_{(i,j)} \in P_i$ obtained from equations (eq4, eq5, eq6) described in section 4.2, the vision-language-model executes three primary steps

1. Receives the image patch and corresponding bounding box metadata $(p_{(i,j)}, bb_{(i,j)}) \in P_i$ (eq6)
2. Extracts the image patch $p_{(i,j)}$
3. Generates a semantic textual description $d_{(i,j)}$ for the given image patch $p_{(i,j)}$

This process is mathematically represented as:

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

Where

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

(3) The output of equation (eq6), the textual description $d_{(i,j)}$, is stored alongside the image patch and bounding-box metadata within P_i as follows

$$P_i = \{(p_{(i,1)}, bb_{(i,1)}, d_{(i,1)}), (p_{(i,2)}, bb_{(i,2)}, d_{(i,2)}), \dots, (p_{(i,j)}, bb_{(i,j)}, d_{(i,j)})\} \qquad (eq8)$$

Where

P_i represents metadata of the frame at index i-th

$p_{(i,j)}$ denotes the image patch at frame i-th, index j-th

$bb_{(i,j)}$ is the bounding-box containing metadata of the detected object

$d_{(i,j)}$ is the semantic textual description of image patch $p_{(i,j)}$

(4) After generating semantic descriptions for the patches in frame i, the element P_i is stored in a temporary sliding-window memory structure called FramesState. This sliding window maintains a fixed size equal to the batch frame length n. When the number of elements in FramesState exceeds n, the oldest elements P_i is removed from the current window and moved into a permanent storage area known as ConsistentMemory. This persistent storage preserves processed data for subsequent verification tasks or extended analysis.

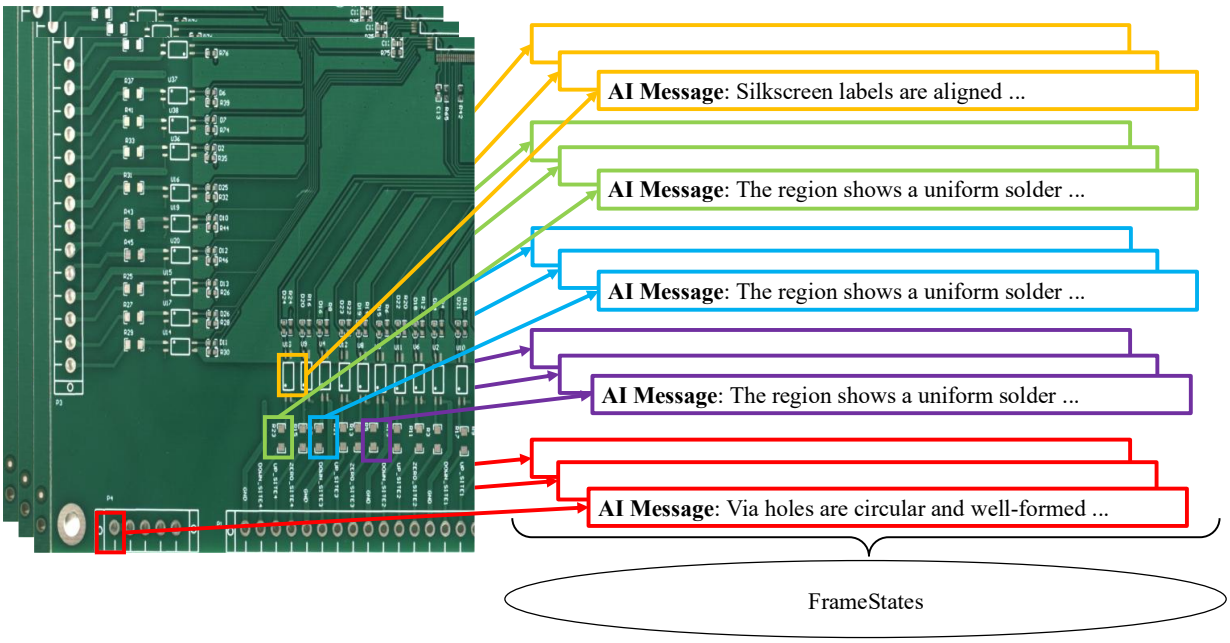

FIG.5 illustrates an example of aggregating semantic textual data for multiple patches across multiple frames

The sliding-window mechanism maintains real-time stability, prevents memory overload, and ensures retention of sufficient contextual information required for reasoning steps

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

Where

P_i is metadata of the frame at index i-th

B represents a batch of nnn frames as previously defined in (eq1)

n denotes the total number of frames in one batch

(5) When the size of the sliding window exceeds the batch frame length n

$$|FrameStates| > n$$

the oldest element P_{i-n} is removed from FramesState and transferred to ConsistentMemory. This mechanism is represented by

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

Where

$ConsistentMemory$ is a long-term, stable storage repository (such as a database)

n represents the maximum number of frames retained in the sliding window (batch size)

In certain implementations, ConsistentMemory may be realized through a vector database system such as PGVector, ChromaDB, or equivalent, facilitating semantic querying or contextual searches.

(6) Employing the VLM for semantic description provides several technical advantages, enhances integration with logical reasoning systems by representing defects semantically rather than as pure pixel-based features; supplies input for AI agents within the system, enabling context matching, temporal analysis, or knowledge querying; improves explainability due to clear textual descriptions, helping human supervisors understand classification decisions; facilitates flexible handling of uncertain regions by utilizing confidence score thresholds.

4.4. Retrieval-Augmented Generation

According to an embodiment of the invention, the system integrates a Retrieval-Augmented Generation (RAG) mechanism to provide additional contextual support during the inference process. In this step, the Research Agent initiates the RAG mechanism by combining the current query with relevant contextual information retrieved from a vectorized database.

This database stores semantic representations derived from quality assurance (QA) documents, which have been processed and input into the system by human operators. Information retrieved from this database is subsequently provided as contextual input to a large language model (LLM), enabling the generation of inference-enhanced textual responses.

This process is mathematically modeled as follows

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

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

Where

$QADB$ is a vectorized database containing semantic embeddings extracted from human-curated quality assurance documents.

$Retrieve$(FrameStates, QADB) is a retrieval function that extracts relevant context from QADB based on the content of FrameStates.

$LLM(FrameStates, C)$ is a large language model receiving FrameStates as the query and C as retrieved context, producing an augmented inference response.

$RAG(FrameStates$ is the resulting textual output generated by the Retrieval-Augmented Generation mechanism, comprising both original query information and supplementary context.

4.5. Description of multi-agent inference mechanism

Another important aspect of the invention is structuring the defect inference process as a set of Artificial Intelligence (AI) agents, each assigned a specific functional role within the information processing pipeline. This design enhances scalability, supports parallel processing, and optimizes real-time performance. The system employs a cooperative multi-agent architecture, where individual components interact through a shared memory structure (FramesState).

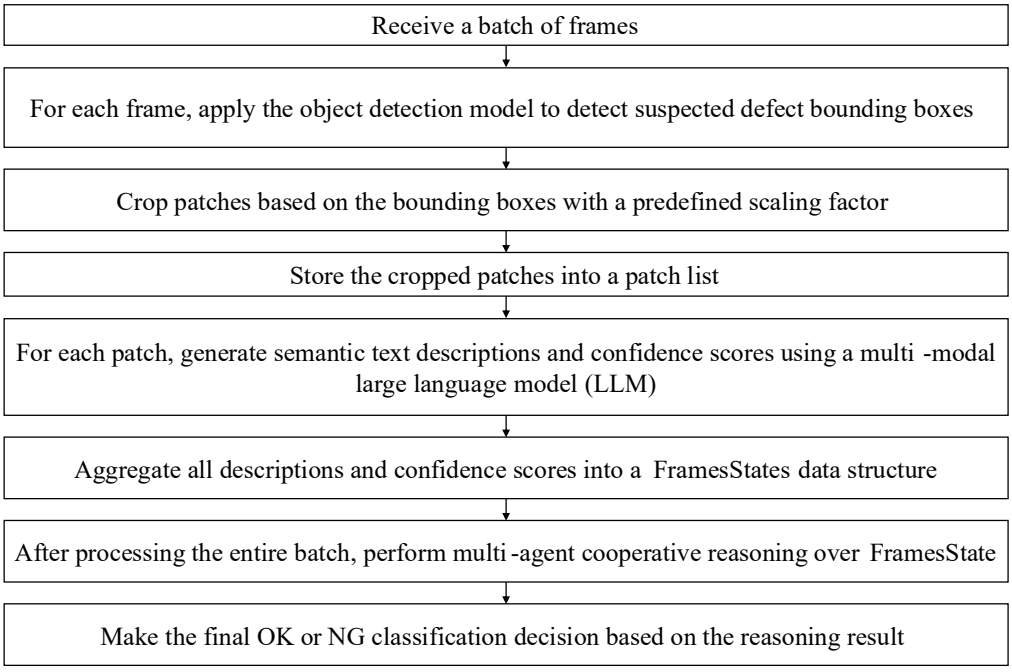

FIG. 6 illustrates the schematic flowchart of defect detection across an image sequence using batch aggregation

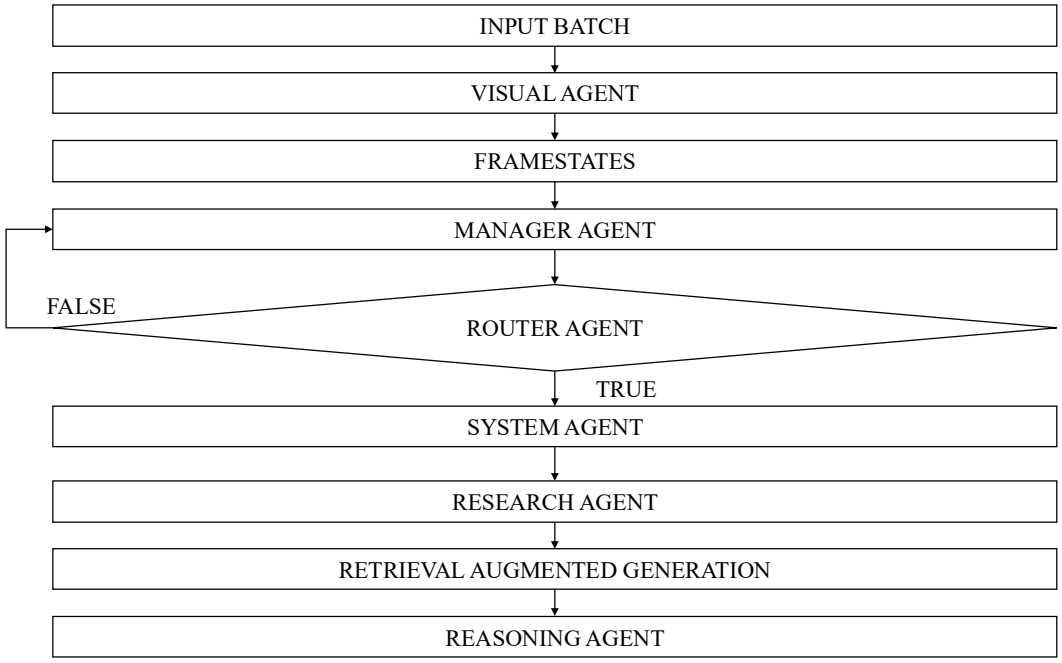

FIG. 7 illustrates the system inference coordination workflow

The system inference coordination workflow comprises following steps

(1) Visual Agent receives data units $P_i \in FramesState[i]$ from the semantic description module (section 4.3). This agent transforms image patches and accompanying information into descriptive natural-language messages.
(2) Manager Agent evaluates the validity and clarity of messages received from the Visual Agent. 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) Router Agent routes information to other agents based on logical conditions, specifically
    - 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.
    - 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.
(4) 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.
(5) System Agent aggregates information received from the Visual and Research Agents to construct FramesStates, facilitating the final inference step.
(6) 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.

4.6. Hardware Configuration for Visual Inspection

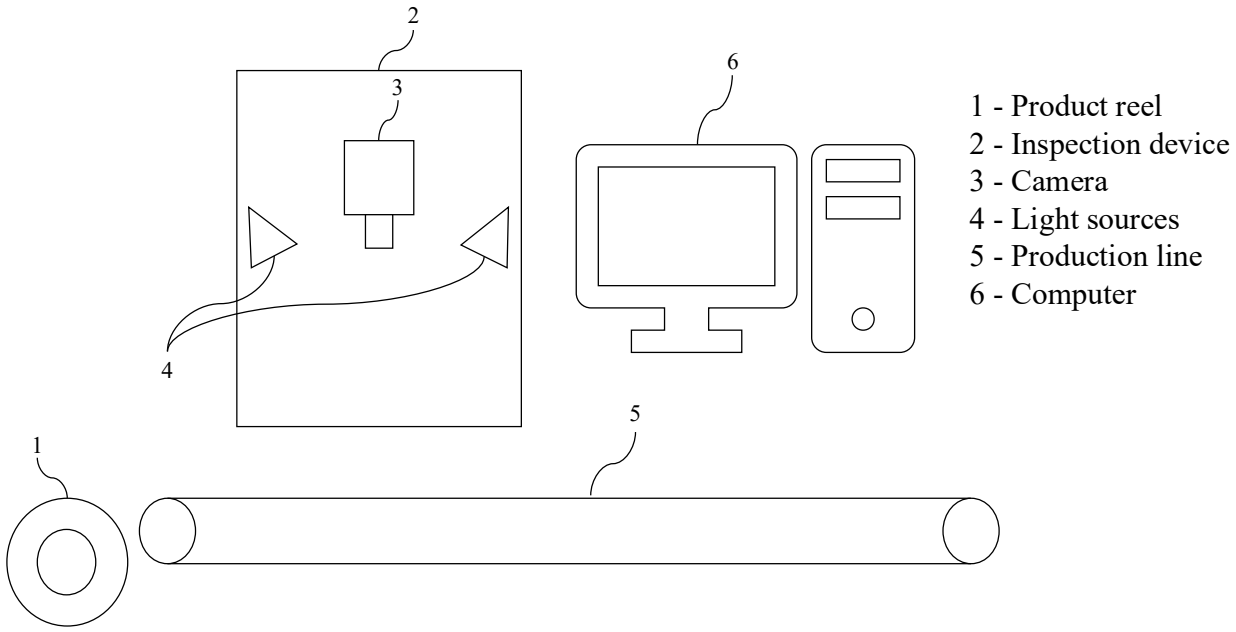

FIG. 8 illustrates the hardware module device used for product inspection

An essential aspect of the invention lies in the hardware configuration designed to support automated visual defect inspection. This configuration ensures accurate, stable, and synchronized image acquisition aligned with the software pipeline described in Sections 4.1 through 4.5. The hardware system is intended for continuous operation in high-speed manufacturing environments and offers flexible integration with existing production lines.

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.

During operational flow, the product is unwound from the reel (1) and transported via the production line (5) through the inspection zone. The camera (3), synchronized with the lighting system (4), captures surface images of the moving product within the inspection device (2). The acquired images are then transmitted to the computer (6), which executes the complete software pipeline—including object detection models, vision-language semantic description, temporal data aggregation, and final decision-making via multi-agent reasoning architecture.

4.7. Fine object detection model

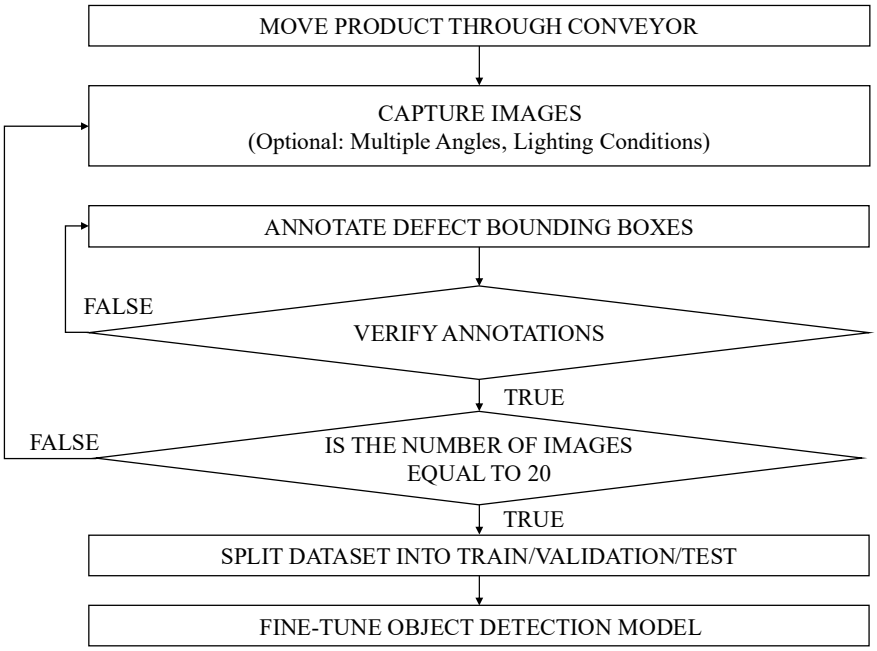

FIG. 9 illustrates the data collection process and the fine-tuning procedure for the object detection model

According to an embodiment of the invention, fine-tuning the object detection module does not require collecting or labeling a large-scale dataset. Instead, a small dataset containing representative examples of the target visual defects is sufficient for adapting the parameters of a pre-trained 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.

This approach of using a small-scale fine-tuning dataset yields several technical advantages, (1) labeling efficiency, significantly reduces annotation workload and associated data collection costs; (2) training efficiency, it help minimizes training time and computational resources due to the reduced number of epochs and smaller dataset size; (3) system stability, ensures robustness in production environments by requiring only initial-level detection accuracy,

which is later enhanced through contextual inference and memory-based correction via the FramesState and reasoning agents.

References

Academic Research Publications

Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. arXiv preprint arXiv:1804.02767. https://arxiv.org/abs/1804.02767

Radford, A., Kim, J. W., Hallacy, C., et al. (2021). Learning transferable visual models from natural language supervision. arXiv preprint arXiv:2103.00020. https://arxiv.org/abs/2103.00020

Alayrac, J. B., Donahue, J., Luc, P., et al. (2022). Flamingo: a visual language model for few-shot learning. arXiv preprint arXiv:2204.14198. https://arxiv.org/abs/2204.14198

Zhang, Q., Huang, Y., & Wu, Z. (2023). Multimodal semantic annotation for defect analysis in PCB inspection systems. IEEE Transactions on Industrial Informatics, 19(1), 123–133.

Chase, H., Singh, A., & Wang, Y. (2023). LangChain: Composable language agents for practical reasoning. arXiv preprint arXiv:2304.05333.

Richter, F., Meissner, L., & Götz, T. (2023). Auto-GPT: Autonomous language agents in real-world planning tasks. AI Systems Journal, 5(3), 211–225.

Wojke, N., Bewley, A., & Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. In 2017 IEEE International Conference on Image Processing (ICIP), 3645–3649.

Siemens AG. (2021). SIMATIC Machine Vision Systems: Smart inspection with flexible configuration. Siemens Whitepaper. https://new.siemens.com/global/en/products/automation/industry-software/simatic-machine-vision.html

Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). "YOLOv4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*. Available: https://arxiv.org/abs/2004.10934.

Chen, J., Jiang, Y., Lu, J., & Zhang, L. (2024). S-Agents: Self-organizing Agents in Open-ended Environments. *ArXiv*. https://arxiv.org/abs/2402.04578.

Tung, C., Kelleher, M. R., Schlueter, R. J., Xu, B., Lu, Y., Thiruvathukal, G. K., Chen, Y., & Lu, Y. (2018). Large-Scale Object Detection of Images from Network Cameras in Variable Ambient Lighting Conditions. *ArXiv*. https://arxiv.org/abs/1812.11901

Zheng, Y., Jing, Y., Zhao, J., & Cui, G. (2024). LAM-YOLO: Drones-based Small Object Detection on Lighting-Occlusion Attention Mechanism YOLO. *ArXiv*. https://arxiv.org/abs/2411.00485

Dilmaghani, M. S., Shariff, W., Ryan, C., Lemley, J., & Corcoran, P. (2024). Autobiasing Event Cameras. *ArXiv*. https://arxiv.org/abs/2411.00729

Sun, G., et al. (2022). *Efficient One-stage Video Object Detection by Exploiting Temporal Consistency*. ECCV 2022. ACM Digital Library+4GitHub+4European Computer Vision Association+4

Du, Y., et al. (2022). *Learning to Prompt for Open-Vocabulary Object Detection with Vision-Language Model*. CVPR 2022. CVF Open Access+1arXiv+1

Li, Y., et al. (2023). *TCOVIS: Temporally Consistent Online Video Instance Segmentation*. ICCV 2023. CVF Open Access

Pham, T.-N., Nguyen, V.-H., Kwon, K.-R., Kim, J.-H., & Huh, J.-H. (2024). Improved YOLOv5 based deep learning system for jellyfish detection. *IEEE Access, 12*, 64925–64937. https://doi.org/10.1109/ACCESS.2024.3405452

Yang, D., Simoulin, A., Qian, X., Liu, X., Cao, Y., Teng, Z., & Yang, G. (2025). DocAgent: A Multi-Agent System for Automated Code Documentation Generation. *ArXiv*. https://arxiv.org/abs/2504.08725

Patent Documents

US20170147905A1 – Systems and methods for end-to-end object detection. https://patents.google.com/patent/US20170147905A1/en

CN107527009B – Remnant object detection method based on YOLO target detection. https://patents.google.com/patent/CN107527009B/en

US12205356B2 – Semantic image capture fault detection. https://patents.google.com/patent/US12205356B2/en

US11237933B2 – Multi-agent plan recognition. https://patents.google.com/patent/US11237933B2/en

US20200184383A1 – User intent classification using multi-agent reinforcement learning. https://patents.google.com/patent/US20200184383A1/en

US11847775B2 – Automated machine vision-based defect detection. https://patents.google.com/patent/US11847775B2/en

US11987236B2 – Monocular 3D object localization from temporal aggregation. https://patents.google.com/patent/US11987236B2/en

US10957031B2 – Intelligent defect detection from image data. https://patents.google.com/patent/US10957031B2/en