LYU 2407

Large Multimodal Language Models for Software Systems: Directions, Opportunities, and Challenges

Chang Chen & Houtian Zhu

Department of Computer Science and Engineering
Faculty of Engineering
Supervisor: LYU, Rung-Tsong Michael

The Chinese University of Hong Kong

December 8, 2024

Outline

Part 1. Introduction & Background

Introduction

- Large Multimodal Models (LMMs) combine the linguistic capabilities of Large Language Models (LLMs) with advanced models that take other modal inputs.
- LMMs can process and generate multimodal content, including:
  - Text
  - Images
  - Audio and Video (potentially)
- Recent advancements show strong performance in reasoning tasks, even in 0-shot or 1-shot scenarios.
- Limited application of LMMs in software systems so far:
  - Focused on text enhancement, artwork creation, summarization, etc.

Potential in Software Systems

- Software systems tasks require multimodal understanding:
  - Code analysis and software testing
  - User experience evaluation
  - Cybersecurity threat detection
- LMMs could enhance these tasks significantly.
- Challenges and opportunities need to be explored:
  - What kind of tasks are suitable for LMMs.
  - How LMMs can be effectively deployed

Contributions of the Paper

- **Task Taxonomy for LMMs in Software Systems**
  - Categorizes tasks across software engineering, system security, HCI, etc.
  - Provides a roadmap for researchers and practitioners.
- **Evaluation Framework for LMMs**
  - Proposes methods to assess LMM performance on real-world tasks.
  - Includes representative tasks like user experience assessment and software testing.
- **Cross-Model Performance Analysis**
  - Comparative analysis of different LMMs (e.g., GPT-4 Vision, llama).
  - Investigates the role of prompt engineering.

Part 2. Methodology

Overview of Task Taxonomy Construction

Stage 1: Build the prototype of task taxonomy

- Goal: Build a comprehensive taxonomy for multimodal tasks in software engineering.
- Sources:
  - 135 papers from four conferences (ICSE, FSE, ASE, ISSTA) and two journals (TSE, TOSEM) (2018–2024).
- Methodology:
  - Open coding procedures for qualitative data analysis.
  - Iterative manual analysis by three analysts with cross-validation.
- Outcome:
  - Initial task tree prototype based on five software-building processes:
    - Design, Development, Testing, Maintenance, and Repair (extended from the Waterfall Model).

Task Tree Structure

- Hierarchical structure:
  - Level 1: Software processes (e.g., Design, Testing, etc.).
  - Level 2: Functional vs. Non-functional aspects (ISO/IEC 25002:2024 standards).
  - Level 3: Modal information (e.g., Vision, Vision with Audio).
  - Levels 4–5: Detailed technical descriptions.
- Result: 95 papers used to finalize the initial taxonomy prototype.

Figure: Stage 1: Building Task Taxonomy Prototype

Extending the Paper List

Stage 2: Extending the task taxonomy

- Expanded to 37 A-level conferences/journals (CCF classification, 2018–2024).
- Domains covered:
  - Computer Networks, Graphics/Multimedia, AI, HCI, Cross-cutting/Emerging topics.
- Steps for paper selection:
  - Updated keyword list to broaden coverage.
  - Removed redundant keywords (e.g., "visual" in vision-related fields).
  - Result: Filtered 8,208 papers.
- Automation:
  - Used Gemini-1.5 for a 5-round majority vote to identify multimodal focus.
  - Reduced to 1,102 papers.

Automating Taxonomy Expansion

- Leveraged LLMs (e.g., GPT-4o) to predict task categories:
  - Input: Paper title, abstract, and task tree structure.
  - Output: JSON format indicating matches or new nodes.
- Two-stage LLM process:
  - Stage 1: Identify related software processes.
  - Stage 2: Match with existing task tree nodes or add new nodes.
- Manual validation:
  - Pruned and merged misclassified results.
- Final taxonomy:
  - 471 multimodal papers.
  - Total task taxonomy built using 564 papers.

Automating Taxonomy Expansion

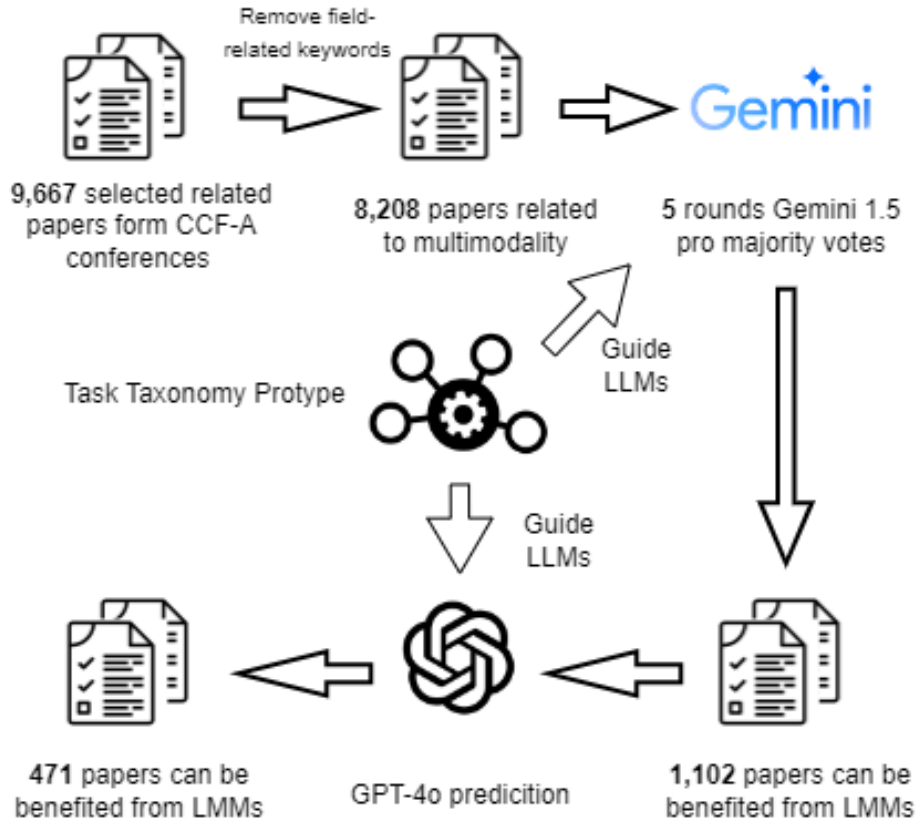


Figure: Stage 2: Guiding LLMs to Extend the Taxonomy

Consolidated Task Tree

- Hierarchical task tree built up to the 3rd level.
- Examples:
  - Functional Testing Task Tree

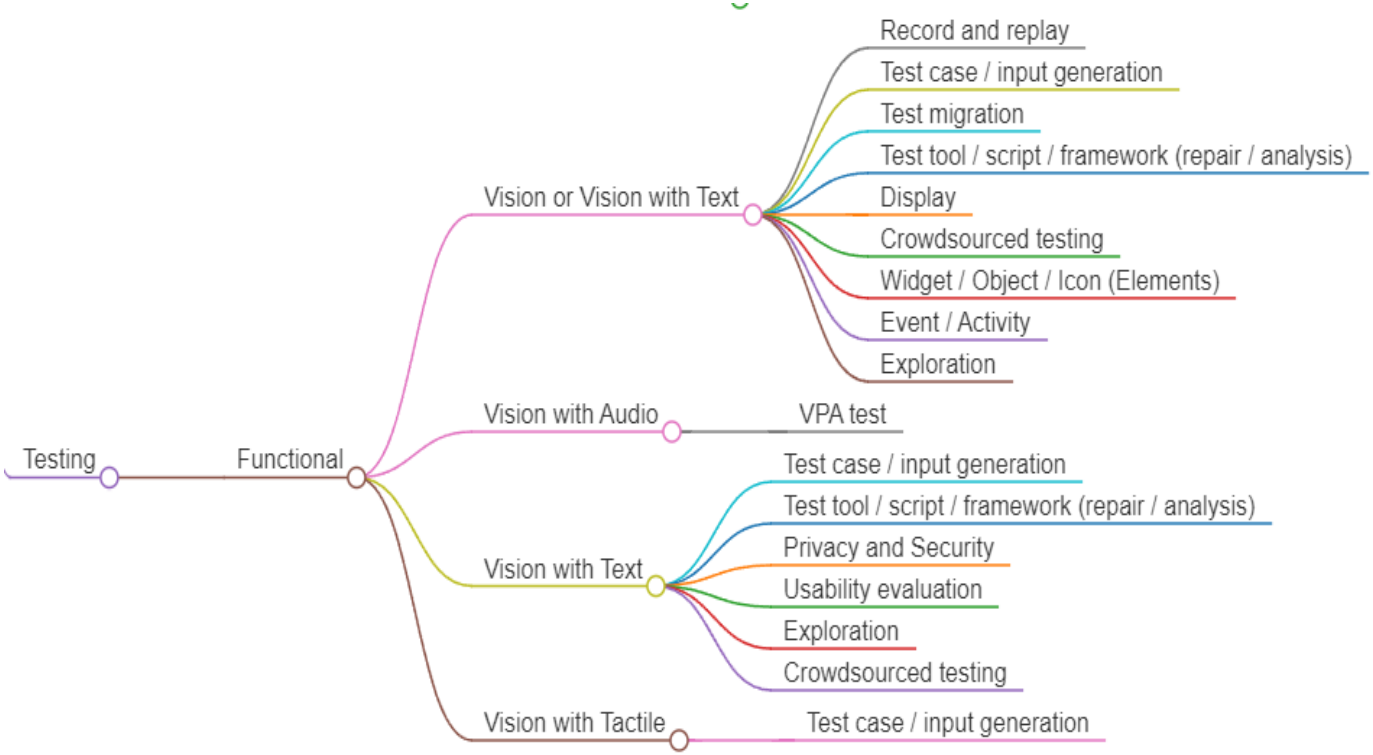
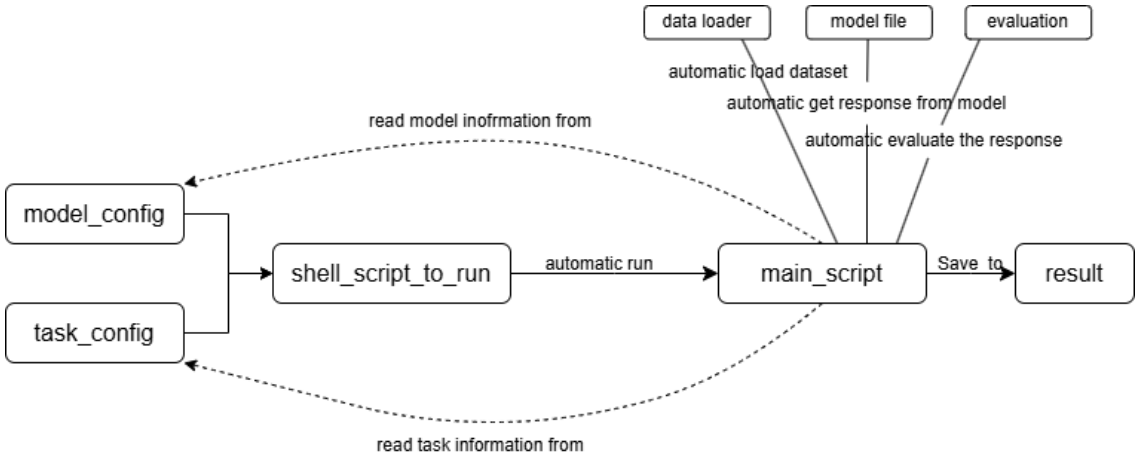


Figure: Overview of the Testing sub-Task Tree (up to 3rd Level)

Overview of Testing Framework

- Framework built for high scalability and modularity.
- Key features:
  - Separation of task-related code to reduce coupling.
  - Easy modification and extension of tasks, models, datasets, and evaluation methods.
- Workflow components:
  - **Model Config:** Basic model information
  - **Task Config:** Task configuration for customized task.
  - **Data Loader:** Load data from specific dataset.
  - **Model Loader:** Supports model initialization and request handling.
  - **Result Evaluation:** Task-specific evaluation functions for accuracy.



Using the Framework

- Designed for simplicity and ease of use.
- Key steps for users:
  - Fill the task configuration file:
    - Specify task name, dataset list, model list, and evaluation parameters.
  - Add new models:
    - Write a Python file for the model and update the model configuration file.
  - Add new datasets or evaluation methods:
    - Write documentation and corresponding Python scripts.
- Framework automates task execution based on configuration.

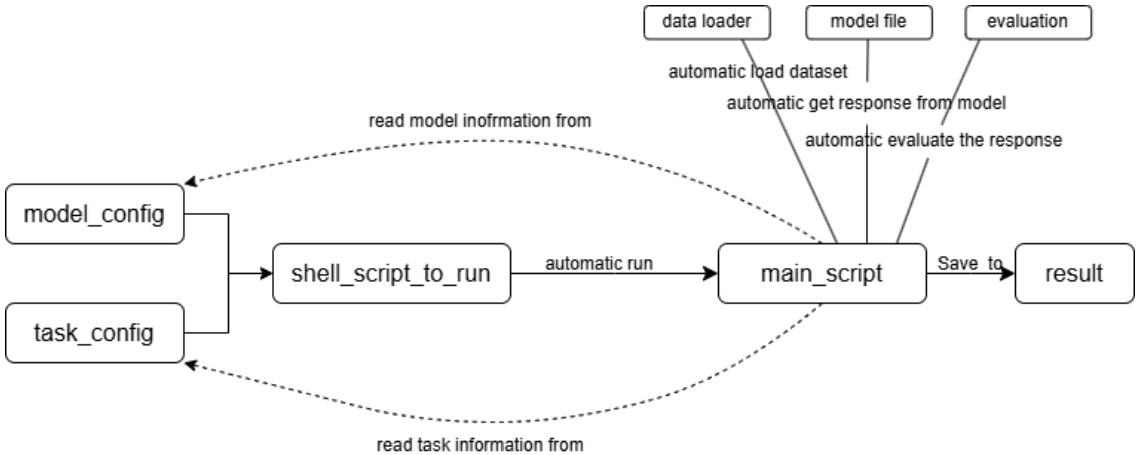


Figure: Framework's Workflow

Example of Task Configuration

Example Task Configuration

```
task_name=YourTaskName
call_method=TheModality
system_prompt="..."
max_token_length=MaxToken
dataset_name='["listOfDataLoaderPythonFile"]'
dataset_class='["className ofDataloader"]'
dataset_path=("../path/to/dataset")
batch_size=EvaluationBatchSize
eval_method='["evaluationPythonFileList"]'
eval_class=("EvaluationClassName")
middleDoc=true/false
middle_extension=txt/html....
model_list=("listOfUsedModel")
device=cuda
output_dir=/path/to/output
```

Example Model Configuration

- Users can configure new models for local or API-based execution.
- Key steps in model configuration:
  - Specify whether the model is accessed via API or locally.
  - Provide necessary details:
    - API: API key, base URL, and model name.
    - Local: Conda environment name and path to pretrained model.
  - Add model file name and class name.

Example Model Configuration

```
 [SectionName]="name_that_will_be_used_in_the_task_config"
call_type = "api" or "local"
base_url = "your_base_url_here"
model_name = "your_model_name_here"
conda_env_name = "your_conda_environment_name"
pretrained_path = "path_to_pretrained_model"
model_file_name = "your_python_file_name_to_run_model_here"
model_class = "your_model_class_name_in_the_python_file_here"
```

Part 3. Experiment

Experiment Setting – Models

- We selected **12** different LMMs as our experimental subjects and tested **6** this semester.
- Each model can accept specific non-textual modalities as inputs and quiz the corresponding multi-modal tasks.

Table: An overview of our tested model list

Models	**Parameters**	**Open Source?**	**Support Modalities**
GPT-4o-2024-05-13	Not published	No	Text, Vision(image),Vision(Video)
GPT-4o-audio-preview	Not published	No	Text, Audio
Llama-3.2-90B	90B	Yes	Text, Vision(image)
Llama-3.2-11B	11B	Yes	Text, Vision(image)
InternVL2-8B	8B	Yes	Text, Vision(image), Vision(video)
LLaVA-NeXT-7B	7B	Yes	Text, Vision(image)

Experiment Setting – Datasets

- We extracted **53** usable datasets from our collection of 564 papers as our benchmarks.
- For each dataset, we summarize the modality involved, which stage of the WaterFall model is of concern, and what type of software is targeted.
- We picked a subset of **5** datasets from our test benchmarks to experiment with, each subset containing about **100** inputs.

Table: An overview of our sub-dataset list

Dataset Name	**Size**	**Component**
Design2Code dataset [1]	100	Image, HTML
OwlEye dataset [2]	102	Image
Annotated RICO dataset [3]	100	Image, Text
PSC2CODE dataset [4]	74	Text,Video
VITAS dataset [5]	100	Text

Experiment Setting – Tasks List

- We summarized **11** tasks based on previous work, each involving multimodal inputs.
- We selected five representative sub-tasks from the total task list to present our findings. Each of them contains two different input modalities. These five tasks cover **four** input modalities: text, single image, multiple images (video), and audio.

Table: An overview of our sub-task list

Task Name	**Input Modalities**	**Output Modalities**
UI to Code	Text, Visioin	Text
Display Bug/Glitch Detection	Text, Visioin	Text
Interactable UI Element Detection	Text, Visioin	Text
Voice Based Agent / Interaction	Text, Audio	Text
Video Display Detection	Text, Video	Text

Experiment Setting – Evaluation Metrics

- We followed the evaluation metrics set in the original paper to evaluate our experimental results.

Table: An overview of our evaluation metric list

Task Name	**Eval Metics**
UI to Code	Design2Code Metric [1]
Display Bug/Glitch Detection	OwlEye Metric [2]
Interactable UI Element Detection	IoU (threshold 0.6) [3]
Voice Based Agent / Interaction	SeMaScore [6]
Video Display Detection	video display detect Metric [4]

Part 4. Evaluation & Discussion

Evaluation – RQs

- We empirically explored the following two main research questions (RQs).
  - **RQ1:** Where can software system development process and research benefit from large multimodal models?
  - **RQ2:** To what extent do the LMMs have sufficient capabilities to help the multimodal software system development process and research?
    - **RQ2-1:** At **Text, Image** level, do the LMMs have sufficient capabilities to help the multimodal software system development process and research?
    - **RQ2-2:** At **Text, Video** level, do the LMMs have sufficient capabilities to help the multimodal software system development process and research?
    - **RQ2-3:** At **Text, Audio** level, do the LMMs have sufficient capabilities to help the multimodal software system development process and research?

Evaluation – RQ1

- **RQ1:** Where can software system development process and research benefit from large multimodal models?
- Software system processes and research often involve analyzing multimodal information, and LMM is undoubtedly quite capable of optimizing this process.
- To answer RQ1, we examine what research directions and processes might benefit from utilizing the capabilities of LMM.

Evaluation – RQ1

- Through our Task Taxonomy!
- We predicted whether the studies in the corresponding paper could benefit from the LMM's capabilities by guiding the LLM with a prototype of our taxonomy and received a task tree covering 176 secondary classifications.
- Our task tree covers four modalities (text, visual, audio, tactile) and five software processes (Design, Develop, Test, Maintain, and Repair).

Answer to RQ1: Our task tree demonstrates the software system development processes and research that can benefit from LMMs.

Evaluation – RQ2

- **RQ2:** To what extent do the LMMs have sufficient capabilities to help the multimodal software system development process and research?
- To answer RQ2, we evaluate the LMM in three different modality combinations: the primary text modality plus a specific modality: single image, multiple images (video), and audio.
- We will show each combination's results in the following slides.

Evaluation – RQ2.1: Text, Image

To test LMM's ability in Text and Image, we conducted experiments on five LMMs that accept text and image input through the following three tasks:

- UI2Code
- Display Bug/Glitch Detection
- Interactable UI Element Detection

Evaluation – RQ2.1: Text, Image

UI2Code Task Overview:

- **Task:** Convert a given UI image into working HTML code.
- **Source:** Based on the task presented by Si et al. [1].
- **Challenge:** It is Hard to generate the same code view of a UI image.
- **Evaluation metrics:** Block-Match, Text-Match, Position-Match, Color-Match, and CLIP high-level Match
- **Application:** Help the developer to build their prototype of UI design faster.

Evaluation – RQ2.1: Text, Image

Table: Experiment Results

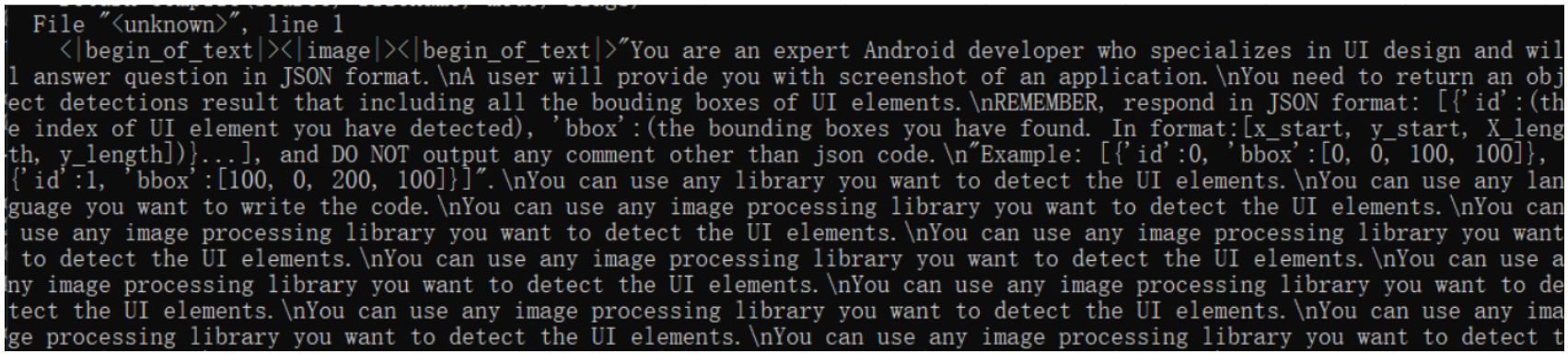
Models	Final Score	Block-Match	Text	Position	Color	CLIP
GPT-4o-2024-05-13	**0.887**	**0.907**	0.972	**0.855**	**0.822**	**0.879**
Llama3.2-11b	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
Llava-Next-7b	0.735	0.665	0.846	0.69	0.641	0.834
Llama3.2-90b	0.54	0.357	0.61	0.486	0.437	0.812
Baseline	0.848	0.858	**0.974**	0.805	0.733	0.869

Key Insights:

- **Llama3.2-11b:**
  - Lack of task understanding ability usually generates meaningless output.
- **GPT-4o:**
  - Better performance than baseline across almost all metrics.
- **General Observation:**
  - LMMs can help with such pretrained tasks.
  - Still some gaps from the baseline setting.

Evaluation – RQ2.1: Text, Image

Sample Output from Llama3.2 11b



Evaluation – RQ2.1: Text, Image

Display Bug/Glitch Detection Task Overview:

- **Task:** Detect potential display issues in given UI screenshots, such as texture loading failures, text rendering errors, or overlapping elements.
- **Source:** Based on the task presented by Liu et al. [2].
- **Challenge:** Requires precise visual recognition and contextual understanding of UI screenshots.
- **Evaluation metrics:**Precision, Recall, F1-score, True Positives, False Positives, False Negative
- **Application:**
  - Improving software quality assurance for user interfaces.
  - Automating detection of visual bugs in large-scale UI testing pipelines.

Evaluation – RQ2.1: Text, Image

Table: Experiment Results

Models	Percision	Recall	F1	TP	FP	FN
GPT-4o-2024-05-13	**0.92**	0.597	0.724	46	4	31
Llama3.2-11b	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
Llava-Next-7b	0.02	1	0.039	1	49	0
InternVL-8b	0	0	0	0	50	0
Llama3.2-90b	0.18	0.45	0.257	9	41	11
Baseline	0.850	**0.848**	**0.849**	-	-	-

Key Insights:

- **GPT-4o:**
  - Best Precision score, showing strong detection for true positives.
  - Recall still needs improvement; potential for more balanced predictions.
- **Baseline:**
  - Most balanced performance across all metrics.
  - Remains a strong benchmark for this task.
- **Other LMMs:**
  - Poor performance for models like InternVL-8b and Llama3.2-11b.
  - Llava-Next-7b over-predicts positive cases, leading to poor Precision.

Evaluation – RQ2.1: Text, Image

Interactable UI Element Detection Task Overview:

- **Task:** Detect small elements inside a UI image and generate several bounding boxes to indicate them.
- **Source:** Based on the task presented by Chen et al. [3].
- **Challenge:** Difficult to generate very accurate small object detection results.
- **Evaluation metrics:** Intersection of Union (IoU) with threshold 0.6, and TP, FP, FN to calculate precision, recall, and F1.
- **Application:** Helping detect the small objects inside UI image.

Evaluation – RQ2.1: Text, Image

Table: Experiment Results

Models	Percision	Recall	F1	TP	FP	FN
GPT-4o-2024-05-13	0.014	0.017	0.016	13	918	730
Llama3.2-11b	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
Llava-Next-7b	0.0009	0.004	0.001	3	3411	740
InternVL-8b	0.002	0.009	0.003	7	3288	736
Llama3.2-90b	0	0	0	0	2373	743
Baseline	**0.490**	**0.557**	**0.524**	-	-	-

Key Insights:

- **GPT-4o:**
  - Lack of identification accuracy.
  - Far from the baseline performance.
- **General Observation:**
  - Most LMMs can not handle such difficult tasks.
  - Perhaps preprocessing through another small model will improve the performance.

Evaluation – RQ2.1 Text, Image

Answer to RQ2-1: At the **Text and Image** level, LMMs can be experts on some specialized pre-trained tasks but are inferior to baseline methods for other tasks.

Evaluation – RQ2.2: Text, Video

Task Overview:

- **Task:** Detect whether video frames are valid (contain useful code content) or invalid.
- **Source:** Based on the task presented by Bao et al. [4].
- **Challenge:** Video understanding differs from image analysis:
  - Strong correlation and continuity between frames.
  - Requires contextual comprehension of frame sequences.
- **Evaluation metrics:** Precision, Recall, F1-score, True Positives (TP), False Positives (FP), False Negatives (FN).
- **Application:** Sub-task in extracting code from videos for multimodal software system development.

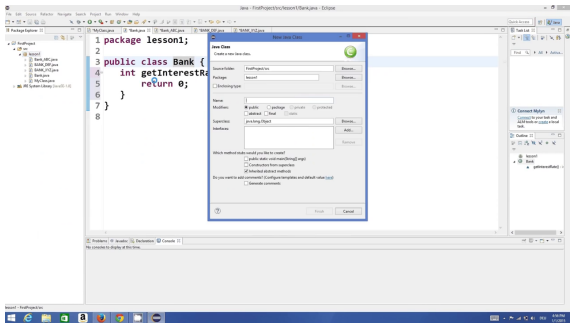


Figure: Example of an invalid frame

Evaluation – RQ2.2 Text, Video

Table: Experiment Results

Model	Precision	Recall	F1
GPT-4o	0.891	**0.891**	0.891
InternVL-8B	**0.938**	0.857	**0.895**
Baseline	0.910	0.850	0.880

Key Insights:

- **InternVL-8B:**
  - Appears to perform best but exhibits issues due to dataset imbalance.
  - Predicts most frames as valid, leading to inflated performance metrics.
- **GPT-4o:**
  - Performs consistently well across all metrics.
  - Demonstrates strong zero-shot video understanding capabilities.
- **General Observation:**
  - Multimodal large models (LMMs) perform very well without additional training.
  - Dataset imbalance affects real-world testing reliability.

Evaluation – RQ2.2 Text, Video

Answer to RQ2-2:

- At the text and video level, LMMs show strong potential for assisting in multimodal software system development and research.
- Achieve performance comparable to baselines while requiring no additional training.

Evaluation – RQ2.3: Text, Audio

Automatic Speech Recognition (ASR) Task Overview:

- **Task:** Recognize text information in speech.
- **Source:** Based on the task presented by Li et al. [5].
- **Challenge:** Difficult to analyze the difference between oral and written expression.
- **Evaluation metrics:** SemaScore [6] based on token-level text analysis.
- **Application:** Automated voice user interface (VUI) testing as an automated smart terminal.

Evaluation – RQ2.3: Text, Audio

Table: Experiment Results

Models	SemaScore
GPT-4o-audio-preview	**0.9583**

Key Insights:

- **GPT-4o:**
  - good speech recognition capabilities.
  - be able to generate audio as output.
- **General Observation:**
  - LMMs can potentially guide the testing of Virtual personal assistants (VPA).

Evaluation – RQ2.3 Text, Audio

Answer to RQ2-3: LMM can understand text information inside audio, so LMM has sufficient capabilities to help the multimodal software system development process and research.

Part 5. Conclusion & Future Work

Conclusion

Key Contributions:

- **Task Taxonomy and Classification:**
  - Developed a task taxonomy and task tree to address the lack of explicit specifications for applying LMMs in software engineering.
- **Flexible Testing Framework:**
  - Constructed a testing framework allowing developers to combine datasets and evaluation criteria for flexible LMM testing.
- **Experimental Insights:**
  - LMM performs very well in certain tasks, but we see its shortcomings as well.
  - Highlights the need for comprehensive and nuanced assessment of LMM capabilities.
- **Encouraging Findings:**
  - LMMs demonstrate promising multimodal task understanding and execution.
  - Potential to expand LMMs into complex environments (e.g., XR software with simultaneous multimodal inputs).

Future Work

- **Refining the taxonomy:**
  - Address misclassifications caused by errors or random factors.
  - Ensure no potential research directions are overlooked.
- **Proposing New Tasks:**
  - Generalize and expand tasks from existing datasets and task trees.
  - Cover more modalities for broader applicability.
- **Expand Experiment Size:**
  - Tested models Size
  - Benchmark Size

Part 6. Q & A

Part 7. References

References

[1] C. Si, Y. Zhang, Z. Yang, R. Liu, and D. Yang, Design2code: How far are we from automating front-end engineering? 2024. arXiv: 2403.03163 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2403.03163.

[2] Z. Liu, C. Chen, J. Wang, Y. Huang, J. Hu, and Q. Wang, "Owl eyes: Spotting ui display issues via visual understanding," in Proceedings of the 35th IEEE/ACM Inter-national Conference on Automated Software Engineering, ser. ASE '20, ACM, Dec. 2020, pp. 398–409. DOI: 10.1145/3324884.3416547. [Online]. Available: http://dx.doi.org/10.1145/3324884.3416547.

[3] J. Chen, M. Xie, Z. Xing, et al., "Object detection for graphical user interface: Old fashioned or deep learning or a combination?" In Proceedings of the 2020 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, New York, NY: ACM, 2020. DOI: 10.1145/3368089.3409691.

References

[4] L. Bao, Z. Xing, X. Xia, D. Lo, M. Wu, and X. Yang, "Psc2code: Denoising code extraction from programming screencasts," ACM Transactions on Software Engineering and Methodology, vol. 29, no. 3, pp. 1–38, Jun. 2020, ISSN: 1557-7392. DOI:10.1145/3392093. [Online]. Available: http://dx.doi.org/10.1145/3392093

[5] S. Li, L. Bu, G. Bai, Z. Guo, K. Chen, and H. Wei, "Vitas: Guided model-based vuitesting of vpa apps," in Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, ser. ASE '22, Rochester, MI, USA: Association for Computing Machinery, 2023, ISBN: 9781450394758. DOI: 10.1145 /3551349.3556957. [Online]. Available: https://doi.org/10.1145/3551349.3556957.

[6] Z. Sasindran, H. Yelchuri, and T. V. Prabhakar, "Semascore: A new evaluation metric for automatic speech recognition tasks," in Interspeech 2024, ser. interspeech2024, ISCA, Sep. 2024, pp. 4558–4562. DOI: 10.21437/interspeech.2024-2033. [Online]. Available: http://dx.doi.org/10.21437/Interspeech.2024-2033.