Final Report - Checklists and LLM prompts for efficient and effective test creation in data analysis

by John Shiu, Orix Au Yeung, Tony Shum, Yingzi Jin

## Executive Summary

The global artificial intelligence (AI) market is expanding rapidly and demanding for robust quality assurance for machine learning (ML) systems to prevent risks such as misinformation, social bias, financial losses, and safety hazards. FixML addresses these challenges by offering an automated code review tool embedded with best practices for ML test suites, curated from ML research and industry standards.

Our approach includes developing the tool in Python package based on Large Language Models (LLMs) and creating comprehensive checklists to enhance ML software’s trustworthiness, quality, and reproducibility. The tool analyzes ML projects, compares test suites against best practices, and delivers evaluations and test specifications, which can significantly reduce the time and effort required for manual assessments.

To ensure reliability, we defined two success metrics: accuracy (comparison with human expert judgments) and consistency (standard deviation across multiple runs). Our findings indicated that while our tool is effective, there is room to improve in both metrics, which requires further prompt engineering and refinement for enhanced performance.

The FixML package is available on PyPI and can be used as a CLI tool and a high-level API, which makes it user-friendly and versatile. Future improvements will focus on specialized checklists, enhanced evaluators, customized test specifications, and other optimization to further improve ML system quality and user experience.

## Introduction

### Problem Statement

The global AI market is growing exponentially (Grand-View-Research 2021), driven by its ability to autonomously make complex decisions impacting various aspects of human life, including financial transactions, autonomous transportation, and medical diagnosis.

However, ensuring the software quality of these systems remains a significant challenge (Openja et al. 2023). Specifically, the lack of a standardized and comprehensive approach to testing ML systems introduces potential risks to stakeholders. For example, inadequate quality assurance in ML systems can lead to severe consequences, such as misinformation (Belanger 2024), social bias (Nunwick 2023), substantial financial losses (Regidi 2019) and safety hazards (Shepardson 2023)

Therefore, defining and promoting an industry standard and establishing robust testing methodologies for these systems is crucial. But how?

### Our Objectives

We propose to develop testing suites diagnostic tools based on LLMs and curate checklists based on ML research papers and best practices to facilitate comprehensive testing of ML systems with flexibility. Our goal is to enhance applied ML software’s trustworthiness, quality, and reproducibility across both the industry and academia (Kapoor and Narayanan 2022).

## Data Science Methods

### Current Approaches

To ensure the reproducibility, trustworthiness, and lack of bias in ML systems, comprehensive testing is essential. We outlined some traditional approaches for assessing the completeness of ML system tests with their advantages and drawbacks as follows.

1. **Code Coverage**

Code coverage measures the proportion of source code of a program executed when a particular test suite is run. Widely used in software development, it quantifies test quality and is scalable due to its short processing time. However, it cannot indicate the reasons or specific ML areas where the test suites fall short under the context of ML system development.

1. **Manual Evaluation**

Manual evaluation involves human experts reviewing the source code, whom can take the business logic into considerations and identify vulnerabilities. It often provides context-specific improvement suggestions and remains one of the most reliable practices (Openja et al. 2023), (Alexander et al. 2023). However, it is time-consuming and not scalable due to the scarcity of human experts. Moreover, different experts might put emphasis on different ML test areas and lack a comprehensive and holistic review of the ML system test suites.

### Our Approach

Our approach is to deliver an automated code review tool with the best practices of ML test suites embedded. This tool aims to educate ML users on best practices while providing comprehensive evaluations of their ML system codes.

To establish these best practices, we utilized data from ML research papers and recognized online resources. In collaboration with our partner, we researched industrial best practices (Team 2023), (Jordan 2020) and academic literature (Openja et al. 2023), and consolidated testing strategies into a human-readable and machine-friendly checklist that can be embedded into the automated tool.

For development, we collected 11 GitHub repositories of ML projects as studied in (Openja et al. 2023). These Python-based projects include comprehensive test suites. Our tool should be able to analyze these test suites, compare them with embedded best practices, and deliver evaluations.

We expect that our approach will provide scalable and reliable test suite evaluations for multiple ML projects. However, we recognize that our current best practices only focus on a few high-priority test areas due to time constraints. We plan to expand this scope in the future. While our tool’s evaluations are not yet as reliable as human evaluations, we will quantify its performance.

### Success Metrics

To properly assess the performance of our tool which leverages LLMs capability, we have taken reference of the methods in (Alexander et al. 2023) and defined two success metrics: accuracy and consistency. These metrics will help users (researchers, ML engineers, etc.) gauge the trustworthiness of our tool’s evaluation results.

1. **Accuracy vs Human Expert Judgement**

We run our tool on ML projects from (Openja et al. 2023) to obtain evaluation results for each ML checklist item. These results are then compared with our manually assessed ground truth data based on the same criteria. Accuracy is calculated as the proportion of matching results to the total number of results.

1. **Consistency**

We perform multiple runs on each ML project to obtain evaluation results for each checklist item. Consistency is measured by calculating the standard deviation of these results across multiple runs for each project.

## Data Product & Results

### Data Products

Our solution includes a curated checklist for robust ML testing and a Python package for checklist-based evaluation of ML project’s testing robustness using LLMs. The package is publicly available on the Python Packaging Index (PyPI).

Justifications for these products are:

* Checklists have been shown to reduce errors in software systems and promote code submissions (Gawande 2010), (Pineau et al. 2021).
* Python is widely used in ML, compatible with various OSes, and integrates well with LLMs. These ensure the ease of use and development.

#### How to use the product

There are two ways to make use of this package:

1. **As a CLI tool.** A runnable command fixml is provided by the package. Once installed, users can perform codebase evaluations, generate test function specifications, and more by running subcommands under fixml in the terminal.
2. **As a high-level API.** Users can import necessary components from the package into their own systems. Documentation is available through docstrings.

By offering it as both CLI tool and API, our product is user-friendly to interact with, and versatile to support various use cases such as web application development and scientific research.

#### System Design

(FIXME To be revised)

|  |  |
| --- | --- |
| |  | | --- | |  |   Figure 1: Diagram of FixML system design |

The design of our package follows object-oriented and SOLID principles, which is fully modular. Users can easily switch between different prompts, models, and checklists, which facilitates code reusability and collaboration to extend its functionality.

There are five components in the system of our package:

1. **Code Analyzer**

It extracts test suites from the input codebase, to ensure only the most relevant details are provided to LLMs given token limits.

1. **Prompt Templates**

It stores prompt templates for instructing LLMs to generate responses in the expected format.

1. **Checklist**

It reads the curated checklist from a CSV file into a dictionary with a fixed schema for LLM injection. The package includes a default checklist for distribution.

1. **Runners**

It includes the Evaluator module, which assesses each test suite file using LLMs and outputs evaluation results, and the Generator module, which creates test specifications. Both modules feature validation, retry logic, and record response and relevant information.

1. **Parsers**

It reads the report templates and converts Evaluator’s responses into evaluation reports in various formats (QMD, HTML, PDF) using the Jinja template engine, which enables customizable report structures.

#### Checklist Design

The embedded checklist contains best practices for testing ML pipelines, and is curated from ML research and recognized online resources. Prompt engineering is applied to further improve the LLM performance. This helps mitigate LLM hallucinations (Zhang et al. 2023) by ensuring strict adherence to the checklist.

Table 1: Structure of the checklist in CSV format. Users can easily modify and expand the checklist by adding new rows to the CSV file.

| Column | Description |
| --- | --- |
| ID | Unique Identifier of the checklist item |
| Topic | Test Area of the checklist item |
| Title | Title of the checklist item |
| Requirement | Prompt for the checklist item to be injected into LLMs for evaluation |
| Explanations | Detailed explanations for human understanding |
| Reference | References for the checklist item, e.g. academic papers |
| Is Evaluator Applicable | Indicates if the checklist item is used during evaluation (0 = No, 1 = Yes) |

|  |  |
| --- | --- |
| |  | | --- | |  |   Figure 2: An example of the checklist exported in PDF format. Users can easily read and distribute the checklist. |

#### Artifacts

Using our package results in three artifacts:

1. **Evaluation Responses**

These responses include both LLM evaluation results and process metadata stored in JSON format. This supports various downstream tasks, such as report rendering and scientific research, by selectively extracting information.

|  |  |
| --- | --- |
| |  | | --- | |  |   Figure 3: An example of the evaluation responses. It includes call\_results for evaluation outcomes and details about the model, repository, checklist, and the run. |

1. **Evaluation Report**

This report provides a well-structured presentation of evaluation results for ML projects. It includes a summary of the completeness score and a detailed breakdown with explanations for each checklist item score.

|  |  |
| --- | --- |
| |  | | --- | |  |   Figure 4: An example of the evaluation report exported in PDF format using our default template. Users can customize their reports by creating their own templates. |

1. **Test Specification Script**

These are generated test specifications stored as Python scripts.

|  |  |
| --- | --- |
| |  | | --- | |  |   Figure 5: An example of the generated test specifications |

### Evaluation Results

As described in Success Metrics, we conducted 30 iterations on each repository from (Openja et al. 2023) and examined the breakdown of the completeness score to assess our tool’s evaluation quality.

1. **Accuracy**

Accuracy is our primary consideration since our tool’s value depends on its ability to align with expert judgment. We targeted 3 of the repositories for human evaluation: [lightfm](https://github.com/lyst/lightfm), [qlib](https://github.com/microsoft/qlib), [DeepSpeech](https://github.com/mozilla/DeepSpeech). We compared and plotted the graph to illustrate how well our tool’s outputs align with the ground truth.

import pandas as pd  
gt = pd.read\_csv('../data/processed/ground\_truth.csv')  
gt

Table 2: Ground truth scores for 3 repositories per checklist item based on human evaluation. (1 = fully satisfied, 0.5 = partially satisfied, 0 = not satisfied)

|  | id | title | DeepSpeech | lightfm | qlib |
| --- | --- | --- | --- | --- | --- |
| 0 | 2.1 | Ensure Data File Loads as Expected | 0.0 | 1.0 | 0.5 |
| 1 | 3.2 | Data in the Expected Format | 0.0 | 1.0 | 1.0 |
| 2 | 3.5 | Check for Duplicate Records in Data | 0.0 | 0.0 | 0.0 |
| 3 | 4.2 | Verify Data Split Proportion | 0.0 | 1.0 | 0.5 |
| 4 | 5.3 | Ensure Model Output Shape Aligns with Expectation | 0.0 | 0.5 | 1.0 |
| 5 | 6.1 | Verify Evaluation Metrics Implementation | 0.0 | 1.0 | 1.0 |
| 6 | 6.2 | Evaluate Model's Performance Against Thresholds | 0.0 | 1.0 | 1.0 |

import altair as alt  
import pandas as pd  
  
df\_repo\_\_stat = pd.read\_csv('../data/processed/score\_stat\_by\_repo\_3.5-turbo.csv')  
gt = pd.read\_csv('../data/processed/ground\_truth.csv')  
gt = gt.melt(id\_vars=['id', 'title'], var\_name='repo', value\_name='ground\_truth')  
  
df\_repo\_\_stat\_with\_gt = df\_repo\_\_stat.merge(gt, on=['id', 'title', 'repo'])  
  
base = alt.Chart(  
 df\_repo\_\_stat\_with\_gt.query('repo in ["lightfm", "qlib", "DeepSpeech"]')  
).transform\_calculate(  
 min="max(0, datum.mean-datum.std)",  
 max="min(1, datum.mean+datum.std)"  
)  
   
# generate the points  
points = base.mark\_point(  
 filled=True,  
 size=50,  
 color='black'  
).encode(  
 x=alt.X('mean:Q').scale(domainMin=0, domainMax=1).title("Score").axis(  
 labelExpr="datum.value % 0.5 ? null : datum.label"  
 ),  
 y=alt.Y('id\_title:N', title=None, axis=alt.Axis(labelPadding=10, labelLimit=1000, grid=False))#.scale(domainMin=0, domainMax=1).title('Score'),  
)  
  
# generate the points for ground truth  
gt\_points = base.mark\_point(  
 filled=True,  
 size=200,  
 color='green',  
 shape="diamond"  
).encode(  
 x=alt.X('ground\_truth:Q'),  
 y=alt.Y('id\_title:N')  
)  
  
# generate the error bars  
errorbars = base.mark\_errorbar().encode(  
 x=alt.X("min:Q").title('1 SD'), #"id:N",  
 x2="max:Q",  
 y="id\_title:N"  
)  
  
(gt\_points + points + errorbars).facet(  
 column=alt.Column('repo:N').title(None)  
).configure\_axis(   
 labelFontSize=12,   
 titleFontSize=12  
)

|  |
| --- |
| alt.FacetChart(...)  Figure 6: Analysis of accuracy of the scores per checklist item. The black dot and line represent the mean and standard deviation of scores from the tool, while the green diamond represents the ground truth score for a single repository. It shows our tool tends to underrate satisfactory cases. |

When examining accuracy, we observed that our tool effectively identifies non-satisfying cases. However, it often classifies fully satisfied items as partially satisfied and partially satisfied items as not satisfied.

This indicates that our tool achieves a certain degree of accuracy. The next questions we consider are:

* Are there other factors that impact the performance of our tool?
* In what direction can we improve our tool?

1. **Consistency**

Consistency is another consideration because it directly impacts the reliability of the evaluation results from user perspective. Given that LLM-generated completeness scores contain some randomness, we plotted the uncertainty of these scores across checklist items and repositories to show how consistent these results were.

stds = df\_repo\_\_stat[['repo', 'std', 'id\_title']].pivot(index='repo', columns='id\_title').copy()  
stds.columns = [col[1] for col in stds.columns]  
stds = stds.reset\_index()  
stds = stds.melt(id\_vars='repo', var\_name='id\_title')  
  
base = alt.Chart(stds)  
  
box = base.mark\_boxplot(  
 color='grey',  
 opacity=0.5,  
 size=20,  
).encode(  
 x=alt.X('value:Q').title('Standard Deviation of Scores'),  
 y=alt.Y('id\_title:N', title=None, axis=alt.Axis(labelPadding=10, labelLimit=1000, grid=False))  
)  
  
stripplot = base.mark\_circle(size=100).encode(  
 y=alt.Y(   
 'id\_title:N',  
 axis=alt.Axis(ticks=False, grid=True, labels=True),   
 scale=alt.Scale(),   
 ),   
 x='value:Q',  
 yOffset="jitter:Q",  
 color=alt.Color('id\_title:N', legend=None),  
 tooltip='repo'  
).transform\_calculate(  
 # Generate Gaussian jitter with a Box-Muller transform  
 jitter="sqrt(-2\*log(random()))\*cos(2\*PI\*random())"  
)  
  
(  
 box + stripplot  
).configure\_view(   
 stroke=None  
).configure\_axis(   
 labelFontSize=12,   
 titleFontSize=12  
).properties(  
 height=300,   
 width=600,  
)

|  |
| --- |
| alt.LayerChart(...)  Figure 7: Analysis of the uncertainty of scores (measured in standard deviation on a scale of 0 to 1) per checklist item. Each dot represents the uncertainty of scores from 30 runs of a single repository. It shows different patterns across checklist items. |

When we examined the consistency, we observed various patterns and sought to identify potential causes, which could provide ideas for improvements. We identified two categories with diverging patterns:

1. **High Uncertainty**

Items like 6.1 Verify Evaluation Metrics Implementation showed high standard deviations across repositories (median = 0.12). This might suggest potential issues with prompt quality for the LLM to produce consistent results, which could be mitigated through improved prompt engineering.

1. **Outliers with High Uncertainty**

Items like 2.1 Ensure Data File Loads as Expected had outliers with exceptionally high standard deviations, which is possibly due to unorthodox repositories. A careful manual examination is required for a more definitive conclusion.

#### Comparison among LLMs

To evaluate if newer LLMs improve performance, we obtained outputs from gpt-4o and gpt-4-turbo on the lightfm repository and plotted the graph to compare how our tool performs in terms of accuracy and consistency when switched the LLMs.

By comparing with the results from gpt-3.5-turbo (shown in [Figure 6](#fig-accu-mean-sd-plot)), we observed an increase in consistency using newer LLMs given the smaller standard deviations. However, we found that gpt-4o returned “Satisfied” for all items, while gpt-4-turbo deviated more from the ground truth for item 3.5 Check for Duplicate Records in Data, 5.3 Ensure Model Output Shape Aligns with Expectation compared to gpt-3.5-turbo.

df\_repo\_4o\_\_stat = pd.read\_csv('../data/processed/score\_stat\_by\_repo\_4o.csv')  
df\_repo\_4o\_\_stat\_with\_gt = df\_repo\_4o\_\_stat.merge(gt, on=['id', 'title', 'repo'])  
df\_repo\_4o\_\_stat\_with\_gt['model'] = 'gpt-4o'  
  
df\_repo\_4turbo\_\_stat = pd.read\_csv('../data/processed/score\_stat\_by\_repo\_4-turbo.csv')  
df\_repo\_4turbo\_\_stat\_with\_gt = df\_repo\_4turbo\_\_stat.merge(gt, on=['id', 'title', 'repo'])  
df\_repo\_4turbo\_\_stat\_with\_gt['model'] = 'gpt-4-turbo'  
  
df\_model\_comp = pd.concat(  
 (df\_repo\_4turbo\_\_stat\_with\_gt, df\_repo\_4o\_\_stat\_with\_gt),   
 axis=0  
)  
  
base = alt.Chart(  
 df\_model\_comp  
).transform\_calculate(  
 min="max(0, datum.mean-datum.std)",  
 max="min(1, datum.mean+datum.std)"  
)  
   
# generate the points  
points = base.mark\_point(  
 filled=True,  
 size=50,  
 color='black'  
).encode(  
 x=alt.X('mean:Q').scale(domainMin=0, domainMax=1).title("Score").axis(  
 labelExpr="datum.value % 0.5 ? null : datum.label"  
 ),  
 y=alt.Y('id\_title:N', title=None, axis=alt.Axis(labelPadding=10, labelLimit=1000, grid=False)),  
)  
  
# generate the points for ground truth  
gt\_points = base.mark\_point(  
 filled=True,  
 size=200,  
 color='green',  
 shape="diamond"  
).encode(  
 x=alt.X('ground\_truth:Q'),  
 y=alt.Y('id\_title:N')  
)  
  
# generate the error bars  
errorbars = base.mark\_errorbar().encode(  
 x=alt.X("min:Q").title('1 SD'),  
 x2="max:Q",  
 y="id\_title:N"  
)  
  
(  
 gt\_points + points + errorbars  
).facet(  
 column=alt.Column('model:N').title(None)  
).configure\_axis(   
 labelFontSize=12,   
 titleFontSize=12  
)

|  |
| --- |
| alt.FacetChart(...)  Figure 8: Analysis of the scores per checklist item on repository lightfm using different GPT versions. The black dot and line represent the mean and standard deviation of scores from the tool, while the green diamond represents the ground truth score. |

The graph suggests a potential improvement in consistency and/or accuracy when switching to newer LLMs. However, it also indicates that what works well with the current LLM may not necessarily perform well with newer models. This implies the need for exploration in different structures, such as prompt engineering for gpt-4-turbo.

## Conclusion

### Wrap Up

The development of FixML has been driven by the need of better quality assurance in ML systems and the current limitations of traditional testing methods on ML projects. FixML provides curated checklists and automated tools that enhance the evaluation and creation of test suites for ML projects. This in return, significantly reduces the time and effort required to assess the completeness of ML test suites, and thus promotes thorough and efficient assessment on ML projects.

### Limitation & Future Improvement

While FixML provides substantial benefits, there are limitations and areas to be addressed in future development:

1. **Specialized Checklist**

Although the default checklist is general and may not cover all requirements for different ML projects, the current checklist structure allows users to edit checklist easily (shown in [Table 1](#tbl-checklist-schema)). Future development will focus on creating specialized checklists for tailored evaluations across various domains and project types. Collaboration with ML researchers is welcomed for creating specialized checklists based on specific use cases.

1. **Enhanced Test Evaluator**

As shown in session Evaluation Results, there are potential accuracy and consistency issues on the evaluation results using OpenAI gpt-3.5-turbo model (shown in [Figure 6](#fig-accu-mean-sd-plot), [Figure 7](#fig-cons-sd-box-plot)). Future improvements involves better prompt engineering techniques and support for multiple LLMs for enhanced performance and flexibility. User guidelines in prompt creation will be provided to facilitate collaboration with ML developers.

1. **Customized Test Specification**

As shown in [Figure 5](#fig-testspec), the current generator produces general test function skeletons without project-specific details. Future developments will integrate project-specific information to produce customized test function skeletons. This may further encourage users to create comprehensive tests.

1. **Further Optimization**

The cost associated with LLM usage is an important consideration for users of our tool. Future improvements will include sharing our cost data and providing calculations for estimated costs (e.g. cost per line of code). This will help users estimate their expenses and conduct a cost-benefit analysis to make informed decisions when using our tool.

By addressing these limitations and implementing future improvements, we aim for FixML to achieve better performance and contribute to the development of better ML systems, and ultimately enhance human life.

## References

Alexander, Rohan, Lindsay Katz, Callandra Moore, and Zane Schwartz. 2023. “Evaluating the Decency and Consistency of Data Validation Tests Generated by LLMs.” *arXiv Preprint arXiv:2310.01402*.

Belanger, Ashley. 2024. “Air Canada Must Honor Refund Policy Invented by Airline’s Chatbot.” Ars Technica. <https://arstechnica.com/tech-policy/2024/02/air-canada-must-honor-refund-policy-invented-by-airlines-chatbot/>.

Gawande, Atul. 2010. *Checklist Manifesto, the (HB)*. Penguin Books India.

Grand-View-Research. 2021. “Artificial Intelligence Market Size, Share & Trends Analysis Report by Solution, by Technology (Deep Learning, Machine Learning), by End-Use, by Region, and Segment Forecasts, 2023 2030.” Grand View Research San Francisco.

Jordan, Jeremy. 2020. “Effective Testing for Machine Learning Systems.” <https://www.jeremyjordan.me/testing-ml/>.

Kapoor, Sayash, and Arvind Narayanan. 2022. “Leakage and the Reproducibility Crisis in ML-Based Science.” *arXiv Preprint arXiv:2207.07048*.

Nunwick, Alice. 2023. “ITutorGroup Settles AI Hiring Lawsuit Alleging Age Discrimination.” Verdict. <https://www.verdict.co.uk/itutorgroup-settles-ai-hiring-lawsuit-alleging-age-discrimination/>.

Openja, Moses, Foutse Khomh, Armstrong Foundjem, Zhen Ming, Mouna Abidi, Ahmed E Hassan, et al. 2023. “Studying the Practices of Testing Machine Learning Software in the Wild.” *arXiv Preprint arXiv:2312.12604*.

Pineau, Joelle, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d’Alché-Buc, Emily Fox, and Hugo Larochelle. 2021. “Improving Reproducibility in Machine Learning Research (a Report from the Neurips 2019 Reproducibility Program).” *Journal of Machine Learning Research* 22 (164): 1–20.

Regidi, Asheeta. 2019. “SEBI’s Circular: The Black Box Conundrum and Misrepresentation in AI-Based Mutual Funds.” Firstpost. <https://www.firstpost.com/business/sebis-circular-the-black-box-conundrum-and-misrepresentation-in-ai-based-mutual-funds-6625161.html>.

Shepardson, David. 2023. “GM’s Cruise Recalling 950 Driverless Cars After Pedestrian Dragged in Crash.” Reuters. <https://www.reuters.com/business/autos-transportation/gms-cruise-recall-950-driverless-cars-after-accident-involving-pedestrian-2023-11-08/>.

Team, Microsoft Industry Solutions Engineering. 2023. “Testing Data Science and MLOps Code.” *Testing Data Science and MLOps Code - Engineering Fundamentals Playbook*. <https://microsoft.github.io/code-with-engineering-playbook/machine-learning/ml-testing/>.

Zhang, Yue, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, et al. 2023. “Siren’s Song in the AI Ocean: A Survey on Hallucination in Large Language Models.” <https://arxiv.org/abs/2309.01219>.