Supporting Information

Image-Based Food Quality Analysis Framework Driven by Large Language Models

Xiao-Yue Yin, Hai-Long Wu, Luo-Yuan Han, Ye He, Xiao-Zhi Wang*, Tong Wang*, Ru-Qin Yu

State Key Laboratory of Chemo and Biosensing, College of Chemistry and Chemical Engineering, Hunan University, Changsha 410082, P. R. China.

* Corresponding author 1: Tong Wang
State Key Laboratory of Chemo and Biosensing, College of Chemistry and Chemical Engineering, Hunan University, Changsha 410082, P. R. China
E-mail: wangtong@hnu.edu.cn

* Corresponding author 2: Xiao-Zhi Wang
State Key Laboratory of Chemo and Biosensing, College of Chemistry and Chemical Engineering, Hunan University, Changsha 410082, P. R. China
E-mail: wangxiaozhi@hnu.edu.cn

Contents

Section S1. General Information

S1.1 Large Language Models (LLMs)

This study involves three representative LLMs: GPT-5, GPT-4o, and DeepSeek-R1. GPT-5 and GPT-4o are developed and maintained by OpenAI, and represent state-of-the-art autoregressive transformer-based architectures capable of understanding and generating human-like natural language. These models are trained on extensive multimodal datasets and optimized through reinforcement learning with human feedback, enabling them to perform complex reasoning, contextual understanding, and domain-specific knowledge transfer. DeepSeek-R1, developed by DeepSeek, is the company's first-generation reasoning model designed to enhance structured problem-solving and logical inference. By combining vast pre-training with fine-tuning on reasoning-intensive tasks, DeepSeek-R1 demonstrates competitive performance in numerical reasoning, symbolic manipulation, and multi-step deduction.

S1.2 Application Programming Interface (API)

To interact with LLMs efficiently, this study utilizes their APIs, which provide standardized endpoints for model invocation. Through an API call, structured input data (e.g., pre-processed food image features and contextual prompts) are transmitted to the model, and the corresponding outputs (e.g., predicted labels) are returned in real time.

S1.3 Machine Learning Methods

■ *k*-nearest neighbor (*k*-NN)

k-NN is a similarity-based method, which core idea follows the principle that "similar instances are grouped together"[1]. By calculating the distance (e.g., Euclidean distance) between the unknown sample and each training sample, the closer the distance to the unknown sample is, the higher the weight of the sample. Then, the class label assigned to the unknown sample is determined based on the majority class among these *k* nearest neighbors. Although *k*-NN is a simple and intuitive algorithm that does not require an explicit training phase, it may be computationally intensive during prediction and is highly sensitive to outliers. Therefore, selecting an appropriate k value and distance metric is critical to optimizing the performance of a *k*-NN model.

■ Random forest (RF)

RF is an ensemble learning algorithm based on the idea of bagging[2]. It trains and predicts samples by constructing multiple decision trees (DTs). Each tree randomly selects samples and

features during the training process, and finally establishes a strong classification or regression model suitable for different data sets by voting or averaging the prediction results of each decision tree. In the modeling process of RF, randomness is reflected in two key aspects: random sampling and random feature selection. By performing bootstrap sampling on the training data and randomly selecting feature subsets at each split, the diversity between RF-based learners can be improved, the risk of overfitting of a single decision tree can be reduced, and the generalization performance of the model can be enhanced.

- Partial least squares discriminant analysis (PLS-DA)

PLS-DA is a supervised classification method based on Partial Least Squares Regression (PLSR)[3]. It projects high-dimensional and collinear variables into a low-dimensional latent space while maximizing the covariance between predictor variables and categorical response variables. A key advantage of PLS-DA is its ability to focus and enhance between-group separation. Furthermore, PLS-DA combines dimensionality reduction and discriminant analysis in a single algorithm, making it particularly suitable for modeling high-dimensional (HD) data. In classification tasks, PLS-DA converts class labels into dummy variables and constructs latent variables that best distinguish between different classes. The interpretability of these latent variables also helps in understanding which features contribute most to class differentiation, leading to the wide application of PLS-DA in numerous fields such as medical diagnostics[4], food analysis[5], and metabolomics[6].

S1.4 Deep Learning Methods

- ResNet-18

The ResNet family of models was initially proposed by He et al. and is an architecture based on CNN. Its core concept is residual connections[7]. As a member of the ResNet family, ResNet-18 is the smallest and lightest model. ResNet-18 consists of 17 convolutional layers, one 3×3 max-pooling layer, and one fully connected layer. A classic ResNet-18 model contains 33.16 million parameters, with ReLU activation and batch normalization (BN) applied to the "building blocks" of all convolutional layers. The residual building blocks are the fundamental structure of the ResNet-18 network. Instead of forcing each layer to learn a completely new representation, it allows each layer to learn a residual between the input and the output. This is achieved through skip connections, which can bypass one or more layers, making gradients flow more easily during training. This architectural innovation makes it possible to train neural networks with hundreds or even thousands of layers.

- DenseNet-121

DenseNet-121 is a fully connected CNN that is considered a model that emphasizes the use of

convolutional neural networks, allowing them to remain effective as network depth increases by leveraging the smaller inter-layer correlations[8]. In the DenseNet-121, each layer is connected to all other layers in the grid below it. This structure facilitates feature reuse, reduces parameter redundancy, and improves gradient flow. To maintain feedforward capability, each layer receives input from deeper layers and passes feature maps to all subsequent layers. Each Kth layer has K input blocks and contains all the convolutional feature map blocks preceding it. DenseNet-121 contains different blocks, such as dense blocks and transition blocks, which differ from other pooling and convolutional layers. Details of the DenseNet-121 are as follows: 5-convolution and pooling layers, 3-transition layers (6, 12, 24), 1-Classification layer (16) and 2-denseblock (1 × 1 and 3 × 3 conv).

■ MobileNet_v2

MobileNet_v2 is a lightweight CNN architecture designed for resource-constrained environments such as mobile devices and embedded systems, and has been widely used in image recognition and computer vision tasks[9]. A key feature of this network is depthwise separable convolution, which breaks down the standard convolution operation into two steps: depthwise separable convolution and pointwise separable convolution. This reduces computational cost while efficiently extracting image features. Furthermore, MobileNetv2 introduces an inverse residual structure, enhancing the network's nonlinearity and making it more suitable for handling a wide range of image features, especially in edge cases and low-quality images. The lightweight design of MobileNetv2 makes it ideal for embedded devices and mobile applications.

■ VGG-16

VGG is a work published by the Visual Geometry Group at the University of Oxford at the ILSVRC 2014 conference[10]. This work primarily investigates the impact of convolutional neural network depth on the accuracy of large-scale image recognition. Its main contribution lies in using a very small (3×3) convolutional filter architecture to comprehensively evaluate the effects of increasing network depth. VGG-16 is a structure within the VGG network, consisting of a series of consecutive 3×3 convolutional layers, max-pooling layers, and fully connected layers. Its simple and unified architecture makes it a powerful benchmark model for image recognition tasks. Although its computational cost is higher than lightweight models, VGG-16 provides powerful performance and well-structured feature layers, and has been widely applied in medicine, biology, geography, and other fields.

S1.5 Dataset Details and Few-Shot Strategy

■ Dataset Details

This paper collects and uses three different food image datasets for three different food quality

assessment tasks. Fig. S1 shows examples of food images from each dataset. In Dataset I, for maturity classification, unripe bell peppers are green, while ripe bell peppers turn red due to the accumulation of carotenoids such as lycopene. In Dataset II, for freshness detection, fresh apples have smooth surfaces and bright colors, while rotten or damaged apples have scars and darker areas. In Dataset III, for authenticity identification, genuine black truffles exhibit a distinct marbled network pattern inside, with irregular, dense, and natural veins and clear differences in color depth. Fake truffles, on the other hand, have regular, overly smooth textures and lack natural transitions.

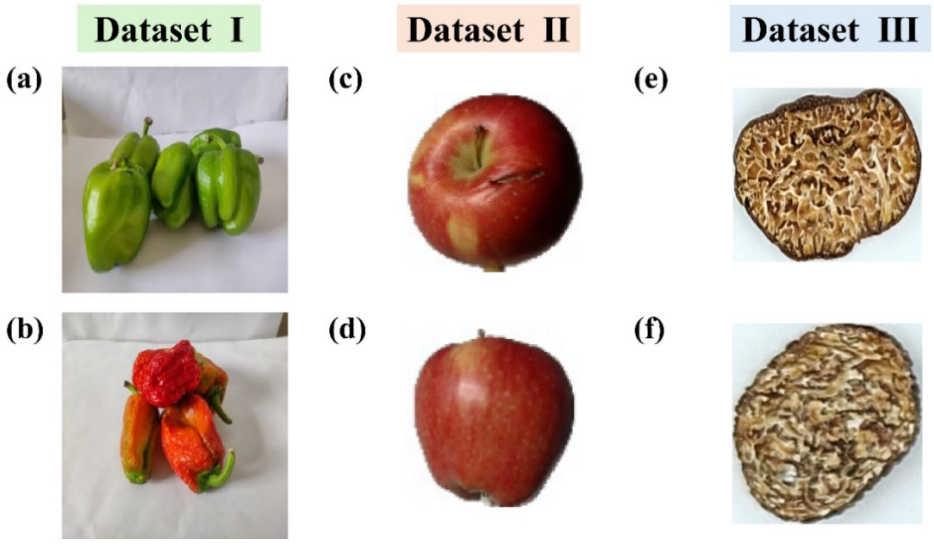

Fig. S1 Representative sample images from the three datasets. (a) unripe bell pepper, (b) ripe bell pepper, (c) rotten apple, (d) fresh apple, (e) fake Asian black truffle, and (f) real Asian black truffle.

■ Few-Shot Strategy

Due to differences in how input data is processed and the input limitations of the context window, the two LLMs modalities differ substantially in the number of training samples that can be feasibly included in a single prompt. In the text-only inference, numerical feature vectors are serialized into text tokens, resulting in minimal token consumption. Consequently, the full training data can be incorporated as in-context examples without approaching the model's context-length limits. In contrast, in the vision-language inference, each raw image must be encoded by the model's visual backbone, producing a large number of visual tokens. Excessive images within a single prompt would cause input overflow or token truncation. To ensure stable inference, we adopted a controlled few-shot strategy by randomly sampling 40 images from each training set and uniformly resizing them to 224 × 224 pixels prior to input. This configuration was not determined by the theoretical upper limits of GPT-5 or GPT-4o, but rather by empirical observations: in this work, across multiple trials, a 40-image subset consistently achieved reliable execution without token truncation while

preserving sufficient visual diversity for effective in-context learning. This design choice provides a balanced trade-off between computational feasibility and experimental comparability across datasets.

Section S2. Applying LLMs to Image-Based Food Quality Analysis

S2.1 Image Feature Processor

To reduce the entry barrier for users and ensure the reproducibility of feature processing, we developed a graphical user interface (GUI) for image feature processing. The detailed usage instructions for the "*Image Feature Processor*" GUI are shown in the Fig. S2.

(1) Upload the original image data. You can upload your divided dataset, including training set, validation set, and test set, in the corresponding part of the control panel.

(2) Extract image features. After uploading the dataset, you can click the "Extract Features" button to extract image features. Here, we use the ResNet-18 model to extract the 512-dimensional features of the images. A prompt window like the one shown in the figure appears, indicating that the image feature extraction is successful.

(3) Image feature dimensionality reduction. We use the PCA method to reduce the dimensionality of high-dimensional image data. The default dimensionality reduction is 10 dimensions. You can adjust the "PCA components" value to an appropriate value based on your project requirements. Click the "Apply PCA Dimensionality Reduction" button to implement PCA dimensionality reduction. When the following window prompts, the PCA dimensionality reduction is successful.

(4) Save File. Users can customize the feature file save path in the "Output Directory" bar.

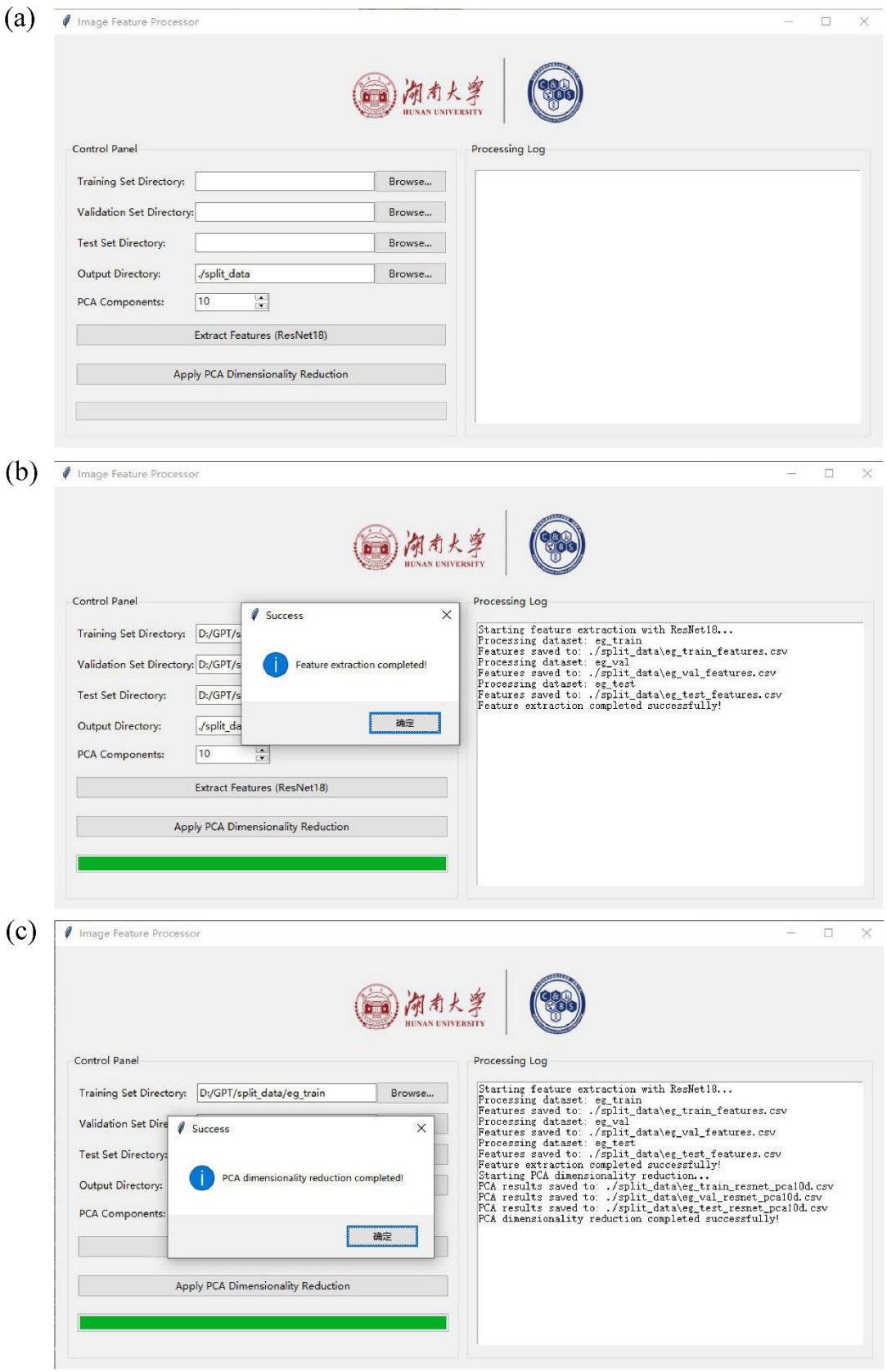

Fig. S2 (a) GUI of the integrated image feature processing tool. (b) Image feature extraction example. (c) Image feature dimensionality reduction example.

S2.2 Interaction via Chat Window

(1) **Text-only inference.** All three large language models (GPT-5, GPT-4o, and DeepSeek-R1) support direct interaction through a chat window interface, where the users can input structured instructions or prompts and receive model outputs in natural language or structured formats. The operational procedure is identical across models. Taking GPT-5 as an example, the workflow involves the following steps: (i) preparing the input prompt that contains pre-processed food image features (e.g., numerical vectors extracted by ResNet-18), (ii) embedding the contextual examples (training set samples with known labels) into the prompt, and (iii) appending the prediction request for the test samples. The model returns the predicted labels in the required structured format (e.g., binary classification labels "0" and "1").

This window-based interaction method is straightforward and does not require programming skills, making it suitable for researchers from non-computer science backgrounds. As shown in Fig. S3, we provide a practical example of food image discrimination using GPT-5 via chat interface, demonstrating the entire process from prompt construction to classification output.

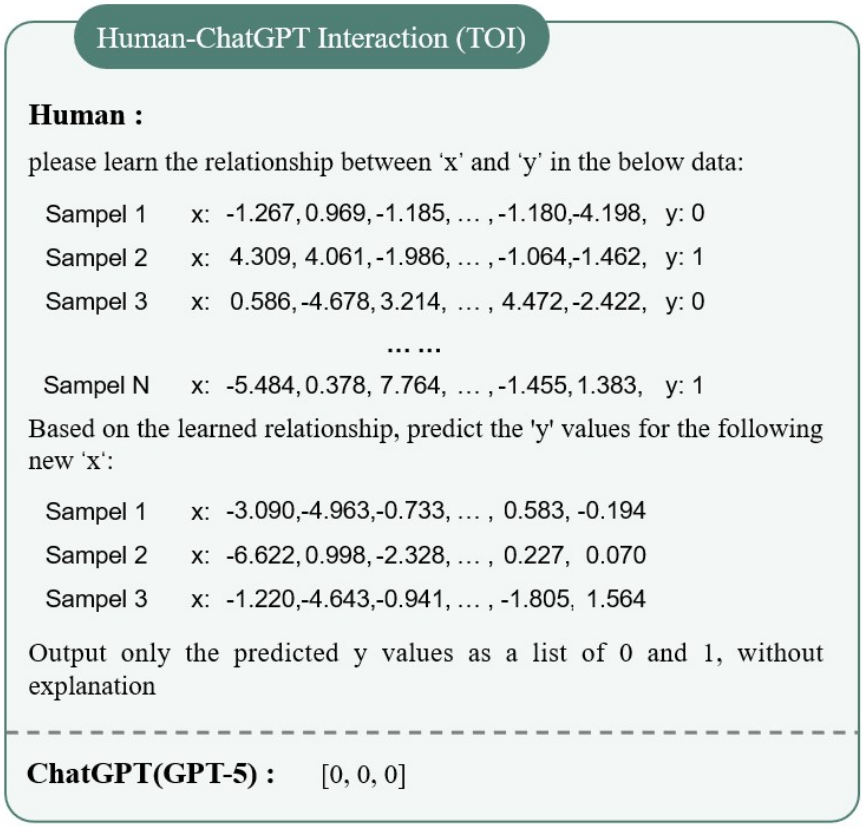

Fig. S3 Schematic diagram of calling the text-only inference of LLMs (using GPT-5 as an example) for food image classification through a dialog window.

(2) **Vision-language inference.** In visual-language inference, users directly provide raw food

images to the LLMs, which uses its built-in visual encoder to extract latent image embeddings, enabling end-to-end multimodal inference. Both GPT-5 and GPT-4o used in this study support image input via a chat interface.

When using the chat interface, users can provide a small number of labeled training images as contextual examples, followed by uploading test images for prediction. However, the chat window currently has technical limitations, allowing a maximum of three image files per message, which restricts the ability to provide batch multimodal prompts. This upload limitation primarily applies to the GPT-5/GPT-4o chat interface, not the underlying model architecture itself. Therefore, large-scale contextual presentation or batch prediction cannot be fully achieved via the chat window. The window-based approach remains suitable for concept presentation, stepwise inference, and few-shot evaluation. As shown in Fig. S4, we provide a practical example demonstrating how GPT-5 can perform multimodal inference using a small number of contextual food images uploaded directly via the chat interface.

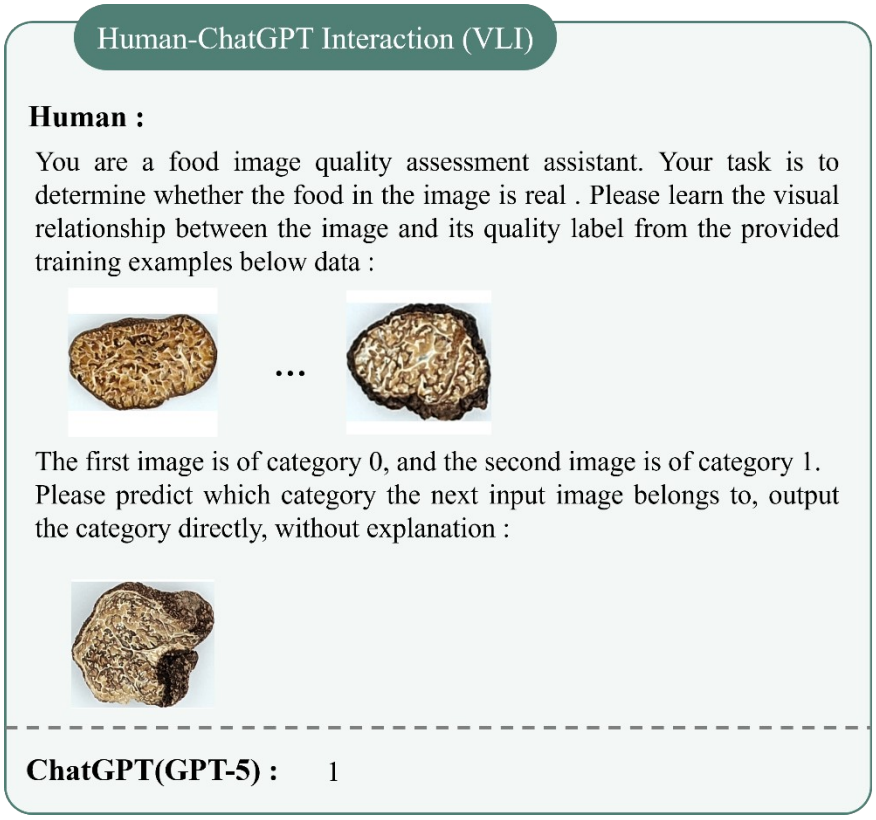

Fig. S4 Schematic diagram of calling the vision-language mode of LLMs (using GPT-5 as an example) for food image classification through a dialog window.

S2.3 Interaction via API

In addition to chat-window interaction, GPT-5, GPT-4o, and DeepSeek-R1 can also be accessed programmatically through their Application Programming Interfaces (APIs). This approach is

particularly suitable for batch processing, automated workflows, and large-scale experiments. In our implementation, API calls are executed in a Python environment (Python 3.8) using the official SDKs provided by each model vendor. To invoke the models via API, users must (i) obtain a valid API key from the provider, (ii) install the required software dependencies (e.g., Python packages such as openai), and (iii) explicitly specify the target model in the code. We provide script examples for both text-only inference and visual-language inference paths, demonstrating how to pass structured food image features or original images and custom hints to the model via API requests, and how to collect the returned prediction results and save them as structured output (CSV format).

(1) **Text-only inference**.

ChatGPT API (TOI)

```
import openai
import pandas as pd

# Initialize API client
client = openai.Client(
    api_key="YOUR_API_KEY",
    base_url="YOUR_URL")

# Build Prompt A: encode training pairs (x, y)
def build_training_chunks(train_x, train_y, chunk_size=90):
    chunks = []
    for i in range(0, len(train_x), chunk_size):
        chunk = ""
        for j in range(i, min(i + chunk_size, len(train_x))):
            chunk += f"x: {train_x[j]}, y: {train_y[j]}\n"
        chunks.append(chunk)
    return chunks

# Build Prompt A + Prompt B
def build_prompt(chunks, test_x_batch):
    prompt_a = (
        "Please learn the relationship between 'x' and 'y' "
        "in the below data :\n")
    prompt_a += "".join(chunks)

    prompt_b = (
        f"\nBased on the learned relationship, predict the 'y' values "
        f"for the following new 'x' : {test_x_batch}\n"
        "Output only the predicted y values as a list of 0 and 1, without explanation.")
    return prompt_a + prompt_b

# Send prompt to the model
def get_prediction(prompt):
    response = client.chat.completions.create(
        model="gpt-5", # Replace with models such as gpt-4o or deepseek-r1
        messages=[
            {"role": "system",
             "content": "You are a helpful assistant that performs binary classification."},
            {"role": "user", "content": prompt}],
        temperature=0.1)
    return response.choices[0].message.content.strip()

# …… (data reading, evaluation code omitted)

# Main workflow
train_x, train_y = ..... # read training data
test_x,  test_y  = ..... # read test data

train_chunks = build_training_chunks(train_x, train_y, chunk_size=90)
all_preds = []
batch_size = 28
for i in range(0, len(test_x), batch_size):
    test_x_batch = test_x[i:i + batch_size]

    prompt = build_prompt(train_chunks, test_x_batch)
    print(f"Processing batch {i//batch_size + 1} ...")

    pred_text = get_prediction(prompt)
    preds = eval(pred_text)   # convert "[0,1,1,...]" to a Python list
    all_preds.extend(preds)

# …… (metrics calculation, saving results)
```

ChatGPT(GPT-5) : test_predictions

x	true_label	pred_label
[-3.0896, -4.9626, -0.7332, 1.9353, 0.5827, -0.1944, -2.6166, 1.3209, -3.0136, 1.362]	0	0
[-6.6225, 0.9982, -2.3283, -2.0345, 0.2266, 0.0704, -0.6239, -1.5211, -0.1669, -0.3149]	0	0
[-1.2195, -4.6427, -0.9413, 0.1282, -1.8051, 1.5636, 2.4954, 0.4041, 1.424, -0.1307]	0	0
[-0.175, -0.2713, 5.6492, 1.2458, 4.5449, 3.8054, 4.6473, -0.1588, 0.6917, -2.8392]	0	1
[1.112, 10.1607, -2.3145, 1.0264, 1.9184, -2.308, 2.7163, 1.553, 0.2381, 0.8912]	1	1

Fig. S5 Schematic diagram of calling the text-only inference of LLMs (using GPT-5 as an example)

for food image classification via the API.

(2) **Vision-language inference.**

ChatGPT API (VLI)

```
from openai import OpenAI
import base64

# Initialize API client
    client = OpenAI(
    api_key="YOUR_API_KEY",
    base_url=" YOUR_URL ")

# Encode an image into base64
def encode_image(path):
    with open(path, "rb") as f:
        return base64.b64encode(f.read()).decode("utf-8")

# Build Prompt C + Prompt D
def build_prompt(train_images, train_labels, test_images):
    content = []
    # Prompt C: Learning from training examples
    content.append({
        "type": "text",
        "text": (
            "Please learn the visual relationship between the image and its "
            "quality label from the provided training examples below [training set].")})
    # Few-shot visual training examples
    for img_path, label in zip(train_images, train_labels):
        content.append({
            "type": "image_url",
            "image_url": f"{encode_image(img_path)}"})
        content.append({
            "type": "text",
            "text": f"Label: {label}"})
    # Prompt D: Predict the category for new images
    content.append({
        "type": "text",
        "text": (
            "Based on the learned relationship, predict the category for the "
            "next input images [test set]. Output only the predicted values "
            "as a list of 0 and 1, without explanation.")})
    # Test images for prediction
    for img_path in test_images:
        content.append({
            "type": "image_url",
            "image_url": f"{encode_image(img_path)}"})
    return content

# Send prompt to the model
def get_prediction(train_images, train_labels, test_images):
    prompt_content = build_prompt(train_images, train_labels, test_images)
    response = client.chat.completions.create(
        model ="gpt-5", # Replace with models such as gpt-4o or deepseek-r1
        messages=[{"role": "user", "content": prompt_content}],
        temperature=0.1 )
    raw_output = response.choices[0].message.content.strip()
    return predictions

# …… (data reading, evaluation code omitted)

# Main workflow
    if __name__ == "__main__":
    train_images, train_labels = ...   # load training data
    test_images = ...                  # load test data
    preds = get_prediction(train_images, train_labels, test_images)
    print(preds)

# …… (metrics calculation, saving results)
```

ChatGPT(GPT-5) : test_predictions

filename	true_label	pred_label
img_0001.jpg	1	1
img_0002.jpg	0	0
img_0003.jpg	1	1
img_0004.jpg	0	0

Fig. S6 Schematic diagram of calling the vision-language inference of LLMs (using GPT-5 as an example) for food image classification via the API.

S2.4 Interpretable Output

Interpretability is crucial for building trust and facilitating applications in the food industry within AI-based food quality analysis. In many practical applications, such as regulatory compliance, quality auditing, and consumer-facing applications, interpretability is critical because understanding the logic behind quality decisions is essential. To address this issue, we designed a prompt to enhance interpretability, requiring the model to provide a classification decision along with a brief explanation

of the visual features that led to the decision. Given that Text-only inference uses high-dimensional feature vectors that are difficult for humans to understand, we only conducted this experiment in visual-language inference. The prompt settings and results are shown in Fig. S7.

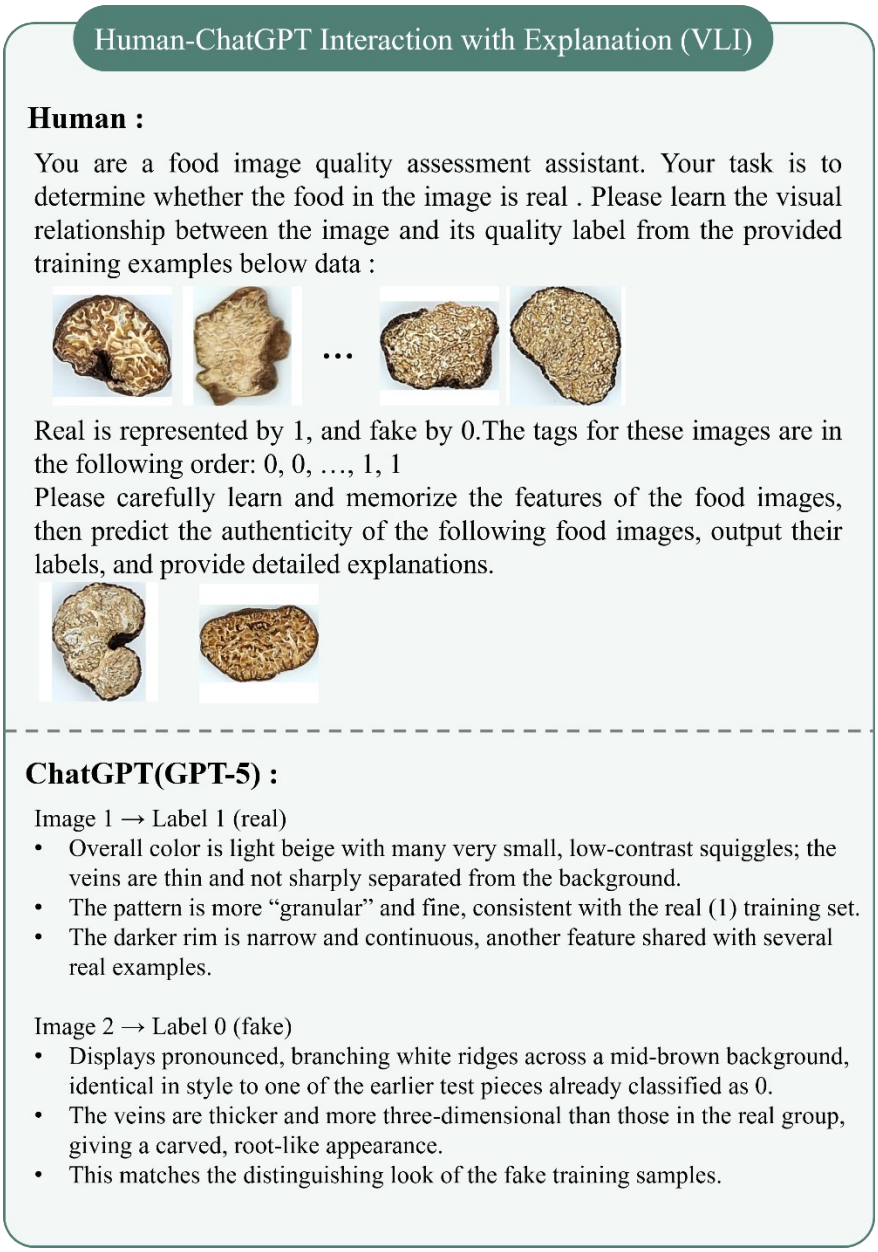

Fig. S7 Schematic diagram of calling the vision-language mode of LLMs (using GPT-5 as an example) for food image classification with explanation.

S2.5 Performance of LLMs on the Test Set for Three Food Quality Analysis Tasks

Confusion matrices can reveal misclassification patterns, which is particularly important for food safety risk assessment. As shown in Figures S7 and S8, we present the confusion matrices for each LLM on the test sets of three food quality analysis tasks in both the TOI and VLI modes, clearly demonstrating the misclassification distribution among different categories.

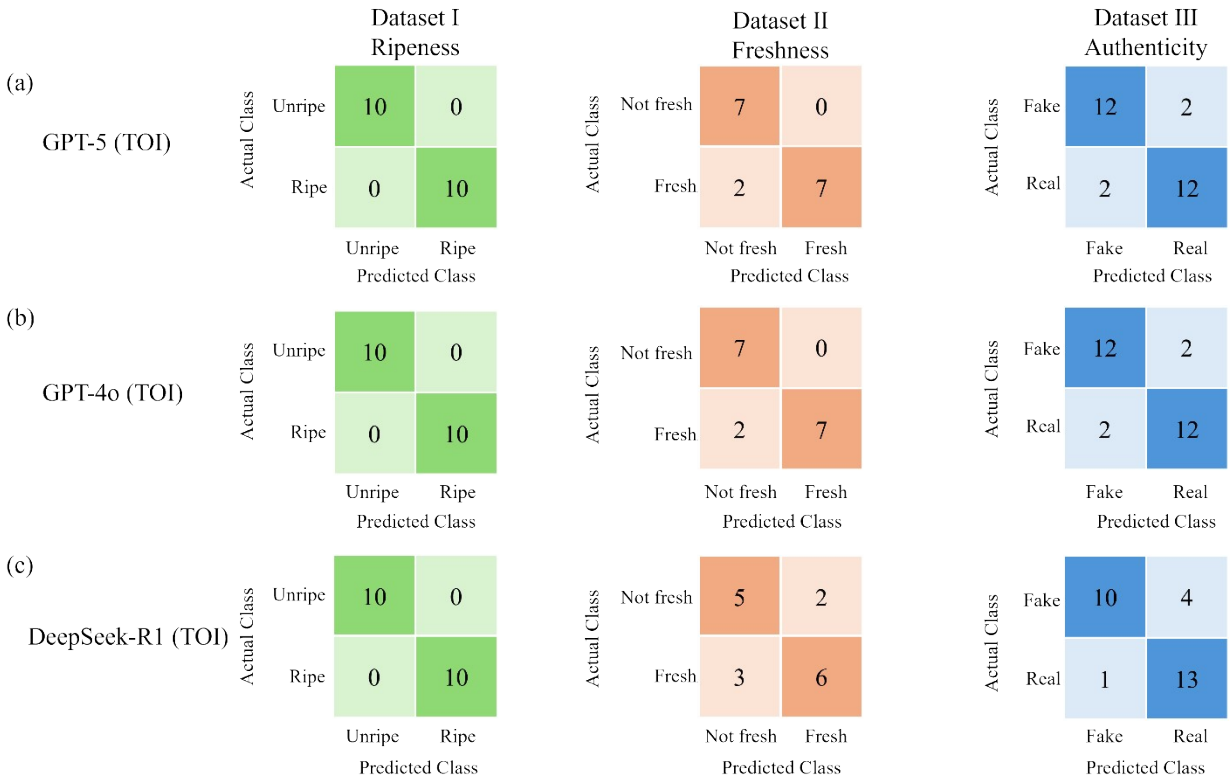

Fig. S8 Confusion matrices of LLMs in TOI mode on three food quality analysis task test sets. (a) GPT-5, (b) GPT-4, (c) DeepSeek-R1.

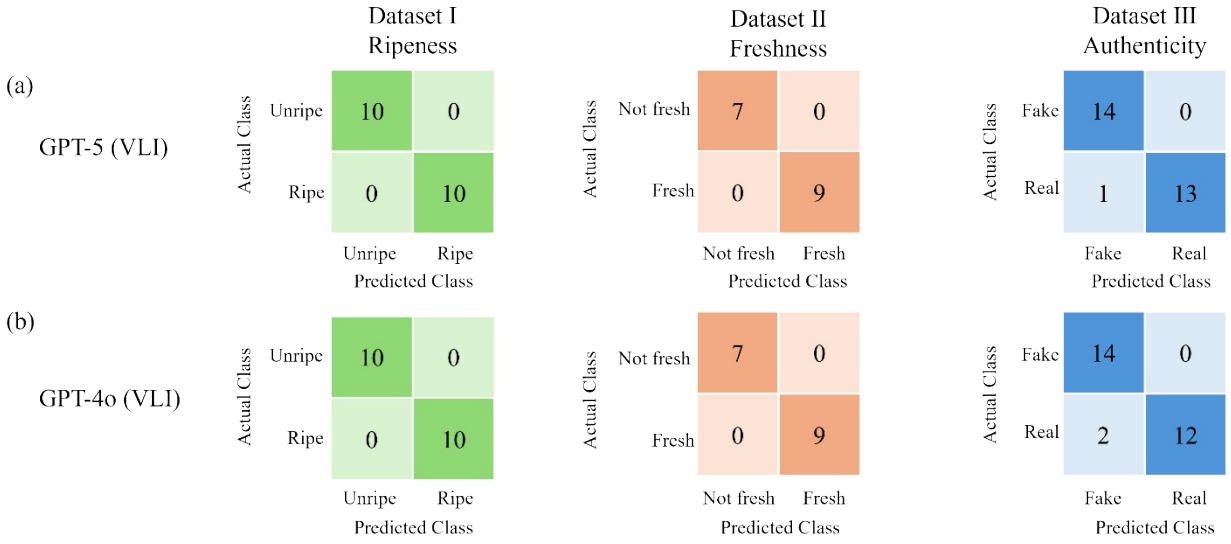

Fig. S9 Confusion matrices of LLMs in VLI mode on three food quality analysis task test sets. (a) GPT-5, (b) GPT-4.

Section S3. Cumulative Variance Contribution Rates

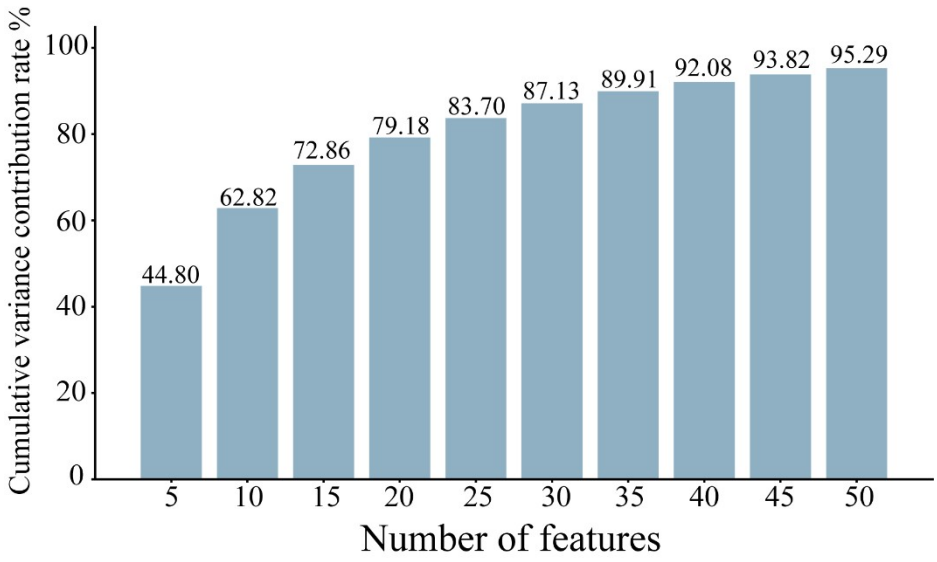

Fig. S10 Cumulative variance contribution rates of different feature numbers.

Section S4. Hyperparameters of the baseline model

Table S1. The hyper-parameters of different machine learning models.

Model	Parameters to be optimized	Results	Package
RF	n_estimators: range(50, 300, 50)	250	scikit-learn (1.3.0) RandomForestClassifier
	min_samples_split: range(2, 10, 2)	2	
	min_samples_leaf: range(1, 10, 2)	1	
k-NN	n_neighbors: range(1, 31)	2	scikit-learn (1.3.0) KNeighborsClassifier
PLS-DA	n_components: range(1, 10)	3	scikit-learn (1.3.0) PLSRegression

Table S2. The hyper-parameters of different deep learning models.

Model	Parameters to be optimized	Results	Package
ResNet-18	The learning rate: [1e-3, 1e-4, 1e-5]	1e-3	torch (2.0.1)
	Batch size: [16, 32, 64]	32	
	The number of epochs: 1000	120	
	The patience of early stop: 50	50	
	The Weight decay: [0, 1e-4, 1e-3]	1e-4	
DenseNet-121	The learning rate: [1e-3, 1e-4, 1e-5]	1e-3	torch (2.0.1)
	Batch size: [16, 32, 64]	32	
	The number of epochs: 1000	110	
	The patience of early stop: 50	50	
	The Weight decay: [0, 1e-4, 1e-3]	1e-4	

MobileNet_v2	The learning rate: [1e-3, 1e-4, 1e-5]	1e-3	torch (2.0.1)
	Batch size: [16, 32, 64]	32	
	The number of epochs: 1000	100	
	The patience of early stop: 50	50	
	The Weight decay: [0, 1e-4, 1e-3]	1e-4	
VGG-16	The learning rate: [1e-3, 1e-4, 1e-5]	1e-3	torch (2.0.1)
	Batch size: [16, 32, 64]	32	
	The number of epochs: 1000	130	
	The patience of early stop: 50	50	
	The Weight decay: [0, 1e-4, 1e-3]	1e-4	

References

1 T. Cover and P. Hart, *IEEE Trans. Inf. Theory*, 1967, 13, 21–27.

2 L. Breiman, *Mach. Learn.*, 2001, 45, 5–32.

3 R. G. Brereton and G. R. Lloyd, *J. Chemom.*, 2014, 28, 213–225.

4 M. Pérez-Enciso and M. Tenenhaus, *Hum Genet* 112, 581–592 (2003).

5 A. M. Jiménez-Carvelo, S. Martín-Torres, F. Ortega-Gavilán and J. Camacho, *Talanta*, 2021, 224, 121904.

6 O. Chovancova, D. Macekova, J. Kostolny, A. Stafurikova and T. Kiskova, in *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, IEEE, Budapest, Hungary, 2019, pp. 298–301.

7 K. He, X. Zhang, S. Ren and J. Sun, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Las Vegas, NV, USA, 2016, pp. 770–778.

8 G. Huang, Z. Liu, L. van der Maaten and K. Q. Weinberger, *arXiv*, 2018, preprint, arXiv:arXiv:1608.06993, DOI: 10.48550/arXiv.1608.06993.

9 M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Salt Lake City, UT, 2018, pp. 4510–4520.

10 K. Simonyan and A. Zisserman, *arXiv*, 2015, preprint, arXiv:arXiv:1409.1556, DOI: 10.48550/arXiv.1409.1556.