RETAIN: Interactive Tool for Regression Testing Guided LLM Migration

Tanay Dixit[1] * **Daniel Lee**[2] **Sally Fang**[2] **Sai Sree Harsha**[2]
Anirudh Sureshan[2] **Akash Maharaj**[2] **Yunyao Li**[2]
[1]University of Illinois Urbana Champaign
[2]Adobe Inc.
dixittanay@gmail.com dlee@adobe.com

Abstract

Large Language Models (LLMs) are increasingly integrated into diverse applications (Kaddour et al., 2023). The rapid evolution of LLMs presents opportunities for developers to enhance applications continuously. However, this constant adaptation can also lead to performance regressions during model migrations. While several interactive tools have been proposed to streamline the complexity of prompt engineering, few address the specific requirements of regression testing for LLM Migrations (Ma et al., 2024). To bridge this gap, we introduce RETAIN (REgression Testing guided LLM migrAtIoN), a tool designed explicitly for regression testing in LLM Migrations. RETAIN comprises two key components: an interactive interface tailored to regression testing needs during LLM migrations, and an error discovery module that facilitates understanding of differences in model behaviors. The error discovery module generates textual descriptions of various errors or differences between model outputs, providing actionable insights for prompt refinement. Our automatic evaluation and empirical user studies demonstrate that RETAIN, when compared to manual evaluation, enabled participants to identify twice as many errors, facilitated experimentation with 75% more prompts, and achieves 12% higher metric scores in a given time frame.

1 Introduction

Large Language Models (LLMs) have demonstrated proficiency in executing a wide array of complex tasks (Achiam et al., 2023; et al., 2024), which previously necessitated custom fine-tuned models. This capability has made the integration of LLMs into applications increasingly attractive, as it significantly reduces the costs associated with developing models from scratch. However, for LLMs to effectively perform these complex tasks, careful prompt design is crucial (Brown et al., 2020; Wei et al., 2022). Prompt engineering is an unstructured process that involves crafting instructions within the prompt or curating a set of in-context examples (Khattab et al., 2022). These design choices are often highly specific to the particular model being prompted.

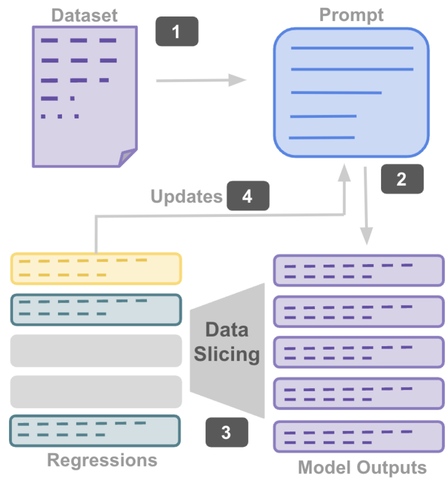

Figure 1: Regression Testing for Prompting LLMs. The process involves: (1) input dataset, (2) initial prompt, (3) data slicing algorithm to identify behavioral differences (regressions) across models, and (4) prompt refinement to address identified regressions.

The rapidly evolving landscape of LLMs, compels application developers to continually update to newer versions to maintain optimal performance. Moreover, applications utilizing LLM APIs often face forced transitions as older models are deprecated and discontinued[1]. This creates a recurring challenge of re-engineering prompts for different LLMs to achieve the same task and maintain consistent model behavior, a process we define as **LLM migration**.

Migrations to newer LLMs are difficult due to

* Work done as intern at Adobe.

[1]https://platform.openai.com/docs/deprecations

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing System Demonstrations, pages 301–310
November 12-16, 2024

model regressions (Ma et al., 2024), necessitating the development of custom tools for analyzing discrepancies in model behaviors. Such regression tests must focus both on pattern discovery for errors and systematic failure validation (Cabrera et al., 2023; Ma et al., 2024). These patterns can generally be encoded as a subgroup or "slice" of model outputs, with a corresponding metric that characterizes the observed behavior, and are often discovered in an iterative and manual manner by prompt developers (Shankar et al., 2024).

Figure 1 illustrates a high-level regression testing process for prompting, drawing parallels with software engineering techniques. The main challenge in regression testing based prompting, is to design a systematic method of identifying regressions. While numerous tools and frameworks have been developed to assist in prompt engineering, ranging from interactive platforms (Wu et al., 2022; Arawjo et al., 2024; Cabrera et al., 2023) to automated systems (Khattab et al., 2023; Zhou et al., 2022), few address the specific needs of regression testing in prompting. Existing tools often lack support for data slicing (Figure 1), which requires manual inspection to identify regressions and group data points into slices. Furthermore, current tools provide insufficient support for analyzing model behaviors at various granularities.

To bridge this gap, we propose RETAIN (**RE**gression **T**esting guided LLM migr**A**t**I**o**N**) - designed explicitly for regression testing in prompting and enables flexible analysis of model behaviors at various granularities. RETAIN aims to reduce the effort required in identifying regressions by automatically detecting differences in model behaviors across different data subsets (§4.4). Our tool features an interactive interface supporting the analysis of various prompt iterations across multiple granularity levels: aggregate metric scores, distribution analysis of metric scores, and side-by-side comparisons at the instance level (§4.3). Furthermore, RETAIN integrates prompt updating capabilities, making it a self-contained solution for the entire prompting process. Through user studies, we demonstrate that RETAIN, compared to manual prompting approaches, aids users in identifying twice as many errors, facilitates iteration over 75% more prompts, and achieves 12% higher metric scores in a given time frame.

2 Related Work

2.1 Prompting tools

Prompting has emerged as new paradigm (Liu et al., 2023) based on language models that model the probability of text directly. To effectively leverage the pre-trained knowledge of large language models (LLMs), carefully designed prompts are required (Wei et al., 2022). To facilitate analyzing and experimenting with different prompts several commercial prompting tools and libraries, such as Promptify (Pal, 2022), Lang Chain (Langchain, 2023) and Guidance (AI, 2023) have been developed. Several interactive prompting tools like Strobelt et al. (2022); Mishra et al. (2023); Wu et al. (2022) aim to reduce the workload in experimenting with several prompts. Tools like Zeno (Cabrera et al., 2023) provide support for analysing models performance on different data slices but are limited to only datasets that contain meta-data, which is often not available for majority NLP tasks. A new emergent area involves automatic prompt engineering techniques (Khattab et al., 2023; Yuksekgonul et al., 2024) which aim to treat the prompting process as an optimization task.

2.2 Exploratory Analysis and Automated Discovery

Automatic pattern discovery is a well studied problem with several classical methods in ML (Manning and Schutze, 1999) such as topic modeling (Blei et al., 2003) to extract major topical variations. Our task is different from these traditional settings as it requires error discoveries in the form of natural language predicates, which are interpretable and can express abstract concepts. Several works like Zhong et al. (2023); Wang et al. (2023); Zhong et al. (2022) show that LLMs are capable of extracting distributional differences between two text corpora. We leverage these ideas for building our data slicing module (Figure 2-D).

3 User Challenges in Regression Testing for LLM Migrations

To understand users' workflows in regression testing for LLM Migrations, we conducted a formative study and collaborative design process, adapted from the methodology described in (Zhang et al., 2022). Our study included semi-structured interviews with researchers and engineers, focusing on their experiences in LLM Migrations.

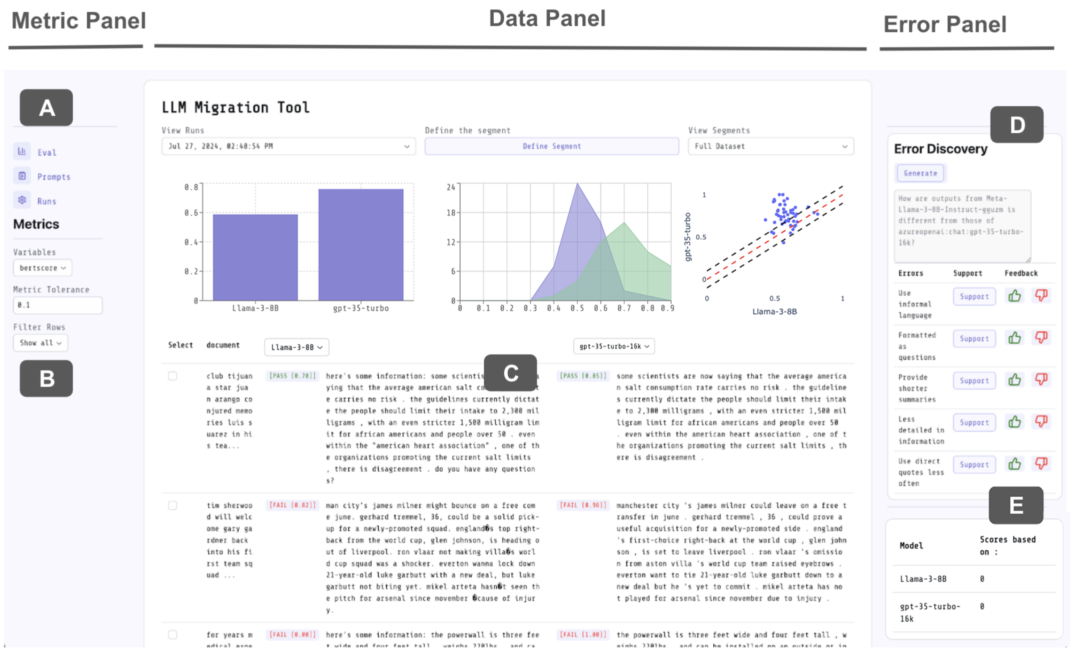

Figure 2: RETAIN comprises of three main Panels: Metric Panel, Data Panel, and Error Analysis Panel. It features three pages (A) designed for various prompt engineering tasks, (B) Users can set metrics, (C) compare model outputs through charts and side-by-side comparisons, and (D) conduct in-depth analysis of failure cases using the error discovery module. Additionally, users can define LLM assertions to evaluate outputs across different prompts. (E)

Our findings revealed several key challenges: difficulty in identifying differences in model outputs (regressions), struggle to understand causes of variations in metric scores, and lack of systematic tracking for the effects of prompt edits on model outputs. In cases of migrations, ensuring consistent LLM behavior is critical, underscoring the importance of regression testing. Based on these insights, we identified three primary design goals:

- **DG1:** Develop methods to automatically identify behavioral changes across prompts or models, and intelligently suggest data slices, especially when metadata is unavailable.
- **DG2:** Provide tools for examining LLM behavior at various levels, from aggregate metrics to individual instance comparisons, supporting diverse analytical needs.
- **DG3:** Integrate capabilities for systematic tracking and analysis of prompt modifications, enabling users to iterate and improve prompts based on regression testing results.

4 System

In this section, we demonstrate RETAIN using a scenario where a researcher or engineer utilizes our tool for LLM migration in the task of prompt migration (Ma et al., 2024) for a summarization task (Hermann et al., 2015). The user is migrating a prompt optimized for `gpt3.5-turbo-16k` to `Llama-3-8b`. It's important to note that RETAIN is versatile and applicable to any prompt engineering setup. The user initiates the process by creating a simple declarative configuration file (detailed in Appendix §A). This file contains essential information such as model names, access keys, initial prompts, metrics, and test data (Promptfoo, 2023). With this configuration in place, the user can launch the RETAIN tool. For implementation details, readers are directed to Appendix A.

4.1 Pages

RETAIN consists of three tabs: (1) Eval, (2) Prompts, and (3) Runs (Figure 2-A). The Eval Page comprises three key panels: (i) Metric Panel, (ii) Data Panel, and (iii) Error Analysis Panel. The Prompts page (Figure 6) displays the model's prompt, which in this case is the prompt for `Llama-3-8B` model. For the task of migration, the user begins with the same prompt as `gpt3.5-turbo` and iteratively refines it to optimize the Llama prompt, aiming to achieve behavior comparable to `gpt3.5-turbo`. The Runs page (Figure 7) offers a tabular view of the metric scores for both models. This structure is designed to provide a comprehensive overview of the prompt engineering process, offering users a bird's-eye view of the entire migration workflow.

4.2 Metrics Panel

The Metrics Panel displays all user-defined metrics from the configuration file within the Metrics Card's variables toggle (Figure 2-B). To address the challenge of non-determinism in LLM regression testing (Ma et al., 2024), we introduce the concept of Metric Tolerance. This feature is analogous to confidence intervals in hypothesis testing and represents the acceptable margin of difference between two metric scores for them to be considered equivalent. The panel features a dropdown menu for filtering the data table to display only test data points where metric score differences exceed the set tolerance. This enables users to focus on discrepancies between model outputs, aiding in efficient analysis and debugging.

4.3 Data Panel

The Data Panel (Figure 2-C) consists of aggregate-level visualizations and instance-level side-by-side comparisons (DG2)

Visualizations The panel incorporates three visualizations to facilitate model analysis. First, the Aggregate Metric Score Chart provides a performance summary. However, recognizing that aggregate scores may not fully capture model behavior (Cabrera et al., 2023; Ribeiro et al., 2020), we include additional visualizations. The Metric Score Distribution Chart allows users to compare the distribution of metric scores between the models. Lastly, the Regressions Chart (Promptfoo, 2023), designed to address our goal of regression-based prompting.

Side-by-Side Comparisons To complement the aggregate visualizations, we provide instance-level comparisons through a side-by-side tabular interface. This feature is crucial to identify specific slices of interest and observe qualitative patterns in model outputs (Kahng et al., 2024). By allowing direct comparison of individual instances, users can gain deeper insights into the model's behavior.

4.4 Error Analysis Panel

A significant challenge in prompt engineering is understand why and where the model performs poorly with respect to the given metrics (DG1). To address

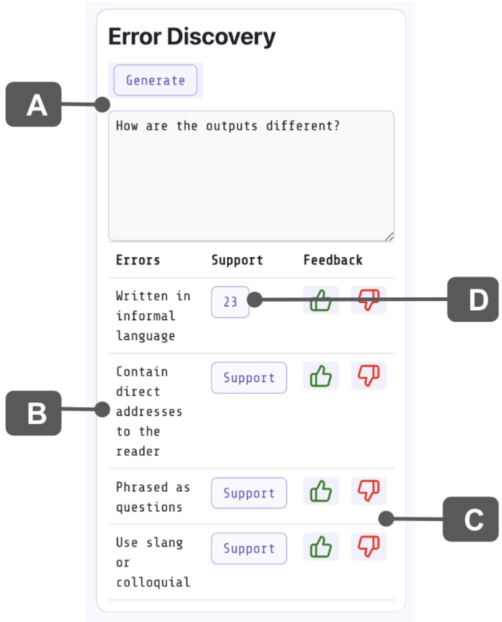

Figure 3: Error Discovery Module Interaction. (A) Users initiate error generation to identify discrepancies among model outputs in the side-by-side comparison table. (B) For errors of interest, users can employ the support feature (D) to highlight specific model outputs containing the selected error type. (C) The thumbs up/down feature allows users to create or remove custom LLM metrics based on error descriptions.

this, we introduce Goal-driven error discovery, designed to streamline the error identification process and facilitate targeted prompt refinements.

Goal-Driven Error Discovery Figure 3 shows the various interactions with the module. Our error discovery module, inspired from Zhong et al. (2022) and Zhong et al. (2023), aims to identify distributional differences between model outputs that are relevant to user-defined goals. This approach not only helps users understand why the model is under performing on given metrics but also provides textual descriptions of errors, which can be directly incorporated into subsequent prompt edits. To help users identify the model outputs containing a given error type, we employ a *selector module*. The *selector module* highlights the model outputs containing the specific error in the side-by-side comparison tables. We implement two distinct pipelines for these tasks. For building the goal-oriented error discovery, we prompt (Table 3) GPT-4 to identify differences between the groups of outputs of the two models for a given goal. For the *selector module* for every model output, we prompt (Table 4) GPT-3.5 to classify whether the outputs contains the given error or not. Additional implementation details in Appendix B.

Defining LLM Assertions Shankar et al. (2024) emphasize the importance of LLM assertions in detecting data quality errors made by language models. Building on this concept and Zheng et al. (2024), we enable users to define custom LLM-based metrics that specifically evaluate errors of interest. Users can create these metrics by clicking on the thumbs-up icon (Figure 3-C) associated with a particular error description. In formulating these metrics, we incorporate the error descriptions to ensure relevance and specificity. Additional implementation details and we adopt the prompts from Kim et al. (2024) for this task.

4.5 Features for Iterative Prompt Engineering

To support the iterative nature of prompt engineering (DG3), we offer several additional features. The *View Runs* feature in the Data Panel (Figure 2) enables users to track and compare performance across different prompt versions. The *Define Segments* feature helps users define custom data slices and persist them across runs (DG2), addressing the need for fine-grained performance analysis identified in our formative studies. Users can customize which model outputs are displayed in the side-by-side comparison tables. This feature, combined with the error discovery module, allows for detailed analysis of how prompt edits affect model behavior across subgroups of data for different versions.

5 Evaluations

To evaluate our system comprehensively, we employ two approaches: (1) an automatic evaluation (§5.1) to assess the accuracy of our LLM-based error discovery method in detecting distributional differences between model outputs, and (2) a user study (§5.2) to compare RETAIN's impact on the prompt migration process against current practices.

5.1 Automatic Evaluations

The goal-oriented error discovery module is designed to streamline the identification of differences between model outputs. Evaluating such a system poses significant challenges due to the unsupervised nature of error discovery and the absence of labelled data. To address this, we develop a synthetic dataset to assess the system's ability to recover known differences between two artificially constructed corpora. This approach allows us to quantitatively evaluate the effectiveness of our error discovery mechanism in a controlled setting.

5.1.1 Dataset Generation and Metrics

We follow a methodology similar to Zhong et al. (2023) to evaluate the error discovery module. We employed a LLM to generate two corpora (A and B) that differ along two dimensions: a goal-relevant dimension and a distractor dimension. For example, if the goal is to *understand how Corpus A differs from Corpus B in terms of topic*, then we would synthesize Corpus A to be on politics while Corpus B on sports (goal-relevant dimension being varying topic). Additionally, we would vary the corpus on another dimension eg. writing style (distractor dimension). Corpus A would be more informal while Corpus B would be formal. The system's task is to identify the goal-relevant dimension i.e., the topic. The process of generating the dataset involves randomly sampling both dimensions from a predefined set of attributes. Corpus A and B were generated such that all samples incorporated the distractor dimension, while a fixed percent of the samples also incorporated the goal-relevant dimension. We synthesized 100 test data points to create our evaluation dataset. For evaluation, we adopted the metrics used by Zhong et al. (2023). We used Error Relevance to assess the module's effectiveness in generating errors relevant to the gold error type. To evaluate the selector module (Error Coverage), we employed precision and recall metrics to evaluate the module's ability to identify data points in the corpora containing the given error type.

5.1.2 Performance Analysis

Table 1 shows how the goal-oriented error discovery module significantly enhances the detection of relevant errors, compared with a baseline prompting approach (see Appendix B for details). Regarding the identification of data points with specific errors, the system demonstrates higher precision (0.69) compared to recall (0.38). This higher precision is particularly beneficial in our context, as it ensures that the system highlights rows that are highly likely to contain the error in question, reducing the burden on users by minimizing the number of rows requiring manual inspection.

Metric	w/ goal	w/o goal
Error Relevance	0.87	0.72
Error Coverage		
- Precision	0.69	0.70
- Recall	0.38	0.36

Table 1: Performance Evaluation of Goal-Oriented Error Discovery. The incorporation of user-defined goals substantially enhances the accuracy of error detection, demonstrating the efficacy of our approach in identifying relevant discrepancies between model outputs.

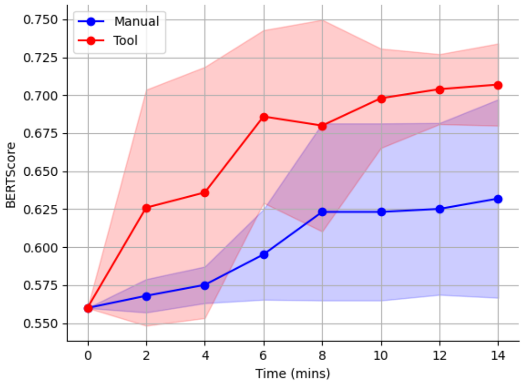

Figure 4: BERTScore Progression Over Time. The solid line is the average score while the shaded region is the standard deviation. We can observe that using our tool participants could achieve higher scores in lesser time.

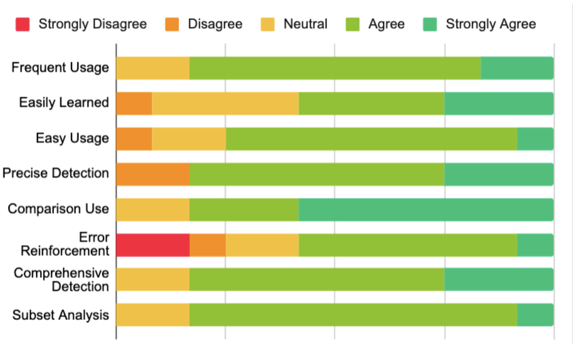

Figure 5: Post-Study Psychometric Evaluation Results. The x-axis labels are simplified for readability and the full questions are available in Section §C.2.

5.2 User Study

To evaluate RETAIN, we conducted a comprehensive two-phase user study designed to assess two critical aspects of our system across 12 participants proficient in prompt engineering. This dual-phase approach allows us to examine both the analytical capabilities of RETAIN and its practical application in real-world prompt engineering scenarios.

5.2.1 Phase 1: Error Identification Task

Phase 1 involved a within-subject study on error identification. Participants had 15 minutes per set to identify and note types of errors between two model outputs. We created a dataset with manually injected errors based on a typical LLM error taxonomy, validated by two independent NLP experts. Participants used both manual (Excel) and RETAIN-assisted methods for error identification. This design compared RETAIN's efficiency and accuracy against traditional methods in detecting and categorizing LLM output discrepancies, aiming to evaluate our system's potential improvements in error detection and classification.

5.2.2 Phase 2: End-to-End Prompt Engineering Experience

Phase 2 used a between-subject design to evaluate prompt engineering, focusing on performing LLM Migrations. Participants had 15 minutes to migrate a prompt optimized for `gpt-35-turbo` to `llama-3-8b`. Group A used a standard jupyter notebook, while Group B used RETAIN, allowing comparison of RETAIN's effectiveness against traditional methods in prompt engineering. After exploring RETAIN, participants completed a post-screen survey using a 5-point Likert scale to assess usability, functionality, utility, cognitive load, and overall satisfaction.

5.2.3 Results

RETAIN significantly outperformed traditional methods in regression testing guided prompt engineering. It identified nearly twice as many errors (165 vs 86) and covered more error categories (2.56 vs 2.22 average). RETAIN-refined prompts achieved higher BERT scores (0.704 vs 0.625) (Figure 5), improving scores by 25% compared to 12% manually within the given timeframe. Users could also experiment more with RETAIN (4.55 vs 2.6 prompt edits). Psychometric evaluation reinforced these findings, with 76.04% positive responses and 83% intending frequent use. Users praised RETAIN's efficiency in data processing, component analysis, and model comparison.

6 Conclusion

We present RETAIN- a tool for regression testing guided LLM Migration. RETAIN comprises of an interactive prompting interface tailored to regression testing needs, and an error discovery module that facilitates understanding differences in model outputs. The tools aims to help users in understanding where and why models score poorly on given metrics. Our user study indicated that the tool enables users identify twice as many errors, iterate with more prompt versions and achieve a higher score on evaluation metrics within the same time frame. We hope that our easy to setup, self-contained tool will facilitate broader adoption among those involved in LLM migration tasks.

7 Limitations

Our user study revealed several opportunities to further enhance RETAIN's analytical capabilities:

- On-the-Fly Metric Creation: Users expressed a desire to create rule-based metrics during analysis to deterministically catch specific error types. This could be implemented using regex-based filtering, allowing for more flexible and immediate error detection.

- Prompt Edit Suggestions: Currently, RETAIN doesn't provide automated prompt edit suggestions. Incorporating automatic prompt engineering techniques, as demonstrated by Khattab et al. (2023), could significantly accelerate the prompt migration process.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Guidance AI. 2023. guidance.

Ian Arawjo, Chelse Swoopes, Priyan Vaithilingam, Martin Wattenberg, and Elena L Glassman. 2024. Chainforge: A visual toolkit for prompt engineering and llm hypothesis testing. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–18.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3(null):993–1022.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Ángel Alexander Cabrera, Erica Fu, Donald Bertucci, Kenneth Holstein, Ameet Talwalkar, Jason I Hong, and Adam Perer. 2023. Zeno: An interactive framework for behavioral evaluation of machine learning. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–14.

Meta AI et al. 2024. The llama 3 herd of models.

Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *NIPS*, pages 1693–1701.

Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. 2023. Challenges and applications of large language models. *arXiv preprint arXiv:2307.10169*.

Minsuk Kahng, Ian Tenney, Mahima Pushkarna, Michael Xieyang Liu, James Wexler, Emily Reif, Krystal Kallarackal, Minsuk Chang, Michael Terry, and Lucas Dixon. 2024. Llm comparator: Visual analytics for side-by-side evaluation of large language models. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pages 1–7.

Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. 2022. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive NLP. *arXiv preprint arXiv:2212.14024*.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*.

Tae Soo Kim, Yoonjoo Lee, Jamin Shin, Young-Ho Kim, and Juho Kim. 2024. Evallm: Interactive evaluation of large language model prompts on user-defined criteria. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA. Association for Computing Machinery.

Langchain. 2023. Langchain.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

Wanqin Ma, Chenyang Yang, and Christian Kästner. 2024. (why) is my prompt getting worse? rethinking regression testing for evolving llm apis. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI*, pages 166–171.

Christopher Manning and Hinrich Schutze. 1999. *Foundations of statistical natural language processing*.

Aditi Mishra, Utkarsh Soni, Anjana Arunkumar, Jinbin Huang, Bum Chul Kwon, and Chris Bryan. 2023. Promptaid: Prompt exploration, perturbation, testing and iteration using visual analytics for large language models. *arXiv preprint arXiv:2304.01964*.

Ankit Pal. 2022. Promptify: Structured output from llms. https://github.com/promptslab/Promptify. Prompt-Engineering components for NLP tasks in Python.

Promptfoo. 2023. Promptfoo.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of nlp models with checklist. *arXiv preprint arXiv:2005.04118*.

Shreya Shankar, Haotian Li, Parth Asawa, Madelon Hulsebos, Yiming Lin, JD Zamfirescu-Pereira, Harrison Chase, Will Fu-Hinthorn, Aditya G Parameswaran, and Eugene Wu. 2024. Spade: Synthesizing assertions for large language model pipelines. *arXiv preprint arXiv:2401.03038*.

Hendrik Strobelt, Albert Webson, Victor Sanh, Benjamin Hoover, Johanna Beyer, Hanspeter Pfister, and Alexander M Rush. 2022. Interactive and visual prompt engineering for ad-hoc task adaptation with large language models. *IEEE transactions on visualization and computer graphics*, 29(1):1146–1156.

Zihan Wang, Jingbo Shang, and Ruiqi Zhong. 2023. Goal-driven explainable clustering via language descriptions. *arXiv preprint arXiv:2305.13749*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. 2022. Promptchainer: Chaining large language model prompts through visual programming. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–10.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. Textgrad: Automatic "differentiation" via text.

Zheng Zhang, Ying Xu, Yanhao Wang, Bingsheng Yao, Daniel Ritchie, Tongshuang Wu, Mo Yu, Dakuo Wang, and Toby Jia-Jun Li. 2022. Storybuddy: A human-ai collaborative chatbot for parent-child interactive storytelling with flexible parental involvement. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA. Association for Computing Machinery.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36.

Ruiqi Zhong, Charlie Snell, Dan Klein, and Jacob Steinhardt. 2022. Describing differences between text distributions with natural language. In *International Conference on Machine Learning*, pages 27099–27116. PMLR.

Ruiqi Zhong, Peter Zhang, Steve Li, Jinwoo Ahn, Dan Klein, and Jacob Steinhardt. 2023. Goal driven discovery of distributional differences via language descriptions. In *Advances in Neural Information Processing Systems*, volume 36, pages 40204–40237. Curran Associates, Inc.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*.

A Additional Details

Tool Implementation Details RETAIN is a web-based application.The entire tool was implemented using Python. For the user interface we used Reflex[2] while for the backend we made use of Langchain and litellm to query the various LLMs.

```
# prompts...
prompts:
  - "Summarize this document"
  - "Summarize this document, concisely and professionally:"
# models...
providers:
  - openai:gpt-35-turbo-16k
  - meta-llama-3-8b
# tests cases
tests:
  - vars:
     document: "file://docs.txt"
    assert:
      - type: bleu
        value: "Summary ... "
      - type: bertscore
        value: "Summary ... "
```

Table 2: Example of a configuration file used to setup RETAIN.

B Error Discovery Implementation Details

The goal-oriented approach has two prompts. The prompt used for generating the errors is in Table 3 and for selecting the model outputs in Table 4. For the generator prompt, it is possible that for some instances all the model outputs might not fit into one prompt, hence we construct multiple prompts with different sets of samples so that GPT-4 can "see" all the different model outputs. We set temperature to be 0 for both the tasks. The baseline (non-goal oriented approach) used the prompt described in Table 5. For generating the synthetic evaluation dataset, we use the following attributes set *topic, writing style, stance, language, formatting, and country* and V was varied from 0.6 to 1.0. We prompted GPT-4 to generate the outputs.

C User Study Details

C.1 Participant Recruitment

We recruited 12 participants for this study, each with at least two years of experience in ML Engineering or prompt engineering with LLMs. All participants were ML Engineers or Research Scientists from industrial settings, regularly working with LLMs for task-oriented use cases. Recruitment was conducted via an internal messaging service, dissemintated to individuals who had no conflicting interest. Participants were selected based on their expertise to ensure informed feedback on the LLM Migration tool. All interviews were conducted in person. Compensation included a single-meal voucher or gift of equivalent value in Califor-

[2] https://reflex.dev/

Given two groups of inputs (Group A and Group B) and a Question, your task is to identify differences that make the groups different according to the specific question. Each input in a group starts with the token [ITEM].
Follow these guidelines:
1. Only generate differences that help answer the question provided.
2. Only generate 4-5 words description for each difference.
3. Each difference description should start on a new line.
4. Each difference should be unique and relevant to the question provided.
5. If there are no differences that make the groups different according to the question, output 'There are no differences that make the groups different according to the question provided'.
Group A: {{Corpus A}}
Group B: {{Corpus B}}
Question: goal
Compared to outputs in Group A, more outputs in Group B

Table 3: Prompt used to generate the various errors as part of the goal oriented error discovery module.

Given two groups of outputs (Model A and Models B) and a Question, your task is to identify textual differences that answer the specific question. Each output in a model starts with the token [ITEM].
Follow these guidelines:
1. Only generate differences that help answer the question provided.
2. Only generate 4-5 words description for each difference.
3. Each difference description should start on a new line.
4. Each difference should be unique and should help answer the question provided.
5. If there are no differences that make the groups different according to the question, output 'There are no differences that make the groups different according to the question provided'.
Model A Outputs: {{Corpus A}}
Model B Outputs: {{Corpus B}}
Question: {{goal}}
To answer the question, we can see that, compared to outputs from Model A, more outputs from Model B are

Table 4: Prompt used to select the various model outputs which contain a given error type.

Figure 6: Prompts Page: The user can edit/update the model prompts using the Prompts tab.

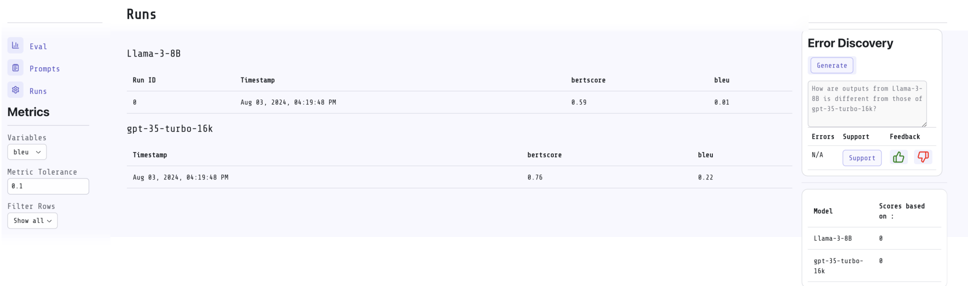

Figure 7: Runs Page: This page provides a tabular visualization of the various prompt versions.

Given two groups of inputs (Group A and Group B), identify all stylistic, syntactic and semantic differences that make the groups different. Some possible differences could be common words, phrases, or patterns in writing style that are present in one group but not in the other group. Each input in a group starts with the token [ITEM]. Only generate 4-5 words description for each difference, and each difference description should start on a new line. Ensure to cover all the above 3 categories of differences. Do not output descriptions that start with words like 'In Group A' or 'Group B ..'.
Group A: {{set_a}}
Group B: {{set_b}}
Compared to outputs in Group A, majority outputs in Group B

Table 5: Prompt used as a baseline to find differences between two groups. This is a standalone, non-goal-oriented prompt

nia.

C.2 Post User Survey Questions

- I think I would like to use this system frequently.
- I would imagine that most people would learn to use this system very quickly.
- I found the system very easy to use.
- The error discovery module helped me identify errors quickly.
- This tool could be useful for comparing two LLMs.
- The error discovery module helped reinforce the errors I had observed.
- The error discovery module helped me to quickly identify the data points with the a common error.
- The tool provided support to analyze different subsets of the data according to the user needs.